mac5811

data

## • PROVAS DE DOUTORADO

| | | | |
|---|---|---|---|
| 1- | 1997/1 | Pág 1 | (J. A. Soares) |
| 2- | 1997/2 | " 15 | |
| 3- | 2006/1 | " 25 | (ours) |
| 4- | 2006/2 | " 38 | |
| 5- | 2005/1 | " 48 | (coelho) |
| 6- | 2005/2 | (não houve exame) | (mandel) |
| 7- | 1998/1 | Pág. 56 | |
| 8- | 1998/2 | " 65 | (Pf) |
| 9- | 2004/2 | " 75 | (carlinhos) |
| 10- | 2004/1 | " 83 | (A. mandel) |
| * 11- | 2007/1 | " 92 | (coelho) |
| * 12- | 2007/2 | " 101 | (cris) |
| 13- | 2003/2 | " 111 | (carlinhos) |
| 14- | 2003/1 | " 117 | |
| 15- | 1999/1 | " 126 | |
| 16- | 1999/2 | " 130 | (cris) |
| * 17- | 2008/1 | " 136 | (J. A. Soares) |
| 18- | 2000/1 | " 154 | (I. Simon) |
| 19- | 2000/2 | " 161 | (P. f) |
| 20. | 2001/1 | " 175 | (I. Simon) |
| 21. | 2001/2 | " 181 | |
| 22. | 2002/1 | " 184 | (P. f) |
| 23. | 2002/2 | " 187 | |

NOT APPROVED

data 1

PROVA DOUTORADO 97/1 (José Soares)

(1) Prove o disprove

(a) Se $f = O(g) \Rightarrow 2^f = O(2^g)$

é falso. Tome $f = n$ e $g = n/2$, assim

$n \leq 4\frac{n}{2}$ $(c = 4)$

Entretanto não é verdade que $2^n \leq d\,2^{n/2}$ para uma constante $d$ e $n \geq n_0$

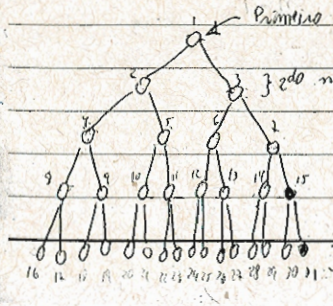
$d \geq 2^{n/2}$ ?

(b) Se $f = O(g) \Rightarrow f = O(f+g)$

- Sim, é verdade (para funções crescentes)
- PROVA: Por definição sabemos que $f(n) \leq c\,g(n)$

$$f(n) \leq c_1 g(n) \leq c_2 g(n) + c_2 f(n)$$
$$\leq c_3 (g(n) + f(n)) \qquad c_3 = \max\{c_1, c_2\}$$
$$= O(g(n) + f(n))$$

(2) Quais posições de um vetor organizado como um HEAP máximo de $(2^n - 1)$ elementos distintos poderiam ser ocupados pelo 3$^{\text{ro}}$ maior elemento? E o terceiro menor elemento? (distintos 2-a-2) (suponha $n \geq 3$)




3$^{\text{ro}}$ maior $\Rightarrow$ Posições: $[2..7]$

3$^{\text{ro}}$ menor $\Rightarrow$ Posições $[2^{n-2}..2^n - 1]$

SCANPIX

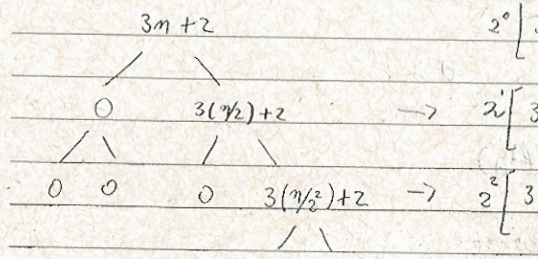
(3) Resolva a recorrência. Você pode supor que $n = 2^k$ para algum inteiro positivo $k$.

$T(1) = 4$

$T(n) = 2T(n/2) + 3n + 2$

SOLUÇÃO:

• Árvore de recorrência




$2^0\left[3\frac{n}{2^0} + 2\right] = 3n + 2.2^0$ (i=0)

$\rightarrow 2^1\left[3\frac{n}{2} + 2\right] = 3n + 2.2^1$

$\rightarrow 2^2\left[3\frac{n}{2^2} + 2\right] = 3n + 2.2^2$

$\rightarrow 2^i\left[3\frac{n}{2^i} + 2\right] = 3n + 2.2^i$

no último nível $\frac{n}{2^i} = 1 \Rightarrow i = \lg n$

Assim $T(n) = \sum_{i=0}^{\lg n - 1}\left(3n + 2.2^i\right) + 2^{\lg n}.4$

$= 3n\lg n + 2\sum_{i=0}^{\lg n - 1} 2^i + 4n$

$= 3n\lg n + 2\left(2^{\lg n} - 1\right) + 4n$ (→ n − 1)

$T(n) = 3n\lg n + 6n - 2$





em $k$, com $n = 2^k$, para $k = 0, 1, 2, \ldots$

• Prova por indução $T(n) = 3n\lg n + 6n - 2$

- Base: $k = 0$, então $n = 1$

$T(1) = 6 - 2 = 4$ (corresponde ao valor da base da recorrência)

- Passo: $k > 0$

$T(n) = 2T(n/2) + 3n + 2$

$= 2\left[3\frac{n}{2}\lg(n/2) + 6\left(\frac{n}{2}\right) - 2\right] + 3n + 2$

$= 3n(\lg n - 1) + 6n - 4 + 3n + 2$

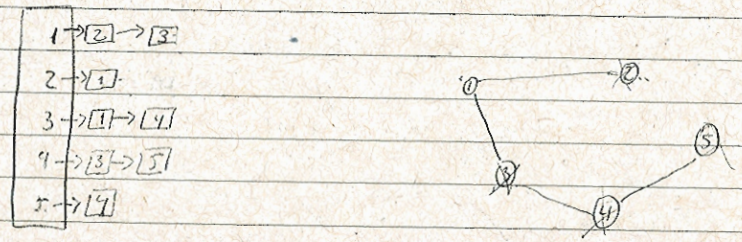
$= 3n\lg n - 3n + 6n - 2 + 3n$

$= 3n\lg n + 6n - 2$ #

Logo $T(n) = O(n\lg n)$

(4) Descreva um algoritmo que, em tempo $O(n)$, determina se um grafo com $n$ vértices, dado por sua lista de adjacências, é uma árvore.
NOTA: "Tempo linear no tamanho da entrada não é suficiente."

1. Seja $G=(V,E)$ tamanho da entrada $(|V|, |E|)$
- Lista de adjacências
- Estratégia: Fazer uma busca em largura, marcando os vértices visitados.







VERIFICA-ARVORE$(V,E)$
1. $n \leftarrow |V|$
2. para $i \leftarrow 1$ até $n$ faça
3. $P[i] \leftarrow 0$
4. VISITA$(1, 0)$ ▷ assumindo vértices $[1..n]$
5. contador $\leftarrow 0$
6. para $i \leftarrow 1$ até $n$ faça
7. se $P[i] = 1$
8. então contador $\leftarrow$ contador $+1$
9. se contador $= n$
10. então devolva VERDADEIRO
11. senão devolva FALSO

$O(|V|) = O(n)$

VISITA$(u, du)$
1. $P[u] \leftarrow P[u] + 1$
2. se $P[u] < 2$
3. então para cada $v \in adj[u]$ faça
4. se $v \neq du$
5. então VISITA$(v, u)$

$O(|V|)$





(5) Esquematizar a prova de que a ordenação de $n$ inteiros toma tempo $\Omega(n \lg$

- A ordenação baseada em comparações de $n$ inteiros toma tempo $\Omega(n \lg n)$.
- Supondo que $\exists$ uma árvore de decisão que permite representar as comparações necessárias. Cada nó da árvore representa uma comparação E cada folha uma ordenação do vetor de entrada. A altura representa # de compa
- Observemos que o número de folhas de uma árvore binária de altura $h$ será $2^h$
- Por outro lado o número de permutações de um vetor de $n$ elementos é $n!$

→ O número de folhas de uma árvore binaria de altura $h$ devera ser maior o igual ao numero de permutações

$$2^h \geq n!$$

$$\boxed{h \geq \lg(n!)}$$

$$(n!)^2 = \underbrace{n \cdot (n-1)}_{1} \underbrace{(n-2)}_{2} \underbrace{\cdots}_{3} \underbrace{(2)}_{(n-1)} \underbrace{(1)}_{(n)}$$

$$= \prod_{i=0}^{n-1} (i+1)(n-i) = \prod_{i=0}^{n-1} (in - i^2 + n - i) = \prod_{i=0}^{n-1} \left[ n + \underbrace{i(n-i-1)}_{\geq 0} \right]$$

$$\geq \prod_{i=0}^{n-1} n = n^n$$

Assim $n! \geq n^{n/2}$

Logo $h \geq \lg(n!) \geq \frac{n}{2} \lg(n)$

$$\boxed{h = \Omega(n \lg n)}$$

(5b) Descreva um algoritmo que, em tempo $O(n)$, ordena $n$ inteiros, cada um deles em $\{1, 2, \ldots, n^3\}$

# dígitos

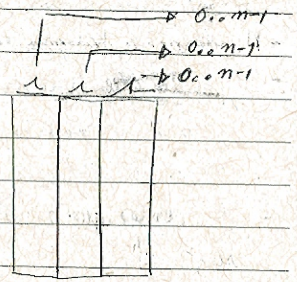
$10 \rightarrow \log_{10}(10) + 1 = 2$

$100 \rightarrow \log_{10}(100) + 1 = 3$

$n \rightarrow \log_n(n) + 1 = 2$

$n^3 \rightarrow \log_n(n^3) + 1 = 4$

↳ número constante de dígitos




ALGORITMO:

1º Converter os $n$ inteiros à base $n$, assim cada número terá um # constante de dígitos

2º Aplicar o algoritmo de ordenação ESTÁVEL (baseado em comparações) como o RADIX-SORT sobre os números.
(Desde o número menos significativo ao mais significativo)

COMPLEXIDADE:

1. $\Theta(n)$
2. $k\ \Theta(n)$ onde $k$ é uma constante

$T(n) = O(kn) = O(n)$

(5c) Não contradiz. Em (b) sabe-se a priori os números que serão tratados. Portanto pode ser feito um alg. LINEAR. Em (a) não se assume nada.

Cabe resaltar que se $\{1, \ldots n^n\} \Rightarrow O(n^2)$.



(6) Considere o problema de se determinar se uma seq. de 0's e 1's tem 2 0's consecutivos.
A única operação permitida sobre a seq é uma função consulta(i) que devolve 0 se o elemento da seq. é 0 e 1 em caso contrário.

(6a) Note que se $n = 1 \bmod 3$ então existe um alg. que faz no máximo $n-1$ comparações.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | | 0 | | | | 0 | | 0 | | | | 0 | | 0 | |

| | ⓔ | m1 | m2 | ⓔ | # accesses |
|---|---|---|---|---|---|
| (a) | X | 1 | 1 | X | 2 |
| (b) | X | 1 | 0 | X | 3 |
| (c) | X | 0 | 1 | X | 3 |
| (d) | X | 0 | 0 | X | 2 |

• Vale destacar que para o caso (b) precisa-se armazenar o valor de X examinado, para ser usado na próxima iteração.

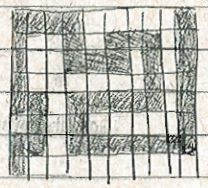
BUSCA-ZEROS $(A, n)$

```
1.  ant ← -1
2.  para i ← 2 até n-2 faça
3.      m1 ← consulta(i)
4.      m2 ← consulta(i+1)
5.      se m1 = 0 e m2 = 0
6.          então devolva VERDADEIRO
7.      se m1 = 1 e m2 = 0
8.          então e ← consulta(i+2)      ▷ consulta extremo direito
9.              se e = 0
10.                 então devolva VERDADEIRO
11.                 senão ant ← e
12.     se m1 = 0 e m2 = 1
13.         então se ant ≠ -1
14.                 então e ← ant
15.                 senão e ← consulta(i-1)
16.             ant ← -1
17.             se e = 0
18.                 então devolva VERDADEIRO
```

(6b) Monte um alg. que no máximo n consultas resolve o problema para qualquer n

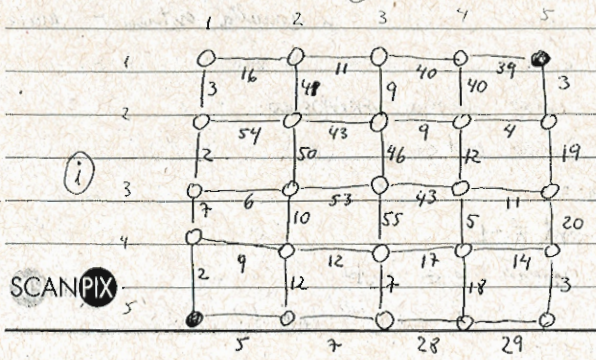
```
ALGORITMO (A, n)                    ▷ para n ≥ 2
1. t ← consulta(1)
2. para i ← 2 até n faça
3.     a ← consulta(i)
4.     se t = 0 e a = 0
5.         então devolva VERDADEIRO
6.         senão t ← a
```

(7) Escreva um alg. que recebe uma matriz M nxm de inteiros e encontra um caminho de (1,n) até (n,1) de forma a minimizar a soma dos valores absolutos das diferenças entre elementos consecutivos do caminho



- Programação dinâmica não seria uma abordagem correta, mas sim grafos, aplicando o algoritmo de DIKSTRA.

(i) $(i,j) = (i-1)*m + j$




Posições importantes:
→ 5 origem
→ 21 destino

Problema: Achar o caminho mais curto do nó 5 até o 21

SCANPIX



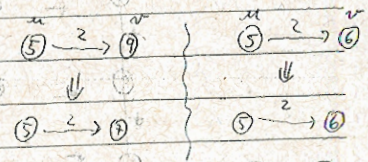
```
CRIAR_LISTA_ADJACENCIAS (A, n)
1. Crie uma lista de pesos w
2. para i ← 1 até n faça
3.     para j ← 1 até n faça
4.         ve ← (i-1)*n + j
5.         se i ≥ 1                          ▷ AGREGAR adj[ve] ← (i-2)*n+j
6.             então w(ve, (i-2)*n+j) ← |M(i-1,j) - M(i,j)|
7.         se i < n
8.             então w(ve, i*n+j) ← |M(i+1,j) - M(i,j)|
9.         se j ≥ 1
10.            então w(ve, ve-1) ← |M(i,j-1) - M(i,j)|
11.        se j < n
12.            então w(ve, ve+1) ← |M(i,j+1) - M(i,j)|
13. devolva lista de pesos w
```

Relaxamento: Para cada vértice $v \in V$ é mantido um atributo $d[v]$, que é um limite superior sobre o peso de um caminho mais curto desde a origem $s$ até $v$.

```
INITIALIZE-SINGLE-SOURCE (G, s)
1. para cada vértice v ∈ V[G] faça
2.     d[v] ← ∞
3.     π[v] ← NIL
4. d[s] ← 0                                  O(n)
```

```
RELAX (u, v, w)
1. se d[v] > d[u] + w(u,v)
2.     então d[v] ← d[u] + w(u,v)
3.           π[v] ← u                        O(1)
```

relax(u,v)
relaxar: consiste em testar se podemos melhorar o caminho mais curto de v encontrado até agora pela passagem através de u, e neste caso, atualizar d[v] e π(v)



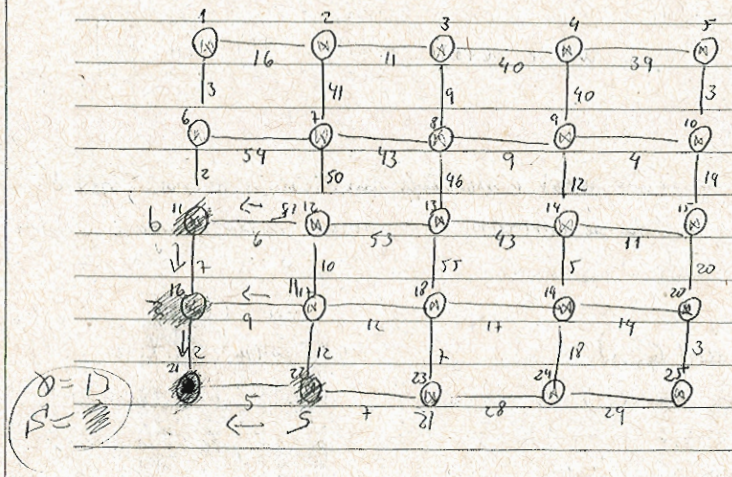

SCANPIX

data

atualizado na primeira etapa → 21

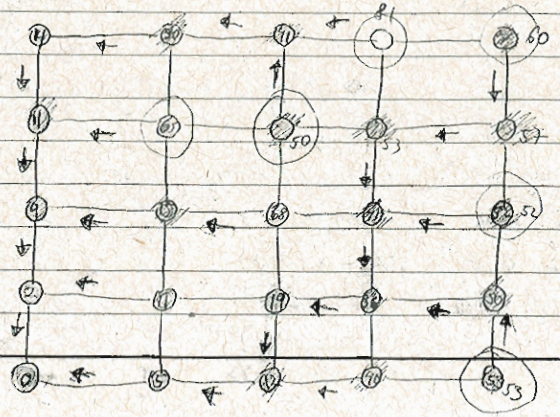
Válido para pesos não negativos

```
DIKSTRA (G, w, s)
1.  INITIALIZE-SINGLE-SOURCE (G, s)
2.  S ← ∅                          ▷ S, gpo de vértices cujos pesos de caminhos mínimos
3   Q ← V[G]                         desde s já foram determinados
4   enquanto Q ≠ ∅ faça            ▷ Q = V - S
5         u ← EXTRACT-MIN(Q)
6         S ← S ∪ {u}
7         para cada vértice v ∈ Adj[u]
8               RELAX(u, v, w)
```



D = 21

Q = D
S =



O exemplo do prova 97/1 está errado!





(8)

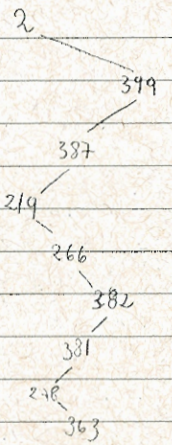
```
ALGORITMO (M, n)
    cria vetor de pesos w e lista de adjacências        Θ(n²)
    fim ← n    origem ← n(n-1)                          Θ(n²)
    DIKSTRA (G, w, origem)
    caminho ← Π[fim]
    enquanto caminho ≠ NIL faça                         } Θ(n-2)
        imprime caminho
        caminho ← Π[caminho]
```

Complexidade $T(n) = \Theta(n^2)$

(8) Veja Pág 138. (Prova 2008/1)

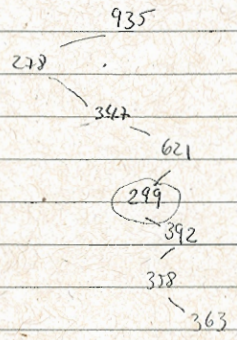
DOUTORADO 97/2

(1) (a)




é possível

(b)

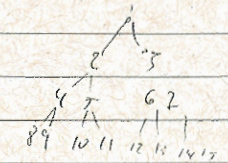



Impossível desde que todos os valores depois de 347 devem ser pelo menos maior o igual. No caso particular 299 não é.

(2) Um heap(max) é uma forma de representar um vetor de elementos com as seguintes propriedade. Seja A um vetor:

$A[j] \leq A[i]$ para todo $i > 1$ e para todo $j > i$

Pai[i] = ⌊i/2⌋
filho_esq[i] = 2i
filho_dir[i] = 2i+1




Pai[i] ≥ filho_esq[i] e
Pai[i] ≥ filho_dir[i]

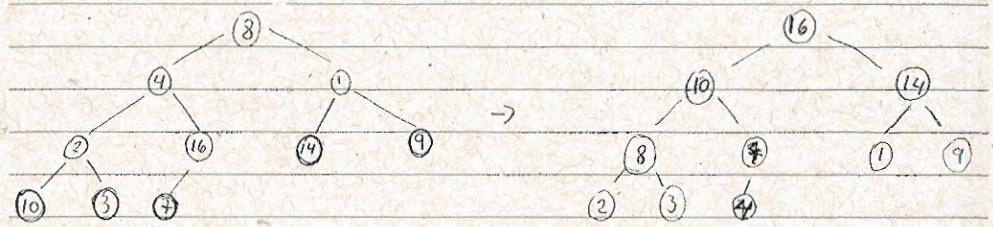




Assim um vetor pode ser representado como uma árvore binária quase completa.

(b)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 4 | 1 | 2 | 16 | 14 | 9 | 10 | 3 | 7 |




(3)

PROCURA($S, n, x$)
Ordenar de forma crescente $O(n \lg n)$
$i \leftarrow 1$
$j \leftarrow n$
enquanto $i < j$ faça
  se $S[i] + S[j] = x$
    então devolva "valor achado"
  se $S[i] + S[j] < x$
    então $i \leftarrow i+1$
    senão $j \leftarrow j-1$
devolva "valor não achado"

(4) Mostre que $\sum_{k=1}^{n} k^{99} = \Theta(n^{100})$

- Provar para $\sum_{k=1}^{n} k^{99} = O(n^{100})$

$$\sum_{k=1}^{n} k^{99} = 1^{99} + 2^{99} + 3^{99} + \dots + n^{99}$$
$$\leq n^{99} + n^{99} + n^{99} + \dots + n^{99}$$
$$\leq n \cdot n^{99} = n^{100}$$

- Provar para $\sum_{k=1}^{n} k^{99} = \Omega(n^{100})$

$$\sum_{k=1}^{n} k^{99} = \sum_{k=1}^{n/2} k^{99} + \sum_{n/2+1}^{n} k^{99}$$

$$\geq \sum_{n/2+1}^{n} (n/2)^{99}$$

$$\geq (n/2)(n/2)^{99} = \left(\frac{1}{2^{100}}\right) n^{100} \quad \text{com } c = 1/2^{100} \text{ e } n \geq 1$$

$$= \Omega(n^{100})$$

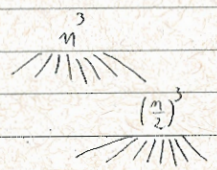




(5)-1 $T(1) = 0$

$T(n) = 8\,T(n/2) + n^3$

(5)-2 Usaremos uma árvore de recorrência para tentar obter a fórmula fechada da recorrência




$$8^0\left(\frac{n}{2^0}\right)^3 = n^3$$
$$8^1\left(\frac{n}{2^1}\right)^3 = n^3$$
$$8^2\left(\frac{n}{2^2}\right)^3 = n^3$$
$$8^i\left(\frac{n}{2^i}\right)^3 = \left(\frac{8^i}{8^i}\right) n^3 = n^3$$

no último nível $\frac{n}{2^i} = 1$ então $\lg n = i$

$$T(n) = \sum_{i=0}^{\lg n - 1} n^3 + 8^{\lg n}$$

$$= n^3 \lg n + n^3 \geq n^3 \lg n = \Omega(n^3 \lg n)$$

(5)-2 Troque 8 por 7 no algoritmo e mostre que $T(n) = O(n^3)$

$$T(n) = n^3 \sum_{i=0}^{\lg n - 1} \left(\frac{7}{8}\right)^i + n^3 \leq n^3 \sum_{i=0}^{\infty} \left(\frac{7}{8}\right)^i + n^3$$

$$\leq n^3 \left[\frac{1}{1 - 7/8}\right] + n^3 = 9n^3 = cn^3 \quad \text{constante } c = 9 \text{ e } n \geq 1.$$

$$= O(n^3)$$

PROVA DOUTORADO 2006/1 (cis) 13/05

(1) Mostre que $\sum_{i=1}^{n} i^2 = \Theta(n^3)$

É sabido que $f(n) = \Theta(n^3)$ se e somente $f(n) = O(n^3)$ e $f(n) = \Omega(n^3)$

- Vou mostrar que $\sum_{i=1}^{n} i^2 = O(n^3)$

$$\sum_{i=1}^{n} i^2 \leq \sum_{i=1}^{n} n^2$$

$$\leq n^3 = O(n^3) \quad \text{Assim } \sum_{i=1}^{n} i^2 = O(n^3) \text{ para } c=1 \text{ e } n \geq 1$$

- Vou mostrar que $\sum_{i=1}^{n} i^2 = \Omega(n^3)$

$$\sum_{i=1}^{n} i^2 = \sum_{i=1}^{n/2} i^2 + \sum_{n/2+1}^{n} i^2$$

$$\geq \sum_{n/2+1}^{n} i^2$$

$$\geq \sum_{n/2+1}^{n} (n/2)^2$$

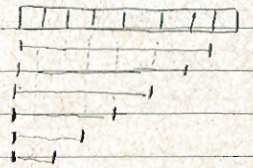
$$\geq \left(\frac{n}{2}\right)^2 \left(\frac{n}{2}\right) \geq \frac{1}{8} n^3 = \Omega(n^3) \text{ para este } c = 1/8 \text{ e } n \geq 1$$





(2) Considere o seguinte função recursiva que determina o índice de um maior elemento de um vetor $v[1..n]$

```
IND_MAXIMO(v, n)
1.  se n = 1
2.      então imax ← n
3.      senão imax ← IND_MAXIMO(v, n-1)
4.            se v[imax] < v[n]
5.                então imax ← n
6.  devolva imax
```




Supondo que a entrada é uma permutação aleatória de $\{1..n\}$ escolhida de acordo com a distribuição uniforme. Qual é o número esperado de vezes que as linhas atribuições das linhas 2 e 5 são executadas? Justifique.

L.2. Será executada somente uma 1. Sendo obvio independente da entrada e do tipo de distribuição

L.5. A linha 5 depende do valor escolhido. Se o $i$-ésimo elemento for maior que os $(i-1)$ elementos a linha 5 será executada

- A prob. do elemento 2 ser maior do que o 1º elemento é $1/2$
- " " " " 3 " " " " os 2 primeiros elem é $1/3$

Assim a prob. da L.5 ser executada será $\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$

$$= \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}\right) - 1$$

$$= O(\ln(n)) + c$$

$$\int_1^n \frac{1}{x}\,dx = \ln x \Big|_1^n = \ln n - \ln 1$$

## (3). LANCHONETE

- Vetor de panquecas: $v[1..n]$ (1ª posição no fundo)
- Único método FLIP$(v, k, n)$ inverte panquecas acima de $k$
- Alg em $O(n)$ chamados a FLIP

ORDENA_PANQUECAS$(v, n)$
1. $i \leftarrow 1$
2. enquanto $i \leq n$ faça
3. $\quad k \leftarrow$ BUSCA_PANQUECA_MAIOR_DIAMETRO$(v, i, n)$
4. $\quad$ FLIP$(v, k, n)$
5. $\quad$ FLIP$(v, i, n)$
6. $\quad i \leftarrow i+1$

Complexidade: Seja $T(n)$ o número de chamadas recursivas (a FLIP) para ordenar um vetor de panquecas de comprimento $n$

$$T(n) = 2n = O(n)$$

BUSCA_PANQUECA_MAIOR_DIAMETRO$(v, i, n)$ — $\Theta(n)$
1. $v_{max} \leftarrow v[1]$
2. para $k \leftarrow 2$ até $n$ faça
3. $\quad$ se $v_{max} < v[k]$
4. $\quad\quad$ então $v_{max} \leftarrow v[k]$

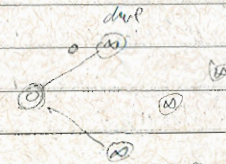




(4) De um alg. eficiente para: Dado um grafo conexo $G$ com $n$ vértices, $m$ arestas e pesos não negativos nas arestas, encontrar um cjto de arestas cuja remoção não desconecta $G$, e que tenha peso máximo. Como sempre, o peso de um cjto de arestas é a soma de seus pesos. (sem remoção de vértices)

- Seja $G = (V, E)$ o grafo de $n$ vértices e $m$ arestas, $|V| = n$ $|E| = m$
- Certamente podemos usar um alg. guloso para achar uma árvore $M$ espanção mínima de peso (e.g. Alg. de PRIM).
- Assim o cjto de arestas cuja remoção não desconecta $G$ será: $E[G] - E[M]$ (i.e. arestas de grafo $G$ menos as arestas do grafo $M$)
- Será usado o alg. MST dado que esse alg. calcula a árvore de espanção de peso mínimo, retendo um cjto de arestas de maior peso.

REMOVE_ARESTAS$(G)$ — $\Theta(mn)$ — (Pesos, raiz)
1. $M \leftarrow$ MST-PRIM$(G, w, 1)$
2. $A \leftarrow E[G] - E[M]$
3. peso $\leftarrow 0$
4. para cada aresta $a \in A$ faça — $\Theta(m)$
5. $\quad$ peso $\leftarrow$ peso $+ a$
6. devolva $(A, \text{peso})$ — TOTAL $= \Theta(nm)$




MST-PRIM$(G, w, r)$
1. para cada vértice $u \in V$ faça — $\Theta(nm)$
2. $\quad$ chave$[u] \leftarrow \infty$
3. $\quad \pi[u] \leftarrow$ NIL
4. chave$[r] \leftarrow 0$
5. $Q \leftarrow V[G]$
6. enquanto $Q \neq \emptyset$ faça
7. $\quad u \leftarrow$ EXTRACT-MIN$(Q)$
8. $\quad$ para cada $v \in$ Adj$[u]$ faça
9. $\quad\quad$ se $v \in Q$ e $w(u,v) <$ chave$[v]$
10. $\quad\quad\quad$ então chave$[v] \leftarrow w(u,v)$
11. $\quad\quad\quad\quad \pi[v] \leftarrow u$
12. devolva $A = \{(u, \pi(u)) : u \in V - \{r\}\}$

(5) Resolva os seguintes recorrências (assuma que $n$ é potência de 2)

$T(1) = a$

(a) $T(n) = 4T(n/2) + n^2$

$n^2 \rightarrow 4^0 (n/2^0)^2 \rightarrow n^2$

$(n/2)^2 \rightarrow 4^1 (n/2^1)^2 \rightarrow n^2$

$(n/2^i)^2 \rightarrow 4^2 (n/2^2)^2 \rightarrow n^2$

$\vdots$

$\rightarrow 4^i (n/2^i)^2 \rightarrow n^2$

no último nível $n = 2^i$ então $\lg n = i$

$$T(n) = \sum_{i=0}^{\lg n - 1} n^2 + 4^{\lg n} a$$

$$T(n) = n^2 \lg n + n^2 a$$

• Prova por Indução em $k$, com $n = 2^k$, para $k = 0, 1, 2, \ldots$

* Base $k=0$, então $n=1$, $T(1) = a$ OK

* Passo $k>0$, então

$$T(n) = 4T(n/2) + n^2$$

$$= 4\left[\left(\frac{n}{2}\right)^2 \lg\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)^2 a\right] + n^2$$

$$= n^2(\lg n - 1) + n^2 a + n^2$$

$$= n^2 \lg n - n^2 + n^2 a + n^2$$

$T(n) = n^2 \lg n + n^2 a$ OK

Assim $T(n) = \Theta(n^2 \lg n)$



$T(1) = a$

(5)-(b) $T(n) = 4T(n/2) + n^2 \lg n$

$n^2 \lg n \longrightarrow 4^0 \left(\frac{n}{2^0}\right)^2 \lg\left(\frac{n}{2^0}\right) =$

$\ldots \left(\frac{n}{2}\right)^2 \lg\left(\frac{n}{2}\right) \longrightarrow 4^1 \left(\frac{n}{2^1}\right)^2 \lg\left(\frac{n}{2}\right) = n^2 \lg\left(\frac{n}{2}\right)$

$\ldots \left(\frac{n}{2^2}\right)^2 \lg\left(\frac{n}{2^2}\right) \longrightarrow 4^2 \left(\frac{n}{2^2}\right)^2 \lg\left(\frac{n}{2^2}\right) = n^2 \lg\left(\frac{n}{2^2}\right)$

no nível $i \longrightarrow n^2 \lg\left(\frac{n}{2^i}\right) = n^2(\lg n - \lg 2^i)$

no último nível $n = 2^i$ então $i = \lg n \longrightarrow 4^i = 4^{\lg n} = n^2$

$$T(n) = \sum_{i=0}^{\lg n - 1} (n^2 \lg n - n^2 i) + n^2 a$$

$$= n^2 \lg^2 n - n^2 \frac{(\lg n - 1)(\lg n)}{2} + n^2 a = n^2 \lg^2 n - \frac{n^2 \lg^2 n}{2} + \frac{n^2 \lg n}{2} + n^2 a$$

$$\boxed{T(n) = \frac{n^2 \lg^2 n}{2} + \frac{n^2 \lg n}{2} + n^2 a}$$

• Prova por indução em $k$, com $n = 2^k$, $k = 0, 1, 2, 3$

• Base $k=0$, então $T(1) = a$ ✓

• Passo

$$T(n) = 4T(n/2) + n^2 \lg n = 4\left[\frac{1}{2}\left(\frac{n}{2}\right)^2 \left[\lg\left(\frac{n}{2}\right)\right]^2 + \frac{1}{2}\left(\frac{n}{2}\right)^2 \lg\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)^2 a\right] + n^2 \lg$$

$$T(n) = \frac{1}{2} n^2 (\lg n - 1)^2 + \frac{1}{2} n^2 (\lg n - 1) + n^2 a + n^2 \lg n$$

$$= \frac{1}{2} n^2 (\lg^2 n - 2\lg n + 1) + \frac{1}{2} n^2 \lg n - \frac{1}{2} n^2 + n^2 a + n^2 \lg n$$

$$= \frac{1}{2} n^2 \lg^2 n - n^2 \lg n + \frac{1}{2} n^2 + \frac{1}{2} n^2 \lg n - \frac{1}{2} n^2 + n^2 \lg n + n^2 a$$

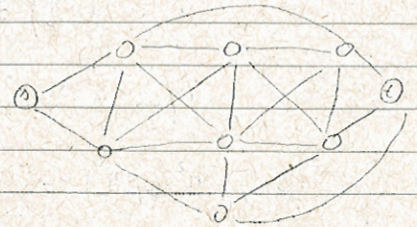
$$= \frac{1}{2} n^2 \lg^2 n + \frac{1}{2} n^2 \lg n + a n^2$$

Assim $T(n) = O(n^2 \lg^2 n)$ Na verdade $T(n) = \Theta(n^2 \lg^2 n)$

(6) Um dado grafo G (com n vértices) e dois vértices s e t

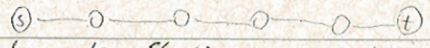
(P1) Determinar se existe um caminho de s a t que passa por no máximo n/2 vértices

(P2) " " " " " " " " " " " mínimo " "

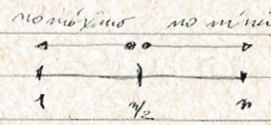



$|V| = 9 \leq n$

$|E| = 13$




caminho + curto $= \delta(s,t) = 5$, com 4 vértices




- com a pesquisa primeiro na expansão podemos achar o caminho mais curto de s a t (considerando o grafo sem pesos)

(É uma limitante inferior - a menor quantidade de arestas)

(a) F. Existe Alg polinomial para P2. P1 é exponencial (com certeza)

(b) V. Para P2 (com certeza)

(c) V. Para P2. Mas Assumindo que $P \neq NP$

(d) F. Somente para P2 existem. Assumindo que $P \neq NP$

Se P é NP então

(a) F

(b) V

(c) F. Existiria para os 2 problemas

(d) V Sim existiriam algs. polinomiais para ambos.





(7) Contadora de pedaços inteiros (CPI)

(8) Seja Y mxn uma matriz semi-ordenada

(8)-(a)

$$\begin{pmatrix} 2 & 3 & 4 \\ 5 & 8 & 9 \\ 12 & 14 & 16 \end{pmatrix} \qquad \begin{pmatrix} 2 & 4 & 8 \\ 3 & 9 & 14 \\ 5 & 12 & 16 \end{pmatrix}$$

$$\begin{pmatrix} \times & 5 & 7 \\ 3 & 6 & 8 \\ 9 & 10 & 11 \end{pmatrix} \to \begin{pmatrix} 11 & 5 & 7 \\ 3 & 6 & 8 \\ 9 & 10 & \times \end{pmatrix} \to \begin{pmatrix} 3 & 5 & 7 \\ 11 & 6 & 8 \\ 9 & 10 & \times \end{pmatrix}$$

$$\begin{pmatrix} 3 & 5 & 7 \\ 6 & 11 & 8 \\ 9 & 10 & \times \end{pmatrix} \to \begin{pmatrix} 3 & 5 & 7 \\ 6 & 8 & 11 \\ 9 & 10 & \times \end{pmatrix}.$$

(8)-(b)

```
EXTRAI_MIN(A, m, n)
1  se A[1,1] = ∞
2     então devolva ∞
3     senão k ← PROCURA_ULTIMO_ELEMENTO(A, m, n)
4           x ← A[1,1]          ▷ o elemento mínimo
5           A[1,1] ← k          ▷ último elemento na primeira posição
6           i ← 1
7           j ← 1
8           enquanto i<m e j<n faça        ▷ caminha pelo menor valor
9
10                se A[i+1, j] > A[i, j+1]
11                   então A[i,j] ↔ A[i,j+1]
12                         j ← j+1
13                   senão A[i,j] ↔ A[i+1, j]
14                         i ← i+1
15    devolva x.
```

```
PROCURA_ULTIMO_ELEMENTO(A, m, n)
1.  i ← 1   j ← 1
2.  enquanto A[i,j] ≠ ∞ e i ≤ m faça
3.        i ← i+1
4.  i ← i-1
5.  enquanto A[i,j] ≠ ∞ e j ≤ n faça
6.        j ← j+1
7.  x ← A[i,j]
    A[i,j] ← ∞
    devolva x
```

j ← j-1

Dica: coloca ∞ na posição (1,1) e caminha pelo menor elemento, assim, não é necessária essa função



$$\begin{pmatrix} 3 & 5 & 7 \\ 6 & 8 & 11 \\ 9 & 10 & 5 \end{pmatrix} \to \begin{pmatrix} 3 & 5 & 7 \\ 6 & 8 & 5 \\ 9 & 10 & 11 \end{pmatrix} \to \begin{pmatrix} 3 & 5 & 7 \\ 4 & 5 & 8 \\ 9 & 10 & 11 \end{pmatrix}$$



$$\begin{pmatrix} 3 & 5 & 7 \\ 3.5 & 6 & 8 \\ 9 & 10 & 11 \end{pmatrix}$$

(8)-(c)

```
INSERE(x, A, m, n)                          ▷ caminha pelo maior valor
1  se A[m,n] ≠ ∞
2     então devolva "Matriz cheia"
3     senão (p,q) ← BUSCA_POSIÇÃO_LIVRE(A, m, n)
4           A[p,q] ← x
5           i ← p
6           j ← q
7           enquanto i>1 e j>1 faça
8                 se A[i,j-1] > A[i-1,j]
9                    então A[i,j] ↔ A[i,j-1]
10                        j ← j-1
11                   senão A[i,j] ↔ A[i-1,j]
12                        i ← i-1
```

```
BUSCA_POSIÇÃO_LIVRE(A, m, n)
    i ← 1
    j ← 1
    enquanto A[i,j] ≠ ∞ e i < m faça
          i ← i+1
    i ← i-1
    enquanto A[i,j] ≠ ∞ e j < n faça
          j ← j+1
    devolva (i,j)
```

Dica: Insere na ultima posição e depois caminha pelo maior valor

$$\begin{pmatrix} 1 & 2 & 4 & 7 \\ 3 & 5 & 8 & 11 \\ 6 & 9 & 12 & 14 \\ 10 & 13 & 15 & 16 \end{pmatrix}$$

(8)-(d) Ordenar(A, n)

```
1.  T ← linha 1 de A
2.  k ← 2
3.  enquanto k ≤ n faça
4.          T ← INTERCALA(T, linha k de A)
5.          k ← k+1
6.  devolva T          ▷ T é um vetor de tamanho n²
```

Complexidade:

- MERGE( {L1}, {L2} ) — $n + n$ na intercalação
- MERGE( {L1, L2}, {L3} ) — $2n + n$ " "
- ⋮
- MERGE( {L1, L2, .., Ln-1}, {Ln} ) — $(n-1)n + n$

$$T(n) = \sum_{i=1}^{n-1} n + n = \frac{n(n-1)}{2}(n) + n$$

$$= \frac{n^3 - n^2}{2} + n = \frac{1}{2}\left(n^3 - n^2 + 2n\right) = O(n^3)$$

Dá para fazer em $O(n \lg n)$

(8)-(e) BUSCA(A, m, n, x)

```
1   i ← 1
2   j ← 1
3   enquanto i ≤ m e x ≥ A[i,j] faça
4           se x = A[i,j]
5                   então devolva "ocorre o valor"
6                   senão i ← i+1
7   enquanto i ≥ 1 e j ≤ n faça
8           se x = A[i,j]
9                   então devolva "ocorre o valor"
10          se A[i,j] > x
11                  então i ← i-1
12                  senão i ← i+1
```





PROVA DOUTORADO 2006/2

(1) Para cada inteiro positivo $i$, sejam $f_i(x)$ e $g_i(x)$ funções tais que $f_i(x) = O(g_i(x))$

i) Defina as funções $F(n)$ e $G(n)$ por $F(n) = \sum_{i=1}^{n} f_i(n)$, $G(n) = \sum_{i=1}^{n} g_i(n)$

É verdade que $F(n) = O(G(n))$

- Rpta: É verdade para funções assintoticamente crescentes ou decrescentes

Sabemos que $f_i(x) \leq c\, g_i(x)$ para constantes $c$ positivas e $x \geq x_0$

Assim $\sum_{i}^{n} f_i(x) \leq \sum_{i}^{n} c\, g_i(x)$

$\sum_{i=1}^{n} f_i(x) \leq c \sum_{i=1}^{n} g_i(x)$ considerando $c = \max\{c_1, c_2, c_3, \ldots, c_n\}$

Logo $F(n) \leq c\, G(n)$

ou $F(n) = O(G(n))$

(2)-(i) Qual o tamanho de entrada como função de $n$ e $k$.

→ a entrada está dada pelo par $(\lg n, \lg k)$. Entretanto, podemos considerar a entrada como $\lg n + \lg k = \lg(nk)$

(2)-(ii) Seja $T(n)$ o número de multiplicações de $n$ (na verdade $n$ vezes) $\rightarrow 1, n, n^2, n^3, \ldots, n^{k-1}$

$$T(n) = \sum_{i=1}^{k} \Theta\left(\lg n \cdot \lg n^{(i-1)}\right)$$

$$= \lg^2 n \sum_{i=1}^{k} \Theta(i-1) \quad \rightarrow 0, 1, 2, \ldots, k-1$$

$$= \Theta\left[\lg^2 n \frac{(k-1)k}{2}\right] = \Theta\left[\frac{1}{2}\lg^2 n\, k^2 - k \lg^2 n\right] = O\left(\frac{1}{2}\lg^2 n\, k^2\right)$$

$$= O\left(k^2 \lg^2 n\right)$$

(2)-(3) O algoritmo não é limitado polinomialmente no tamanho da entrada

$T(n) = \frac{1}{2} k^2 \lg^2 n - k \lg^2 n \leq k^2 \lg^2 n$ para $k$ e $n \geq 1$

- Vou provar pelo absurdo que $T(n)$ não está limitada polinomialmente
- Suponha que $T(n) = O((\lg n + \lg k)^p)$ para $p$ constante $\geq 1$

$$T(n) = k^2 \lg^2 n \leq \lg(nk)^p$$

$$\lg(k^2 \lg^2 n) \leq p \lg \lg(nk)$$

$$p \geq \frac{\lg(k^2 \lg^2 n)}{\lg \lg(nk)} \geq \frac{\lg(\lg^2 k \lg^2 n)}{\lg \lg(nk)}$$

$$p \geq \frac{\lg^2 k \cdot \lg^2 n}{\lg(nk)} = \frac{\lg k \cdot \lg k \cdot \lg n \cdot \lg n}{\lg(nk)}$$

$$p \geq \frac{\lg k \cdot \lg n}{\lg(n \cdot k)} = \frac{\lg n \cdot \lg k}{\lg n + \lg k}$$

$p \geq \frac{\lg n \cdot \lg k}{\lg n + \lg k}$ não é constante

#Absurdo!

$\Theta(x)$ c. $x$

$\Theta(x^2)$

$\Theta(3x)$



(Q.3) Considere a sequência $T(n)$ tal que:

$T(0) = 0$

$T(n) = T(\lfloor n - \sqrt{n} \rfloor) + 1$

Mostre que $T(n) = \Theta(\sqrt{n})$

- Vou provar por indução em $n$, que $T(n) = O(\sqrt{n})$, em particular $T(n) \leq 2\sqrt{n}$, para $n = 0, 1, 2, \ldots$

→ Base: $n = 0$

$T(0) \leq 0$ (verdadeiro)

→ Passo: $n > 0$

$$T(n) = T(\lfloor n - \sqrt{n} \rfloor) + 1$$

$$\leq 2\sqrt{\lfloor n - \sqrt{n} \rfloor} + 1$$

$$\leq 2\sqrt{n - \sqrt{n}} + 1$$

$$\leq 2\sqrt{n - \sqrt{n} + 1/4} + 1 = 2\sqrt{(\sqrt{n} - 1/2)^2} + 1$$

$$\leq 2(\sqrt{n} - 1/2) + 1 = 2\sqrt{n} - 1 + 1$$

$$\leq 2\sqrt{n}$$

Logo $T(n) \leq 2\sqrt{n}$

ou $T(n) = O(\sqrt{n})$

$$T(n) = T(\lfloor n - \sqrt{n} \rfloor) + 1$$

• Vou provar que $T(n) = \Omega(\sqrt{n})$

Provarei por indução em $n$ que $T(n) \geq \sqrt{n}$ para $n = 4, 5, 6, \ldots$

→ Base: $n = 4$ então $T(4) \geq \sqrt{4} = 2$ $\quad T(4) = 2$

→ Passo: $n > 4$ então

$$T(n) = T(\lfloor n - \sqrt{n} \rfloor) + 1$$

$$\geq \sqrt{\lfloor n - \sqrt{n} \rfloor} + 1$$

$$\geq \sqrt{n - \sqrt{n} - 1} + 1 \geq \sqrt{n - 2\sqrt{n} + 1} + 1 \quad \text{desde que } n > 4$$

$$\geq \sqrt{(\sqrt{n} - 1)^2} + 1 = \sqrt{n} - 1 + 1$$

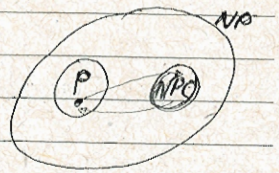
$$\geq \sqrt{n}$$

Logo $T(n) = \Omega(\sqrt{n})$

Portanto $T(n) = \Theta(\sqrt{n})$

(Q.4) $P = NP$ se e só se algum problema em P é NP-completo. (V ou F?)

• ~~Falso~~: Verdadeiro. Veja Pag 85




* Não é verdade que se algum problema em P é NP completo então $P = NP$

Exemplo: raiz quadrada polinomialmente pode ser reduzido para o prob. do CICLO-HAMILTONIANO (NPC)

Entretanto $P = NP$ se existe algum problema NP completo em P





(Q5) Alg. que determina se $x \in v[1..n]$. Calcule o número esperado de iterações da linha 5

• Seja $X_j$ a probabilidade de na iteração $j$ $v[i] = x$, e seja $\overline{X_j}$ a prob. de na iteração $j$ $v[i] \neq x$.

• Observemos que a L.5 será executada $n$ vezes caso o elemento $x$ não esteja no vetor $v$ (isso é o pior caso)

• Observemos também que:

($p_i$ dado pelo RAND)

| iteração ($j$) | Prob. de achar $x$ ($v[i] = x$) |
|---|---|
| 1 | $1/n$ |
| 2 | $(1 - 1/n)(1/n)$ — Prob de achar; Prob. de não achar |
| 3 | $(1 - 1/n)(1 - 1/n)(1/n)$ |
| ⋮ | |
| $n$ | $\left(\frac{n-1}{n}\right)^{n-1}\left(\frac{1}{n}\right)$ → $\frac{1}{n}\left(\sum_{i=0}^{n-1}\left(\frac{n-1}{n}\right)^i\right)$ |

$\frac{j-1}{n} \cdot \frac{n-1}{n}$

Isso obedece uma distribuição geométrica

$$= \frac{1}{n}\left(\left(\frac{n-1}{n}\right)^0 + \left(\frac{n-1}{n}\right)^1 + \left(\frac{n-1}{n}\right)^2 + \ldots\right)$$

$$Pr[X_j = x | n] = (X_j)(\overline{X_j})^{n-1}$$

$$E[X] = \frac{1}{X_i} = \frac{1}{(1/n)} = n$$

Resposta: O número esperado de iterações é $n$

(Q6) $T(1) = 1$ — será que $T(n)$ é $O(n \lg n)$

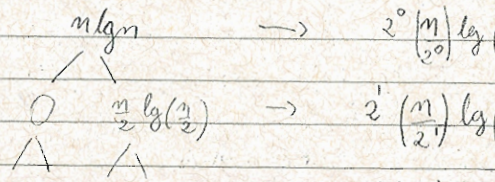
$T(n) = 2\,T(\lfloor n/2 \rfloor) + \lfloor n \lg n \rfloor$

De forma grosseira podemos escrever a recorrência anterior como:

$T(1) = 1$

$T(n) = 2\,T(n/2) + n \lg n$

• Árvore de recursão:




$n \lg n \longrightarrow 2^0 \left(\frac{n}{2^0}\right) \lg\left(\frac{n}{2^0}\right)$

$\longrightarrow 2^1 \left(\frac{n}{2^1}\right) \lg\left(\frac{n}{2^1}\right) = n(\lg n - 1)$

$\longrightarrow 2^2 \left(\frac{n}{2^2}\right) \lg\left(\frac{n}{2^2}\right) = n(\lg n - 2)$

$2^i \left(\frac{n}{2^i}\right) \lg\left(\frac{n}{2^i}\right) = n(\lg n - i)$

no último nível $n = 2^i \Rightarrow i = \lg n$

$$T(n) = \sum_{i=0}^{\lg n - 1} (n \lg n - n i) + 2^{\lg n}$$

$$= n \lg n (\lg n) - n \sum_{i=0}^{\lg n - 1} i + n$$

$$= n \lg^2 n - n\left(\frac{1}{2}\right)(\lg n - 1)(\lg n) = n \lg^2 n - \frac{n \lg^2 n}{2} + \frac{n \lg n}{2}$$

$$= \frac{n}{2} \lg^2 n + \frac{n}{2} \lg n + n$$

• Vou provar por indução em $k$, para $k = 0, 1, 2, \ldots$, com $n = 2^k$, que

$$T(n) = \frac{n}{2} \lg^2 n + \frac{n}{2} \lg n + n$$





→ Base: $k = 0$, então $n = 2^0 = 1$

$T(1) = 1$

→ Passo: $k \geq 1$, então

$T(n) = 2\,T(n/2) + n \lg n$

$$= 2\left[\frac{1}{2}\left(\frac{n}{2}\right)\lg^2\left(\frac{n}{2}\right) + \frac{1}{2}\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) + \frac{n}{2}\right] + n \lg n$$

$$= \frac{n}{2}(\lg n - 1)^2 + \frac{n}{2}(\lg n - 1) + n + n \lg n$$

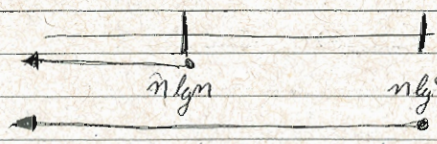
$$= \frac{n}{2}(\lg^2 n - 2 \lg n + 1) + \frac{n}{2} \lg n - \frac{n}{2} + n + n \lg n$$

$$= \frac{n}{2} \lg^2 n - n \lg n + \frac{n}{2} + \frac{n}{2} \lg n - \frac{n}{2} + n + n \lg n$$

$$T(n) = \frac{n}{2} \lg^2 n + \frac{n}{2} \lg n + n$$

$$T(n) = O(n \lg^2 n)$$

Resposta: Não é verdade que $T(n)$ é $O(n \lg n)$, pois $T(n)$ é $O(n \lg^2 n)$



(Q7) ABB(A) embutido em ABB(B)

- Idéia: Criar um alg. recursivo em que primeiro se procure a raiz de A em B, se existir, chamar recursivamente o alg. duas vezes: (1) com a sub-árvore esquerda, e (2) com a sub-árvore direita

EMBUTIDO(A, B)
1. r ← raiz(A)
2. se r = NIL
3. então devolve TRUE ▶ A árvore vazia está embutida em outra
4. x ← PROCURE-ELEMENTO(r, B)
5. se x ≠ FALSE
6. então devolva EMBUTIDO(esq[x], B) e EMBUTIDO(dir[x], B)
7. senão devolva FALSE

Como o algoritmo procura cada elemento de A em B, mantendo a ordem de dependência, então a complexidade é $O(n)$, sendo que $n$ é o número de nós na árvore B.

(Q8) Ordem parcial.

Como pode ocorrer que se $a < b$ e $b < c$ então $a < c$ e que existe a opção de "incomparação", obrigatoriamente devem ser realizadas todas as comparações (2-a-2), Assim são necessários $\Omega(n^2)$ comparações.

SCANPIX

Um algoritmo válido seria o SELECTION-SORT, em que para cada iteração, é escolhido o menor elemento



ORDENA-ORDEM-PARCIAL(A, n)
1. para i ← 1 até n-1 faça
2. menor ← A[i]
3. para j ← i+1 até n faça
4. se COMPARA(menor, A[j]) = '>'
5. então menor ← A[j]
6. A[i] ↔ menor

Acho que também funciona usando o alg. INSERTION-SORT.

Complexidade $O(n^2)$

(Q9) Considere importante que serão usados strings distintos

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| A: | 4 | 5 | 3 | 2 | 0 | 1 | 8 |
| p(i): | 4 | 5 | 3 | 2 | 0 | 1 | 6 |
| δ(i): | -3 | -3 | 0 | 2 | 5 | 5 | 1 |

É uma medida que quão desordenado está o vetor A.

$\delta_A = \sum_{i=1}^{n} \delta(i) = 7$

$\overline{\delta_A} = \sum_{i=1}^{n} |\delta(i)| = 19$

ordene A

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 2 | 3 | 4 | 5 | 8 |
| Posições → | 5 | 6 | 4 | 3 | 1 | 2 | 7 |
| anteriores +1 | -3 | -3 | 0 | 2 | 5 | 5 | 1 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 2 | 3 | 6 | 5 | 4 | 10 | 11 |
| p(i) = | 0 | 1 | 2 | 3 | 6 | 5 | 4 | 7 | 8 |
| δ(i) = | 1 | 1 | 1 | 1 | -1 | 1 | 3 | 1 | 1 |

$\delta_A = 9$ ← # de elementos

$\overline{\delta_A} = 11$

ordene A

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 10 | 11 |
| Posições → | 1 | 2 | 3 | 4 | 7 | 6 | 5 | 8 | 9 |
| +1 → | 0 | 0 | 0 | 0 | -2 | 0 | 2 | 0 | 0 |
| | 1 | 1 | 1 | 1 | -1 | 1 | 3 | 1 | 1 |

SCANPIX

(a) $SA = m$ $\Theta(1)$

(b) ALGORITMO (A, n)

1. • Ordene o vetor A de n elementos mantendo um registro das posições originais
2. • Use o registro das posições para calcular $S_A$. Seja B o vetor de índices registrados de A (ordenado em forma crescente)
3. soma ← 0
4. para i ← 1 até n faça
5. soma ← i − B[i] + 1
6. devolva soma

Complexidade: Dependerá do alg. usado para ordenar o vetor A, usando o HEAP-SORT ou o QUICK-SORT, podemos garantir ter uma complexidade de $O(n^2)$, sendo n o tamanho da instância de entrada (tamanho do vetor) :-)





2005/1 (Cotho)

(Q1)

(a) $T(n) = O(f(n))$ ? para $n \geq n_0$

Existem constantes $c \geq 1$, e $n_0$ tais que $T(n) \leq c\,f(n)$. Isto é $T(n)$ é uma função que está limitada superiormente por $f(n)$

(b) $20n^3 + 10n\lg n + 5 = O(n^3)$ ? verdade

$$20n^3 + 10n\lg n + 5 \leq 20n^3 + 10n^3 + 5n^3$$
$$\leq 35n^3$$
$$= O(n^3)$$

(c) É verdade que $\frac{1}{2}n^2 = O(n)$? falso

Suponha que seja verdade, então existe uma constante c e $n_0$ tais que

$$\frac{1}{2}n^2 \leq cn \quad \text{então} \quad c \geq \frac{1}{2}n \quad \text{o qual é absurdo}$$

(d) $T(n) = \Omega(f(n))$ ?

Existem constantes positivas $c \geq 1$, e $n_0$ tais que $T(n) \geq c\,f(n)$ para $n \geq n_0$

(e) Que significa $T(n) = n + \Omega(n\lg n)$? É um abuso de notação $T(n) = n + f(n)$ onde $f(n) = \Omega(n\lg n)$, ou seja

$T(n) = \Omega(n\lg n)$

?

## (Q2) Algoritmo MAIORES

(a)

| Linha | Complexidade |
|---|---|
| 1 | $O(n)$ |
| 2 | $\Theta(k)$ |
| 3 | $\Theta(k)$ |
| 4 | $\Theta(k \lg n)$ |
| 5 | $\Theta(1)$ |

Total = $O(n) + \Theta(k) + \Theta(k \lg n)$

$T(n) = O(n + k \lg n)$

(b) Resposta em função de $n$.

$T(n)$ será $O(n)$ se $O(n + k \lg n) = O(n)$

ou seja $k \lg n \leq n$

$k \leq n / \lg n$

Portanto se $k = O\left(\frac{n}{\lg n}\right)$ então $T(n) = O(n)$

## (Q3) Algoritmo caixa preta.

Seja $T(n, r)$ o número de vezes que o algoritmo CAIXA PRETA é chamado pelo algoritmo ALGO, a recorrência é definida como:

$T(n, r) = 1$ , se $n = 0$

$T(n, r) = r\,T(n-1, r) + 1$ , se $n > 0$

Exemplo para $n = 3$

$r\,T(2,r) + 1$

$r\,T(1,r) + 1$

$r\,T(0,r) + 1$

$T(0,r) = 1$

$r^3 + r^2 + r + 1$

$r^2 + r + 1$





$$T(n, r) = \sum_{i=0}^{n} r^i$$

Portanto:

- $T(n, r) = \frac{r^{n+1} - 1}{r - 1}$ , se $r \neq 1$
- $T(n, r) = n + 1$ , se $r = 1$ ✓

## (Q4) Algoritmo MIN-ENV (Guloso)

▷ Algoritmo que recebe um vetor $p[1..n]$, em que $0 \leq p[i] \leq 1$ para $i = 1, 2, \ldots n$ e devolve o número mínimo de envelopes

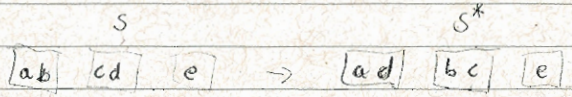
```
MIN-ENV (p, n)
1.  p ← HEAPSORT (p, n)        ▷ Ordena um vetor em forma crescente
2.  env ← 0
3.  i ← 1
4.  j ← n
5.  enquanto i ≤ j faça
6.      se p[i] + p[j] ≤ 1
7.          então i ← i + 1        ▷ coloque i e j no envelope
8.                j ← j - 1
9.                env ← env + 1
10.         senão j ← j - 1        ▷ coloque só j no envelope
11.               env ← env + 1
12. devolva env
```

$T(n) = O(n \lg n)$ dada pelo algoritmo de ordenação

a° Sub-estrutura ótima: Seja $S'$ uma solução ótima, vou provar que $S^*$ também é uma solução ótima.

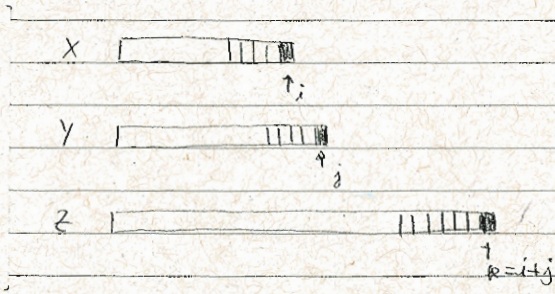
b° Escolha gulosa (propriedade gulosa)

- Em toda iteração utiliza-se no máximo 2 elementos cujos pesos sejam menor ou igual do que 1.




## (Q5) Embaralhamento

- Alg. de decisão: Alg. verificador dados X e Y, e um certificado Z
- Abordagem gulosa OU Prog. dinâmica:




Recorrência: Seja $c[i,j]$ a "decisão" de sequências $X[1..i]$ e $Y[1..j]$ serem um embaralhamento de $Z[1..i+k]$

$c[i,j] = 1$ , se $X[i] = Z[i+j]$ e $c[i-1,j]$

$c[i,j] = 1$ , se $Y[j] = Z[i+j]$ e $c[i,j-1]$

$c[i,j] = 0$ , caso contrário

? $c[0,j] = 1$ , se $Y[1..j] = Z[1..j]$ , o.c.c.

? $c[i,0] = 1$ , se $X[1..i] = Z[1..i]$ , o.c.c.

falltou!





## (Q6)

(a) $\Pi \leq_P \Pi'$

- Significa que $\Pi$ é tão fácil quanto o problema $\Pi'$
- " " $\Pi'$ " " difícil " " " $\Pi$

O problema de decisão $\Pi$ pode polinomialmente ser representado / reduzido ao tipo $\Pi'$

(b) P: Classe de problemas que podem ser "resolvidos" em tempo pol

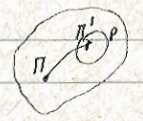
NP: " " " " " " "verificáveis" " " "

dado um certificado.

NP completo: um problema é NP completo se ele está em NP e todo problema em NP é polinomialmente reduzível a ele.

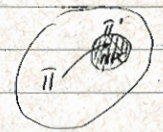
(c) Se $\Pi \leq \Pi'$ e $\Pi'$ está em P então $\Pi$ está em P?

Verdadeiro (P é fechado polinomialmente)




(d) Se $\Pi \leq \Pi'$ e $\Pi'$ é NP-completo, então $\Pi'$ é NP-completo?

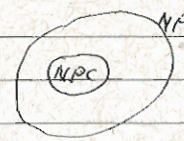
Falso. desde que $P \neq NP$



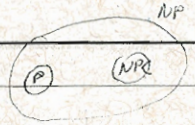

(e) Há problemas em NP que não são NP-completos?

$\exists x \in NP$ tal que $x \notin$ NP-completo?

Verdade, desde que $P \neq NP$



(f) $P \cap NP \neq \emptyset$ ?




verdadeiro $P \subseteq NP$

(g) Existem problemas NP-completos em P?

Não se sabe. Será verdade somente se conseguir-se provar que $P = NP$.

(Q7)

(a) MAKESET ( x )

1. pai[x] ← x
2. rank[x] ← 0

FINDSET ( x )

1. se pai[x] ≠ x
2. então pai[x] ← FINDSET ( pai[x] )
3. devolva pai[x]

UNION ( x, y ) ▷ Supõe x e y estão em árvores diferentes

1. x' ← FINDSET ( x )
2. y' ← FINDSET ( y )
3. se rank[x'] > rank[y']
4. então pai[y'] ← x'
5. senão pai[x'] ← y'
6. se rank[x'] = rank[y']
7. então rank[y'] ← rank[y'] + 1





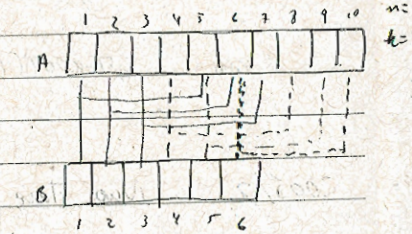
(b) Análise amortizada

- n: MakeSet $O(m \lg^* n)$
- m: FindSet $O(m \alpha(n))$

(Q8) Medianas:

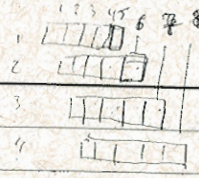
(a) MEDIANAS1 ( A, n, k )

1. m ← ⌊k/2⌋
2. para i ← 1 até n-k+1 faça
3. TA ← A[ i .. i+k-1 ]
4. B[i] ← SELECT-BFPRT (TA, k, m)
5. devolva B.




↳ Seleciona em tempo linear o m-ésimo elemento de um TA de comprimento k em $O(k)$

$T(n) = O(nk)$




(b) Mediana 2

```
Mediana2 (A, n, k)
1. m ← ⌊k/2⌋
2. TREE-BUILD (T, A, 1, k-1)          ▷ Θ(k lg k)
3. para i ← 1 até n-k+1 faça
4.       TREE-INSERT (T, i+k-1)       ▷ O(lg k) → O(n lg k)
5.       B[i] ← GET-ROOT (T)          ▷ O(1) → O(n)
6.       x ← TREE-MINIMUM (T)         ▷ O(lg k) → O(n lg k)
7.       TREE-DELETE (x)              ▷ O(lg k) → O(n lg k).
8. devolva B
```

$T(n) = O(n \lg k)$

O algoritmo mantem em cada iteração uma árvore T com exatamente k elementos, assim todas as operações terão um custo de $O(\lg k)$

A subrotina GET-ROOT(T) devolve o conteúdo da raiz da árvore T.

2005/2 Não tem exame





1998/1 (50)

(Q1)

(1) Defina $O(f)$, $\Omega(f)$ e $\Theta(f)$

- Seja $g = O(f)$. Existem constantes positivas $c$ e $n_0$ tais que $g(n) \leq c\, f(n)$ para $n \geq n_0$
- Seja $g = \Omega(f)$. $g(n) \geq c\, f(n)$ para $n \geq n_0$
- Seja $g = \Theta(f)$ $c_1 f(n) \leq g(n) \leq c_2 f(n)$ para $n \geq n_0$ e constantes $c_1$ e $c_2$.

(2) Se $f = O(g)$ e $g = O(h)$ então $f = O(h)$

Se $f = O(g)$ então por definição $f \leq c_1 g$
Se $g = O(h)$ " " " $g \leq c_2 h$

Assim $f \leq c_1 g \leq c_1 c_2 h$

ou $f \leq c_3 h$ para $c_3 = c_1 \cdot c_2$

Logo $f = O(h)$

(Q2) caso 1: Quando $m = \Theta(n)$, seja $m = n$

- $n^2 m^{1/2} \rightarrow n^2 n^{1/2} = n^{5/2} = O(n^{5/2})$
- $nm + n^2 \lg n \rightarrow n^2 + n^2 \lg n = O(n^2 \lg n)$
- $nm \lg n \rightarrow n^2 \lg n = O(n^2 \lg n)$
- $nm \lg(n^2/m) \rightarrow n^2 \lg n = O(n^2 \lg n)$

ORDEM: 1.- $\{ n^2 m^{1/2} \}$

2.- $\{ nm + n^2 \lg n, nm \lg n, nm \lg(n^2/m) \}$

caso 2: Quando $m = \Theta(n^2)$, seja $m = n^2$

- $n^2 m^{1/2} \rightarrow n^3 = O(n^3)$
- $nm + n^2 \lg n \rightarrow n^3 + n^2 \lg n = O(n^3)$
- $nm \lg n \rightarrow n^3 \lg n = O(n^3 \lg n)$
- $nm \lg(n^2/m) \rightarrow n^3 \lg(1) = O(1)$ ?

ORDEM: 1.- $\{ nm \lg(n^2/m) \}$

2.- $\{ n^2 m^{1/2}, nm + n^2 \lg n \}$

3.- $\{ nm \lg n \}$





(Q3) Computar $n^n$ fazendo no máximo $2 \lg n$ multiplicações.

```
POTENCIA (n, p)
se p = 0
    então devolva 1
se p = 1
    então devolva n
se (p = 0 mod 2)        ▷ se é par
    então x ← POTENCIA(n, p/2)
          devolva x * x
senão x ← POTENCIA(n, ⌊p/2⌋)
      devolva x * x * n
```

POT(8,8)
$8^8$ → POT(8,4)
$8^4$ → POT(8,2)
$8^2$ → POT(8,1)
8

chamada inicial POTENCIA(n, n)

Análise de complexidade: Seja T(n) o número de multiplicações, então
$T(n) \leq T(\lfloor n/2 \rfloor) + 2$ ← Se no pior caso sempre for ímpar.

Grosseiramente podemos analisar T(n) como:
$T(1) = 0$
$T(n) \leq T(n/2) + 2$

2
1
2
1
2
⋮
até $n = 2^i$
ou $i = \lg n$

$T(n) \leq \sum_{i=0}^{\lg n - 1} 2$

$T(n) \leq 2 \lg n$

(Q4) Rotação circular de k posições.

Seja o vetor $V[1..n]$ contendo uma rotação circular.

```
ROTAÇÃO-CIRCULAR ( V, p, r )
1 - se   r - p + 1 > 2
2 -      então  q ← ⌊(p + r)/2⌋
3 -             se V[q] < V[p]        ▷ deslocamento na esq.
4 -                então ROTAÇÃO-CIRCULAR ( V, p, q )
5 -             se V[q] ≥ V[r]
6 -                então ROTAÇÃO-CIRCULAR ( V, q, r )
7 - se   r - p + 1 = 2
8 -      então se V[p] > V[r]
9 -                então devolva V[p]
10 -               senão devolva V[r]
```

Complexidade:

$T(n) = T(n/2) + c$ , em que $c$ é uma constante

Árvore de recorrência:

c
|
c
|
c

até $n = 2^i$ ou $i = \lg n$

$$T(n) = \sum_{i=0}^{\lg n - 1} 1 + c$$

$$T(n) = c \lg n$$

$$T(n) = O(\lg n)$$





(Q5) $n_1 = \left\lceil \frac{2 n_0}{3} \right\rceil$ , $n_2 = \left\lceil \frac{2 n_1}{3} \right\rceil$ , ... , $n_i = \left\lceil \frac{2 n_{i-1}}{3} \right\rceil$

- é verdade que $n_i = \left\lceil \frac{2^i n_0}{3^i} \right\rceil$ ?

Resposta: FALSO, contra-exemplo, tome $n_0 = 2$

$$n_1 = \left\lceil \frac{2 \cdot 2}{3} \right\rceil = 2$$

$$n_2 = \left\lceil \frac{2 \cdot 2}{3} \right\rceil = 2 \neq n_2 = \left\lceil \frac{2^2 \, 2}{3^2} \right\rceil = \left\lceil \frac{8}{9} \right\rceil = 1$$

- Limitação superior:

$$n_1 = \left\lceil \frac{2}{3} n_0 \right\rceil \leq \frac{2}{3} n_0 + 1$$

$$n_2 = \left\lceil \frac{2}{3} n_1 \right\rceil \leq \frac{2}{3} n_1 + 1 \leq \frac{2}{3}\left(\frac{2}{3} n_0 + 1\right) + 1 = \frac{2^2}{3^2} n_0 + \frac{2}{3} + 1$$

$$n_3 = \left\lceil \frac{2}{3} n_2 \right\rceil \leq \frac{2}{3} n_2 + 1 \leq \frac{2}{3}\left(\frac{2^2}{3^2} n_0 + \frac{2}{3} + 1\right) = \frac{2^3}{3^3} n_0 + \frac{2^2}{3^2} + \frac{2}{3} + 1$$

$$n_i \leq \left(\frac{2}{3}\right)^i n_0 + \sum_{k=0}^{i-1} \left(\frac{2}{3}\right)^k = \left(\frac{2}{3}\right)^i n_0 + \frac{\left(\frac{2}{3}\right)^i - 1}{\left(\frac{2}{3}\right) - 1} \longrightarrow -1/3$$

$$n_i \leq \left(\frac{2}{3}\right)^i n_0 - 3\left(\frac{2}{3}\right)^i + 3$$

$$\boxed{n_i \leq \left(\frac{2}{3}\right)^i (n_0 - 3) + 3}$$

prova por indução ✓

Prova por indução em $i$, para $i = 1, 2, 3, \ldots$ que $n_i \leq \left(\frac{2}{3}\right)^i (n_0 - 3) + 3$

- Base: $i = 1$, então

$$n_1 \leq \frac{2}{3} n_0 - 2 + 3 \leq \frac{2}{3} n_0 + 1$$

- Passo: $i > 1$, então

$$n_i = \left\lceil \frac{2}{3} n_{i-1} \right\rceil \leq \frac{2}{3} n_{i-1} + 1$$

$$\leq \frac{2}{3}\left[\left(\frac{2}{3}\right)^{i-1}(n_0 - 3) + 3\right] + 1$$

$$n_i \leq \left(\frac{2}{3}\right)^i (n_0 - 3) + 3$$

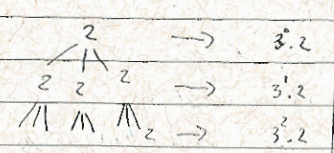
■

(Q6) Recorrência para o alg. TPSORT. Seja $n = j - i + 1$

Dada uma instância $A$ de tamanho $n$ temos:

$$T(n) = 2 \quad , \text{se } n \leq 2$$
$$T(n) = 3T\left(\frac{2n}{3}\right) + 2 \quad , \text{se } n > 2$$

- árvore de recorrência:



$2 \to 3 \cdot 2$

$3 \cdot 2 \to 3^2 \cdot 2$

$3^2 \cdot 2 \to \ldots$

até $\left(\frac{2}{3}\right)^i n = 2$

$$\frac{n}{2} = \left(\frac{3}{2}\right)^i$$

$$i = \log_{(3/2)}(n/2)$$

$$T(n) = \sum_{i=0}^{\log_{(3/2)}(n/2)-1} (2)(3^i) + 2(3)^{\log_{(3/2)}(n/2)}$$

$$= 2\left[\frac{3^{\log_{3/2}(n/2)} - 1}{2}\right] + 2\left(\frac{n}{2}\right)^{\log_{(3/2)} 3}$$

$$= \left(\frac{n}{2}\right)^{\log_{(3/2)}(3)} + 2\left(\frac{n}{2}\right)^{\log_{(3/2)}(3)} - 1$$

$$T(n) = 3\left(\frac{n}{2}\right)^{\log_{(3/2)}(3)} - 1$$





Prova por indução em $n$, para $n = 2, 3, 4, \ldots$

- Base: $n = 2$ temos $T(n) \leq 3\left(\frac{1}{2}\right)^{\log_{(3/2)}(3)} - 1 = 2$ (verdadeiro)

- Passo: Para $n > 2$ temos

$$T(n) \leq 3\,T\left(\frac{2n}{3}\right) + 2$$

$$\leq 3\left[3\left(\frac{2}{3}\frac{n}{2}\right)^{\log_{(3/2)}(3)} - 1\right] + 2 = 3\left[3^{\log_{(3/2)}(3/2)} \cdot 3^{\log_{(3/2)}(2/3\,n/2)} - 1\right] + 2$$

$$\leq 3\left[3^{\log_{(3/2)}(n/2)} - 1\right] + 2 = 3\left(\frac{n}{2}\right)^{\log_{(3/2)}(3)} - 1$$

Logo $T(n) = O\left(n^{\log_{(3/2)}(3)}\right)$ ■

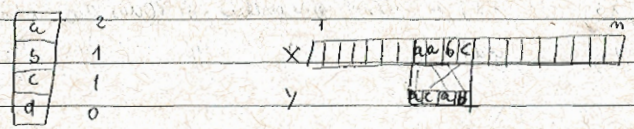
(Q7) Permutação $x = v\,y\,w$ $O(n|A|)$

$y$ é segmento de $x$ se existem $v$ e $w \in A^*$ tais que $x = v\,y\,w$

$|x| = n$ $|y| = m$,

Assume-se que $n \leq m$

Abordagem: contagem de elementos do alfabeto (extensão de frequências)



ALGORITHO( A, X, n, Y, m)

1. para k ← 1 até |A| faça } Θ(|A|)
2. Aff_count[k] ← 0.
3. j ← 1
4. para k ← 1 até |A| faça
5. para j ← 1 até m faça } O(m|A|)
6. se A[k] = Y[j]
7. então Aff_count[k] ← Aff_count[k] + 1
8. para k ← 1 até |A| faça
9. Aff_count_tmp[k] ← 0 } Θ(|A|)
10. calcula freq. para os primeiros m elementos de X } Θ(m|A|)
11. para i ← 1 até n-m+1 faça
12. compara Aff_count com Aff_count_tmp } Θ(|A|)
13. se comparação for igual
14. então devolva SIM } Θ(1)
15. senão Atualiza Aff_count_tmp, tirando o valor
16. correspondente a X[i] e acrescentando X[i+m]

Linhas 11-16: $O(n|A|)$

Complexidade do algoritmo $T(n) = O(n|A|)$.

(Q8) Caixas d-dimensionais.

(a) Seja $(x_1, x_2, \ldots, x_d)$ o vetor contendo os lados da caixa X, e $(y_1, y_2, \ldots, y_d)$ " " " " " " " Y





Abordagem: (Gulosa)

1 - Ordene x e y de forma crescente ( $O(n \lg n)$ se usado heapsort)
2 - Sejam x' e y' os vetores ordenados
3 - Para cada $1 \leq i \leq d$, verifique que seja válido:

$$x'[i] < y'[i]$$

4. Se o passo 3 for verdadeiro, então devolva SIM, caso contrário, devolva NÃO

• Complexidade: $O(d \lg d)$

• Corretude: Qual a sub-estrutura ótima?, Qual a propriedade gulosa?

(b) Usa-se uma variação do RADIX-SORT.

Algoritmo:

1 - Ordene toda caixa Bi, para i=1..n, → $O(nd \lg d)$
2 - para j ← d até 1 faça
3 - Ordene o lado k de toda caixa, preservando as posições originais (estável
Sejam $C_i$ as caixas ordenadas, para $i = 1, \ldots, n$
4 - k ← 1 S ← $C_k$
5 - para i ← 2 até n faça
6 - compara $C_i$ com $C_k$, se todo elemento de $C_k$ for menor do que $C_i$
7 - então k ← i $\sum_i \Theta(d)$
8 S ← S ∪ { $C_k$ }
9. devolva S

L1 = $O(nd \lg d)$ complexidade = $O(nd \lg d + dn \lg n)$
L2-L3 = $O(dn \lg n)$
L4 = $\Theta(1)$
L5-L8 = $O(dn)$

1998 / 2

(Q1) (i) $n^2 = O(n^2 - 100n\lg n)$

- é verdade mas é um abuso de notação.
- Existem constantes $c$ e $n_0$ tais que
$$n^2 \leq c(n^2 - 100n\lg n)$$

(ii) $n = O((n-\sqrt{n})/2)$

- é verdade (abuso de notação)
- Tome $c = 10$ e $n_0 = 100$, então é verdade que:

$$n \leq 10\left(\frac{n-\sqrt{n}}{2}\right) \qquad 100 \leq 10\left(\frac{100-10}{2}\right) = 450$$

(iii) $2^n = O(2^{n/2})$

- Falso
- suponha que seja verdade, então $2^n \leq c\,2^{n/2}$ para um certo $c$ logo

$$n \leq \lg c + \frac{n}{2}$$

$$\frac{n}{2} \leq \lg c \quad \text{ou} \quad 2^{n/2} \leq c$$ o qual é absurdo pois $c$ foi assumida como constante

(iv) $\sqrt{n+1} - \sqrt{n} = O(1/\sqrt{n})$

- verdade
- Tome $c=1$ e $n_0=1$ então:

$$\sqrt{n+1} - \sqrt{n} = \sqrt{n+1} - \sqrt{n}\left(\frac{\sqrt{n}}{\sqrt{n}}\right) = \sqrt{n+1} - \frac{n}{\sqrt{n}}$$

$$= \frac{\sqrt{n^2+n} - n}{\sqrt{\ldots}} \cdot \left(\frac{\sqrt{n^2+n}+n}{\sqrt{\ldots}+n}\right) = \frac{n^2+n-n^2}{\sqrt{\ldots}(\sqrt{\ldots}+n)} \leq \frac{1}{\sqrt{n}}$$





(v) $e^{\sqrt{\lg n}} = O\left(n^{1/\lg\lg n}\right)$

- verdadeiro
- tome $c=1$, $n_0=1$

$$e^{\left\{\sqrt{\lg n}\right\}} = e^{\left\{\sqrt{\lg n}\cdot\frac{\sqrt{\lg n}}{\sqrt{\lg n}}\right\}}$$

$$= e^{\left\{\lg n \cdot \frac{1}{\sqrt{\lg n}}\right\}} = \left(e^{\lg n}\right)^{1/\sqrt{\lg n}}$$

$\sqrt{x} \geq \lg x$ ✓

$$= n^{1/\sqrt{\lg n}}$$

robemos que $\sqrt{\lg n} \geq \lg\lg n$, logo:

$$e^{\sqrt{\lg n}} \leq n^{1/\lg\lg n}$$

(vi) $e^{\sqrt{\lg n}} = \Omega\left((\lg n)^{10^6}\right)$

- verdadeiro

Seja $\lg n = x$ então: $e^{\sqrt{x}} = \Omega\left((x)^{10^6}\right)$

É verdade que uma função exponencial é maior do que uma polinomial, quando $n$ é grande o suficiente.

(Q2) Número de quadrados $= \Theta(n^3)$

- # de quadrados de tamanho 1 $= (n-1)^2$ $= 16$ $4^2$
- " " " " " 2 $= (n-2)^2$ $= 9$ $3^3$
- " " " " " 3 $= (n-3)^2$ $= 4$ $2^2$
- " " " " " 4 $= (n-4)^2$ $= 1$ $1^2$

$\Sigma$

Assim o número de quadrados N será:

$$N = \sum_{i=1}^{n-1} (n-i)^2 = 1^2 + 2^2 + \ldots + (n-1)^2$$

$$= \sum_{i=1}^{n-1} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$

- Vou provar que $\sum_{i=1}^{n-1} i^2 = \Theta(n^3)$

(a) É verdade que $\sum_{i=1}^{n-1} i^2 \leq n^3$

$$\sum_{i=1}^{n-1} i^2 \leq \sum_{i=1}^{n-1} n^2 = n^2(n-1) = n^3 - n^2 \leq n^3$$

Logo $\sum_{i=1}^{n-1} i^2 = O(n^3)$

(b) É verdade que $\sum_{i=1}^{n-1} i^2 \geq n^3$

$$\sum_{i=1}^{n-1} i^2 = \sum_{i=1}^{(n-1)/2} i^2 + \sum_{i=(\frac{n-1}{2})+1}^{n-1} i^2$$

$$\geq \sum_{i=(\frac{n-1}{2})+1}^{n-1} i^2 \geq \sum_{i=(\frac{n-1}{2})+1}^{n-1} \left(\frac{n-1}{2}\right)^2$$

$$\geq \left(\frac{n-1}{2}\right)^2 \left[(n-1) - \left(\frac{n-1}{2}\right)\right] = \left(\frac{n-1}{2}\right)^3 = \frac{1}{8}(n-1)^3$$

$$\geq \frac{1}{8}\left(\frac{n}{2}\right)^3 \geq \frac{1}{64} n^3$$

$$= \Omega(n^3)$$





(Q3) Ordenar k listas, contendo ao todo n elementos, em $O(n \lg k)$ (P2)

CONCATENA $(A, k, n)$ ▷ $A = \{A_1, A_2, \ldots, A_k\}$

O(1) 1. se $k = 1$
2. então devolva $A_1$
3. $B \leftarrow \emptyset$

O(n) 4. para $i \leftarrow 1$ até $k-1$ incrementando em 2 elementos ▷ $i = 1, 3, 5, 7 \ldots$
5. $B \leftarrow B \cup$ INTERCALA $(A_i, A_{i+1})$ ▷ $\{1,2\}, \{3,4\}, \{5,6\}$.

O(1) 6. se $k = 1 \bmod 2$ ▷ se é ímpar
7. então $B \leftarrow B \cup A_k$

$T(n, k/2)$ 8. devolva CONCATENA $(B, \lceil k/2 \rceil, n)$

complexidade: Seja $T(n,k)$ a complexidade de concatenar k listas de ao todo n elementos

$$T(n, k/2) = O(n) + T(n, \lceil k/2 \rceil)$$

ou grosseiramente:

$$T(n,1) = 1$$
$$T(n,k) = T(n, k/2) + n$$

- árvore de recorrência:

$$T(n,k) = \sum_{i=0}^{\lg k - 1} n + 1 = n \lg k + 1$$

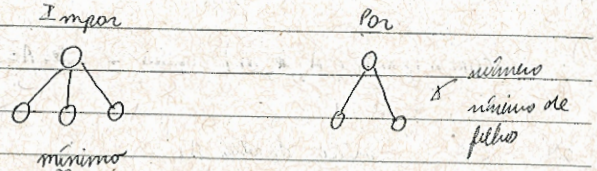
$$\leq n \lg k + n \lg k = 2n \lg k$$

até $\frac{k}{2^i} = 1$ logo $i = \lg k$

$$T(n,k) = O(n \lg k)$$

(Q4) Árvore par-ímpar

Ideia grosseira:




↳ número de filhos = 3 para satisfazer a propriedade

Ou seja a árvore terá no mínimo a forma de uma árvore binária completa

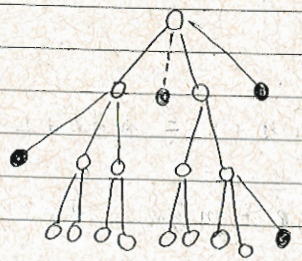
• Sabemos que uma árvore binária de altura $h$ tem $2^{h+1}-1$ nós, assim

$$n \geq 2^{h+1} - 1$$

$$\lg(n+1) \geq h+1$$

$$h \leq \lg(n+1) - 1 \leq \lg(n+n) - 1 = \lg(2n) - 1 = \lg 2 + \lg n$$

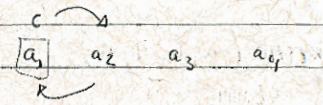
$$h \leq \lg n, \text{ logo } h = O(\lg n)$$







(Q5) Celebridade




▷ Alg. que determina a celebridade $a_i \in A[1..n]$ em que $A$ é um vetor de tamanho $n$

```
CELEBRIDADE (A, n)
1   c ← A[1]
2   i ← 2
3   enquanto i < n faça
4       se A[i] conhece c
5           então se c conhece A[i]
6               então ▷ nenhum dos 2 é celebridade
7                   c ← A[i+1]
8                   i ← i+2
9               senão ▷ A celebridade é a mesma
10                  c ← c
11          senão ▷ c não é mais celebridade
12              se c conhece A[i]
13                  então c ← A[i]
14                  i ← i+1
15              senão ▷ A[i] não é celebridade
16                  c ← A[i+1]
17                  i ← i+2
18  devolva c
```

Complexidade $O(n)$

(Q6) Comparação entre vetores com dados aleatórios.

```
ALGORITMO (a, b, N, n)
1   a[n+1] ← 1
2   b[n+1] ← 1
3   i ← 1
4   enquanto a[i] ≠ 1 ou b[i] ≠ 1 faça
5           i ← i+1
6   se i ≤ n
7       então devolva 1
8       senão devolva 0
```

Temos 4 opções entre 2 elementos de a e b: $\begin{matrix}00\\01\\10\end{matrix}\}\ 3/4$ ; $\begin{matrix}11\end{matrix}\ 1/4$

Seja $x_i$ a probabilidade de na iteração $i$: $a[i] = b[i] = 1$, isto é $1/4$

Seja $\overline{x_i}$ a " " " " $i$: $a[i] \neq 1$ ou $b[i] \neq 1$ isto é $3/4$

Assim, sendo a e b escolhidos ao acaso, o tempo médio $(T(n))$ de execução do algoritmo será:

$$T(n) = x_1 + \overline{x_1}x_2 + \overline{x_1}\,\overline{x_2}x_3 + \ldots + \overline{x_1}\,\overline{x_2}\,\overline{x_3}\ldots\overline{x_{n-1}}x_n$$

$$T(n) = \frac{1}{4} + \frac{3}{4}\cdot\frac{1}{4} + \left(\frac{3}{4}\right)^2\left(\frac{1}{4}\right) + \ldots + \left(\frac{3}{4}\right)^{n-1}\frac{1}{4}$$

$$= \frac{1}{4}\sum_{i=0}^{n-1}\left(\frac{3}{4}\right)^i \leq \frac{1}{4}\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i \leq \frac{1}{4}\left[\frac{1}{1-3/4}\right] \leq 1$$

Logo $T(n) = O(1)$.

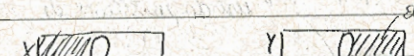




(Q7) Mediana em $O(\lg n)$

- Sequências x e y estritamente crescentes (com elementos diferentes)
- Tome-se cuidado para em cada iteração usar a mesma quantidade de elementos

```
MEDIANALOGN(X, px, rx, Y, py, ry)   ▷ x e y tem sempre o mesmo # de elem.
1. se px ≤ rx
2.     então se rx - px + 1 = 1   ▷ se tem 1 elemento em ambos vetores
3.                 então se X[1] < Y[1]   (pega o menor)
4.                     então devolva X[1]
5.                     senão devolva Y[1]
6.                 senão   ▷ se os vetores tem mais de 1 elemento
7.                     q ← ⌊(rx - px + 1)/2⌋
8.                     se X[px+q] < Y[py+q]
9.                         então devolva MEDIANALOGN(X, px+q, rx, Y, py, py+q)
10.                        senão devolva MEDIANALOGN(X, px, px+q, Y, py+q, ry)
```

(1) 


(2) 


complexidade: $T(n) = \Theta(1) + T(n/2)$, grosseiramente: $T(1) = a$

$T(n) = a + T(n/2)$,

$$T(n) = \sum_{i=0}^{\lg n} a$$

$$= a(\lg n + 1) \leq a(\lg n + \lg n) \leq 2a\lg n = O(\lg n)_{\blacksquare}$$

(Q8) Transposições

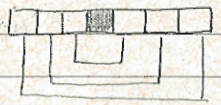
- Transposição é uma bijeção cujo suporte tem cardinalidade 2 (= intercambio)
  Ou seja 2 elementos do vetor não estão na ordem

(i) Considere-se a primeira implementação do PARTICIONE, em que devolve o vetor em 2 grupos tais que $A[1..j] \leq A[j+1..n]$
(devolve j)

No algoritmo PARTICIONE, os elementos são permutados / intercambiados 2-a-2 (i.e. usando uma transposição)

- LIMITE DAS OPERAÇÕES: Os executados no algoritmo PARTICIONE, para um vetor de n elementos

$= \lfloor n/2 \rfloor$



(ii) QS(A, p, r)
se p < r
então q ← PARTICIONE(A, p, r) ← uso do particione de Hoare
QS(A, p, q)
QS(A, q+1, r)

- No pior caso a partição não é balanceada, i.e., um lado da partição fica com 1 elemento e a outra com n-1

- Seja T(n) o número de operações Os, assim

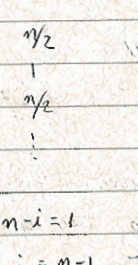
$T(1) = 0$
$T(n) \leq T(n-1) + \lfloor n/2 \rfloor$
↳ estimado em (i)





ou grosseiramente $T(1) = 0$
$T(n) \leq T(n-1) + n/2$

- árvore de recorrência:




$$T(n) \leq \sum_{i=0}^{n-2} (n/2) + 0$$
$$\leq \frac{n}{2}(n-1) = \frac{1}{2}(n^2 - n)$$
$$\leq \frac{1}{2} n^2$$
$$T(n) = O(n^2)$$

(iii) Acha que essa questão está errada pois para um vetor ordenado não é necessária nenhuma transposição. Assim T tem pelo menos 0 elementos

- Quais são as transposições possíveis

$$C\left(\frac{n}{2}\right) = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)}{2}. \quad \text{Logo } T \leq \frac{n^2 - n}{2}$$

(iv) O que quer dizer "n suficientemente grande"? (Formulação errada!)

$$n!^2 = 1.2.3 \cdots (n-1)(n) \cdot (n)(n-1) \cdots 2.1 = \prod_{i=0}^{n-1} (i+1)(n-i) = \prod_{i=0}^{n-1} (in - i^2 + n - i)$$
$$= \prod_{i=0}^{n-1} \left(n + i\underbrace{(n-1-i)}_{\geq 0}\right)$$
$$\geq \prod_{i=0}^{n-1} (n) = n^n$$

Logo $(n!)^2 \geq n^n$ então $n! \geq n^{n/2}$

É verdade que $n^n \geq n! \geq n^{n/2}$ ✓

(Q1) Sejam $f$ e $g$ funções de $\mathbb{N}$ em $\mathbb{N}$

(a) $f(n) = O(g(n))$ implica que $2^{f(n)} = O(2^{g(n)})$

- falso
- Tome $f(n) = 2n$
  $g(n) = n$ então $f(n) = O(g(n))$
  $2n \leq 4n$ para $c=4$, $n \geq 1$

No entanto não é verdade que $2^{2n} = O(2^n)$. Suponha que seja verdade. Então existem constantes $c > 0$ e $n \geq n_0$ tais que $2^{2n} \leq c\,2^n$ então $c \geq 2^n$ o que é absurdo

(b) $f(n) = O(n)$ e $g(n) = O(f(n))$ implica que $g(n) = O(n)$ ?

- verdadeiro

Se $f(n) = O(n)$ então $f(n) \leq c_0 n \quad \forall n \geq n_0$
Se $g(n) = O(f(n))$ então $g(n) \leq c_1 f(n) \quad \forall n \geq n_1$

Logo $g(n) \leq c_1 f(n) \leq c_0 c_1 n \quad \forall n \geq \max\{n_0, n_1\} \Rightarrow g(n) = O(n)$

(c) $f(n) = \Omega(n)$ e $g(n) = O(f(n))$ implica que $g(n) = O(n)$?

- falso
- Se $f(n) = \Omega(n)$ então $f(n) \geq c_0 n$ para todo $n \geq n_0$
- se $g(n) = O(f(n))$ então $g(n) \leq c_1 f(n)$ " " $n \geq n_1$

CONTRAEXEMPLO: $f(n) = n^3 \rightarrow n^3 \geq c_0 n$
$g(n) = n^2 \rightarrow n^2 \leq c_1 n^3$
não implica que $n^2 \leq cn$.

CONTRAEXEMPLO:
$f(n) = n$
$g(n) = \lg n$
Logo $\lg n \geq n$

(d) $f(n) = \Omega(n)$ e $g(n) = O(f(n))$ implica que $g(n) = \Omega(n)$?

- falso
- Se $f(n) = \Omega(n)$ então $f(n) \geq cn$ para este $c$ e $n \geq n_0$
- Se $g(n) = O(f(n))$ então $g(n) \leq c_1 f(n)$ " " $n \geq n_1$





(Q2) Meu manual diz que um problema "está em NP"; concluo então que não existe algoritmo polinomial para o problema. Estou certo ou errado?


$P \subseteq NP$

- Falso, a conclusão está errada!
- Sabemos que todo problema em P está em NP pois com a solução polinomial, podemos criar um certificado para que um alg. polinomial verifique a resposta para SIM
- Portanto, um problema se está em P então existe solução polinomial.

(Q3) Problema um pouco abstrato.

O consumo do algoritmo, certamente, não é polinomial no tamanho da entrada. Na verdade é pseudo-polinomial.

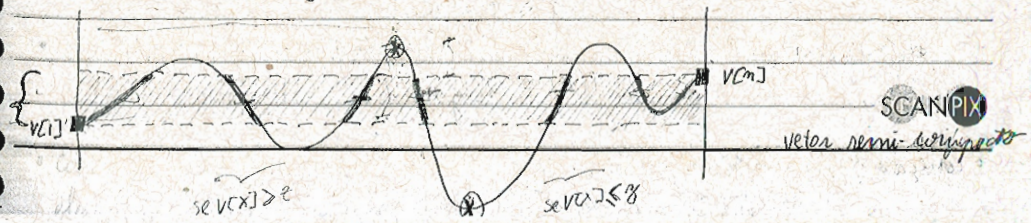
Seja o tamanho da entrada $m$ (número de dígitos), então o consumo de tempo será

$$T(n) = O(n) \quad \text{mas } m = \lg n \text{ ou } n = 2^m$$

$$T(m) = O(2^m)$$

(Q4) $V[1..n]$ de inteiros, Para todo $i$ vale $|V[i] - V[i+1]| \leq 1 \quad 1 \leq i < n$

- Observe-se que $z$ deve estar limitado por $V[1] \leq z \leq V[n]$



```
BUSCA(V, n, z)
1. devolva BUSCA-REC(V, 1, n, z)

BUSCA-REC(V, p, q, z)

1.  se V[p] = z
2.      então devolva p
3.  se V[r] = z
4.      então devolva r
5.  q ← ⌊(p+r)/2⌋
6.  se V[q] = z
7.      então devolva q
8.  se V[q] > z
9.      então devolva BUSCA-REC(V, p+1, q-1, z)
10.     senão devolva BUSCA-REC(V, q+1, r-1, z)
```

- Complexidade: Seja $T(n)$ o consumo de tempo do algoritmo BUSCA-REC para um tamanho de instância $n := p - q + 1$

$T(1) = 2$

$T(n) = T(n/2) + 4$

- árvore de recorrência

4
|
4
|
4
.
.
.

até $i = \lg n$

$$T(n) = \sum_{i=0}^{\lg n - 1} 4 + 2$$

$$= 4 \lg n + 2 \leq 6 \lg n$$

$T(n) \leq 6 \lg n$

$T(n) = O(\lg n)$

- começa o?





(Q5) QUICKSORT com recursão da cauda. → 123456 → 342516

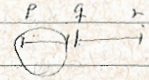
```
QS2(A, p, r)
1.  enquanto p < r faça
2.      q ← PARTICIONE(A, p, r)        Θ(n)
3.      QS2(A, p, q-1)
4.      p ← q+1
```

(a) Mostre que a pilha de recursão pode atingir uma altura proporcional a $n$, $n := r - p + 1$

- Um cenário em que a profundidade da pilha é $\Theta(n)$:
- Sabemos que PARTICIONE custa $\Theta(n)$

- A profundidade da pilha será $\Theta(n)$ quando elege-se sempre exclusivamente ou o maior ou o menor elemento de A entre A[p] e A[r] como pivô para cada iteração do loop (i.e. quando a partição for desbalanceada)

(b)




```
QS2'(A, p, r)
1.  enquanto p < r faça
2.      q ← PARTICIONE(A, p, r)
3.      se q-p > r-q
4.          então QS2'(A, q+1, r)
5.                r ← q-1
6.          senão QS2'(A, p, q-1)
7.                p ← q+1
```

- Escolhe-se a menor metade da partição, já que é mais provável que a pilha não aumente, comparado com a metade MAIOR.



- como o alg. faz chamada recursiva em no MÁXIMO a METADE de elementos. então a profundidade da pilha será $O(\lg n)$

## (Q6) Contador de sub-seqüências

Dados $X[1..m]$ e $Y[1..n]$ contar o número de ocorrências de $X$ em $Y$ em $O(mn)$

- Seja $c[i,j]$ o número de ocorrências de subseq. de $X[1..i]$ em $Y[1..j]$ para $1 \le i \le m$ e $1 \le j \le n$


- Recorrência:

$c[i,j] \leftarrow c[i-1, j-1] + c[i, j-1]$ , se $X[i] = Y[j]$

$c[i,j] \leftarrow c[i, j-1]$ , se $X[i] \neq Y[j]$

Caso base:

$c[0,j] \leftarrow 1$ , $j = 0, \ldots, n$ ▷ sub-seq. vazia pertence a toda seq.

$c[i,0] \leftarrow 0$ , $i = 1, \ldots, m$ ▷ não existe sub-seq. numa seq. vazia

- SUBSEQ-CONT $(X, m, Y, n)$

```
1-  para j ← 0 até n faça
2-      c[0,j] ← 1
3-  para i ← 1 até m faça
4-      c[i,0] ← 0
5-  para i ← 1 até m faça
6-      para j ← 1 até n faça
7-          se X[i] = Y[j]
8-              então c[i,j] ← c[i-1,j-1] + c[i,j-1]
9-              senão c[i,j] ← c[i,j-1]
10- devolva c[m,n]
```

- Consumo de tempo: $O(mn)$



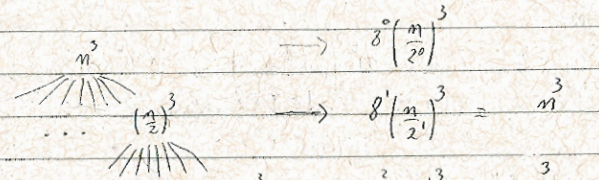


## (Q7) Recorrência.

(a) $T(1) = 0$

$T(n) = 8\,T(n/2) + n^3$

(b) • árvore de recorrência:




até $n = 2^i$

$i = \lg n$

$8^i\left(\frac{n}{2^i}\right)^3 = n^3$

$$T(n) = \sum_{i=0}^{\lg n - 1} n + 0 = n^3 \lg n$$

- Vou provar por indução em $n$, que $T(n) \ge n^3 \lg n$, para $n = 1, 2, 3, \ldots$

- Base: $n = 1$

então $T(1) = 1^3 \lg 1 = 0$ (verdade)

- Passo: $n \ge 1$

então $T(n) = 8\,T(n/2) + n^3$

$$\ge 8\left[\left(\frac{n}{2}\right)^3 \lg\left(\frac{n}{2}\right)\right] + n^3 = n^3(\lg n - 1) + n^3 = n^3 \lg n$$

$\ge n^3 \lg n$

C.Q.D.

$T(1) = 0$

(c) $T(n) = 7\,T(n/2) + n^3$

$$T(n) = \sum_{i=0}^{\lg n - 1} \left(\frac{7}{8}\right)^i n^3 = n^3 \sum_{i=0}^{\lg n - 1} \left(\frac{7}{8}\right)^i$$

$$T(n) \leq n^3 \sum_{i=0}^{\infty} \left(\frac{7}{8}\right)^i = n^3 \left(\frac{1}{1 - 7/8}\right) = 8n^3$$

Logo

$$T(n) = O(n^3)$$

(Q8) ~~Algoritmo: FAZ-ALGO~~

- Base $n = 1$

$T(1) \leq 8$ (verdade)

- Passo $n > 1$

$$T(n) = 7\,T(n/2) + n^3$$

$$\leq 7\left(8\left(\frac{n}{2}\right)^3\right) + n^3$$

$$\leq 7n^3 + n^3$$

$T(n) \leq 8n^3$ C.Q.D.





(Q8) Algoritmo FAZ-ALGO:

```
FAZ-ALGO (E, n)
1   j ← 1
2   t ← 0
3   S[1] ← 0
4   enquanto j ≤ n faça
5       enquanto t > 0 e E[t] ≠ E[j] faça
6           t ← S[t]
7       t ← t + 1
8       j ← j + 1
9       se E[j] = E[t]
10          então S[j] ← S[t]
11          senão S[j] ← t
12  devolva S
```

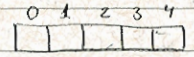
- A afirmação do aluno está correta. O alg. FAZ-ALGO consome $O(n)$
- Claramente a linha 5 será executada n vezes pois j começa com 1 unidade e em cada iteração é incrementada em 1, no entanto a linha 5, dependente dos valores de E será executada no máximo uma vez.
  - → 0 ou 1 vezes se $E[t] = E[j]$ ou
  - → 1 vez se $E[t] \neq E[j]$

2004/1 (A. Mandel)

(Q1) Recorrência : Considere que os índices para vetores em C iniciam em 0

(a) $T(1) = 1$

$T(n) = T(n-1) + 1/n$




Prob

1/4

1/3

1/2 → V[0] < V[1]

1 → mesmo

(b)

$T(n) = \frac{1}{n} + \frac{1}{n-1} + \cdots \frac{1}{2} + 1$

$= \lg n + \Theta(1) \leq 2 \lg n$

$T(n) = O(\lg n)$

(Q2) Listas ordenadas A e B (com elementos distintos) $A[i] \in B$, $\forall i$?

▶ Dados 2 listas encadeadas, com elementos distintos, decida se todo elemento da lista A está na lista B

Propriedade : vol[x] : valor contido em x

prox[x] : próximo elemento da lista encadeada

cada lista está representada por um apontador





```
ALGORITMO-EMB ( x , y )
1.- enquanto y ≠ NIL faça
2.-     se vol[x] = vol[y]
3.-         então x ← prox[x]
4.-               y ← prox[y]
5.-         senão se vol[x] < vol[y]
6.-                 então y ← prox[y]
7.-                 senão devolva "NÃO"
8.- se x = NIL
9.-     então devolva "SIM"   ▶ Se chegou no final de X
10.- devolva "NÃO"            ▶ Se chegou no final de Y
```

(Q3) Para que valores reais $\alpha$ é verdade que $\lg n = O(n^\alpha)$

$\lg n$ será $O(n^\alpha)$ se existem constantes c e $n_0$ tais que

$\lg n < c n^\alpha$ logo $\lg \lg n < \lg c + \alpha \lg n$

$$\alpha > \frac{\lg \lg n - \lg c}{\lg n} = \frac{\lg(\lg n / c)}{\lg n} = \frac{\lg n/c}{n}$$

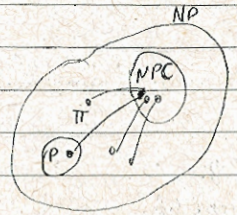
$$\alpha > \left(\frac{1}{c}\right) \frac{\lg n}{n}$$

Sabemos que $\lim_{n \to \infty} \frac{\lg n}{n} = 0$

Portanto para valores de $\alpha > 0$ é verdade que $\lg n = O(n^\alpha)$

(veja Pag 41)

(Q4) P=NP sse algum problema em P é NP-completo (verdadeiro)




Π é NPCompleto
- Se Π está em NP
- Todo problema Π' em NP pode ser reduzível a Π, i.e., $\Pi' \leq_p \Pi$ para todo $\Pi' \in NP$

← • Se P é NPC não garante que P = NP.

→ • Se P=NP então é verdade que algum problema em P é NPC. ✓

SUBSET-SUM

(Q5) ~~Versão booleana~~. Dado A[1..n] e um inteiro S, determinar se existe algum subcjto de A cuja soma é S.

• O problema pode ser visto como dependente de 2 parâmetros: n e S.
→ Se n é pequeno, então a busca exaustiva é suficiente
→ Se S é grande podemos usar Prog. dinâmica

• Algoritmo exponencial: força bruta é $O(n2^n)$ dado que existem $2^n$ possíveis conjuntos e para cada conjunto testamos se a soma é igual a S.

• Algoritmo exponencial + mais eficiente é $O(n2^{n/2})$
- Divida os n elementos em 2 conjuntos de n/2 elementos
- Para cada conjunto calcule a soma dos possíveis $2^{n/2}$ conjuntos armazenando o resultado num vetor de $2^{n/2}$ elementos
- Ordene a primeira metade em forma decrescente e a outra crescentemente $O(2^{n/2} \cdot lg(2^{n/2})) = O(n2^{n/2})$
- Verifique em $O(2^{n/2})$ se um elemento da primeira metade mais um elemento da segunda é igual a S. [se ... então i ← i+1 senão ...]





Recorrência: Seja $s[i,S] = \begin{cases} 1, & \text{se } s[1..i] \text{ tem solução} \\ 0, & \text{c.c.} \end{cases}$

$s[i,0] = 1$ ▶ Existe uma solução vazia para $1 \leq i \leq n$?

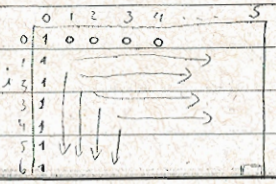
$s[0,S] = 0$ , $S>0$ ▶ não existem elementos que somam S

$s[i,S] = s[i-1,S]$ , se $A[i] > S$

$s[i,S] = \max\{s[i-1,S], s[i-1,S-A[i]]\}$ , se $A[i] \leq S$

se não tá — se ele tá

SUBSET-SUM (A, n, W)
1 aloca s(0..n, 0..W)
2 para i ← 0 até n faça
3 s[i,0] ← 1
4 para S ← 1 até W faça
5 s[0,S] ← 0
6 para i ← 1 até n faça
7 se A[i] > W
8 então s[i,S] ← s[i-1,S]
9 senão se s[i-1,S] = 0
10 então s[i,S] ← s[i-1, S-A[i]]
11 senão s[i,S] ← s[i-1,S]
12 devolva s(n,W)




complexidade: $T(n,W) = \Theta(nW)$

$\Theta(n2^{lg W})$.

pseudo-polinomial

(Q6) Função de otimização? $d(x,y,z) = |x-y| + |y-z| + |z-x|$

(a)
```
ALGORITMO-FORÇA-BRUTA (A, n)   ▷ A[1..n] vetor de inteiros
1.  min ← ∞
2.  para i ← 1 até n faça
3.      para j = 1 até n faça
4.          se i ≠ j
5.              então para k ← 1 até n faça
6.                      se k ≠ i e k ≠ j
7.                          então d ← |A[i] - A[j]| +
8.                                    |A[j] - A[k]| +
9.                                    |A[k] - A[i]|
10.                                 se min < d
11.                                     então min ← d
12.                                           x ← i
13.                                           y ← j
14.                                           k ← k
15. devolva (x, y, z)
```

Complexidade $O(n^3) = n(n-1)(n-2) = O(n^3)$

(b) Sabemos que para qualquer valor de x, y e z, d sempre será igual a zero! Assim, qualquer tripla (x,y,z) minimizam d

```
ALGORITMO-BOBO (A, n)
1. devolva (A[1], A[2], A[3])
```

Questão esquisita!



(Q7) Árvore embutida (Árvore A sendo zona da árvore B) (Veja Pág 45)

```
EMBUTIDO (A, B)
    r ← raiz(A)
    se r = NIL
        então devolve "TRUE"   ▷ árvore vazia está embutida em outra
    x ← PROCURE-ELEMENTO (r, B)
    se x ≠ FALSE
        então devolva EMBUTIDO (esq[x], B) e EMBUTIDO (dir[x], B)
        senão devolva "FALSE"
```

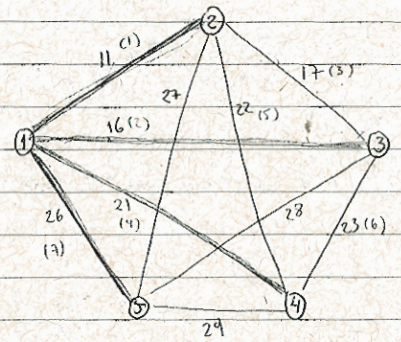
Obs: Tanto A e B devem preservar a ordem. Isto é se xi é descendente de xj em A, xi deverá ser descendente em B.

Complexidade: Como o alg. procura cada elemento de A em B, mantendo a ordem de descendência, então a complexidade é $O(n)$, em que n é o número de nós na árvore B

(Q8) ALGORITMO KRUSKAL $\binom{n}{2} = \frac{n!}{2!(n-2)!}$

- Num grafo com n vértices e $\frac{n^2-n}{2}$ arestas (i.e. grafo completo) com pesos desconhecidos o consumo de tempo será $O(n^2 \lg n^2) = O(n^2 \lg n)$
- Tendo como peso da aresta $(i,j) = \min(i,j) + n \times \max(i,j)$ as arestas escolhidas para o MST serão todas aquelas que tenham o vértice 1 mas a gente desconhece a ordem, assim $T(n) = \Omega(n^2)$ É dado pela ordem do BUILD-HEAP
- Tendo como peso da aresta $(i,j) = \max(i,j) + n * \min(i,j)$, as n primeiras arestas escolhidas serão suficientes para obter uma MST, Assim $T(n) = O(n^2)$

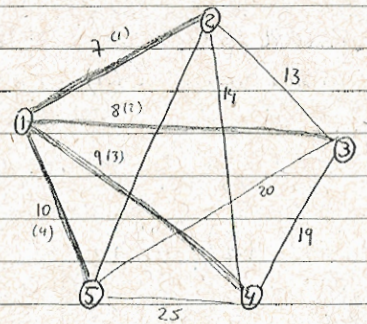
(1)




arestas descartadas pelo KRUSKAL $= \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} i = n(n-1) - \frac{n^2-n}{2}$.

$T(n) = \Omega(n^2)$

(2)




$T(n) = O(n^2)$

- Arestas: 7
  - (1,2) ✓
  - (1,3) ✓
  - (2,3) X
  - (1,4) ✓
  - (2,4) X
  - (3,4) X
  - (1,5) ✓
  - (2,5)
  - (3,5)
  - (4,5)

- Arestas: $(n-1) = 4$
  - (1,2)
  - (1,3)
  - (1,4)
  - (1,5)





2007/1

**(Q1)** Definições

(a) Se $\exists\ a>1, b>1, c>1$ e $n_0>0$ tais que $a^{f(n)} \leq b^{g(n)} \leq c^{f(n)}$ $\forall n \geq n_0$ então $g(n) = \Theta(f(n))$

Resposta:

Sabemos que $a = b^{\log_b(a)}$ e $c = b^{\log_b c}$, assim temos que

$$b^{\log_b(a) f(n)} \leq b^{g(n)} \leq b^{\log_b(c) f(n)}$$

$$\log_b(a) f(n) \leq g(n) \leq \log_b(c) f(n)$$

$$c_1 f(n) \leq g(n) \leq c_2 f(n)$$

Portanto $g(n) = \Theta(f(n))$

(b) Se $g(n) = \Theta(f(n))$ então $\exists\ a>1, b>1, c>1$ e $n_0>0$ tais que $a^{f(n)} \leq b^{g(n)} \leq c^{f(n)}$ $\forall n \geq n_0$

Resposta: Sabemos que $c_1 g(n) \leq f(n) \leq c_2 g(n)$ para um $n \geq n_0$ e constantes $c_1, c_2$

considere $c_1 = \frac{\lg(a)}{\lg(b)}$ e $c_2 = \frac{\lg(c)}{\lg(b)}$ então

$\frac{\lg(a)}{\lg(b)} g(n) \leq f(n) \leq \frac{\lg(c)}{\lg(b)} g(n)$ Logo $\lg(a)\, g(n) \leq \lg(b)\, f(n) \leq \lg c\, g(n)$

Portanto $a^{g(n)} \leq b^{f(n)} \leq c^{g(n)}$ para todo $n \geq n_0$, com $n_0>0$ e $a>1, b>1, c>1$.

(Q2) Problema da tripartição.

- O aluno está certo.

Seja $\Pi$ o problema de decisão correspondente à PARTIÇÃO, e $\Pi'$ o problema da TRIPARTIÇÃO, então vou provar que $\Pi'$ é NP completo baseado numa redução de $\Pi$

- $\Pi'$ está em NP : É fácil decidir em tempo polinomial que, para uma instância $I \in \Pi'$, $I$ corresponde a uma tripartição

- Podemos reduzir polinomialmente $\Pi \leq_p \Pi'$ PART → TRIPART

REDUÇÃO:

- Tome o vetor A de n elementos e estime a somatória de todos os seus elementos, seja $x$ esse valor.
- Crie um novo vetor B com n+1 elementos
- copie os elementos de A em B
- Atribua na posição n+1 de B o valor $x/2$
- Use o algoritmo TRI-PARTICIONE (B, n+1) para "decidir" o problema

Isto é

$$B = A[1..n] \cup \left\{ \frac{\sum_{i=1}^{n} A[i]}{2} \right\}$$

Exemplo $A = \{1, 3, 7, 5, 6\} \longrightarrow$ PARTIÇÃO $= \{1, 3, 7\}\{5, 6\}$

$B = \{1, 3, 7, 5, 6, 11\} \longrightarrow$ TRIPARTIÇÃO $= \{1, 3, 7\}\{5, 6\}\{11\}$

resposta SIM da

- Para toda instância de $I$ de $\Pi$, teremos SIM em $I' \in \Pi'$, e NÃO em caso contrário.





(Q3) Recorrências:

Seja $T(n)$ o consumo de tempo do algoritmo, para uma instância de tamanho $n$. A seguir listamos as recorrências

a) $T(1) = 1$
$T(n) = 5T(n/2) + O(n)$
Recorrência definida para $n$ multiplo de 2 e positivo

b) $T(1) = 1$
$T(n) = 2T(n-1) + O(1)$
Recorrência definida para $n \geq 1$

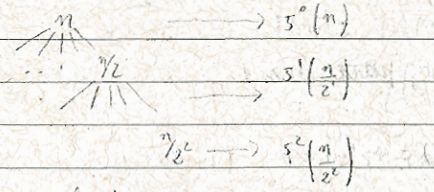
c) $T(1) = 1$
$T(n) = 9T(n/3) + O(n^2)$
Recorrência definida para $n$ multiplo de 3

Resolução de recorrências:

a) Grosseiramente a recorrência (a) podemos descrevê-la como:

$T(1) = 1$
$T(n) = 5T(n/2) + n$

- árvore de recorrência para tentar achar a fórmula fechada.




$$T(n) = \sum_{i=0}^{\lg n - 1} \left(\frac{5}{2}\right)^i n + 5^{\lg n}$$

$$= n\left[\frac{(5/2)^{\lg n} - 1}{5/2 - 1}\right] + n^{\lg 5} = \frac{2}{3} n \left[\frac{n^{\lg 5} - n}{n}\right] + n^{\lg 5}$$

$$= \frac{5}{3} n^{\lg 5} - \frac{2}{3} n = O(n^{\lg 5})$$

Vou provar por indução em $k$, com $n = 2^k$, para $k = 0, 1, 3 \ldots$, que

$T(n) = \frac{5}{3} n^{\log 5} - \frac{2}{3} n$

- Base: $k = 0$, então $T(1) = \frac{5}{3} - \frac{2}{3} = 1$ (verdade)
- Passo: $k \geq 1$, então $T(n) = 5\,T(n/2) + n$

$$= 5\left[\frac{5}{3}\left(\frac{n}{2}\right)^{\log 5} - \left(\frac{2}{3}\right)\left(\frac{n}{2}\right)\right] + n$$

$$= 5\left[\frac{1}{3}\left(\frac{n^{\log 5}}{5}\right) - \frac{n}{3}\right] + n$$

$$= \frac{5}{3} n^{\log 5} - \frac{2}{3} n \qquad \text{C.Q.D}$$

(b) Grosseiramente podemos expressar a recorrência do alg B como:

$T(1) = 1$

$T(n) = 2\,T(n-1) + 1$

$$T(n) = \sum_{i=0}^{n-2} 2^i + 2^{n-1} = (2^{n-1} - 1) + 2^{n-1} = 2^n - 1$$

Prova por indução em $n$, para $n \geq 1$

- Base $n = 1$, então $T(1) = 2^1 - 1 = 1$ (verdade)
- Passo $n > 1$, então $T(n) = 2\,T(n-1) + 1$

$$= 2\left[2^{n-1} - 1\right] + 1$$

$$= 2^n - 1$$

Assim $T(n) = 2^n - 1 = O(2^n)$



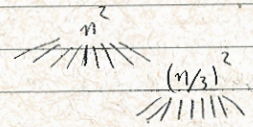


(c) Grosseiramente.

$T(1) = 1$

$T(n) = 9\,T(n/3) + n^2$

- árvore de recorrência



$n^2 \longrightarrow n^2 = n^2$

$(n/3)^2 \longrightarrow 9\,(n/3^1)^2 = n^2$

$(n/3^2)^2 \longrightarrow 9^2\,(n/3^2)^2 = n^2$

$n = 3^i \Rightarrow i = \log_3 n$

$$T(n) = \sum_{i=0}^{\log_3 n - 1} n^2 + 9^{\log_3 n}$$

$$= n^2 \log_3(n) + n^2$$

- Prova por indução em $k$, para $n = 3^k$, com $k = 0, 1, 2, \ldots$
  - Base: $k = 0$, então $n = 1$ $T(1) = 1 \lg(1) + 1 = 1$ (verdade)
  - Passo: $k \geq 1$, então $T(n) = 9\,T(n/3) + n^2$

$$= 9\left[\left(\frac{n}{3}\right)^2 \log_3\left(\frac{n}{3}\right) + \left(\frac{n}{3}\right)^2\right] + n^2$$

$$= n^2 \log_3(n) + n^2$$

Portanto $T(n) = n^2 \log_3 n + n^2 = O(n^2 \log_3(n))$

A: $O(n^{\log 5})$

B: $O(2^n)$

C: $O(n^2 \log_3 n)$

- O algoritmo C é o mais eficiente no pior caso.

(Q4)

• Na linha 2 $i < n$

$k+j+1 < n$

então $i+k+j < i+k+j+1 < 2n$

• Sejam $k, i, j$ os valores de início numa iteração e $k', i', j'$ no final

* Na linha 4: $k' = k$, $i' = i$, $j' = j+1$ $\Rightarrow$ $k'+i'+j' = k+i+j+1$

* Na linha 6: $k' = k$, $i' = i+j+1$, $j' = 0$ $\Rightarrow$ $k'+i'+j' = k+i+j+1$

* Na linha 8: $k' \geq k+j+1$, $i' \geq i+1$, $j' = 0$ $\Rightarrow$ $k'+i'+j' = k+i+j+2$

Logo $k+i+j$ aumenta em pelo menos 1 em cada iteração.

Dado que no início $k+i+j = 1$ e no final $k+i+j \leq 2n$, então são feitas no máximo $2n-1$ iterações, portanto o consumo de tempo é $O(n)$

(Q5) Consumo de tempo de n operações

(a) 1 2 | 3 4 | 5 6 | 7 8 | ...

v a | v a | v a | v a |

$2^0$ 1 | $2^1$ 1 | $2^2$ 1 | $2^3$ 1 |

Número de grupos = $x$ então

$x \cdot 2 \leq n \Rightarrow x \leq n/2$



$T(n) \leq \sum_{i=0}^{n/2} (2^i + 1) = 2^{n/2+1} - 1 + n/2 + 1 \leq 2 \cdot 2^{n/2} + 2^{n/2} = 3 \cdot 2^{n/2}$

$T(n) \leq 3 \cdot 2^{n/2} = O(2^{n/2})$

Portanto uma sequência de n operações consumirá tempo $O(2^{n/2})$

(b)

| v a...a | v a...a | v a...a | ...

$2^0$ $\sqrt{n}$ $\sqrt{n}$ $\sqrt{n}$

Seja $x$ o número de grupos, logo:

$x(1+\sqrt{n}) \leq n \Rightarrow x \leq \frac{n}{(1+\sqrt{n})}$

$T(n) = \sum_{i=0}^{\frac{n}{1+\sqrt{n}}} (2^i + \sqrt{n})$

$\leq \sum_{i=0}^{n/\sqrt{n}} (2^i + \sqrt{n}) = \sum_{i=0}^{\sqrt{n}} (2^i + \sqrt{n}) = 2^{\sqrt{n}+1} - 1 + \sqrt{n}(\sqrt{n}+1) \leq 4 \cdot 2^{\sqrt{n}}$

$T(n) = O(2^{\sqrt{n}})$

(c) | v a | v a a | v aaaaa | v aaaaaaaaa | v ...

1 $2^0$ | 1 $2^1$ | 1 $2^2$ | 1 $2^3$ | 1 $2^4$

Seja $x$ o número de grupinhos então $n \geq \sum_{i=0}^{x} (1+2^i)$

$n \geq \sum_{i=0}^{x} (2^i) = 2^{x+1} - 1$

$n \geq 2^{x+1} - 1 \Rightarrow n+1 \geq 2^{x+1} \Rightarrow x \leq \lg(n+1)$.

$T(n) = \sum_{i=0}^{\lg(n+1)-1} (2^i + 2^i)$

$\leq 2 \sum_{i=0}^{\lg(n+1)-1} 2^i = 2\left(2^{\lg(n+1)} - 1\right) \leq 2 \cdot 2^{\lg(2n)} - 2 = 2\left(2^{\lg n + 1}\right) - 2$

$T(n) = O(2^{\lg n}) = O(n)$ [linear]

(Q6) MIN-GAP

- Idéia: manter uma variável "global" que registre o MIN-GAP, dado um sucessor e predecessor...

(Q7) CONVIDADOS NA FESTA DE ALICE.





(Q8) Seja $Y[1..n]$ a sub-seq contendo $n$ elementos, e seja $c[i,j]$ o maior comprimento de uma sub-seq palíndromo na seq $Y[i..j]$

base
$c[i,j] = 1$ , se $i = j$ ▸ Uma seq de 1 caracter é palíndromo
$c[i,j] = 0$ , se $i > j$ ▸ Não existe subseq contida na seq $Y[i..j]$ com $i > j$

$c[i,j] = 2 + c[i+1, j-1]$ , se $Y[i] = Y[j]$

$c[i,j] = \max\{c[i+1,j], c[i,j-1]\}$ , se $Y[i] \neq Y[j]$

▸ Algoritmo que calcula o maior comprimento de uma subseq palíndromo $Y$, com $Y[1..n]$

```
PALINDROMO-MAX (Y, n)
1. para i ← 1 até n faça
2.     para j ← 1 até n faça
3.         c[i,j] ← -∞        ▸ Inicializa matriz com elementos sentinela
4. devolva LOOKUP-PM(Y, 1, n)
```

▸ Algoritmo que calcula o comprimento maior de uma subseq palindromo de $Y[i..j]$ com $i$ e $j$ inteiros, $i, j = \{1, 2, \ldots, n\}$

```
LOOKUP-PM(Y, i, j)
1- se i = j
2-     então c[i,j] ← 1
3- se i > j
4-     então c[i,j] ← 0
5- se c[i,j] > -∞
6-     então devolva c[i,j]   ▸ Valor já calculado
7-     senão se Y[i] = Y[j]
8-         então c[i,j] ← 2 + LOOKUP-PM(Y, i+1, j-1)
9-         senão q1 ← LOOKUP-PM(Y, i, j-1)
10-            q2 ← LOOKUP-PM(Y, i+1, j)
11-            se q1 < q2
12-                então c[i,j] ← q2
13-                senão c[i,j] ← q1
14- devolva c[i,j]
```

2007/2 (8.25)

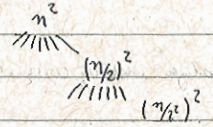
(Q1) Recorrência: Qual o maior inteiro possível para 'a' de tal forma que o ALG-B seja assintoticamente mais eficiente que o ALG-A.

(A) Resolução da recorrência do ALG-A:

$T(1) = 1$

$T(n) = 7\,T(n/2) + n^2$ , para n multiplos de 2

• árvore de recorrência



$n^2 \longrightarrow 7^0 (n/1^0)^2 =$

$(n/2)^2 \longrightarrow 7^1 \left(\frac{n}{2^1}\right)^2 = \left(\frac{7}{4}\right)^1 n^2$

$(n/4)^2 \longrightarrow 7^2 \left(\frac{n}{2^2}\right)^2 = \left(\frac{7}{4}\right)^2 n^2$

até $n = 2^i$

$\lg n = i$

$7^i \left(\frac{n}{2^i}\right)^2 = \left(\frac{7}{4}\right)^i n^2$

$$T(n) = \sum_{i=0}^{\lg n - 1} \left(\frac{7}{4}\right)^i n^2 + 7^{\lg n}$$

$$= n^2 \sum_{i=0}^{\lg n - 1} \left(\frac{7}{4}\right)^i + n^{\lg 7} = n^2 \left[\frac{\left(\frac{7}{4}\right)^{\lg n} - 1}{\frac{7}{4} - 1}\right] + n^{\lg 7} = n^2 \left(\frac{4}{3}\right)\left[\frac{n^{\lg 7} - 1}{n^2}\right] + \ldots$$

$$= \frac{4}{3}\left(n^{\lg 7} - n^2\right) + n^{\lg 7} = \frac{7}{3} n^{\lg 7} - \frac{4}{3} n^2$$

$T(n) = O(n^{\lg 7})$

• Prova por indução: Será provado que $T(n) = \frac{7}{3} n^{\lg 7} - \frac{4}{3} n^2$ por indução em k, com $n = 2^k$, para $k = 0, 1, 2, \ldots$



→ Base: Para $k = 0$ então $T(1) = \frac{7}{3} - \frac{4}{3} = 1$ (verdade)



→ Passo: Para $k \geq 1$ temos

$$T(n) = 7\,T(n/2) + n^2$$

$$= 7\left[\frac{7}{3}\left(\frac{n}{2}\right)^{\lg 7} - \frac{4}{3}\left(\frac{n}{2}\right)^2\right] + n^2 = 7\left[\frac{7}{3}\,\frac{n^{\lg 7}}{7} - \frac{4}{3}\,\frac{n^2}{4}\right] + n^2$$

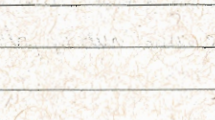
$$= \frac{7}{3} n^{\lg 7} - \frac{7}{3} n^2 + n^2 = \frac{7}{3} n^{\lg 7} - \frac{4}{3} n^2 \quad \text{c.q.d.}$$

(B) Resolução da recorrência do ALG-B

$T'(1) = 1$

$T'(n) = a\,T'(n/4) + n^2$

• árvore de recorrência:



$n^2 \longrightarrow a^0 \left(\frac{n}{4^0}\right)^2 = \left(\frac{a}{16}\right)^0 n^2$

$(n/4)^2 \longrightarrow a^1 \left(\frac{n}{4^1}\right)^2 = \left(\frac{a}{16}\right)^1 n^2$

$(n/4^2)^2 \longrightarrow a^2 \left(\frac{n}{4^2}\right)^2 = \left(\frac{a}{16}\right)^2 n^2$

até $n = 4^i$

$i = \log_4(n)$

$$T'(n) = \sum_{i=0}^{\log_4(n) - 1} \left(\frac{a}{16}\right)^i n^2 + a^{\log_4(n)} = n^2 \sum_{i=0}^{\log_4(n) - 1} \left(\frac{a}{16}\right)^i + n^{\log_4(a)}$$

*Se $a = 16$ então:

$T'(n) = n^2 \log_4(n) + n^{\log_4(a)}$

* Se $a \neq 16$ então

$$T'(n) = n^2 \left[ \frac{(a/16)^{\log_4(n)} - 1}{(a/16) - 1} \right] + n^{\log_4(a)} = n^2 \left[ \frac{\frac{n^{\log_4 a}}{n^2} - 1}{\frac{(a-16)}{16}} \right] + n^{\log_4(a)}$$

$$= \frac{16}{(a-16)} \left[ \frac{n^{\log_4(a)} - n^2}{n^2} \right] + n^{\log_4(a)}$$

$$T'(n) = \frac{a}{(a-16)} n^{\log_4(a)} - \frac{16}{(a-16)} n^2 = O(n^{\log_4(a)})$$

Vamos investigar para valores de $a > 16$

Sabemos que assintoticamente o ALG-A é $T(n) = O(n^{\lg 7})$ e para valores de $a \neq 16$ sabemos que ALG-B é $T'(n) = O(n^{\log_4(a)})$

Assim o maior inteiro de a deverá cumprir $T(n) \geq T'(n)$

$$O(n^{\lg 7}) > O(n^{\log_4(a)}) \quad \text{ou}$$

$$n^{\lg 7} > n^{\log_4(a)}$$

$$\lg 7 > \log_4(a) = \frac{\lg a}{\lg 4} = \frac{\lg a}{2}$$

$$\lg 7^2 > \lg a \qquad \text{Assim} \quad a < 7^2$$

$$a < 49$$

Portanto o maior valor inteiro de a é 48 para que o ALG-B seja assintoticamente mais eficiente que o ALG-A.





(Q2) CONTA INVERSÕES : Usando uma variação do ALGORITMO MERGE-SORT

▶ Algoritmo que recebe um vetor A[1..n] de inteiros todos distintos, e devolve o número de inversões, ou seja, o número de pares (i,j) do vetor A tal que $i < j$ e $A[i] > A[j]$ para $0 \leq i < j \leq n$. Alg. também ordena.

```
CONTA-INVERSOES (A, n)
1.  devolve CONTA-INV-MSORT ( A, 1, n)

CONTA-INV-MSORT ( A, p, r)
1.  se p < r
2.      então q ← ⌊(p+r)/2⌋
3.          i1 ← CONTA-INV-MSORT ( A, p, q)        ▷ O(n/2)
4.          i2 ← CONTA-INV-MSORT ( A, q+1, r)      ▷ O(n/2)
5.          i3 ← CONTA-INV-MERGE ( A, p, q, r)     ▷ Θ(n)
6.          devolva i1+i2+i3
7.      senão devolva 0

CONTA-INV-MERGE ( A, p, q, r)
1.  n1 ← q-p+1
2.  n2 ← r-q
3.  para i ← 1 até n1 faça
4.      L[i] ← A[i+p-1]
5.  L[n1+1] ← ∞
6.  para j ← 1 até n2 faça
7.      R[j] ← A[j+q]
8.  R[n2+1] ← ∞
9.  i ← 1
10. j ← 1
11. inv ← 0
12. para k ← 1 até r-p+1 faça
13.     se L[i] > R[j]
14.     então A[k] ← R[j]
15.         j ← j+1
16.         inv ← inv+1+(n1-i)
17.     senão A[k] ← L[i]
18.         i ← i+1
19. devolva inv
```



Complexidade $T(n) = 2T(n/2) + n \rightarrow T(n) = O(n \lg n)$

• Recorrência $T(1) = 1$

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$

ou grosseiramente $T(1) = 1$

$T(n) = 2T(n/2) + n$

Vou provar por indução em $n$ que $T(n) = n \lg n + n$ para $n = 1, 2, 4, 8, \ldots$

• Base: $n = 1$

$T(1) = 1 \lg(1) + 1 = 1$ (verdadeiro)

• Passo: $n > 1$

$T(n) = 2T(n/2) + n$

$= 2\left[\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) + \frac{n}{2}\right] + n = n \lg n - n + n + n$

$T(n) = n \lg n + n$

Assim o alg CONTA-INVERSÕES terá consumo $O(n \lg n)$

• correção:

A contagem de inversões é realizada principalmente no algoritmo CONTA-INV-INTERCE e assumindo termos e vetores (L e R) ordenados em forma crescente, podemos notar que se, numa dada iteração, $L[i] > R[j]$, então o número de inversões a considerar será $1 + n_1 - i$ dado que todos os elementos posteriores a $L[i]$ também são maiores que $R[j]$ (linha 16). Assim todas as inversões serão consideradas na contagem.





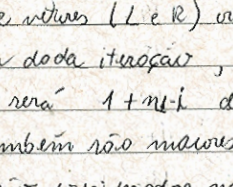
(Q3) Algoritmo que recebe um vetor $A[p..r]$ de números inteiros e um (PSA) inteiro $i$, tais que $1 \leq i \leq r-p+1$ e devolve o $i$-ésimo menor elemento de $A[p..r]$

```
SELECT ( A , p, r, i )
1.  se p < r
2.     então q ← MEDIANA (A,p,r)
3.           se q = i
4.              então devolva A[q]
5.              senão se i < q
6.                       então devolva SELECT ( A , p , q-1, i )
7.                       senão devolva SELECT ( A, q+1 , r, i-q )
8.     senão devolva A[p]
```

• complexidade: Seja $T(n)$ o consumo de tempo do algoritmo, com $n = r-p+1$

Recorrência grosseira $\begin{cases} T(1) = 1 \\ T(n) = T(n/2) + n \end{cases}$

- Árvore de recorrência:




$T(n) = \sum_{i=0}^{\lg n - 1} n\left(\frac{1}{2}\right)^i + 1$

$= n\left[\frac{\left(\frac{1}{2}\right)^{\lg n} - 1}{\frac{1}{2} - 1}\right] + 1 = n\left[\frac{\frac{1}{n} - 1}{\left(-\frac{1}{2}\right)}\right] + 1$

$= (-2)\left(\frac{1}{n} - 1\right) + 1 = 2n - 1 = O(n)$

Prova por indução em $n$, para $n = 1, 2, 4, 8, \ldots$

• Base: $n = 1$ então $T(1) = 1$ (verdade)

• Passo: $n > 1$ então $T(n) = T(n/2) + n = 2\left(\frac{n}{2}\right) - 1 + n = 2n - 1$ c.q.d.

Logo $T(n) = O(n)$.

(Q4) BIT-ALEATÓRIO.





(Q5) TORRES-GÊMEAS

- Observe-se que o problema é similar ao problema de prog. dinâmica para achar uma SSCO-MÁXIMA dados 2 vetores.
- Seja $c[i,j]$ o número máx de lajotas comuns entre os torres $A[1..i]$ e $B[1..j]$

- Recorrência:

$c[i,0] = 0$ para $1 \le i \le n$ } como base quando somente uma torre
$c[0,j] = 0$ para $1 \le j \le m$ } tem lajotas, assim a altura é zero.

$$c[i,j] = c[i-1,j-1] + 1, \text{ se } A[i] = B[j]$$
$$c[i,j] = \max\{c[i,j-1], c[i-1,j]\}, \text{ se } A[i] \neq B[j]$$

- Algoritmo:

```
TORRES-GÊMEAS (A, n, B, m)
1.  para i ← 1 até n faça
2.      c[i,0] ← 0
3.  para j ← 1 até m faça
4.      c[0,j] ← 0
5.  para i ← 1 até n faça
6.      para j ← 1 até m faça
7.          se A[i] = B[j]
8.              então c[i,j] ← 1 + c[i-1,j-1]
9.              senão se c[i,j-1] > c[i-1,j]
10.                 então c[i,j] ← c[i,j-1]
11.                 senão c[i,j] ← c[i-1,j]
12. devolva c[n,m]
```

• consumo de tempo: Seja $T(n,m)$ o consumo de tempo do algoritmo TORRES-GÊMEAS, então $T(n,m) = \Theta(nm)$

• corretude:

→ Se a lajota $i$-ésima da torre A, tem o mesmo raio da $j$-ésima lajota da torre B, então podemos reduzir o problema calculando a altura para os $(i-1)$-ésimos lajotas de A com os $(j-1)$-ésimos lajotas de B.

Se os lajotas comparados tem raio diferentes, então a altura maior corresponderá à comparação dos $(i-1)$ lajotas de A com os $j$-ésimos lajotas de B, ou a comparação dos $i$-ésimo lajotas de A com os primeiros $(j-1)$ lajotas de B.

(Q6) TEM-CICLO

(a) Invariante? Dados P1 e P2, em cada iteração mantem-se que na lista ligada do nó (elemento) P1 até P2 não existe ciclo.

O algoritmo terminará se P2 chegou até o final da lista ligada, ou se P2 conseguiu alcançar P1 num ciclo (i.e., P2=P1)

(b) A linha 2 é executada, no pior caso, $n-1$ vezes, que é quando P2 atinge a posição de P1 num ciclo.

Segundo o algoritmo P1 é incrementado numa posição, entretanto P2 é incrementado em 2 posições para cada incremento de P1; Assim, para que P2 = P1, a linha 2 deve ser executada no máximo $n-1$ vezes





(Q7) FAZ-ALGO

• A linha 4 é executada $n$ vezes dado que $j$ é incrementado em uma unidade em cada iteração

• Entretanto a linha 5 é executada no máximo 1 vez (ou melhor, é executada ou 0 ou 1 vezes); Assim, o algoritmo FAZ-ALGO tem o consumo de tempo $T(n) = O(n)$.

(Q8) Algoritmo TEM-PARTIÇÃO.

• OBS: As linhas 5 e 10 do algoritmo estão errados

L.5. para $j \leftarrow 1$ até 5 faça $T(0,j) \leftarrow$ FALSO

L.10. se $A[i] < j$ e $T[i-1, j-A[i]]$ = VERDADEIRO

(a) Seja $T(n,A)$ o consumo de tempo do algoritmo TEM-PARTIÇÃO para determinar se o vetor $A[1..n]$ tem uma partição tal que

$$\sum_{i \in I} A[i] = \sum_{i \notin I} A[i]$$

em que $I$ é um subconjunto de $\{1..n\}$

$$T(n,A) = O\left(n \sum_{i=1}^{n} A[i]\right) = O(nS)$$ o tempo não é polinomial no tamanho da entrada.

(b) A solução mostrada não é um algoritmo que decide em tempo polinomial no tamanho da entrada se o vetor tem ou não uma partição

O alg. é dependente não somente do tamanho de elementos de A, e se também dos valores de todos os elementos de A.

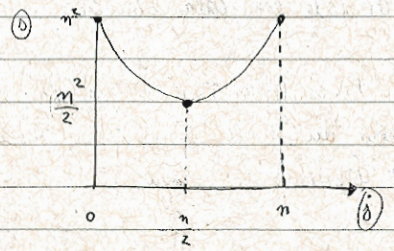
2003/2

~~1998/1~~

(Q1) $(n-j)^2 + j^2$ $\quad 0 \le j \le n$

Seja $S = (n-j)^2 + j^2$

$S = n^2 - 2nj + 2j^2$




- Os valores inteiros que maximizam $S$ são 1 e $(n-1)$.
- O valor inteiro que minimiza $S$ é $n/2$

Primeira derivada: $\frac{dS}{dj} = -2n + 4j = 0$

$j = n/2$

(Q2) Prove que $\sum_{i=1}^{n} \frac{i}{2^i} \le 2$

$S = \sum_{i=1}^{n} \frac{i}{2^i} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \dots$

$\frac{1}{2}$

$\frac{1}{2^2}$ $\frac{1}{2^2}$

$\frac{1}{2^3}$ $\frac{1}{2^3}$ $\frac{1}{2^3}$

$\frac{1}{2^4}$ $\frac{1}{2^4}$ $\frac{1}{2^4}$ $\frac{1}{2^4}$

$$S(n) = \sum_{i=1}^{n} \left( \sum_{j=i}^{n} \left(\frac{1}{2}\right)^j \right)$$

$$= \sum_{i=1}^{n} \left[ \sum_{j=0}^{n} \left(\frac{1}{2}\right)^j - \sum_{j=0}^{i-1} \left(\frac{1}{2}\right)^j \right]$$

$$= \sum_{i=1}^{n} \left[ \frac{\left(\frac{1}{2}\right)^{n+1} - 1}{\left(\frac{1}{2} - 1\right)} - \frac{\left(\frac{1}{2}\right)^{i} - 1}{\left(\frac{1}{2}\right) - 1} \right]$$





$$S(n) = \sum_{i=1}^{n} \left[ (-2) \left( \left(\frac{1}{2}\right)^{n+1} - 1 + 1 - \left(\frac{1}{2}\right)^{i} \right) \right]$$

$$= \sum_{i=1}^{n} \left[ (2) \left( \left(\frac{1}{2}\right)^{i} - \left(\frac{1}{2}\right)^{n+1} \right) \right] = 2 \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{i} - 2 \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{n+1}$$

$$= 2 \left[ \frac{\left(\frac{1}{2}\right)^{n+1} - 1}{\left(-\frac{1}{2}\right)} - 1 \right] - 2n \left(\frac{1}{2}\right)^{n+1}$$

$$= 2 \left[ -2 \left(\frac{1}{2}\right)^{n+1} + 2 - 1 \right] - 2n \left(\frac{1}{2}\right)^{n+1} = 2 \left[ 1 - 2 \left(\frac{1}{2}\right)^{n+1} \right] - 2n \left(\frac{1}{2}\right)^{n+1}$$

$$S(n) = 2 - 4 \left(\frac{1}{2}\right)^{n+1} - 2n \left(\frac{1}{2}\right)^{n+1} = 2 - \left(\frac{1}{2}\right)^{n+1} (4 + 2n) \le 2 \quad \text{para } n \ge 1$$

- Vou provar por indução que $S(n) = 2 - \left(\frac{1}{2}\right)^{n+1} (4+2n)$ para $n = 1, 2, 3, \dots$

- Base $n = 1$: $S(1) = 2 - 4\left(\frac{1}{2}\right)^2 - 2\left(\frac{1}{2}\right)^2 = 2 - 6\left(\frac{1}{4}\right) = 2 - \frac{3}{2} = \frac{1}{2}$ (verdade).

- Passo $n > 1$: suponha que seja válido para $n-1$, então

$$S(n) = S(n-1) + \frac{n}{2^n}$$

$$= 2 - \left(\frac{1}{2}\right)^{n} \left(4 + 2(n-1)\right) + \frac{n}{2^n} = 2 - \left(\frac{1}{2}\right)^{n} \left(2 + 2n\right) + n \left(\frac{1}{2}\right)^{n}$$

$$= 2 - \left(\frac{1}{2}\right)^{n} \left[ \left(2 + 2n\right) - n \right] = 2 - \left(\frac{1}{2}\right)^{n} \left[ 2 + n \right]$$

$$= 2 - \left(\frac{1}{2}\right)^{n} \left[ \left(\frac{1}{2}\right) (4 + 2n) \right]$$

$$= 2 - \left(\frac{1}{2}\right)^{n+1} (4 + 2n)$$

c.q.d.

(Q3) Problema de decisão: Existem $(i, j)$ tal que $V[i]+V[j]=x$ ?

- Seja $V[1..n]$ um vetor ordenado crescentemente.
- 
- ALGORITMO $(V, n, x)$

```
1. i ← 1
2. j ← n
3. enquanto i < j faça
4.     se V[i] + V[j] = x
5.         então devolve "SIM"
6.         senão se V[i]+V[j] > x
7.             então j ← j-1
8.             senão i ← i+1
9. devolva "NÃO"
```

- Complexidade: $O(n)$
- Correção: Dado que o vetor está ordenado a pesquisa começa somando o menor e o maior valor, se for igual ao valor procurado, então devolve SIM, se for maior que $x$ então usa-se o segundo maior, com o primeiro e o segundo menor, e así sucessivamente.

(Q4) Número de operações na somatória

- Sabemos que o número de dígitos para representar na base 2 um número $n$ é $\lfloor \lg(n) \rfloor + 1 \quad n \geq 1$





- Limitante inferior: Assumindo somatória de 0+1, $n$ vezes:

$$\text{soma} = \sum_{i=1}^{n} (1+1) \geq 2n = \Omega(n)$$

- Limitante superior: Assumindo somatória de $n$ mas a somatória dos $(n-1)$ primeiros inteiros

$$\text{soma} = \sum_{i=1}^{n} \left[ \lg\left(\frac{(n-1)(n)}{2}\right) + 1 + \lg(n) + 1 \right]$$

$$= \sum_{i=1}^{n} \left[ \lg(n^2 - n) - \cancel{1} + \cancel{1} + \lg n + 1 \right] \leq \sum_{i=1}^{n} \left[ \lg n^2 + \lg n + 1 \right]$$

$$\leq \sum_{i=1}^{n} \left[ 3 \lg n + 1 \right] \leq 3n \lg n + n$$

$$\text{soma} = O(n \lg n)$$

(Q5) Contador binário

| | |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| 10 | 1 0 1 0 |
| 11 | 1 0 1 1 |
| ⋮ | |
| | k bits |

- Método agregado: Para $n$ operações incrementa começando em zero

$$= n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + \frac{n}{2^{k-1}}$$

$$= n \sum_{k=1}^{k} \left(\frac{1}{2}\right)^{i-1} \leq n \cdot \sum_{i=0}^{n} \left(\frac{1}{2}\right)^{i} = 2n$$

$$\leq 2n$$

Assim, o custo de $n$ operações incrementa é $O(n)$, sendo o custo amortizado $O(1)$ para cada operação.

(Q6) Complexidade

de deriçaõ

(1) Problema NP-completo é uma classe de problemas que mantem os seguintes propriedades:

a) O problema está em NP, que quer dizer que são verificáveis em tempo polinomial, dado um certificado (uma solução)

b) Todo problema em NP pode ser polinomialmente reduzível a ele.

Do ponto de vista prático, se um problema é determinado como NP-completo significa que atualmente não existe algum algoritmo que consiga resolver ou decidir em tempo polinomial o problema.

(2) 1 - verdadeiro se $P \neq NP$

2 - falso, atualmente não se conhece nenhum algoritmo para o prob.

3 - verdadeiro, pois $P \subseteq NP$

4 - verdadeiro

5. falso, somente poderia se concluir que A é NP.

(Q6) Mediana: Solução na página 72, Questão 7 da prova 1998/2

(Q8) Programação de tarefas. (abordagem gulosa)

Seja P[1..n] o vetor de prazos associados as tarefas, tal que P[i] corresponde ao prazo Pi da tarefa i, para i = 1, 2, .., n.

Seja M[1..n] o vetor de multas, tal que M[j] corresponde a multa mj a pagar se a tarefa j não foi executada no prazo P[j]

SCANPIX



* Exemplo

| tarefa: | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| P: | 3 | 2 | 5 | 4 | 1 | 3 |
| M: | 20 | 80 | 100 | 10 | 90 | 20 |

• ordene vetores em forma decrescente de multas

| tarefa | 3 | 5 | 2 | 1 | 6 | 4 |
|---|---|---|---|---|---|---|
| P: | 5 | 1 | 2 | 3 | 3 | 4 |
| M: | 100 | 90 | 80 | 20 | 20 | 10 |

• Abordagem gulosa: coloque a tarefa de maior multa no último dia de prazo

Programação:

| Dia: | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| tarefa: | 5 | 2 | 1 | 4 | 3 | **6** |
| P: | 1 | 2 | 3 | 4 | 5 | 3 |
| M: | 90 | 80 | 20 | 10 | 100 | **20** |

Multa = 20

Se o dia já estiver ocupado por uma tarefa, atribua a tarefa no último dia disponível (pois de qualquer jeito será paga a multa)

* ALGORITMO:

▸ Algoritmo que recebe o número inteiro n de tarefas, e dois vetores: P[1..n] M[1..n], correspondentes aos prazos e multas, respectivamente.

PROGRAMAR-TAREFA (P, M, n)

1. ordene os vetores P e M em forma decrescente de multa } O(n lg n)
2. Para cada tarefa i do vetor ordenado de multas faça
3. atribua a tarefa i no dia P[i], tomando o cuidado: Se o dia já estiver ocupado com outra tarefa, atribua no último dia disponível } O(n)
4. devolva a atribuição de tarefas } O(1)

SCANPIX

* complexidade: O(n lg n) ordem dada pela ordenação

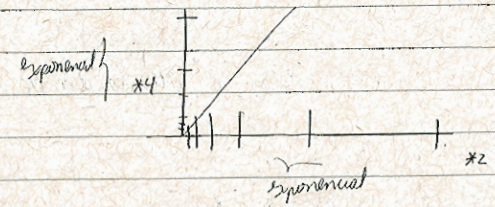
* correção ?

2003/1

(Q1) Ordem de crescimento: (absurda!)

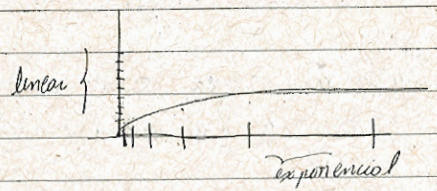
- Deve tomar cuidado com a discretização / amortização

- Algoritmo A: $t_A(n) = n^2$ ou é $\Theta(n^2)$

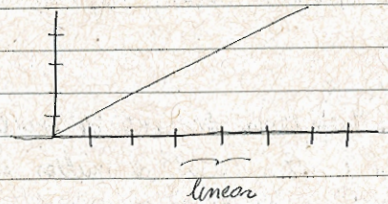



em $4n^2$ ✓
$4n$ $16n^2$ ✓
$8n$ $32n^2$ ✓

- Algoritmo B: $t_B(n) = \lg n$ ou é $\Theta(\lg n)$ ?




- Algoritmo C: $t_C(n) = \frac{1}{2}n$ ou é $\Theta(n)$ ✓








(Q2) Imprimir os $k$ menores elementos

▶ Algoritmo que recebe um vetor $A[1..n]$ de $n$ elementos e um número natural $k$ e imprime os $k$ menores elementos de $A$

```
IMPRIME-MENORES (A, n, k)
1.  Q ← BUILD-MIN-HEAP(A, n)    ▶ constroi um HEAP MÍNIMO Q
2.  para i ← 1 até k faça
3.      x ← EXTRACT-MIN(Q)
4.      imprime x
```

Consumo de tempo:

| Linha | Consumo |
|---|---|
| 1 | $O(n)$ |
| 2 | $\Theta(k)$ |
| 3 | $O(k \lg n)$ |
| 4 | $\Theta(k)$ |
| Total | $O(n + k \lg n)$ |

$\leq n$
$n + 2k + k \lg n$

(Q3) READDIR em linux: (dinossauro!)

(Q4) Grafo numerado e propriedade JÓIA.

- JÓIA: Dado um grafo $G$, decidir se $G$ é jóia: Problema está em NP
- PJÓIA: " " " ", " " existe uma renumeração de $G$ que é jóia: está em NP

a) Se JÓIA está em P, PJÓIA está em P? NÃO pode se concluir se $P \neq NP$

b) Se JÓIA está em NP, PJÓIA está em NP? SIM, o verificador JÓIA também verifica PJÓIA.

(Q5) Recorrências & comparações:

- Seja A o algoritmo que resolve o problema em tempo $T_A(n) = \Theta(n^2)$

- Seja B o algoritmo descrito, cujo consumo de tempo $T_B(n)$ é

$$T_B(n) = 1 \quad , \text{se } n \leq 1$$
$$T_B(n) = \underbrace{n \lg n}_{\text{divisão}} + T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \underbrace{n \lg n}_{\text{combinação}}$$

Grosseiramente podemos representar a recorrência anterior como:

$$T_B(n) = 1 \quad , \text{se } n = 1$$
$$T_B(n) = 2T(n/2) + 2n \lg n \quad , \text{se } n > 1 \qquad \text{válido para potências de 2}$$

- Resolução da recorrência:

$2n \lg n \rightarrow 2^1 \left(\frac{n}{2^0}\right) \lg\left(\frac{n}{2^0}\right) \quad 2n(\lg n - 0)$

$2\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) \rightarrow 2^2 \left(\frac{n}{2^1}\right)\lg\left(\frac{n}{2^1}\right) = 2n(\lg n - 1)$

$2\left(\frac{n}{2^2}\right)\lg\left(\frac{n}{2^2}\right) \rightarrow 2^3 \left(\frac{n}{2^2}\right)\lg\left(\frac{n}{2^2}\right) = 2n(\lg n - 2)$

até $n = 2^i$, $\lg n = i$ $\qquad 2^{i+1}\left(\frac{n}{2^i}\right)\lg\left(\frac{n}{2^i}\right) = 2n(\lg n - i)$

$$= 2n \lg n - 2ni$$

$$T_B(n) = \sum_{i=0}^{\lg n - 1} \left(2n \lg n - 2ni\right) + 2^{\lg n}$$





$$T_B(n) = 2n \lg n \left(\lg n\right) - 2n \sum_{i=0}^{\lg n - 1} i + n$$

$$T_B(n) = 2n \lg^2 n - 2n\left[\frac{1}{2}(\lg n - 1)\lg n\right] + n = 2n \lg^2 n - n \lg^2 n + n \lg n + n$$
$$= n \lg^2 n + n \lg n + n$$

- Prova por indução em $n$, que $T_B(n) = n \lg^2 n + n \lg n + n$, para $n = 1, 2, 3, \ldots$

- Base: Para $n = 1$ temos $T_B(1) = (1)(0) + (1)(0) + 1 = 1$ (verdadeiro)

- Passo: Para $n > 1$ temos

$$T_B(n) = 2\,T_B(n/2) + 2n \lg n$$

$$= 2\left[\left(\frac{n}{2}\right)\lg^2\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)\right] + 2n \lg n$$

$$= n(\lg n - 1)^2 + n(\lg n - 1) + n + 2n \lg n$$

$$= n(\lg^2 n - 2\lg n + 1) + n \lg n - n + n + 2n \lg n$$

$$= n \lg^2 n - 2n \lg n + n + n \lg n - n + n + 2n \lg n$$

$$T_B(n) = n \lg^2 n + n \lg n + n \qquad \text{c.q.d.}$$

$$T_B(n) = O(n \lg^2 n)$$

Resumindo: Alg A: $T_A(n) = \Theta(n^2)$

Alg B: $T_B(n) = O(n \lg^2 n)$

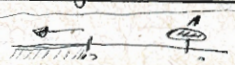
Portanto o algoritmo B é assintoticamente mais eficiente.

$$\frac{n \cdot n}{n \cdot \lg^2 n} = \frac{\lg n}{\lg(e \lg n)} = \frac{\lg n}{1 + \lg \lg n}$$

$$\lim_{n \to \infty} \left(\frac{\lg n}{1 + \lg \lg n}\right) = \infty$$

$\therefore n^2 \gg n \lg^2 n$

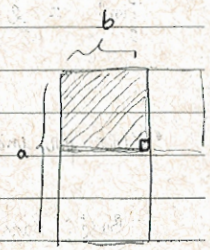




## (Q6) TRIANGULO DE PASCAL (enunciado muito vago)

Exemplo

| a \ b | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 1 | 3 | 6 | 10 | 15 | 21 |
| 3 | 1 | 4 | 10 | 20 | 35 | 56 |
| 4 | 1 | 5 | 15 | 35 | 70 | 126 |
| 5 | 1 | 6 | 21 | 56 | 126 | 252 |




(a) $T(a,b) = 1$ se $a=0$ ou $b=0$

$T(a,b) = T(a-1,b) + T(a,b-1) + 1$ c.c.

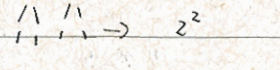
Se vale que $a=b$ então ⟶ ?

$T(a,a) = 1$ se $a=0$

$T(a,a) = T(a-1,a) + T(a,a-1) + 1$ se $a>0$

$T(a,a) = 2T(a-1,a) + 1$ dado que $T(a-1,a) = T(a,a-1)$

• árvore de recorrência




logo: $T(a) = \sum_{i=0}^{a-1} 2^i = 2^a - 1$

$T(a) = \Omega(2^a)$

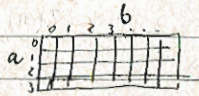
ou melhor $T(a,b) = \Omega\left(2^{\min\{a,b\}}\right)$





(b) Observe que Pascal $(a,b) \equiv$ Pascal $(b,a)$ ▶ valores são simétricos

▶ Algoritmo pascal que recebe 2 valores a e b, inteiros e devolve o valor correspondente ao triângulo de Pascal




```
PASCAL ( a, b)
1-  se a > b
2-      então troque a ⇔ b        ▶ assumimos que a ≤ b no alg.
3-  aloque vetor P[0..a]                } espaço Θ(min{a,b})
4-  para i ← 0 até a faça                } Θ(a)
5-          P[i] ← 1                     }
6-  para j ← 1 até b faça                }
7-      para i ← 1 até a faça            } Θ(a.b)
8-          P[i] ← P[i] + P[i-1]         }
9-  devolva P[a]                         } Θ(1)
```

complexidade: $\Theta(a.b)$

Consumo de espaço: claramente será $\Theta(\min\{a,b\})$

## (Q7) QUICKSORT HÍBRIDO COM INSERTION-SORT


| QS (A, p, r) ▶ Versão original | QS' (A, p, r, k) |
|---|---|
| 1- se p < r | 1- se r - p > k |
| 2- então q ← PARTICIONE (A, p, r) | 2- então q ← PARTICIONE (A, p, r) |
| 3- QS (A, p, q) | 3- QS' (A, p, q, k) |
| 4- QS (A, q+1, r) | 4- QS' (A, q+1, r, k) |

complexidade no pior caso $\Theta(n^2)$

Resposta em função de k

- Complexidade no pior caso: Corresponde quando o particione divide o vetor em forma desigual, escolhendo como pivô ou o maior elemento ou o menor em cada iteração. Assim, a recorrência grosseira será:

$$T(n) = 0 \quad , \text{se } n = k$$
$$T(n) = T(n-1) + n \quad , \text{se } n > k$$

- árvore de recorrência

$n-0$
$n-1$
$n-2$
⋮

até $n-i = k$
$i = (n-k)$

$$T(n) = \sum_{i=0}^{(n-k)-1} (n-i)$$

$$= \sum_{i=0}^{(n-k)-1} n \; - \sum_{i=0}^{(n-k)-1} i$$

$$= n(n-k) - \frac{1}{2}\left[(n-k-1)(n-k)\right]$$

$$= n^2 - nk - \frac{1}{2}\left[n^2 - nk - nk + k^2 - n + k\right]$$

$$= n^2 - nk + nk - \frac{1}{2}\left(n^2 + k^2 - n + k\right)$$

$$T(n) = \frac{1}{2}\left(n^2 - k^2 + n - k\right)$$

$$\leq n^2 - k^2 + n^2 = 2n^2 - k^2$$

$$T(n) = O(2n^2 - k^2)$$

- Complexidade no melhor caso: é obtido quando o PARTICIONE divide em duas partes iguais de elementos. Uma recorrência grosseira que representa esse é:

$$T(n) = 0 \quad , \text{se } n = k$$
$$T(n) = 2T(n/2) + n \quad , \text{se } n > k$$





- árvore de recorrência.

$n$
$n/2 \quad n/2$
$n/2^2 \quad \ldots$

$2^0(n/2^0) = n$
$2^1(n/2^1) = n$
$2^2(n/2^2) = n$

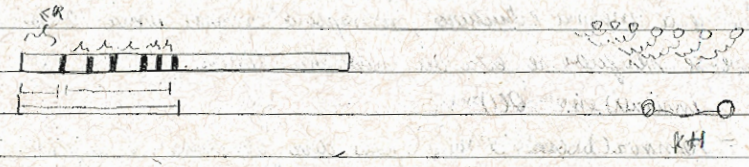
até $\frac{n}{2^i} = k$

$$\lg(n/k) = i$$

$$T(n) = \sum_{i=0}^{\lg(n/k)-1} n = n \lg(n/k)$$

$$T(n) = O(n \lg(n/k))$$

(b)




```
INSERTION-SORT (A, n)
1. para i ← 2 até n faça
2.     j ← i - 1
3.     enquanto A[j] > A[i] e j > 0 faça
4.         troca A[j] ⇔ A[i]
5.         j ← j - 1
```

$O(n^2)$

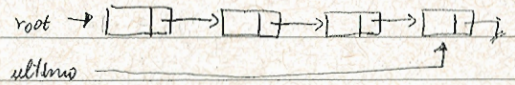
No INSERTION-SORT o número máximo de trocas que poderia existir, caso um vetor semi-ordenado, é de k elementos, portanto a complexidade é de $O(nk)$

(c) VERIFIQUE-SEMI-ORD (A, n) *

```
1. maior ← A[1]
2. para i ← k+2 até n faça
3.     se A[i] < maior
4.         dê exibição de uma falha
5.         se maior < A[i-k]
6.             então maior ← A[i-k]
```

(Q8) Estrutura de dados

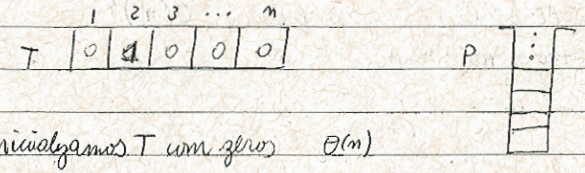
(a) Uso de uma lista ligada, contendo um apontador ao primeiro e ao último elemento (tarefa)




- suporta duplicações
- insira(x) em O(1)
- remova() em O(1)

(b) Use-se a mesma estrutura, mas para inserir uma tarefa previamente deve-se verificar se esta ou não na lista
- insira(x) em O(1)
- remova() em O(n), pois deve verificar na lista toda

(c) como n é pequeno o suficiente, podemos usar (a) um vetor T[1..n] conjuntamente com (b) uma pilha P de no máximo n elementos




- inicializamos T com zeros Θ(n)

- insira(x): verifique se x está no vetor T, i.e. verifique se T[x] ≠ 1 se não está, então coloque x na pilha P . O(1)

- remova(): extraia um elemento de P (i.e. pop(P)), seja i a tarefa extraída. Atribua T[i] = 0 . O(1)

Uso de espaço O(2n) = O(n)





1999/1 (2a)

(Q1) $2^{n+1} = O(2^n)$ ?

• SIM. $2^{n+1} = 2.2^n \leq 4.2^n = O(2^n)$

tome $c=4$ e $n \geq 1$ então $2^{n+1} = O(2^n)$

$2^{2n} = O(2^n)$

• NÃO. Demonstração pelo absurdo. Suponha que seja válido que $2^{2n} = O(2^n)$ então existem constantes $c$ e $n_0$ tais que

$2^{2n} \leq c\,2^n$ para $n \geq n_0$

$c \geq 2^n$ o qual é absurdo

(Q2)

1. Se $f(n) = \Omega_1(g(n))$ então $f(n) > Ag(n)$ para um $n > n_0$ (segundo a definição)
Logo é facil ver que

$f(n) > Ag(n)$ porque existem infinitos $n \geq n_0$

2. CONTRA-EXEMPLO

se $f(n) = \Omega_2(g(n))$ então $f(n) > Ag(n)$ para infinitos valores de $n$

Seja $f(n) = \text{sen}(x)$ então $f(n) = \Omega_2(g(n))$ porque existe $A > 0$ tal que $f(n) > Ag(n)$ para infinitos valores de $n$

mas não existe um $n_0$ tal que $f(n) > Ag(n)$ $\forall\, n > n_0$

m
n [ ]

(Q3) Seja $T(n,m)$ o número de operações necessários para calcular o elemento maior da matriz.

(1) $T(n,m) = n(m-1) + n-1 = nm - \cancel{n} + \cancel{n} - 1 = nm - 1$

(2) $T(n,m) = m(n-1) + m-1 = nm - \cancel{m} + \cancel{m} - 1 = nm - 1$

Independente dos valores de $n$ e $m$, ambas abordagens consomem $nm-1$ operações para calcular o maior elemento na matriz $A_{n \times m}$.

(Q4) Ordem em tempo $O(n)$

- Assume-se $n$ número $\in \{1, 2, \ldots, n^3\}$

- ALGORITMO

1º Mude à base $n$, os $n$ números. Teremos $n$ números de 3 dígitos (no máximo)
2. Para cada dígito (do menor ao maior dígito significativo) aplique o algoritmo de ordenação por contagem - estável (counting-sort)
3. Mude à base 10 os $n$ números

| Linha | Consumo |
|---|---|
| 1 | $\Theta(n)$ |
| 2 | $O(3n)$ |
| 3 | $\Theta(n)$ |
| Total | $O(n)$ |





(Q5) RAIO-CÓSMICO

- Solução baseada em comparações não ochoda dado que não é possível acessar a posições específicos das strings...

(Q6) Caminho mínimo entre 2 vértices

(1) Complexidade do algoritmo:

| Linha | Consumo de tempo |
|---|---|
| 1 | $\Theta(n^2)$ |
| 2-8 | $\Theta(n^3)$ |
| 9 | $\Theta(1)$ |
| Total | $\Theta(n^3)$ |

caminho: $\Pi$ indica de onde eles vêm para chegar a $j$
se $i = j$ ▶ não existe caminho de $i$ a $j$
então $\Pi_{ij} = NIL$
se $w_{ij} = \infty$ ▶ não existe aresta
então $\Pi_{ij} = NIL$

(2) Custo e caminho mínimo de 2 vértices:

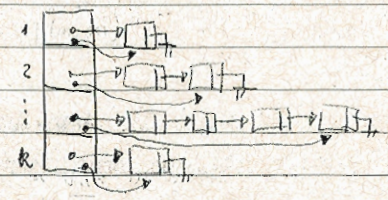
1. $D \leftarrow W$ } $\Theta(n^2)$
2- para $i \leftarrow 1$ até $n$ faça
3- para $j \leftarrow 1$ até $n$ faça
4- se $w_{ij} = \infty$ ou $i = j$
5- então $\Pi_{ij} \leftarrow NIL$
6- senão $\Pi_{ij} \leftarrow i$ } $\Theta(n^2)$
7- para $k \leftarrow 1$ até $n$ faça
8- para $i \leftarrow 1$ até $n$ faça
9- para $j \leftarrow 1$ até $n$ faça
10- $d_{ij} \leftarrow \min\{d_{ij}, d_{ik} + d_{kj}\}$
11- se $d_{ij} > d_{ik} + d_{kj}$ ▶ passou por $k$
12- então $\Pi_{ij} \leftarrow \Pi_{kj}$ } $\Theta(n^3)$
13- devolva $(D, \Pi)$

Para imprimir faça-se um backTracking:

1. $p \leftarrow \Pi[i,j]$
2. enquanto $p \neq NIL$ ou $p \neq i$ faça
3. Imprime $p$
4. $p \leftarrow \Pi[i,p]$

(Q7) FILA DE PRIORIDADES

(i) Use-se uma tabela HASH no lugar de um MIN-HEAP, ou um vetor de listas ligadas




a) inicialização $\Theta(k)$
b) inserção $\Theta(1)$
c) extração $\Theta(1)$
d) reclassificação $\Theta(2) = \Theta(1)$

(ii) O passo mais custoso do alg. de Kruskal é a ordenação ( $O(n \lg n)$ ), dos pesos dos arestas; assim, tendo somente pesos no intervalo $[0..k]$ a ordenação pode ser realizada em tempo linear usando COUNTING-SORT

ALGORITMO $(G(V,E), w)$ ◦ função de peso

1. ordena arestas não decrescente seja L ▶ $O(V)$, dada H1 e H2
2. Make-set (v) $\forall v \in V$
3. $A = \emptyset$ ▶ conj. vazio de arestas
4. para cada $(u,v) \in L$ faça
5. se FINDSET(u) $\neq$ FINDSET(v)
6. então UNION(u,v)
7. $A \leftarrow A \cup \{(u,v)\}$
8. devolva A ▶ conjunto de arestas que conformam a MST

◦ complexidade

| Linha | Consumo |
|---|---|
| 1 | $O(V)$ |
| 2 | $O(V)$ |
| 3 | $O(1)$ |
| 4 | $O(V)$ |
| 5-7 | $O(V \lg^* V)$ |
| 8 | $\Theta(1)$ |
| TOTAL | $\Omega(10V)$ |

SCANPIX



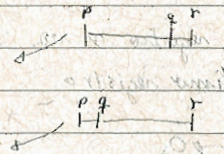
1999/2 (Agosto)

(Q1) (a) Algoritmo limitado superiormente por uma função do conjunto $\Theta(n^2 \lg n)$
(b) " " inferiormente " " " " " " $\Theta(n^2)$

(Q2) QUICKSORT (espaço da pilha log) {se resolve primeiro a metade mais pequena, então a pilha não crescerá muito comparado com a outra metade}

O espaço da pilha deve ser logarítmico no tamanho da entrada; a abordagem que pode ser usada é a seguinte:
- Para toda iteração use-se a menor metade para ordenar-la.

```
QS'(A, p, r)        ▶ QS. com recursão de calda
1. enquanto p < r faça
2.     q ← PARTICIONA(A,p,r)
3.     se q-r > r-q
4.     então QS'(A, q+1, r)
5.           r ← q-1
6.     senão QS'(A, p, q-1)
7.           p ← q+1
```




(Q3) (a) Uma estrutura de dados usada comumente em problemas de ordenação e filas de prioridade.

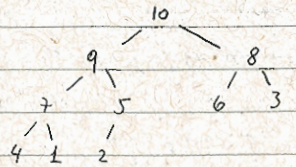
filho_esq[x] := 2x
filho_dir[x] := 2x+1

pai[x] := $\lfloor x/2 \rfloor$

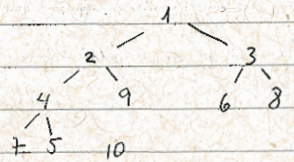
Então A será um MAX-HEAP, se pai[x] $\geq$ filho_esq[x]
pai[x] $\geq$ filho_dir[x] ∀ elemento x

SCANPIX

(6) MAX-HEAP




MIN-HEAP

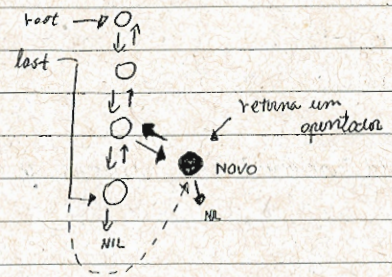



(Q4) ESTRUTURA DE DADOS

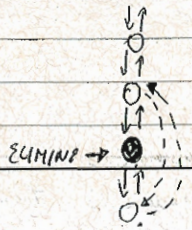
Usaremos uma lista duplamente encadeada

(a) Criação de uma coleção $O(1)$: pois somente é necessário ter um apontador para o primeiro e o último registro (i.e. ambos apontam a NIL.

(b) Incluir registro $x$ na coleção $O(1)$: Dado que tem-se um apontador ao último registro




(c) Exclusão dado um apontador em $O(1)$








(Q5) Ordenar usando $n-1$ comparações num vetor de $n$ elementos

- Idéia: crie-se 2 vetores cada um contendo somente 1 tipo de valor (ordenação não-estável)

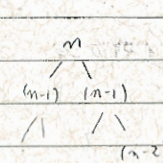
```
ORDENE (A, n)
1.  pivo ← A[1]
2.  n1 ← 0
3.  n2 ← 0
4.  para k ← 2 até n faça
5.      se pivo ≤ A[k]
6.          então n1 ← n1 + 1
7.                B[n1] ← A[k]
8.          senão n2 ← n2 + 1
9.                C[n2] ← A[k]
10. para i ← 1 até n1 faça
11.     A[i] ← B[i]
12. A[n1+1] ← pivo
13. para j ← 1 até n2 faça
14.     A[j+n1] ← C[j]
```

} necessárias e suficientes $n-1$ comparações


(Q6) Impressão de ASTERISCOS (recorrência)

$$T(0) = 0$$
$$T(n) = T(n-1) + n + T(n-1)$$
$$= 2T(n-1) + n$$

• árvore de recorrência




até $(n-i) = 0$

$i = n$

$$T(n) = \sum_{i=0}^{n-1} 2^i (n-i)$$

$$T(n) = n \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} i 2^i$$

$$T(n) = n\left(2^n - 1\right) - \sum_{i=1}^{n-1} \left( \sum_{j=i}^{n-1} 2^j \right)$$

$$\left( \sum_{j=0}^{n-1} 2^j - \sum_{j=0}^{i-1} 2^j \right)$$

$$2^n - 1 - 2^i + 1$$

$$- \sum_{i=1}^{n-1} 2^n + \sum_{i=1}^{n-1} 2^i$$

$$T(n) = n2^n - n - \left(2^n(n-1)\right) + \left(2^n - 1 - 1\right)$$

$$T(n) = n2^n - n - n2^n + 2^n + 2^n - 2$$

$$T(n) = 2 \cdot 2^n - n - 2 = 2^{n+1} - n - 2$$

• Prova por indução em $n$, para $n = 0, 1, 2, \ldots$

- Base: $n = 0$, então $T(1) = 2^1 - 0 - 2 = 0$ (verdade

- Passo: $n \geq 0$ então

$$T(n) = 2T(n-1) + n$$
$$= 2\left[2^n - (n-1) - 2\right] + n$$
$$= 2^{n+1} - 2n - 2 + n$$

$T(n) = 2^{n+1} - n - 2$ C.Q.D.

$T(n) = O(2^n)$





(Q7) MOCHILA FRACCIONÁRIA.

$v = v_1, v_2, \ldots, v_n$ (valor)
$p = p_1, p_2, \ldots, p_n$ (Peso)
$P$ = número natural

} números naturais

Encontrar números reais $x_1, x_2, \ldots, x_n$, tal que $0 \leq x_i \leq 1$ para todo $i$

$p_1 x_1 + p_2 x_2 + \ldots + p_n x_n \leq P$ e

$v_1 x_1 + v_2 x_2 + \ldots + v_n x_n$ seja máximo

(1) ALGORITMO: guloso

$\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \frac{v_3}{p_3} \geq \ldots$

MOCHILA-FRAC $(v, p, P, n)$

1. Ordene o vetor de valores $v$ e $p$ de forma não-decrescente tal que seja 'vo' os valores ordenados e 'po' seus respectivos pesos
2. para $i \leftarrow 1$ até $n$ faça
3. se $vo[i] \leq P$
4. então $xo[i] \leftarrow 1$
5. $P \leftarrow P - p[i]$
6. senão $xo[i] \leftarrow P / p[i]$
7. $P \leftarrow 0$
8. devolva $xo$

(3) O consumo de tempo maior é dado pela linha 1 (ordenação) e é $O(n \lg n)$. As linhas 2-7 é realizado em $O(n)$

Consumo total $O(n \lg n)$

e $v[m] \leq P$ $\forall$

(2) Escolha gulosa: se $v[m]/p[m] \geq v[i]/p[i]$ para todo $i$ então $m$ pertence a uma solução ótima (Prova)

Estrutura ótima: se $R$ é uma solução ótima para a instância $(v, p, P, n)$, então $R - \{m\}$

(Q8) COMPLEXIDADE : Emparelhamento e cobertura.





2008/1

(Q1) $f(n) = O(g(n))$

(1) Significa que dadas constantes $c$ e $n_0$ é verdade que

$$f(n) \leq c\, g(n) \quad \text{para todo } n \geq n_0$$

Uma definição "informal" seria que $g(n)$ é uma função limitante superior à função $f(n)$

(2) Não faz sentido pois a notação $O$, como definida no item (1) representa um limitante superior, e o fato de descrever o consumo de tempo do alg. A como "pelo menos" é uma mal uso da notação (contradição)

(Q2) Seja $S[1..n]$ um vetor de inteiros e $x$ um inteiro. O algoritmo de decisão deve determinar se existem $i$ e $j$ (índices) tais que $S[i] + S[j] = x$, para $i \neq j$ com $1 \leq i \leq n$, $1 \leq j \leq n$

```
BUSCA-SOMA-ELEMENTOS (S, n, x)
1   Ordene o vetor S em forma crescente (use o heapsort)
2   i ← 1
3   j ← n
4   enquanto i < j faça
5       se S[i] + S[j] = x
6           então devolva "SIM"
7           senão se S[i] + S[j] > x
8                   então j ← j - 1
9                   senão i ← i + 1
10  devolva "NÃO"
```

• Complexidade: O consumo de tempo, em notação assintótica será influenciado pela ordenação (linha 1). As linhas 2 até 10, claramente, tem um consumo de $O(n)$. Portanto o consumo total será de $O(n \lg n)$

Uma alternativa para o mesmo problema é o seguinte.

```
BUSCA-SOMA-ELEMENTO (A, n, x)
1.  ordene A em forma crescente                    O(n lg n)
2.  para i ← 1 até n-1 faça                        Θ(n)
3.      se BUSCA-BINARIA (A, i+1, n, x-A[i])       } O(n lg n)
4.          então devolva "SIM"
5.  devolva "NÃO"                                  Θ(1)
                                                   O(n lg n)

BUSCA-BINARIA (A, p, r, x)
1  se p < r
2      então q ← ⌊(p+r)/2⌋
3            se A[q] = x
4                então devolva "SIM"
5                senão se A[q] < x
6                        então devolva BUSCA-BINARIA (A, q+1, r, x)
7                        senão devolva BUSCA-BINARIA (A, p, q-1, x)
8  devolva "NÃO"
```

Complexidade: Seja $T(n)$ o consumo de tempo para uma instância de tamanho $n$ para o alg. BUSCA-SOMA-ELEMENTO; Então $T(n) = O(n \lg n)$





## (Q3) MAJORITÁRIO

Seja $A[1..n]$ um vetor de inteiros, não necessariamente com elementos ordenados. O alg projetado será dividido em 2 partes. A primeira corresponde à busca de um candidato majoritário, e a segunda correspondente à verificação do candidato no vetor A.

▶ Algoritmo que recebe um vetor $A[1..n]$ de inteiros e devolve o elemento majoritário (se existir), caso contrário devolve $\nexists$

```
MAJORITÁRIO (A, n)
1  x ← MAJORITARIO-CANDIDATO (A, n)
2  count ← 0
3  para i ← 1 até n faça
4      se x = A[i]
5          então count ← count + 1
6  se count ≥ n/2
7      então devolva x
8      senão devolva ∄
```

SOLUÇÃO ELEGANTE:

```
MAJORITARIO-CANDIDATO (A, n)
1.  m ← 0
2.  para i ← 1 até n faça
3.      se m = 0
4          então m ← 1
5              ele ← A[i].
6          senão se ele = A[i]
7              então m ← m+1
8              senão m ← m-1
9.  devolva ele
```

▶ Alg. que recebe um vetor $A[1..n]$ e procura um candidato elemento majoritário

```
MAJORITÁRIO-CANDIDATO (A, n)
1   j ← 0
2   para i ← 1 até n-1 com incremento de 2 faça
3       se A[i] = A[i+1]
4           então j ← j+1
5                 B[j] ← A[i]
6   se j = 1
7       então devolva B[j]
8   se j = 0
9       então devolva A[n]
10. se j > 1
11.     então se n mod 2 = 1 ▶ ímpar
12.              então j ← j+1
13.                   B[j] ← A[n]
14.     devolva MAJORITARIO-CANDIDATO (B,
```

• consumo de tempo dos algoritmos:

*Alg. MAJORITÁRIO-CANDIDATO: Observe-se que tamanho máximo de B, em cada chamada recursiva, dada um vetor A, de tamanho n é de n/2, dado que as linhas 2 a 5 consulta 2-a-2 elementos consecutivos em A. Assim, uma recorrência grosseira que representa a complexidade, T(n), no pior caso será:

$T(1) = 0$

$T(n) = T(n/2) + n$

Note que a comparação A[i] = A[i+1] na linha 3 será executada no máximo O(n)
Claramente $T(n) \leq 2n$

- Prova por indução em n; para n = 1, 2, 3, ...
  - Base: n=1 então $T(1) = 0 \leq 2$ (verdade)
  - Passo: n > 1 então $T(n) = T(n/2) + n$

$$\leq 2\left(\frac{n}{2}\right) + n = 2n$$

$T(n) \leq 2n$

Portanto $T(n) = O(n)$

*Alg. MAJORITÁRIO: Seja $T'(n)$ o consumo de tempo do algoritmo para uma instância de tamanho n

| Linha | Consumo |
|---|---|
| 1 | $O(n)$ |
| 2 | $\Theta(1)$ |
| 3-5 | $O(n)$ |
| 6-8 | $O(1)$ |

$T'(n) = O(n)$

Concluimos que o algoritmo MAJORITÁRIO terá um consumo linear.

$T'(n) = O(n)$



(Q4) Recorrências

(a) Algoritmo A1: Seja $T(n)$ o consumo de tempo do alg. para uma instância de tamanho n

$T(n) = 1$ , se $n = 1$

$T(n) = 8\,T(n/2) + \Theta\left((3n)^{\lg 7}\right)$ , para $n = 2^1, 2^2, 2^3, \ldots$

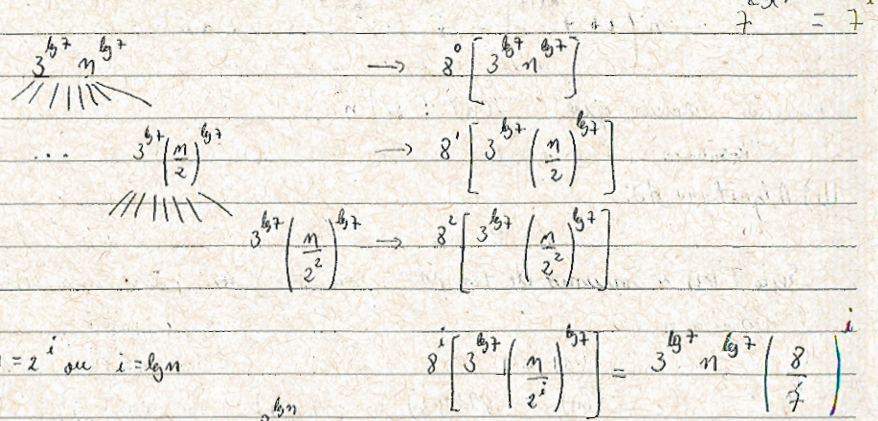
Grosseiramente podemos escrever a recorrência acima como:

$T(n) = 1$ , para $n = 2^0$

$T(n) = 8\,T(n/2) + (3n)^{\lg 7}$ , para $n = 2^1, 2^2, \ldots$

• Resolução da recorrência

(i) árvore de recorrência:




até $n = 2^i$ ou $i = \lg n$

$$8^i\left[3^{\lg 7}\left(\frac{n}{2^i}\right)^{\lg 7}\right] = 3^{\lg 7} n^{\lg 7}\left(\frac{8}{7}\right)^i$$

$$T(n) = \sum_{i=0}^{\lg n - 1} (3n)^{\lg 7}\left(\frac{8}{7}\right)^i = (3n)^{\lg 7}\left[\frac{\left(\frac{8}{7}\right)^{\lg n} - 1}{\frac{8}{7} - 1}\right] = 7(3n)^{\lg 7}\left[\frac{8^{\lg n}}{7^{\lg n}} - 1\right]$$

$$= 7(3n)^{\lg 7}\left[\frac{n^3}{n^{\lg 7}} - \frac{n^{\lg 7}}{n^{\lg 7}}\right] = 7\,3^{\lg 7} n^{\lg 7}\left[\frac{n^3 - n^{\lg 7}}{n^{\lg 7}}\right] = 7(3^{\lg 7})\left[n^3 - n^{\lg 7}\right]$$

$$= 7^{(\lg 3 + 1)}\left(n^3 - n^{\lg 7}\right) + n^3 = n^3\left[1 + 7^{(\lg 3 + 1)}\right] - 7^{(\lg 3 + 1)} n^{\lg 7}$$

(ii) vou provar por indução em $n$, que $T(n) = n^3\left[1 + 7^{(\lg 3+1)}\right] - 7^{(\lg 3+1)}\, n^{\lg 7}$, para $n = 2^0, 2^1, 2^2, \ldots$

- Base $n=1$, então $T(1) = 1$ (verdadeiro)
- Passo $n > 1$, então

$$T(n) = 8\,T(n/2) + (3n)^{\lg 7}$$

$$= 8\left[\left(\frac{n}{2}\right)^3\left(1 + 7^{(\lg 3+1)}\right) - 7^{(\lg 3+1)}\left(\frac{n}{2}\right)^{\lg 7}\right] + (3n)^{\lg 7}$$

$$= n^3\left(1 + 7^{\lg 3+1}\right) - 8\,7^{\lg 3+1}\cdot\frac{n^{\lg 7}}{7} + 7^{\lg 3}\cdot n^{\lg 7}$$

$$= n^3\left(1 + 7^{\lg 3+1}\right) - (7)(7^{\lg 3}\, n^{\lg 7})$$

$$= n^3\left(1 + 7^{\lg 3+1}\right) - 7^{\lg 3+1}\, n^{\lg 7} \qquad \text{c.q.d.}$$

Podemos concluir que $T(n) = O(n^3)$

(b) O algoritmo A2:

Seja $T'(n)$ o consumo de tempo do algoritmo A2, para uma instância de tamanho $n$

$$T(n) = 1 \qquad \text{se } n = 3^0$$
$$T(n) = 3\,T(n/3) + O(n \lg n) \qquad \text{se } n = 3^1, 3^2, \ldots$$

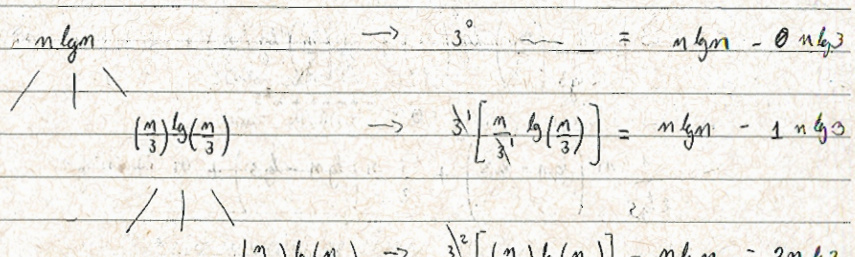
Grosseiramente podemos escrever a recorrência anterior como:

$$T(n) = 1 \qquad \text{se } n = 3^0$$
$$T(n) = 3\,T(n/3) + n\lg n \qquad \text{se } n = 3^1, 3^2, \ldots$$





(i) resolução da recorrência mediante uma árvore de recorrência




até $n = 3^i$

$i = \log_3 n$

no $i$-ésimo nível → $n \lg n - i n \lg 3$

$$T(n) = \sum_{i=0}^{\log_3 n - 1}\left[n\lg n - i n \lg 3\right] + 3^{\log_3 n}$$

$$= n\lg n\left(\log_3 n\right) - \frac{n\lg 3}{2}\left(\log_3 n - 1\right)\left(\log_3 n\right) + n$$

$$= n\lg n\left(\frac{\lg n}{\lg 3}\right) - n\left(\frac{\lg 3}{2}\right)\left[\left(\frac{\lg n}{\lg 3}\right)^2 - \left(\frac{\lg n}{\lg 3}\right)\right] + n$$

$$= \left(\frac{1}{\lg 3}\right) n\lg^2 n - \left(\frac{1}{2\lg 3}\right) n\lg^2 n + \frac{n}{2}\lg n + n$$

$$T(n) = \left(\frac{1}{2\lg 3}\right) n\lg^2 n + \frac{n}{2}\lg n + n$$

(ii) prova por indução em $n$, para $n = 3^0, 3^1, 3^2, \ldots$

- Base: $n = 1$ então $T(1) = 1$ (verdadeiro)
- Passo: $n > 1$ então

$$T(n) = 3\,T(n/3) + n\lg n$$

(continua na outra folha →)

$$T(n) = 3T(n/3) + n\lg n$$

$$= 3\left[\frac{1}{2\lg 3}\left(\frac{n}{3}\right)\lg^2\left(\frac{n}{3}\right) + \frac{1}{2}\left(\frac{n}{3}\right)\lg\left(\frac{n}{3}\right) + \left(\frac{n}{3}\right)\right] + n\lg n$$

$\lg^2 n - 2\lg n \lg 3 + \lg^2 3$

$$= \frac{1}{2\lg 3} n\left(\lg n - \lg 3\right)^2 + \frac{1}{2} n\left(\lg n - \lg 3\right) + n + n\lg n$$

$$= \frac{1}{2\lg 3} n\lg^2 n - n\lg n + \frac{n}{2}\lg 3 + \frac{1}{2} n\lg n - \frac{n}{2}\lg 3 + n + n\lg n$$

$$= \frac{1}{2\lg 3} n\lg^2 n + \frac{1}{2} n\lg n + n \qquad \text{c.q.d.}$$

Portanto $T(n) = O(n\lg^2 n)$

## (Q5) SELECT

▶ Algoritmo que recebe vetores $A[p..r]$, $W[p..r]$, de números inteiros, e inteiros $p, r$ e $i$ e devolve o $i$-ésimo menor valor da lista formada por $W[p]$ cópias de $A[p]$, $W[p+1]$ cópias de $A[p+1]$ ...

```
SELECT(A, W, p, r, i)
1  q ← ⌊(p+r)/2⌋
2  x ← SELECT-BFPRT(A, p, r, q)      ▶ procura o valor da mediana
3  PARTITION(A, W, p, r, x)          ▶ particiona A usando x como pivô.
                                       W deve manter a ordem de A.
4  w1 ← 0
5  para k ← 1 até q-1 faça
       w1 ← w1 + W[k]
```







```
7.  se w1 ≥ i
8.      então devolva SELECT(A, W, p, q-1, i)
9.      senão se w1 + W[q] ≥ i          ▶ w1 + W[q] ≥ i ≥ w1
10.         então devolva A[q]
11.         senão devolva SELECT(A, W, q+1, r, i - w1 - W[q])
```

Complexidade: Seja $T(n)$ o consumo de tempo do algoritmo, para um tamanho de entrada $n$.

| Linha | Consumo | |
|---|---|---|
| 1 | $\Theta(1)$ | |
| 2 | $O(n)$ | como mostrado no CLRS |
| 3 | $\Theta(n)$ | particiona usando como pivô $x$ |
| 4 | $\Theta(1)$ | |
| 5-6 | $O(n/2)$ | dado que $q$ é o índice da mediana |
| 7 | $\Theta(1)$ | |
| 8-11 | $T(n/2) + \Theta(1)$ | |

$$T(n) = T(n/2) + O(n)$$

ou grosseiramente $T(n) = 1$ se $n = 1$

$T(n) = T(n/2) + n$ se $n > 1$

claramente $T(n) \leq 2n$

- Prova por indução em $n$, para $n = 1, 2, 3, \ldots$
  - Base, $n = 1$, então $T(1) = 1 \leq 2$ (verdadeiro)
  - Passo, $n > 1$, então $T(n) = T(n/2) + n$
    $\leq 2(n/2) + n \leq 2n$
    $T(n) \leq 2n$ c.q.d.

Portanto $T(n) = O(n)$

- corretude: O algoritmo está correto pois para cada nível na recursão é conhecido se o elemento da mediana corresponde ao $i$-ésimo valor da lista. Se não for, então procura-se na metade

(Q6) ESCALONAMENTO DE TAREFAS (Veja Pág 115)

(Q7)

(1) Existe uma solução polinomial:

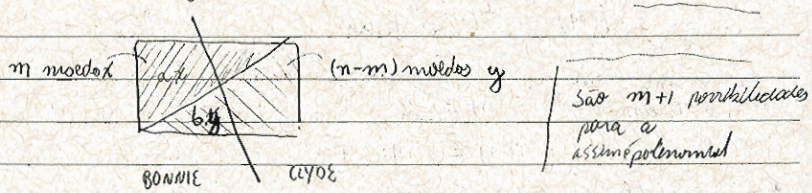
- Suponha que existam $m$ moedas $x$, assim existirão $(n-m)$ moedas $y$

A quantidade total $S = mx + (n-m)y$, sendo que cada pessoa terá $S/2$ dolares

- Suponha agora que Bonnie tome $a$ moedas de $x$, $0 \leq a \leq m$. Assim, deve-se verificar quantas moedas $y$ Bonnie pode conseguir.

com $a$ moedas: $ax \leq S/2$ $\quad b = \left[\frac{S/2 - ax}{y}\right] \leq (n-m)$

Se $ax+by = S/2$ então existe solução ($a$ e $b$ inteiros)




são $m+1$ possibilidades para a assim é polinomial

(2) do livro CLRS

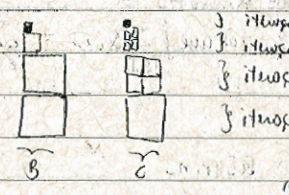
Existem $n$ moedas, com um número arbitrário de valores diferentes, mas todos potências de 2. ($1, 2, 4, 8, \ldots$)





Algoritmo: (E)

1. Ordenar as moedas em forma decrescente de valor
2. Dê a Bonnie a moeda com maior valor dm
3. Se a soma das moedas restantes é menor a dm então não existe solução
4. Caso contrário dê a Clyde tantas moedas quanto necessárias para atingir dm
5. Se ainda existem moedas faça o passo (2)
6. devolva "SIM"



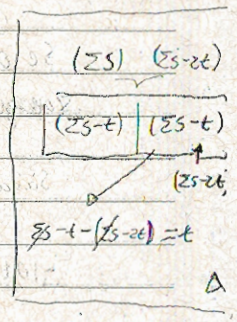

CONNECTED-DIVISION / Partição

(2) [da prova] NP-completo BC Divisão (BCD) é NP completo

(a) BCD é NP: É fácil ver que dada uma divisão de dirigíveis, podemos verificar qual a soma é a mesma

(b) SUB-SET-SUM $\leq_p$ BCD (PARTIÇÃO)

- Problema SUBSET-SUM:
  - Entrada: Vetor de inteiros $S[1..n]$ e inteiro $t$
  - saída: $I \subseteq \{1..n\}$ talque $\sum_{i \in I} S[i] = t$




- Problema BCD (PARTIÇÃO):
  - Entrada: Vetor de inteiros $P[1..n]$
  - saída: $I \subseteq \{1..n\}$ talque $\sum_{i \in I} P[i] = \sum_{i \notin I} P[i]$

Podemos fazer uma redução do problema SUBSET-SUM para BCD da seguinte forma

$P = S[1..n] \cup (\sum_{i=1}^{n} S - 2t)$

Se o problema BCD consegui solucionar o problema dado o vetor P, então conseguimos solucionar o problema SUBSET-SUM da seguinte maneira: Sabemos que BCD dará 2 conjuntos cada um somando $(\sum_{i=1}^{n} S - t)$ Uma das partições terá o elemento $(\sum S - 2t)$, se tirarmos esse elemento, então os elementos restantes somaram $t$

(3) Seja BCP100 o problema de partição de dinheiro aceitando uma diferença de 100 unidades

(a) BCP 100 é NP : Dado um certificado é fácil verificar em tempo polinomial a solução

(b) BCD $\leq_p$ BCP100 :

- Problema BCP100 :
  - Entrada : Vetor de inteiros Q[1..n]
  - Saída : $I \subseteq \{1..n\}$ tal que $\left| \sum_{i \in I} Q[i] - \sum_{i \notin I} Q[i] \right| \leq 100$

- Problema BCD (PARTIÇÃO)
  - Entrada : Vetor de inteiros P[1..n]
  - Saída : $I \subseteq \{1..n\}$ tal que $\sum_{i \in I} P[i] = \sum_{i \notin I} P[i]$

Podemos fazer uma redução do problema BCD ao problema BCP100 da seguinte forma:

$Q[i] = P[i] * 101$ para $i = \{1..n\}$

Se o problema BCP100 consegue solucionar o problema dado Q, então conseguimos solucionar o problema BCD da seguinte maneira.

Saída de BCP100 $I \subseteq \{1..n\}$ tal que $\left| \sum_{i \in I} Q[i] - \sum_{i \notin I} Q[i] \right| \leq 100$

$$-101 < \sum_{i \in I} P[i]*101 - \sum_{i \notin I} P[i]*101 < 101$$

$$-1 < \sum_{i \in I} P[i] - \sum_{i \notin I} P[i] < 1$$

ou $\sum_{i \in I} P[i] - \sum_{i \notin I} P[i] = 0$

ou $\sum_{i \in I} P[i] = \sum_{i \notin I} P[i]$





(Q8) Vetor K ordenado:

(a) Um vetor 1-ordenado é um vetor ordenado em forma não-decrescente, i.e.

$A[i] \leq A[i+1]$ para $i = 1..n-1$

(b) → 1 2 3 4 5 6 7 8 10 9 $k=2$

→ 1 3 2 5 4 7 6 9 8 10 $k=2$

(c) • Primeira parte: Se A está k ordenado temos

$$\frac{1}{k} \sum_{j=i}^{i+k-1} A[j] \leq \frac{1}{k} \sum_{j=i+1}^{i+k} A[j] \quad \text{para } i = 1, 2, \ldots, n-k$$

$$\sum_{j=i}^{i+k-1} A[j] \leq \sum_{j=i+1}^{i+k} A[j]$$

Sabemos que $\sum_{j=i}^{i+k-1} A[j] = \sum_{j=i+1}^{i+k} A[j] + A[i] - A[i+k]$ para $i = 1, 2, \ldots n-k$

Portanto:

$$\sum_{j=i+1}^{i+k} A[j] + A[i] - A[i+k] \leq \sum_{j=i+1}^{i+k} A[j]$$

$A[i] \leq A[i+k]$ para $i = 1, 2, \ldots, n-k$

• Segunda parte: Se $A[i] \leq A[i+k]$ então temos

$$\sum_{j=i+1}^{i+k} A[j] + A[i] \leq \sum_{j=i+1}^{i+k} A[j] + A[i+k] \quad \text{para } i = 1, 2, \ldots, n-k$$

$$\sum_{j=i+1}^{i+k} A[j] + A[i] - A[i+k] \leq \sum_{j=i+1}^{i+k} A[j]$$

Usando (I) temos: $\sum_{j=i}^{i+k-1} A[j] \leq \sum_{j=i+1}^{i+k} A[j]$

finalmente $\frac{1}{k} \sum_{j=i}^{i+k-1} A[j] \leq \frac{1}{k} \sum_{j=i+1}^{i+k} A[j]$ c.q.d.

(d) Algoritmo para k-ordenar um vetor de n elementos em $O(n \lg(n/k))$

▶ Algoritmo que recebe um vetor A[1..n] e o k ordena, i.e. arranja o vetor A tal que $A[i] \leq A[i+k]$ para $i = 1, \ldots, n-k$

```
K-ORDENA (A, n, k)
1.  K-ORDENA-QS (A, 1, n, k)

K-ORDENA-QS (A, p, r, k)
1.  se r - p > k
2.      então q ← ⌊(p+r)/2⌋
3.      x ← SELECT-BFPRT (A, p, r, q)      ▶ seleciona a mediana
4.      PARTICIONE (A, p, r, x)            ▶ x é o pivô para particionar
5.      K-ORDENA-QS (A, p, q-1)               os elementos menores ou
6.      K-ORDENA-QS (A, q+1, r)               iguais a x ficarão na esq.
                                              e os maiores na direita
                                              de q
```

* CORREÇÃO: K-ORDENA-QS

ordena em forma crescente somente se o tamanho do vetor for maior a k, caso contrário não ordena.

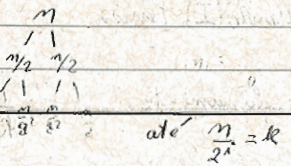
* COMPLEXIDADE: Seja $T(n)$ o consumo de Tempo de K-ORDENA-QS para um tamanho de instância $n := r - p + 1$

$$T(n) = 1 \quad , \text{se } n \leq k$$
$$T(n) = O(n) + 2T(n/2) \quad , \text{se } n > k$$

Grosseiramente a recorrência podemos escrevê-la como:

$$T(n) = 1 \quad , \text{se } n \leq k$$
$$T(n) = 2T(n/2) + n \quad , \text{se } n > k$$

- Resolução: Árvore de recorrência.






$$T(n) = \sum_{i=0}^{\lg(n/k)-1} n + 2^{\lg(n/k)}$$

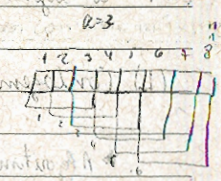
$$T(n) = n \lg(n/k) + n/k$$

- Prova por indução em n,

  - Base $n = k$ então: $T(k) = k \lg(1) + 1 = 1$ (verdade)

  - Passo $n > k$ então

$$T(n) = 2T(n/2) + n$$
$$= 2\left[\left(\frac{n}{2}\right)\lg\left(\frac{n}{2k}\right) + \frac{n}{2k}\right] + n$$
$$= n \lg\left(\frac{n}{k} \cdot \frac{1}{2}\right) + \frac{n}{k} + n$$
$$= n\left[\lg\left(\frac{n}{k}\right) + \lg\left(\frac{1}{2}\right)\right] + \frac{n}{k} + n$$
$$= n \lg\left(\frac{n}{k}\right) + \left(\frac{n}{k}\right) \quad \text{c.q.d.}$$




(e) Um vetor k-ordenado pode ser ordenado em $O(n \lg k)$

▶ Algoritmo que recebe um vetor A[1..n] de inteiros e um inteiro k, que representa a sua k-ordenação e devolve o vetor A ordenado de forma não decrescente.

```
ORDENA K VETOR (A, n, k)
1.  H ← BUILDHEAP (A, 1, k)        ▶ constrói um MIN HEAP usando A[1..k]   } O(k)
2.  para i ← 1 até n-k faça        ▶                                       } O(n-k+1) = O(n)
3.      INSERT-HEAP (H, A[i+k])                                            } n · O(lg k)
4.      min ← EXTRACT-HEAP (H)                                             } n · O(lg k)
5.      B[i] ← min                                                         } O(n)
6.  para i ← n-k+1 até n faça                                              }
7.      B[i] ← EXTRACT-HEAP (H)                                            } O(k lg k)
8.  devolva B
```

Total = $O(n \lg k)$ ■

(f) Consumo $k$-ordenar em $\Omega(n \lg n)$ no pior caso.

- Para um vetor $A[1..n]$ podemos ter um inteiro $k$, $1 \leq k \leq n-1$
- Quando $k = n-1$, então serão necessários poucos operações para $k$-ordenar o vetor $A$. Entretanto, se $k=1$ então, então os operações necessárias será proporcional a uma ordenação baseada em comparações. (esse é o pior caso)

Sabemos que qualquer ordenação baseada em comparações consome tempo $\Omega(n \lg n)$, portanto o algoritmo para $k$-ordenar um vetor, sendo $k$ variante, será $\Omega(n \lg n)$ ■

(Q9) Contagem de caminhos / contagem de caminhos mínimos

(1) Contagem de caminhos.

▶ Algoritmo que recebe um grafo dirigido acíclico $G = (V, E)$, e vértices $s, t \in E$, e devolve o número de caminhos de $s$ até $t$.

CONTA-CAMINHOS $(G, s, t)$
1. $T \leftarrow$ ORDENA-TOPOLOGICAMENTE$(G)$ ▶ $T$ é um vetor contendo os vértices ordenados em forma crescente de término
2. $T' \leftarrow$ PODA-VERTICES$(T, s, t)$ ▶ $T'$ terá os vértices de $T$ que vem $\geq s$ e $\leq t$ na ordenação topológica
3. paths$[T'[1]] \leftarrow 1$
4. para $i \leftarrow 2$ até $|T'|$ faça
5. paths$[T'[i]] \leftarrow 0$
6. para $i \leftarrow 2$ até $|T'|$ faça
7. para cada vértice $j$ = incidente$[T'[i]]$ faça
8. paths$[T'[i]] \leftarrow$ paths$[T'[i]] +$ paths$[j]$
9. devolva paths$[t]$



• Correção: A chave do algoritmo é a ordenação topológica, a qual garante que todo vértice $j$ incidente em $i$ estava antes do vértice $i$. Cada vértice terá um novo atributo chamado 'paths' que indica o número de caminhos existentes até o vértice.

Assim, para o problema de achar o número de caminhos de $s$ a $t$, inicialmente atribuimos paths$[s] \leftarrow 1$, os demais vertices terão um valor de 0. Facilmente, usando uma abordagem de P.D. temos que

$$\text{paths}[i] := \sum_{(j,i) \in E} \text{paths}[j]$$

paths$:= \sum_{(j,i) \in E}$ (paths$[j]$) se peso$[j] =$ min

min $=$ min$_{(j,i) \in E}$ { peso$[j] + w(j,i)$ }

caso dos caminhos mínimos

• complexidade:

| Linha | Consumo | |
|---|---|---|
| 1 | $O(\|V\|+\|E\|)$ | |
| 2 | $O(\|V\|)$ | |
| 3 | $\Theta(1)$ | |
| 4-5 | $O(\|V\|)$ | |
| 6-8 | $O(\|V\|+\|E\|)$ | ▶ dado que somente uma vez é examinado todo vértice e toda aresta |
| 9 | $\Theta(1)$ | |
| TOTAL | $O(\|V\|+\|E\|)$ | |

Se $T(n)$ é o consumo de tempo, para uma instância de tamanho $n := (|V|+|E|)$, então $T(n) = O(n)$.

ORDENAÇÃO-TOPOLOGICA$(G)$ ▶ $O(|V|+|E|)$

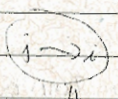
Faça uma busca em profundidade (DFS) no grafo $G$ e anote sequencialmente o número de visita e o número de término, logo faça a ordenação em forma não-crescente de valor de término. Com isso garante-se, que todo vertice $j$ incidente ao vertice $i$, estava antes do vertice $i$.

COMPLEXIDADE: Como DFS foi a busca em tempo $O(|V|+|E|)$, a nossa versão para a ordenação topológica consome também $O(|V|+|E|)$.

(b) Contagem de caminhos mínimos $w(i,j)$

▶ Algoritmo que um grafo dirigido acíclico $G=(V,E)$, em que cada aresta contem um valor de peso, e vértices $s, t \in V$, e devolve o número de caminhos mínimos de $s$ até $t$.

```
CONTA-CAMINHOS-MINIMOS ( G, s, t)
1.  T ← ORDENA-TOPOLOGICAMENTE                          ┐ do algoritmo
2.  T' ← PODA-VERTICES ( T, s, t)                       ┘ anterior
3.  peso [ s ] ← 0 , paths [ s ] ← 1
4.  para i ← 2 até |T'| faça
5.      peso [ T'[i] ] ← ∞ , paths [T'[i]] ← 0
6.  para i ← 2 até |T'| faça
7.      min ← ∞
8.      para cada vértice j incidente [ T'[i] ] faça    ┐
9.          se min > w(j, T'[i]) + peso [ j ]           │ acha o menor
10.             então min ← w(j, T'[i]) + peso [j]      ┘ caminho até T'[i]
11.     para cada vértice j incidente [ T'[i] ] faça    ← mínimo
12.         se (w(j, T'[i]) + peso[j] ) = min.
13.             então paths [ T'[i]] ← paths [T'[i]] + paths [j]
14.                 peso [ T'[i]] ← min
15. devolva paths [t]
```




• Corretude: Dada a ordenação topológica do grafo $G$, garante-se que a contagem de caminhos mínimos para todo vertice $j$ incidente em $i$, será calculado com antecedência.
→ 2 fases:
a° Calculo do caminho de peso mínimo (Linhas 8-10)
b° Contagem dos caminhos de peso mínimo (Linhas 11-14)

• Complexidade: Continua sendo $O(|V|+|E|)$, no que cada aresta é analisada 2 vezes (uma em cada fase).





25/07/2008

2000/1

(Q1) Uma árvore rubro-negra (RED-BLACK) é uma árvore binária balanceada na qual dados 2 filhos de um nó, o caminho para chegar as folhas de um é no máximo o dobro do outro (árvore binária em que cada nó um atributo: COR)

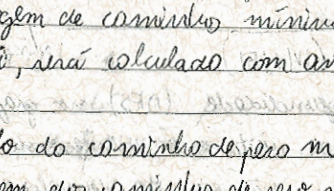
PROPRIEDADES:
a - Toda árvore tem uma raiz PRETA e Todo nó é vermelho ou preto
b - Um nó vermelho tem obrigatoriamente 2 nos pretos
c - Toda folha é preta
d - Para todo nó, o caminho até as folhas tem igual número de nós pretos

Se a cor da raiz é mudada, então não é mais RUBRO-PRETA, pois não seria mais garantida a propriedade (d), portanto seria uma árvore desbalanceada.

(Q2) Recorrências:

(a) $T(1) = 1$
$T(n) = 3\,T(n/2) + n^2$

árvore de recorrência




até $n = 2^i$
$i = \log n$

$T(n) = \sum_{i=0}^{\log n - 1} n^2 \left(\frac{3}{4}\right)^i + 3^{\log n}$

$\leq n^2 \sum_{i=0}^{n} \left(\frac{3}{4}\right)^i + n^{\log 3}$

$\leq n^2 \left(\frac{1}{1-3/4}\right) + n^{\log 3} = 4n^2 + n^{\log 3}$

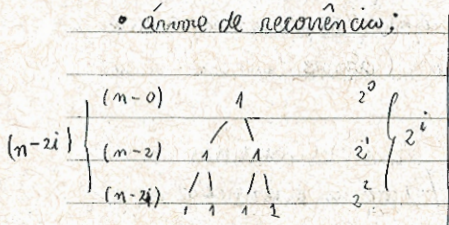
$= O(n^2)$

Prova por indução.

(b) $T(1) = 1$

$T(n) = 2\,T(n-2) + 1$

• árvore de recorrência:




até $n - 2i = 1$, $\frac{n-1}{2} = i$

$$T(n) = \sum_{i=0}^{\frac{(n-1)}{2}-1} 2^i + 2^{\frac{n-1}{2}}$$

$$T(n) = \sum_{i=0}^{\frac{(n-1)}{2}} 2^i = \frac{2^{\frac{(n-1)}{2}+1} - 1}{(2-1)}$$

$$T(n) = 2^{(n+1)/2} - 1$$

• Proposição: $T(n) = 2^{(n+1)/2} - 1$

• Prova por indução em $n$; para $n = 1, 2, \ldots$

- Base: $n=1$ então $T(1) = 2^1 - 1 = 1$ o qual é verdadeiro

- Passo: $n > 1$ então

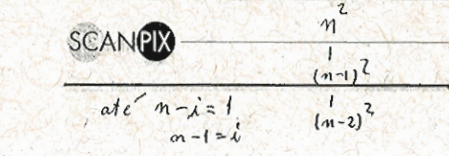
$$T(n) = 2\,T(n-2) + 1$$
$$= 2\left[2^{(n-1)/2} - 1\right] + 1$$
$$T(n) = 2^{(n+1)/2} - 1 \quad \text{c.q.d.}$$

Portanto $T(n) = O(2^{n/2})$ $\qquad 2^{n/2+1/2} = \sqrt{2}\cdot 2^{n/2} = O(2^{n/2})$

(c) $T(1) = 1$

$T(n) = T(n-1) + n^2$

• árvore de recurrência:




$$T(n) = \sum_{i=0}^{(n-1)-1} (n-i)^2 + 1$$

$$= \sum_{i=0}^{n-2} \left(n^2 - 2ni + i^2\right) + 1$$

$$= n^2(n-1) - 2n\left[\frac{1}{2}(n-2)(n-1)\right] + \Theta(n^3) + 1$$



$$T(n) = n^3 - n^2 - (n^2 - 2n)(n-1) + \Theta(n^3) + 1$$

$$= n^3 - n^2 - \left[n^3 - n^2 - 2n^2 + 2n\right] + \Theta(n^3) + 1$$

$$= n^3 - n^2 - n^3 + 3n^2 - 2n + \Theta(n^3) + 1$$

$$= \Theta(n^3) + 2n^2 - 2n + 1 = O(n^3)$$

$\sum_{i=0}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$

Poderia ter sido usada mas aqui maiores.

Proposição: $T(n) \leq n^3 + n^2$

Prova por indução em $n$, para $n = 1, 2, 3, \ldots$

- Base: $n=1$, então $T(1) = 1 \leq 2$ (verdadeiro)

- Passo: $n > 1$, então

$$T(n) = T(n-1) + n^2$$
$$\leq (n-1)^3 + (n-1)^2 + n^2$$
$$\leq n^3 - 3n^2 + 3n - 1 + n^2 - 2n + 1 + n^2$$
$$\leq n^3 - n^2 + n \leq n^3 + n \leq n^3 + n^2 \quad \text{para } n \geq 1$$

$$T(n) \leq n^3 + n^2$$

Portanto $T(n) = O(n^3)$ c.q.d.

(Q3) Mr. B.A. VACA

- Usando comparações pode-se criar algoritmos que trabalhem com estruturas de dados para filas de prioridades de tal modo que a inserção e o Máximo sejam executados no pior caso $O(1)$. Mas se existe um algoritmo para extrair o Máximo em $O(1)$, então esse alg. poderia ser usado para ordenar qualquer vetor em tempo $O(n)$, dada que basta $n$ extrações de máximos. O qual é uma contradição para o limitante inferior $\Omega(n \lg n)$ para a ordenação baseada em comparações.

(Q4) Frequencia de accesso:

| $n$ = 10000 | 4000 | 6000 |
|---|---|---|
| Pessoas | 0.40 sons (a) | 0.60 ruins (b) |
| # Acessos | 0.60 | 0.40 |

(1) $n = 10\,000$

busca binária é $O(\lg n) = \lg(10000) = 13.29$

(2)


$\lg(4000)$ para obter 1 som

$\lg(6000)$ para obter 1 ruim

$0.60 * \lg(4000) + 0.40 * \lg(6000)$

$0.60\,(11.97) + 0.40\,(12.55)$

$7.182 + 5.020 = 12.202$

Portanto (2) apresenta a melhor tempo de consulta.

(Q5) MAX-HEAP: Descrição do algoritmo BUILD-MAXHEAP

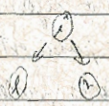
CONSTROI-HEAP ( A, n )

1 para $i \leftarrow \lfloor \frac{n}{2} \rfloor$ até 1 faça

2. MAX-HEAPIFY ( A, n, i )

MAX-HEAPIFY ( A, n, i )

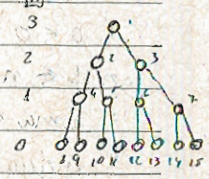
1. $l \leftarrow 2i$
2. $r \leftarrow 2i + 1$







3. se $l \leq n$ e $A[l] > A[i]$
4. então maior $\leftarrow l$
5. senão maior $\leftarrow i$
6. se $r \leq n$ e $A[r] > A[\text{maior}]$
7. então maior $\leftarrow r$
8. se maior $\neq i$
9. então troca $A[i] \leftrightarrow A[\text{maior}]$
10. MAX-HEAPFY ( A, n, maior )




* Complexidade:

- MAXHEAPFY é linear na altura dos nós
- Para um heap, na altura h temos $\leq \left\lceil \frac{n}{2^{h+1}} \right\rceil$ nós
- A altura de um heap de n nós é $\lfloor \lg n \rfloor$
- MAXHEAPFY consome $O(h)$ para ordenar algum nó de altura h
- Portanto o consumo de tempo para o CONSTROI-HEAP será:

$$T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} O(h) \left\lceil \frac{n}{2^{h+1}} \right\rceil = n \sum_{h=0}^{\lfloor \lg n \rfloor} O\left(\frac{h}{2^h}\right)$$

↳ maiorando, o certo é $h=1$

$$= n \sum_{h=0}^{\lfloor \lg n \rfloor} h \left(\frac{1}{2}\right)^h$$

$$\leq n \sum_{n=0}^{\infty} h \left(\frac{1}{2}\right)^h = n \left[ \frac{1/2}{(1-1/2)^2} \right] = 2n$$

$T(n) \leq 2n$

$T(n) = O(n)$

(Q6) Intercalação de k listas : Veja Questão 3 (1998/2) Pág 68

(Q7) Seqüência de operações, para p lido iterativamente

```
ALGORITMO(p)
1. a ← 0
2. b ← n
3. soma ← 0
4. enquanto a<n e b>0 faça
5.     if p>0
6.     então a ← a+1      } +1/-1/+1/...
7.     senão a ← a-1
8.           b ← b-1      ▶ b só decrementa
9.     soma ← soma + 1
10.    p ← LEIAP()
```

Resposta: O valor máximo para a soma é 3n-2 que corresponde a execução de:

(i) 2(n-1) operações para p>0 ou p<0 (indistintamente)

(ii) n operações para p>0

- Note-se que para qualquer valor de p (>0, <0) o valor da soma se incrementa em uma unidade.

n=3

| | a (0 → ) | b (→ n) | |
|---|---|---|---|
| >0 | +1 | | 2(n-1) = 2(2) = 4 operações |
| <0 | -1 | -1 | |
| >0 | +1 | | |
| <0 | -1 | -1 | |
| >0 | +1 | | n = 3 operações |
| >0 | +1 | | |
| >0 | +1 | | |

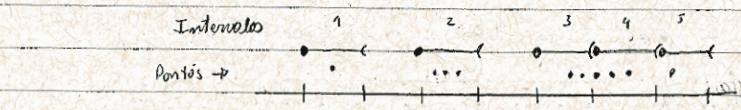
} 7 operações

soma = 7





(Q8) Menor conjunto de intervalos de comprimento unitário




▶ Algoritmo que recebe um vetor P[1..n] de pontos na reta real e devolve o menor conjunto C de intervalos unitários que contém todos os n pontos

```
MENOR-CONJUNTO ( P, n )
1. P ← MERGE-SORT ( P, n )     ▶ Ordene de forma não-decrescente P
2. k ← 0
3. i ← 0
4. enquanto i < n faça
5.     i ← i+1
6.     p1 ← ⌊P[i]⌋
7.     p2 ← ⌈P[i]⌉
8.     enquanto P[i+1] < p2 faça
9.         i ← i+1
10.    k ← k+1
11.    C[k] ← p1
12. devolva C                  ▶ C[1..k] contém os números inteiros correspondentes
                                  aos intervalos unitários.
```

O(n) pois cada ponto i é examinado uma única vez.

- Correção: A chave está na ordenação dos pontos. O algoritmo começa com o menor ponto e procura-se pegar todos os pontos maiores a ele que ainda estejam no próprio intervalo unitário (abordagem gulosa), com isso assegura-se que cada ponto esta únicamente num intervalo.

- Consumo: Seja T(n) o consumo de tempo para uma instância de tamanho n (n pontos), então $T(n) = O(n \lg n)$

A complexidade está influenciada pela ordenação (MERGESORT)

PROVA 2000/2 (PF)

(Q1) Recorrência:

$T(1) = 1$

$T(n) \leq T(n-1) + O(n)$ ▷ Existem constante $C>0$ e $n_0$ t.q. $T(n) \leq T(n-1) + cn$

- Discussão: Significa que a função T, para um dado inteiro $i$ é menor ou igual à remotoria da função para $i-1$ com alguma instância de $O(n)$.
- Resolução:

$T(n) = 1$ , se $n=1$

$T(n) \leq T(n-1) + n$ , se $n>1$

- árvore de recorrência:

$n$
$n-1$
$n-2$

$T(n) \leq \sum_{i=0}^{n-1} (n-i) = n + (n-1) + \dots + 2 + 1$

$\leq \sum_{i=1}^{n} i = \frac{1}{2}(n+1)n = \frac{n^2+n}{2}$

até $n-i=1$, $i=n-1$ $T(n) \leq \frac{n^2}{2} + \frac{n}{2}$

- Proposição: $T(n) \leq \frac{n^2}{2} + \frac{n}{2}$

- Prova por indução em $n$, para $n = 1, 2, 3, \dots$

(i) base: $n=1$, então $T(1) \leq \frac{1}{2} + \frac{1}{2} \leq 1$ (verdadeiro)

(ii) passo: $n>1$, então

$T(n) \leq T(n-1) + n$

$\leq \frac{(n-1)^2}{2} + \frac{(n-1)}{2} + n$



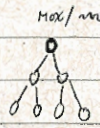
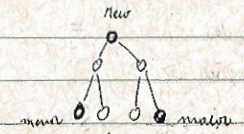


$T(n) \leq \frac{1}{2}\left[ n^2 - 2n + 1 + n - 1 + 2n \right]$

$T(n) \leq \frac{1}{2}n^2 + \frac{1}{2}n$ c.q.d.

Portanto $T(n) = O(n^2)$ ▷ pertence ao conjunto de funções $O(n^2)$

(Q2) HEAP vs ABB para uso em filas de prioridades

| | HEAP MÁXIMO / MÍN | ABB |
|---|---|---|
| |   |   |
| construção | $O(n)$ ← | $O(n \lg n)$ ← |
| inserção | $O(\lg n)$ | $O(\lg n)$ |
| eliminação | $O(\lg n)$ | $O(\lg n)$ |
| máximo | $\Theta(1)$ ← | $O(\lg n)$ ← |
| extrai máximo | $O(\lg n)$ | $O(\lg n)$ |

- Favorável quando precisa-se conhecer rapidamente o maior ou o menor elemento
- Ordenação em $O(n \lg n)$, que corresponde a $n$ vezes extrai Máximo/Mínimo

- Favorável quando deseja-se conhecer o elemento de prioridade mediana
- Ordenação, dada uma ABB, em $\Theta(n)$ pois precisa-se percorrer uma vez cada $n$

(Q3) Sequência de n operações / A. amortizada

(1) Seja S uma sequência de n operações

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^0$ | $2^1$ | 1 | $2^2$ | 1 | 1 | 1 | $2^3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $2^4$ | 1 |

Seja C o custo total das primeiras n operações:

$$C = n + \sum_{i=0}^{\lfloor \lg n \rfloor} (2^i - 1)$$

$$= n + \sum_{i=0}^{\lfloor \lg n \rfloor} 2^i - \sum_{i=0}^{\lfloor \lg n \rfloor} 1$$

$$\leq n + \left(2^{\lg n + 1} - 1\right) - (\lg n + 1)$$

$$\leq n + 2n - 1 - \lg n - 1 = 3n - \lg n - 2$$

$$< 3n$$

Método agregado: Soma dos custos de todas as operações para determinar o custo amortizado de cada operação

Portanto, o custo amortizado para cada operação será de $\frac{3n}{n} = 3 = O(1)$

(2) Função Potencial: $\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$

↳ custo real — diferença de potencial

↳ custo amortizado

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 1 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 |
| operação | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| custo | $2^0$ | $2^1$ | 1 | $2^2$ | 1 | 1 | 1 | $2^3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $2^4$ | 1 |
| | 1 | 3 | 4 | 8 | 9 | 10 | 11 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 42 | 43 |
| | 0 | 0 | 2 | 0 | 2 | 4 | 6 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 0 | 2 |

k

i

custo

custo (soma)

Potencial

$16 - 2^{1+3} = 0$



- $2^{(i+1)} - 2^i = 2^i$ é a distância entre 2 potências de 2 operações sequenciais
- Assim, podemos amortizar o custo da seguinte operação cara considerando uma carga adicional constante (de pelo menos 2 por operação) para operações baratas ~~$c(2^i) \geq 2^{i+1}$~~ ~~$c \geq 2$~~

Seja $c \geq 2$ uma constante e para $i = 1, 2, 3, \ldots$ definimos

$$\Phi(i) = c\left(i - 2^{\lfloor \lg i \rfloor}\right)$$

Isto é, se temos um inteiro m tal que $2^m \leq i$ e k tal que

$$i = 2^m + k \quad \text{então}$$

$$\Phi(i) = ck$$

Temos 2 casos.

$k > 0$: $\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$

$= 1 + ck - c(k-1) = 1 + ck - ck + c = 1 + c = O(1)$

$k = 0$: $\hat{c_i} = 2^m + 0 - c\left[\left(2^m - 1\right) - 2^{(m-1)}\right]$

$= 2^m - c\left[2^m - 1 - \left(\frac{1}{2}\right)2^m\right] = 2^m - c\left[\left(\frac{1}{2}\right)2^m - 1\right]$

$= 2^m\left[1 - \frac{1}{2}c\right] + c = O(1)$

(2) Função Potencial

| $i$: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| custo: | $2^0$ | $2^1$ | 1 | $2^2$ | 1 | 1 | 1 | $2^3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $2^4$ | 1 | 1 | 1 | 1 |
| energia Potencial: | 0 | 0 | 2 | 0 | 2 | 4 | 6 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 0 | 2 | 4 | 6 | 8 |

- Existe uma distância de $|2^{(i+1)} - 2^{(i)}| = 2^i$ elementos entre 2 potencias consecutivas

- Assim, podemos amortizar o custo da seguinte operação cara, considerando uma carga adicional constante (de pelo menos 2 unidades por operação) nas operações baratas

$$c(2^i) \geq 2^{i+1}$$
$$c \geq 2$$

Função potencial: $\Phi(1) = 0$ , para $i = 0$

$$I(i) = 2i - 2^{1+\lfloor \lg i \rfloor} \text{ , para } i > 0$$

Portanto, para $i = 1$ temos

$$\hat{c_i} = c_i + \Phi(1) - \Phi(0)$$
$$= 1 + [2 - 2^1] - 0 = 1 \quad \text{(constante)}$$

Para $i > 1$, e não potência de 2:

$\lfloor \lg(i-1) \rfloor = \lfloor \lg i \rfloor$ dado que $i$ não é potência de 2 !!

$$\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$$
$$= 1 + [2i - 2^{1+\lfloor \lg i \rfloor}] - [2(i-1) - 2^{1+\lfloor \lg(i-1) \rfloor}]$$
$$= 1 + 2i - 2 - 2^{1+\lfloor \lg i \rfloor} + 2^{1+\lfloor \lg(i-1) \rfloor}$$
$$= 3 \quad \text{(constante)}$$





Para $i > 0$, é $i = 2^j$ (i potência de 2) , para $j = 0, 1, 2, \ldots$ $\quad i - 1 = 2^j - 1$

$$\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$$
$$= 2^j + 0 - \left[ 2(2^j - 1) - 2^{1+\lfloor \lg(2^j - 1) \rfloor} \right]$$
$$= 2^j - 2 \cdot 2^j + 2 + 2^{1+\lfloor \lg(2^j - 1) \rfloor}$$
$$= -2^j + 2 + 2^{1+\lfloor \lg(2^j - 1) \rfloor} \quad \rightarrow \quad \lfloor \lg(2^j - 1) \rfloor = \lfloor \lg 2^j \rfloor - 1 = (j-1)$$
$$= 2 \quad \text{(constante)}$$

(3) método contábil:

Atribua para cada operação 3 reais; de tal forma que o custo amortizado seja o seguinte ($\hat{c_i}$)

Seja $c_i$ o custo da $i$-ésima operação

$$c_i = \begin{cases} i, & \text{se } i \text{ é potência de 2} \\ 1, & \text{cc.} \end{cases}$$

- Se $i$ não é potência exata de 2, pague com 1 real, e armazene 2 como crédito
- " $i$ é potência exata de 2, pague $i$ reais usando o crédito armazenado

Do exercício (1) sabemos que $\sum_{i=1}^{n} c_i \leq 3n$

Então $\sum_{i=0}^{n} \hat{c_i} \geq \sum_{i=0}^{n} c_i$ crédito = custo amortizado − custo real $\geq 0$



Como o custo amortizado em cada operação é constante $O(1)$ e a quantidade nunca é negativa; $n$ operações levam custo $O(n)$

❼ Outro exercício parecido:

• Uma seq. de n operações é realizada numa estrutura de dados. A i-ésima operação custa $\sqrt{i}$, se i é um quadrado perfeito, e 1 em caso contrário determine o custo amortizado por operação.

$(i)^2$ $(j+1)^2$

| i: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| custo: | $\sqrt{1}$ | 1 | 1 | $\sqrt{4}$ | 1 | 1 | 1 | $\sqrt{9}$ | 1 | 1 | 1 | 1 | 1 | 1 | $\sqrt{16}$ | 1 | 1 | 1 | 1 | |
| | 0 | ½ | 1 | 0 | ½ | 1 | 3/2 | 2 | 0 | ½ | 1 | 3/2 | 2 | 5/2 | 3 | 0 | ½ | 1 | 3/2 | 2 |

(1) Método agregado:

Seja C o custo dos primeiros n operações na sequencia, com $n>0$, assim.

$$C = n + \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} (\sqrt{(i)^2} - 1)$$

$$= n + \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} (i-1) = n + \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} i - \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} 1$$

$$\leq n + \frac{\sqrt{n}(\sqrt{n}+1)}{2} - \frac{2\sqrt{n}}{2} = n + \frac{1}{2}\left[n + \sqrt{n} - 2\sqrt{n}\right]$$

$$\leq n + \frac{1}{2}n - \frac{1}{2}\sqrt{n} < \frac{3}{2}n$$

$C < 3/2\, n$

Portanto o custo amortizado por operação é $\frac{C}{n} = \frac{3/2\, n}{n} = \frac{3}{2} = O(1)$

(2) Método contábil:

Atribua a cada operação 1.5 reais e faça a amortização da seguinte forma:
- Se i não é quadrado perfeito, pague com 1 real e armazene 0.5 reais como crédito
- Se i é quadrado perfeito, pague $\sqrt{i}$ reais com o crédito armazenado.





Do exercício anterior (1) sabemos que $\sum_{i=1}^{n} c_i < 3/2\, n$, então

$$\sum_{i=1}^{n} \hat{c}_i > \sum_{i=1}^{n} c_i \qquad \text{crédito} = \text{custo amortizado} - \text{custo real} > 0$$

Portanto cada operação tem um custo amortizado de 3/2 reais $= O(1)$

(3) Método potencial:

• Sabemos que a distância entre 2 elementos quadrados perfeitos é

$$(i+1)^2 - (i)^2 = \cancel{i^2} + 2i + 1 - \cancel{i^2} = 2i+1$$

• Assim, podemos amortizar o custo de uma operação cara, mediante uma carga constante (de ao menos 1/2 unidades) nas operações baratas.

$$c(2i+1) \geq (i+1)^2$$

$$c \geq \frac{(i+1)^2}{2i+1} \geq \frac{1}{2}\left(\frac{i+1}{i+1}\right)^2, \quad \boxed{c \geq 1/2} \qquad \text{majorando}$$

seja $\hat{c}_i$ o custo amortizado para a operação i-ésima

Função potencial $\Phi(i) = 0$, se $i = 0$

$\Phi(i) = \frac{1}{2}i - \frac{1}{2}\lfloor\sqrt{i}\rfloor^2$, se $i > 0$

Quando $i = 1$ então

$$\hat{c}_i = c_i + \Phi(1) - \Phi(0)$$

$$= 1 + \left[\frac{1}{2} - \frac{1}{2}\right] - 0 = 1 = O(1) \qquad \text{constante}$$

Quando $i > 1$, e i não é quadrado perfeito — como i não é quadrado perfeito $\lfloor\sqrt{i}\rfloor = \lfloor\sqrt{i-1}\rfloor$

$$\hat{c}_i = c_i + \Phi(i) - \Phi(i-1)$$

$$= 1 + \left[\frac{1}{2}i - \frac{1}{2}\lfloor\sqrt{i}\rfloor^2\right] - \left[\frac{1}{2}(i-1) - \frac{1}{2}\lfloor\sqrt{i-1}\rfloor^2\right]$$

$$= 1 + \frac{1}{2}i - \frac{1}{2}i + \frac{1}{2} - \frac{1}{2}\left[\cancel{(i)^2} + \cancel{(1)^2}\right] = \frac{3}{2} \qquad \text{(constante)}$$

Quando $i > 1$, e $i$ é quadrado perfeito, seja $i = (j)^2$

$$\hat{c}_i = c_i + \Phi(i) - \Phi(i-1)$$

$$= \sqrt{i} + 0 - \left[\frac{1}{2}(i-1) - \frac{1}{2}\left\lfloor\sqrt{i-1}\right\rfloor^2\right]$$

$$= j = \frac{1}{2}(j^2-1) + \frac{1}{2}\left\lfloor\sqrt{j^2-1}\right\rfloor^2$$

$$= j - \frac{1}{2}j^2 + \frac{1}{2} + \frac{1}{2}\left\lfloor\sqrt{j^2-i}\right\rfloor^2 \qquad \left\lfloor\sqrt{j^2-1}\right\rfloor = j-1$$

pois $i$ é quadrado perfeito

$$= j - \frac{1}{2}j^2 + \frac{1}{2} + \frac{1}{2}(j-1)^2$$

$$= j - \frac{1}{2}j^2 + \frac{1}{2} + \frac{1}{2}(j^2 - 2j + 1)$$

$$= \cancel{j} - \cancel{\frac{1}{2}j^2} + \frac{1}{2} \pm \cancel{\frac{1}{2}j^2} - \cancel{j} + \frac{1}{2} = 1 = O(1)$$

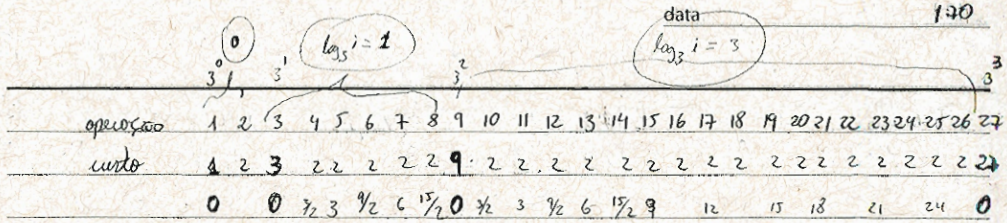
Finalmente, podemos concluir que o custo amortizado de uma operação na sequência é $O(1)$

- Ainda outro parecido:

Uma seq. de $n$ operações é executada sobre uma estrutura de dados. Para $i = 1, \ldots, n$, a $i$ésima operação custa $i$, se $i$ é potência inteira 3 e custa 2 c.c. Determine o custo amortizado.








(1) MÉTODO AGREGADO:

Seja $C$ o custo das primeiras $n$ operações, na sequência, com $n > 0$

$$C = 2n + \sum_{i=0}^{\lfloor \log_3 n \rfloor} (3^i - 1)$$

$$= 2n + \sum_{i=0}^{\lfloor \log_3 n \rfloor} 3^i - \sum_{i=0}^{\lfloor \log_3 n \rfloor} 1 \qquad \frac{4n+3n}{2}$$

$$\leq 2n + \left\lfloor \frac{3^{\log_3 n + 1} - 1}{2} \right\rfloor - \left\lfloor \log_3 n + 1 \right\rfloor = 2n + \frac{1}{2}\left\lfloor 3n - 1 \right\rfloor - \left\lfloor \log_3 n + 1 \right\rfloor$$

$$C \leq \frac{7n}{2} - \frac{1}{2} - \log_3 n - 1 < \frac{7}{2}n = O(n)$$

Portanto o custo amortizado por operação é $\frac{C}{n} = \frac{\frac{7}{2}n}{n} = \frac{7}{2} = O(1)$

(2) MÉTODO CONTÁBIL:

- atribua a cada operação $2 + 3/2 = 7/2$ reais e faça a amortização da seguinte forma:
  (i) Se $i$ não for potência inteira de 3, então pague a operação com 2 reais e armazene 3/2 como crédito
  (ii) Se $i$ for potência inteira de 3, então pague a operação com o crédito armazenado

(3) MÉTODO POTENCIAL:

- Sabemos que a distância entre 2 elementos potência de 3 é $|3^{i+1} - 3^i| = 2 \cdot 3^i$
- Assim podemos amortizar o custo de uma operação cara, considerando incrementos constantes (de pelo menos 3/2 unidades) nas operações baratas (cargo)

$c(2 \cdot 3^i) \geq 5^{i+1}$ logo $(C \geq 3/2)$

Função potencial $\Phi(i) = 0$ , se $i = 0$

$\Phi(i) = \frac{3}{2} i - \frac{3}{2} 3^{\lfloor \log_3 i \rfloor}$ , se $i > 0$

Seja $\hat{c_i}$ o custo amortizado pela i-ésima operação

Temos 3 casos para i:

- Quando $i = 1$, então

$$\hat{c_i} = c_i + \Phi(1) - \Phi(0)$$
$$= 1 + \left[ \frac{3}{2} - \frac{3}{2} \right] = 1 = O(1)$$

- Quando $i > 1$, e $i$ é potência inteira 3: , i.e. $i = 3^j$

$$\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$$
$$= i + 0 - \left[ \frac{3}{2}(i-1) - \frac{3}{2} 3^{\lfloor \log_3 (i-1) \rfloor} \right]$$
$$= i - \frac{3}{2} i + \frac{3}{2} + \frac{3}{2} 3^{\lfloor \log_3 (i-1) \rfloor} \quad \text{como: } \lfloor \log_3 (i-1) \rfloor = \log_3 i - 1$$
$$= \frac{3}{2} - \frac{1}{2} i + \frac{3}{2} 3^{\lfloor \log_3 (i-1) \rfloor} \quad \text{seja } i = 3^j$$
$$= \frac{3}{2} - \frac{1}{2} 3^j + \frac{3}{2} 3^{j-1} = \frac{3}{2} - \frac{3^j}{2} + \frac{3^j}{2}$$
$$= \frac{3}{2} = O(1)$$





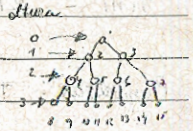
- Quando $i > 1$, e não é potência inteira de 3

$$\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$$
$$= 2 + \left[ \frac{3}{2} i - \frac{3}{2} 3^{\lfloor \log_3 i \rfloor} \right] - \left[ \frac{3}{2}(i-1) - \frac{3}{2} 3^{\lfloor \log_3 (i-1) \rfloor} \right]$$
$$= 2 + \frac{3}{2} i - \frac{3}{2} i + \frac{3}{2} - \frac{3}{2} 3^{\lfloor \log_3 i \rfloor} + \frac{3}{2} 3^{\lfloor \log_3 (i-1) \rfloor}$$

dado que para $i$ não potência de 3 é válido que $\lfloor \log_3 i \rfloor = \lfloor \log_3 (i-1) \rfloor$

$$= 7/2 = O(1)$$

(Q4) Seja $i$ um nó de um heap $A[1..n]$. Mostre que a altura de um nó $i$ é $\lfloor \log(\frac{n}{i}) \rfloor$.




Todo heap de altura $h$ tem entre $2^h$ e $2^{h+1} - 1$ nós

- O heap tem $h$ níveis: $0, 1, 2, \ldots, h$
- cada nível $j = 0, 1, 2, \ldots, h$ tem $\leq 2^j$ nós
- O nível $h$ tem $x$ nós: $1 \leq x \leq 2^h$
- O número total de nós para o heap será:
  - se $x = 2^h$: $2^0 + 2^1 + 2^2 + \ldots + 2^h = 2^{h+1} - 1$
  - se $x = 1$: $2^0 + 2^1 + 2^2 + \ldots + 2^{h-1} + 1 = 2^h$
- Portanto o número total de nós estará entre $2^h$ e $2^{h+1} - 1$

- Seja $h_i$ a altura do nó $i$
- $h_i$ é o comprimento da sequência $(i, 2i, \ldots, 2^{h_i} i)$

onde $2^{h_i} \leq n < 2^{h_i+1}$

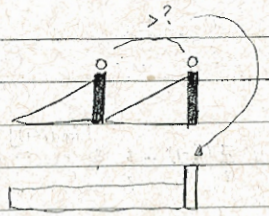
$2^{h_i} i \leq n < 2^{h_i+1} i$

$2^{h_i} \leq n/i < 2^{h_i+1}$

$h_i \leq \log(\frac{n}{i}) < h_i + 1$

$h_i = \lfloor \log(\frac{n}{i}) \rfloor$ C.Q.D.

(Q5) SLOWSORT




Em cada chamada recursiva do SLOWSORT é colocado o maior elemento do vetor de n elementos e atribuído na posição n. É feito esse processo para o tamanho de instancia n-1.

$T(n) = 1$

$T(n) = 2T(n/2) + T(n-1) + 1$

Proposição: $T(n) \geq 2^9 n^9 = 2^9 + 1$

Prova: Por indução em n, para $n = 2^0, 2^1, 2^2, \ldots$

- Base $n=1$ $\quad T(1) \geq 1$ (verdadeiro)
- Passo $n > 2$

$$T(n) = 2T(n/2) + T(n-1) + 1$$
$$\geq 2\left(\frac{n}{2}\right)^9 + \left(\frac{n-1}{2}\right)^9 + 1$$

SCANPIX



(Q6) Mínimo (Veja Q1 - 2004/1) Pag 83

$T(n) = 1$ se $n=1$

$T(n) = T(n-1) + 1/n$ se $n > 1$

$$T(n) = \frac{1}{n} + \frac{1}{(n-1)} + \ldots + \frac{1}{2} + 1$$
$$= \lg n + \Theta(1)$$

- Proposição: $T(n) = \lg n + 1$
- Prova por indução em n, para $n = 1, 2, \ldots$
  - Base: $n=1$, então

$$T(1) = 1 \quad \text{(verdadeiro)}$$

  - Passo: $n > 1$, então

$$T(n) = T(n-1) + 1/n$$
$$= \left[\lg(n-1) + 1\right] + \frac{1}{n} = \underbrace{\lg(n-1) + \frac{1}{n}}_{\lg n} + 1$$
$$= \lg n + 1$$

C.Q.D.

(Q7) BUILD HEAP (HEAP MÍNIMO)

```
1  A[i] ∈ A[n]
2  se i = 1 ou A[i] ≥ A[⌊i/2⌋]
3      então MIN-HEAPFY(A, n, i)
4      senão enquanto i > 1 e A[⌊i/2⌋] > A[i] faça
5          A[i] ↔ A[⌊i/2⌋]
6          i ← ⌊i/2⌋
```

SCANPIX

2001/1

(Q1) Recorrência $f(1) = 0$

$f(2n+1) = f(2n) = f(n) + \lg n$ , para $n > 1$

| $i$ | $f(i)$ |
|---|---|
| 1 | 0 |
| 2 | $0 + \lg 1$ |
| 3 | $0 + \lg 1$ |
| 4 | $\lg 1 + \lg 2 = \lg(1.2)$ |
| 5 | $\lg 1 + \lg 2 = \lg(1.2)$ |
| 6 | $\lg 1 + \lg(3) = \lg(1.3)$ |
| 7 | $\lg + \lg 3 = \lg(1.3)$ |
| 8 | $\ldots = \lg(1.2.4)$ |
| 9 | $\ldots = \lg(1.2.4)$ |

$\leq \lg(1.2)$ ... $\leq \lg(1.2)$ ... $\leq \lg(1.3)$ ... $\leq \lg(1.2.4)$ ... $\leq \lg\left(\frac{n}{2}\right) + \lg\left(\frac{n}{2}\right) + \lg\left(\frac{n}{2}\right) = O(\lg^2 n)$

($\lg\left(\frac{n}{2}\right)$ vezes)

Outro limite não tão justo:

Logo $f(n) \leq \lg\left(\left(\frac{n}{2}\right)!\right)$ Sabemos que $n^{n/2} \leq n! \leq n^n$

$\leq \left(\frac{n}{2}\right).\lg\left(\frac{n}{2}\right) = \frac{n}{2}\left[\lg n - 1\right] \leq n \lg n$ $\Big\}$ $\lg(n!) \leq n \lg n$

$f(n) = O(n \lg n)$

Portanto a opção (d) é o melhor limitante superior

(Q2)

| | T | Inserção | Remoção | ocla mais próximo | Construção |
|---|---|---|---|---|---|
| a) vetor ordenado | $O(n)$ | $O(n)$ | $O(n)$ | $O(\lg n)$ | $\downarrow$ |
| b) vetor não ordenado | $O(n)$ | $O(1)$ | $O(n)$ | $O(n)$ | |
| c) heap | $O(n \lg n)$ | $O(\lg n)$ | $O(\lg n)$ | $O(n \lg n)$ | $O(n)$ |
| (d) árvore balanceada binária | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ | $O(n \lg n)$ |
| e) hashing | $O(n)$ | $O(n)$ | $O(n)$ | $O(1)$ | |

A árvore balanceada minimiza T.



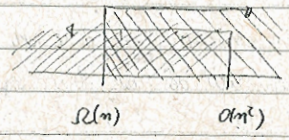


(Q3) Árvore rubro-preta. $2^{k+1} - 1$ ? Dado que é uma árvore binária completa

(Q4) Consumo de tempo

Algoritmo A. $O(n^2)$

Algoritmo B $\Omega(n)$



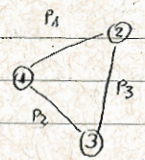

Não. Dado que as limitantes são diferentes, o fato de falar $\Omega(n)$ não garante que o algoritmo B seja melhor que A.

Tome $T_B(n) = n^3$ e $T_A(n) = n^2$ então o alg B tem maior consumo de tempo.

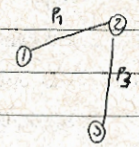
(Q5) Complexidade. Completamente errado. Se o algoritmo está em NP, estão de pertence a classe de problemas que podem ser verificáveis em tempo polinomial para a resposta SIM. Por outro lado, sabemos que $P \subseteq NP$. Então se o algoritmo está em NP, poderia se tratar de um algoritmo que está em P, portanto existe algoritmo polinomial para o problema.


(Q6) MST. O algoritmo guloso não garante a construção de uma árvore geradora mínima, pois arestas com pesos altos poderiam ser considerados na árvore

Veja um exemplo com 3 vértices:



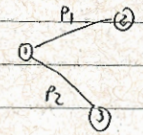

- com $P1 \leq P2 \leq P3$
- Suponha que a leitura dos vértices seja realizada em forma crescente (i.e, 1, 2, 3)




Árvore geradora. Solida

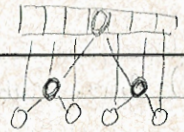
Peso = $P_1 + P_3$

$\geq$




Árvore geradora mínima

Peso = $P_1 + P_3$



(Q7) Árvore de busca binária

- Ordene os n chaves em O(n) [Transformando-as à base n e aplicando Radix sort]
- Construa a árvore de busca binária da seguinte maneira
  - O valor na posição $\lfloor \frac{n}{2} \rfloor$ deve ser inserido como raiz
  - O filho esquerdo da raiz deve ser o elemento $\lfloor n/2^2 \rfloor$
  - " " direito " " " " " " $\lfloor 3n/2^2 \rfloor$

(Q8) Heap Mínimo

Num Min-Heap deve-se sempre que $A[i] \leq A[2i]$
e $A[i] \leq A[2i+1]$ para todo $i$

Prova: $A[i] = \lg\left(\frac{i}{i+1}\right)$ <

$= \lg(i) - \lg(i+1)$

$A[i] < \lg(i) - \lg(i+1/2) = \lg\left(\frac{i}{i+1/2}\right) = \lg\left(\frac{2i}{2i+1}\right) = A[2i]$

(Prova similar para $A[i] \leq A[2i+1]$)

- Como a função logarítmica é crescente, então $A[i] \leq A[i+1]$, para todo $i \geq 1$, portanto $A[i] \leq A[i+1]$ e $A[i] \leq A[2i+1]$

(Q9) Segmento crescente de comprimento máximo.

1





```
Algoritmo 1te (A, n)
1  comp ← 1
2. compMax ← 1
3. para i ← 2 até n faça
4.       se A[i-1] < A[i]
5.             então comp ← comp + 1
6.             senão comp ← 1
7.       se comp > compMax
8.             então compMax ← comp
9. devolva compMax
```

✓ inicialmente 0

```
compMaxREC (A, n, max)
1. se n = 1
2.    então devolva 1
3. se n > 1
4.    então comp ← compMaxREC (A, n-1, max)   ? ruim.
5.          se A[n-1] < A[n]
6.             então se comp > max
7.                   então devolva comp
8.                   senão devolva 1
```

(Q10) Aluguel de esquis: Alocação de esquis

Se n = m então é suficiente ordenar a lista de esquis e a lista de esquiadores.

esquis: $s_1 \leq s_2 \leq s_3 \leq \ldots \leq s_n$

Esquiadores: $h_1 \leq h_2 \leq h_3 \leq \ldots \leq h_3$

para posteriormente atribuir esquiador(i) ← esqui(i) para todo $1 \leq i \leq n$

- complexidade $O(2n\lg n + n) = O(n \lg n)$ (ordenação)
- Prova de corretude?

Sejam $m$ esquis e $n$ esquiadores. A solução pede $p(i) \leftarrow j$ (j do esqui, i esquiador) estando em ordem (crescente ou decrescente)

- Suponha que $p$ é uma solução ótima que contem uma inversão, i.e., existe $i$ e $j$ tal que $p(i) > p(j)$, entao existe (pelo menos) uma inversão com $j = i+1$

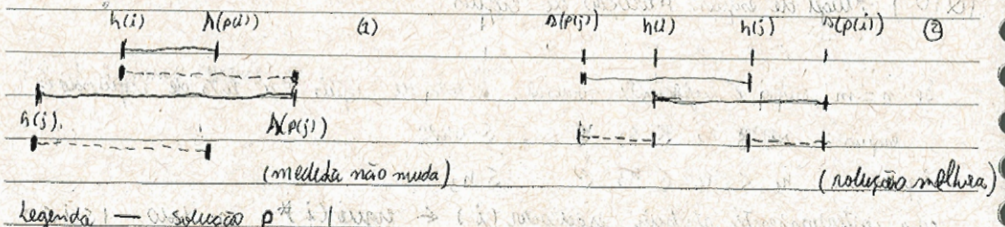
- Vamos trocar esquiadores $i$ e $j$ seus esquis, tendo uma solução $p^*$
  $p^*(i) \leftarrow j$
  $p^*(j) \leftarrow i$
  e para todo $k \in \{1..n\} \setminus \{i, j\}$ temos $p^*(k) \leftarrow k$

- Será que $p^*$ é uma solução correta? SIM
  Ainda é mantida uma atribuição de esquis

- Será que $p^*$ é tão boa quanto $p$?
  Como para os esquiadores diferentes de $i$ e $j$ não mudou a atribuição, então vamos somente analisar a diferença (dos esquis) entre $i$:

$$|h(i) - s(p(i))| + |h(j) - s(p(j))| \quad \text{vs} \quad |h(i) - s(p(j))| + |h(j) - s(p(i))|$$

Existem $c(\frac{4}{2}) = 6$ casos diferentes (dependendo da ordem relativa dos 4 posições)




Legenda — solução $p^*$
--- solução $p^*$

Fazendo isso para todos os casos vemos que a solução nova $p^*$ é tão boa quanto $p$. Portanto fazendo esse processo para todas as inversões observaremos que a atribuição ao esqui $i$ para o esquiador $i$ é ótima.





(# esquis, # esquiadores)

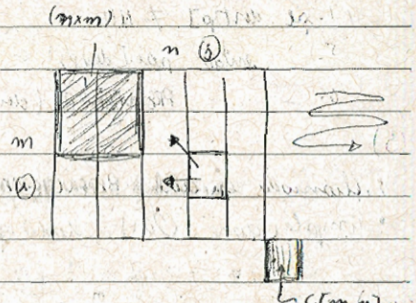
Se $m > n$ então o problema se torna em um de P.D. (misturado com a escolha gulosa mostrada.

Recorrência: Seja $c[i,j]$ o valor ótimo (da soma dos valores absolutos) ao atribuir os $i$ esquis ao conjunto $\{1, \ldots, j\}$ de esquiadores

$$c[i,j] = \begin{cases} \infty & \text{, se } i = 0 \\ \text{Atribua-se em ordem crescente (sol anterior)} & \text{, se } i = j \\ \min\{\, c[i-1, j] \;,\; c[i-1, j-1] + |h(i) - s(j)| \,\} & \text{, se } i > j \end{cases}$$

(ele não tá) (ele tá na solução)

devolva $c[m,n]$




(Q11) slavsoa!

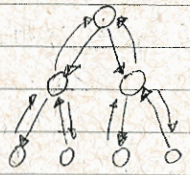
- Prova da corretude por indução
- $T(n) = \Omega\left(n^{(\ln n)/(2+\varepsilon)}\right)$ em que $\varepsilon > 0$ dado por Jorge Stolfi.

2001/02

(Q1) $O(n \lg \sqrt{n})$

Sim é possível dado que $O(n \lg(n^{1/2})) = O(n \lg n)$ que ainda é uma limitante para a ordenação baseada em comparações, como o Quicksort "assintoticamente boca"

(Q2) Árvores binárias:



(a) PREENCHE-PAI ( p )

1. se esq[p] ≠ NIL
2. então pai[esq[p]] ← p
3. PREENCHE-PAI (esq[p])
4. se dir[p] ≠ NIL
5. então pai[dir[p]] ← p
6. PREENCHE-PAI (dir[p])

(b)

- Chamada inicial: PREENCHE-PAI (raiz)
- complexidade $O(n)$, dado que percorre 1 vez todo nó.

(Q3) Multiplicação de $n^m$ em $2 \lg n$

base exp

MULT ( b, n )

1. se n = 0
2. então devolva 1
3. se n = 1
4. então devolva b
5. se n > 1
6. então p ← MULT (b, ⌊n/2⌋)
7. se ⌊n/2⌋ = n/2 ▶ se é divisível por 2
8. então p ← p * p
9. senão p ← p ⊗ p ⊗ b
10. devolva p

$T(n) \leq T(n/2) + 2$

$T(n) \leq \sum_{i=0}^{\lg n} 2 = 2 \lg n$



+ Eficiente

(Q4) (a) $k = \Theta(2^n)$, RadixSort $\Theta(n^2)$ HeapSort

(b) $k = \Theta(2^{(\lg n)^2})$, RadixSort $\Theta(n \lg^2 n)$ "

(c) $k = \Theta(n^3)$, RadixSort $\Theta(n)$ RadixSort

(Q5) comb 1: $\geq 1 + 2T(n-1, n)$

$T(m,n) = 1 + T(m-1, n) + T(m, n-1)$, se $m \geq 1$, $n \geq 1$

$T(m,n) = 1$, se $m = 1$ ou $n = 1$

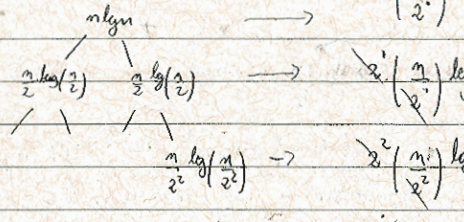
$T(m,n) = \Omega\left(2^{\min\{m,n\}}\right)$ [polinomial]

comb 2: $T(m,n) = \Theta(mn)$ [exponencial]

(Q6) $T(1) = 1$

$T(n) = 2T(n/2) + n \lg n$

- árvore de recorrência:




$2^0 \left(\frac{n}{2^0}\right) \lg\left(\frac{n}{2^0}\right) = n \lg n - 0n$

$2^1 \left(\frac{n}{2^1}\right) \lg\left(\frac{n}{2^1}\right) = n \lg n - 1n$

$2^2 \left(\frac{n}{2^2}\right) \lg\left(\frac{n}{2^2}\right) = n \lg n - 2n$

no último nível $n = 2^i$, $i = \lg n$

$2^i \left(\frac{n}{2^i}\right) \lg\left(\frac{n}{2^i}\right) = n \lg n - ni$

$T(n) = \sum_{i=0}^{(\lg n - 1)} [n \lg n - ni] + 2^{\lg n}$

$\leq n \lg^2 n - n \sum_{i=0}^{(\lg n - 1)} i + n = n \lg^2 n - n\left(\frac{1}{2}\right)(\lg n - 1) \lg n + n$

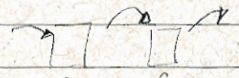
$\leq n \lg^2 n - \frac{1}{2}\left[n \lg^2 n - n \lg n\right] + n = \frac{1}{2} n \lg^2 n + \frac{n}{2} \lg n + n$

$T(n) = O(n \lg^2 n)$ Prove por indução! Portanto não é $O(n \lg n)$

(Q7) A ⊆ B embutível. $A \rightarrow B$

EMBUTIDO ( A, B )
1. A' ← Ordena A em ordem raiz - esq - dir
2. B' ← Ordena B em ordem raiz - esq - dir
3. count ← 0
4. j ← 1
5. para i ← 1 até |A'|
6. enquanto A'[i] ≠ B'[j] faça
7. j ← j+1
8. se j ≤ |B'| e A'[i] = B'[j]
9. então count ← count + 1
10. j ← j+1
11. se count = |A'|
12. então devolva "SIM" complexidade $O(|A|+|B|)$
13. senão devolva "NÃO" linear

(Q8) Fila usando 2 pilhas (E, D)




INSERE (x)
1. push (D, x)

EXTRAI ( )
1. se EMPTY(E) ▸ Se E está vazio
2. então enquanto não EMPTY(D) faça
3. push (E, pop(D))
4. devolva pop(E)

Método contábil: Para cada operação de inserção pague 2 reais. Um real usado no empilhamento em D e guarde um real de crédito para o empilhamento do mesmo elemento em E.

Portanto cada operação terá um custo amortizado de 2 ou $O(1)$, sendo que



2002/1

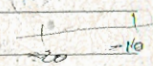
(Q1) Veja Q3 Prova 2006/1 (Pág 27)

(Q2)
P1: Encontrar árvore geradora T que minimize $c_1(T)$ É custo
P2: " " " " " " $c_2(T)$ maior ∀ custo: $uv \in T$ ∀

SIM: É verdade que P1 envolve P2.
NÃO. é verdade que P2 é solução de P1.

Para P2 use-se o algoritmo de Kruskal ou de Prim para achar a MST.
$O(V + E \lg E)$

(Q3) (1) SIM
(2) verdade se P = NP
(3) falso
(4) verdadeiro se P ≠ NP
(5) Não se sabe ainda. Verdadeiro se P = NP




(Q4) Segmento ALGORITHO (A, n) ▸ Segmento de soma mínima.
1 - soma ← 0 somaMin ← 0
2 - e ← 0 d ← -1 i ← 1
3 - para f ← 1 até n faça
4 - se soma > 0
5 - então i ← f
6 - soma ← A[f]
7 - senão soma ← soma + A[f]
8 - se soma < somaMin
9 - então somaMin ← soma
10 - e ← i d ← f
11 - devolva (e, d, somaMin)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -1 | 10 | -10 | -20 | 10 | 1 | |

f = ~~1 2 3 4 5 6~~ 7
i = ~~1 2~~ 3
e = ~~0 1~~ 3
d = ~~-1 2~~ 5
soma = ~~0 -1 10 0 -20~~ -10
somaMin = ~~0 -1 -10~~ -20

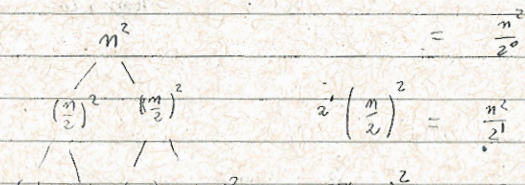
(Q5) $T(1) = 1$

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n^2)$

Aproximadamente: $T(1) = 1$

$T(n) = 2T(n/2) + n^2$

• árvore de recorrência



$n^2 = \frac{n^2}{2^0}$

$2^1\left(\frac{n}{2}\right)^2 = \frac{n^2}{2^1}$

$2^2\left(\frac{n}{2^2}\right)^2 = \frac{n^2}{2^2}$

até $n = 2^i$, $i = \lg n$ ... $2^i\left(\frac{n}{2^i}\right)^2 = \frac{n^2}{2^i}$

$$T(n) = \sum_{i=0}^{(\lg n)+1} n^2\left(\frac{1}{2}\right)^i + n$$

$$= n^2\left[\frac{\left(\frac{1}{2}\right)^{\lg n} - 1}{\frac{1}{2} - 1}\right] + n = n^2\left[-2\left(\frac{1}{n} - \frac{n}{n}\right)\right] + n$$

$$= n^2\left[\frac{-2}{n}\left(1 - n\right)\right] + n = -2n(1-n) + n = n + 2n^2 - 2n = O(n^2)$$

• Prova por indução em $n$

base: $n = 1$ (verdadeiro)

Passo: $n > 1$ $\quad T(n) = 2T(n/2) + n^2$

$$= 2\left[\frac{n}{2} + 2\left(\frac{n}{2}\right)^2 - 2\left(\frac{n}{2}\right)\right] + n^2$$

$$= n + n^2 - 2n + n^2 \qquad T(n) = O(n^2)$$

$$= n + 2n^2 - 2n \quad \text{c.q.d.}$$

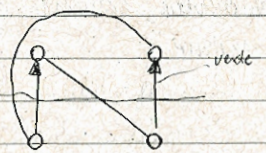


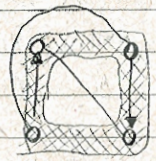


(Q6) $F_1$: $T(x,y) = \Omega(2^{\min\{x,y\}})$ (Exponencial)

$F_2$: $T(x,y) = O(xy)$ (Técnica de P.D.)

(Q7) Grafo Tricolor



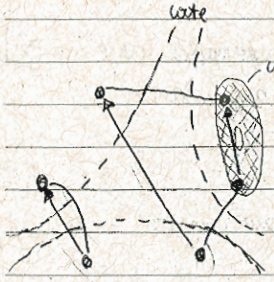



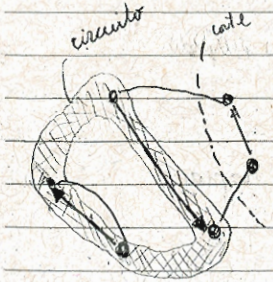
corte: Verde - vermelho

- Não tem arcos amarelos
- arcos verdes tem mesma orientação

circuito verde - amarelo (não nec. aresta

- não tem arcos vermelhos
- arcos verdes orientados consistentemente (no circuito)




↗ aresta verde

```
CORTE-AO-CIRCUITO ( G, (a,b) )
1   y ← a   j ← 1   V[j] ← a
2   enquanto y ≠ b faça                          é amarelo
3   :  { seja x o vértice vizinho de b tal que (b,x) ou (x,b)
4   amarelo { j ← j+1
5   :       { V[j] ← x
6   :  { se existe aresta verde (x,y) verde, para y ≠ x
7   verde { então j ← j+1
8   :     { V[j] ← y   b ← y
9   :  Se existe aresta verde (y,x) verde
10  :  então devolva (V, "corte")
11  devolva (V, "circuito")
```

Ideia: "verde - amarelo"

(Q8) Veja Q2. 2007/2 Pág 104.

2002/01 $n = r - p + 1$

(Q1) $T(n) = 0$ se $n = 1$

$T(n) = T(n-1) + 1$

$T(n) = n - 1 = \Theta(n)$

(Q2) Falso. BUILDHEAP como montado no CLRS consome $O(n)$. Mesmo sendo baseado em comparações, ele não ordena todo o vetor, somente rearranja de tal modo que $A[i] \geq A[2i]$ e $A[i] \geq A[2i+1]$ no MAXHEAP

(Q3) Se Co-NP $\neq$ NP então P $\neq$ NP

Sabemos que $P \subseteq NP \cap$ Co-NP

Então Se Co-NP $\neq$ NP sabemos que P $\neq$ NP

(Q4) Não podemos concluir, de forma geral, qual das funções cresce mais rápido. A resposta será dada para casos particulares.

Seja $f(n) = n$, $g(n) = n$ { então ambos crescem igualmente

Seja $f(n) = n$, $g(n) = 2$ { então $f(n)$ cresce mais rápido

Seja $f(n) = 2$, $g(n) = n^2$ { então $g(n)$ cresce mais rápido

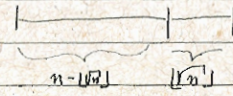




(Q5) Use-se o algoritmo SELECT-BFPRT de Blum e colaboradores (1973) que permite achar o $i$-ésimo menor elemento em tempo linear do tamanho da entrada. Logo imprima-se os $\lfloor \sqrt{n} \rfloor$ maiores elementos

```
ALGORITMO ( A, n)
1. x ← SELECT-BFPRT(A, n, n - ⌊√n⌋)
2. PARTICIONE( A, n, x )   ▷ Particiona A usando x como pivô
3. para i ← n - ⌊√n⌋ até n faça
4.       imprimir A[i]
```

O algoritmo funciona devido a que não se ordena o vetor inteiro, e sem se particiona o vetor A



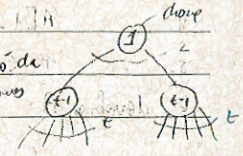

Consumo de tempo $O(n + n + n) = O(n)$

(Q6) Sabemos que numa árvore B de $n$ elementos e $t$ chaves a altura é

$$h \leq \log_t \left( \frac{n+1}{2} \right)$$

Portanto a altura deve ser

$$\left[ \log_{200}\left(\frac{n+1}{2}\right), \log_{100}\left(\frac{n+1}{2}\right) \right]$$




$$n \geq 1 + \sum_{i=0}^{h-1} 2t^i (t-1)$$

$$n \geq 1 + 2t^h - 2 \quad (t-1)$$

$$\frac{n+1}{2} \geq t^h$$

$$h \leq \log_t \left( \frac{n+1}{2} \right)$$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | $2t$ |
| 4 | $2t^2$ |
| 5 | $2t^3$ |
| ⋮ | |
| $h$ | $2t^{h-1}$ |

(Q7) MERGE DE VETORES

custo ao Agrupar de $A_0 \to A_k$

| | | |
|---|---|---|
| □ | $2^0$ | — |
| □□ | $2^1$ | $(2^0)+2^1$ |
| □□□□ | $2^2$ | $(2^0+2^1)+2^2$ |
| □□□□□□□□ | $2^3$ | $(2^0+2^1+2^2)+2^3$ |

$n = 2^{k+1}-1$

$n+1 = 2^{k+1}$

$\lg(n+1) = k+1$

$\lg(n+1)-1 = k$

Em cada nível (iteração) $2^{i+1}-1$

No total temos $\sum_{i=1}^{k}(2^{i+1}-1) = 2\sum_{i=1}^{k}(2^i-1)$

$= 2\left(2^{k+1}-1\right) - k = 2n-k$

$= 2n - \lg(n+1) + 1$

$= \Theta(n)$

▶ Algoritmo que recebe uma sequência de k vetores e os agrupa

AGRUPA ( $A_0, A_1, A_2, \ldots, A_k$ )

1. $A[1..1] \leftarrow A_0[1..1]$
2. $m \leftarrow 1$
3. para $i \leftarrow 1$ até $k$ faça
4. $\quad A[1..m+2^i] \leftarrow$ INTERCALA ( $A[1..m], A_i[1..2^i]$ )
5. $\quad m \leftarrow m+2^i$
6. devolva $A$

▶ Intercala 2 vetores em tempo linear, dado que ambos estão em forma crescente.

Complexidade $T(n) = \Theta(n)$

(Q8) Veja Pág 183 a análise amortizada.

fim das provas 06/08/2008