de X em Y então existe uma subsequência palíndromo de Y de maior comprimento.

Prova: Se existe uma ssco de X em Y isto quer dizer que existem índices $i_1 < i_2 .. < i_m$ tais que $X[j] = Y[i_j]$ para $j = 1, \dots, m$ e como X é Y invertido, nada mais é que a maior subsequência com índices $i_m < i_2 < i_1$ tais que $X[j] = Y[i_j]$ ou seja pode ser lida da esquerda p/ direita ou da direita para a esquerda, que é a definição de palíndromo.

Subestrutura ótima: Suponha Z ssco máxima de $X[1..n]$ e $Y[1..n]$

(1) Seja $X[n] = Y[n]$ então $Z[K] = X[n] = Y[n]$ e $Z[1..K-1]$ é ssco máxima de $X[1..n-1]$ e $Y[1..n-1]$

(2) Seja $X[n] \neq Y[n]$ então $Z[K] = X[n]$ e $Z[K] \neq Y[n]$ implica que $Z[1..K]$ é ssco máxima de $X[1..n]$ e de $Y[1..n-1]$

(3) Seja $X[n] \neq Y[n]$ então $Z[K] = Y[n]$ e $Z[K] \neq X[n]$ implica que $Z[1..K]$ é ssco de $X[1..n-1]$ e de $Y[1..n]$

Prova: (1) Suponha que Z^*_{k-1} não é ssco máxima de comprimento $K-1$ de X_{n-1} e Y_{n-1}, então existe uma outra ssco máxima de comprimento maior de $K-1$, tal que adicionada ao elemento n é melhor que Z^* o que é uma contradição.

(2) Suponha que Z^*_K não é ssco máxima de comprimento K de X_n e Y_{n-1}, então existe uma outra solução de comprimento maior que K para X_n e Y_{n-1} ou seja de comprimento maior que K, uma contradição

(3) simétrico a 2

Prova 2006/2

Questão 1 - Para cada inteiro positivo i, sejam $f_i(x)$ e $g_i(x)$ funções tais que $f_i(x) = O(g_i(x))$. Defina as funções $F(n)$ e $G(n)$ por $F(n) = \sum_{i=1}^{n} f_i(n)$ e $G(n) = \sum_{i=1}^{n} g_i(n)$. É verdade que $F(n) = O(G(n))$?

Para $i = 1, 2, 3 \dots$ existe uma constante $c > 0$ e uma constante $x_0 > 0$ tal que $f_i(x) \leq c.g_i(x)$ para $\forall\, x \geq x_0$

Como $F(n) = \sum_{i=1}^{n} f_i(n)$
$= f_1(n) + f_2(n) + f_3(n) + \dots f_n(n)$ e

$G(n) = \sum_{i=1}^{n} g_i(n)$
$= g_1(n) + g_2(n) + g_3(n) + \dots g_n(n)$

$$\boxed{\begin{array}{l} F(n) \leq \max_{1 \leq i \leq n} \{f_i(n)\} \\ G(n) \leq \max_{1 \leq i \leq n} \{g_i(n)\} \end{array}}$$

$$\boxed{\begin{array}{l} \text{Como } f_1(n) \leq c.g_1(n) \\ f_2(n) \leq c.g_2(n) \\ \vdots \\ \max f_i(n) \leq \max g_i(n) \end{array}}$$

$F(n) = \sum_{i=1}^{n} f_i(n) \leq \max_{1 \leq i \leq n} \{f_i(n)\}$ e $G(n) = \sum_{i=1}^{n} g_i(n) \leq \max_{1 \leq i \leq n} \{g_i(n)\}$

$F(n) \leq \max_{1 \leq i \leq n} \{f_i(n)\} \leq \max_{1 \leq i \leq n} \{g_i(n)\}$

$F(n) \leq \underset{c}{1}.\max_{1 \leq i \leq n} \{g_i(n)\}$ para $\forall\, n > n_0$

$F(n) \leq c.G(n) = F(n) = O(G(n))$ //

/ /

Questão 2 - Considere o seguinte problema:

ENTRADA: inteiros positivos n e K

SAÍDA: n^K.

1- Qual o tamanho da entrada como função de n e K?

2- Considere o seguinte algoritmo para resolver o problema:

produto := 1

para i := 1 até K faça — $O(K)$

produto := produto · n; — $\Theta(\log(n^{K-1}) \cdot \log(n)) = K-1 \log n \cdot \log n$ $K-1 \cdot \log^2 n$

Assuma que o algoritmo da escola primária é usado para fazer as multiplicações. Para multiplicar um número com x dígitos por um número com y dígitos o algoritmo da escola primária usa $\Theta(x \cdot y)$ passos. Estime da melhor forma possível o tempo de processamento do algoritmo acima.

3- Explique porque o algoritmo não é limitado polinomialmente no tamanho da entrada.

1- O tamanho é $(\lceil \log(n+1) \rceil, \lceil \log(K+1) \rceil)$

2- linha

1 - $O(1)$

2 - $O(\log K+1))$

3 -

Linha 3

$\sum_{i=1}^{K} \Theta(\log n^{i-1} \cdot \log n)$

$= \Theta(\log n \sum_{i=1}^{K} (i-1) \log n$

$= \Theta(\log^2 n \sum_{i=1}^{K} (i-1))$

$= \Theta(\log^2 n \cdot [\sum_{i=0}^{K} i - \sum_{i=1}^{K} 1$

$= \Theta(\log^2 n \cdot [\frac{K(1+K)}{2} - K)$

$= \Theta \log^2 n \cdot \frac{K^2}{2}$ //

3- pois n e K não expressam o real tamanho dos valores. Caso $n = 10$ ou 1000 fará diferença no tempo de consumo do algoritmo.

/ /

Questão 3 - Considere a sequência $T(n)$ tal que:

$T(0) = 0$

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

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

Para simplificar:

$T(0) = 0$

$T(n) \leq T(n - \sqrt{n}) + 1$

Vou provar por indução em n que $T(n) = \Theta(\sqrt{n})$

Teorema: $T(n) \leq 2\sqrt{n}$ para $n \geq 0$

Prova:

base: $n = 0$

$T(0) \leq 0 = 2 \cdot \sqrt{0} = 0$ //

passo: $n > 1, 2, 3 \ldots$

H.I: $T(n - \sqrt{n}) \leq 2\sqrt{n - \sqrt{n}}$

$$\begin{aligned}
T(n) &\leq T(n - \sqrt{n}) + 1 \\
&\overset{H.I}{\leq} 2\sqrt{n - \sqrt{n}} + 1 \\
&\leq 2\sqrt{n - \sqrt{n} + \tfrac{1}{4}} + 1 \\
&\leq 2\sqrt{n - \tfrac{1}{2}\sqrt{n} + \tfrac{1}{4}} + 1 \\
&\leq 2\sqrt{(\sqrt{n} - 1/2)^2} + 1 \\
&= 2(\sqrt{n} - 1/2) + 1 \\
&= 2\sqrt{n} - 1 + 1 \\
&= 2\sqrt{n} \text{ //}
\end{aligned}$$

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

Teorema: $T(n) \geq \frac{1}{2}\sqrt{n}$ para $n \geq 0$

Prova por indução em n

base: $n = 0$

$T(0) = 0 \geq \frac{1}{2}\sqrt{0} \geq 0$

passo: $n \geq 1, 2, 3.$

H.I.: $T(n - \sqrt{n}) \geq \frac{1}{2}\sqrt{n - \sqrt{n}}$

$\frac{\sqrt{2}}{2}$

$T(n) \overset{H.I}{\geq} \frac{1}{2}\sqrt{n - \sqrt{n} - n} + 1$

$= \frac{1}{2}\sqrt{n} - \frac{1}{2}n + 1$

$\geq \frac{1}{2}\sqrt{n} - \frac{1}{2}n + n$ para $n \geq 1$

$\geq \frac{1}{2}\sqrt{n} - \frac{1}{2}n + \frac{1}{2}n = \frac{1}{2}\sqrt{n}$ //

Questão 4 - P=NP se e só se algum problema em P é NP-completo. Essa afirmativa é verdadeira ou falsa? Justifique

Verdadeira. Se P fosse NP significaria que qualquer algoritmo teria uma solução polinomial, ou seja, todos os problemas poderiam ser reduzidos a um problema em P, sendo esse problema em P também portanto NP-completo.

Por outro lado se um problema em P fosse NP-completo significaria que todos os problemas se reduziriam a um problema em P tendo portanto solução polinomial e dessa forma P=NP.

Questão 5 - pg

Questão 6 - Considere a sequência $T(n)$ tal que:

$T(1) = 1$

$T(n) = 2T(\lfloor n/2 \rfloor) + \lfloor n \lg n \rfloor$. Mostre que $T(n)$ não é $O(n \lg n)$

$T(1) = 1$

$T(n) \le 2T(n/2) + n \lg n$

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

$= 2^2(2T(n/8) + n/4 \lg n/4) + n \lg n/2 + n \lg n/2$

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

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

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

$= n + n \cdot \sum_{k=0}^{i-1} \lg n - n \cdot \sum_{k=0}^{i-1} k \cdot \lg 2$

$= n + n \cdot \lg n \cdot \lg n - n \sum_{k=0}^{i-1} k$

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

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

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

$\frac{n}{2^i} = 1$

$n = 2^i$

$\boxed{i = \lg n}$

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

Vou provar que $T(n) \le \frac{1}{2} n \lg^2 n + n + \frac{1}{2} n \lg n$

Prova por indução em n:

base: $n = 1$

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

passo: $n \ge 2^1, 2^2 \ldots$

$T(n) \le 2T(n/2) + n \lg n$

$\overset{H.I}{\le} 2 \cdot \left(\frac{1}{2}\frac{n}{2} \lg^2 n/2 + n/2 + \frac{1}{2} \cdot \frac{n}{2} \cdot \lg n/2\right) + n \lg n$

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

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

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

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

Assim $T(n) = O(n \lg^2 n)$ e não $O(n \lg n)$ //

Questão 7 - Para uma árvore de busca binária (ABB) e um item x armazenado nela, seja $N(x)$ o nó da árvore contendo x. Dizemos que uma ABB A é embutível em uma ABB B se:

- Todo item armazenado em A também está em B, e
- Para todos os pares x, y de itens, se $N(x)$ é descendente de $N(y)$ em A, então $N(x)$ é descendente de $N(y)$ em B.

Dê um algoritmo que, dadas A e B, decide em tempo linear se A é embutível em B.

O algoritmo EH_EMBUTIVEL recebe duas árvores binárias A e B e devolve 0 caso A seja embutível em B e >0 caso contrário.

/ /

EH-EMBUTÍVEL(A, B, raizA)

```
Se raizA ≠ NIL
  então x ← BUSCA(B, N(raizA))
    se x=-1 então devolve Não
    Resp ← VERIFICA-EMBUTIDO(A, B, raizA, x)
    se Resp > 0
       então devolva Não
       senão devolva SIM
  senão devolva SIM
```

VERIFICA-EMBUTIDO(A, B, raizA, x)

```
se raizA ≠ NIL e x = NIL
    então devolva 1
se raizA ≠ NIL
    então e ← VERIFICA-EMBUTIDO(A, B, esq[raizA], esq[x])
        se e≠0
            então devolva e
            senão se A[raizA] ≠ B[x]
                então devolva e+1
        e ← VERIFICA-EMBUTIDO(A, B, dir[raizA], dir[x])
        se e≠0
            então devolva e
devolva 0.
```

O algoritmo EH_EMBUTIDO utiliza duas subrotinas. BUSCA que implementa uma busca binária simples que recebe uma árvore binária e o valor ou chave a ser procurado e possui um consumo de $O(\lg m)$ sendo m nº de elementos de B. E utiliza tb o algoritmo VERIFICA-EMBUTIDO que recebe duas árvores e suas respectivas raiz e nó e devolve ≥ 1 se A não confere com B e $= 0$ se confere. Esta subrotina tem tempo $O(n)$ onde n.

NOME:

E-MAIL

MATÉRIA

PROFESSOR

SALA

HORÁRIOS

HORÁRIO	SEGUNDA	TERÇA	QUARTA	QUINTA	SEXTA	SÁBADO

PROVAS/NOTAS		TRABALHOS	

ANOTAÇÕES

/ /

é o nº de nós de A.

linha

1 O(1) ?

2 O(lg m)

3 O(n)

4-7 O(1)

tamanho da instância: (n, m)

consumo total O(n + lg m)

corretude: Invariante em todas as chamadas VERIFICA-EMBUTIDO - raiz A é raiz de uma subárvore em A.

Início: No algoritmo EH-EMBUTIDO é feita uma busca em B pelo índice correspondente a raiz de A na primeira chamada de VERIFICA-EMBUTIDO. Todos os nós abaixo da raiz de A serão comparados com B.

Manutenção: No algoritmo VERIFICA-EMBUTIDO recursivamente faz-se conferências nas subárvores esquerda e direita sucessivamente; as operações esq e dir garantem a manutenção.

Término: Ao chegar a folhas da subárvore esquerda e direita o algoritmo volta à raiz com todos os nós conferidos com B e devolve > 0 ou = 0 devolvendo a resposta nas linhas 4-7.

/ /

Questão 8 - Uma ordem parcial em um conjunto é uma relação binária, denotada usualmente por $\prec$, tal que (i) $a \nprec a$ para cada a do conjunto e (ii) se a, b e c são tais que $a \prec b$ e $b \prec c$, então $a \prec c$. Se a, b são tais que nem $a \prec b$ nem $b \prec a$, eles são incomparáveis. Um vetor $A[1..n]$ de elementos dessa ordem estende a ordem se, para todo i, j, se $A[i] \prec A[j]$, então $i < j$.

Suponha definido um tipo Op e dada uma função compara (Op x, Op y), que devolve:

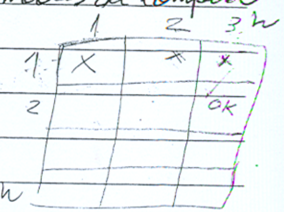
'<' se $x \prec y$; '>' se $y \prec x$; '|' se x e y são incomparáveis

Considere o problema: dado um vetor $A[1..n]$ do tipo Op, reordenar esse vetor de modo a estender a ordem.

Mostre que qualquer algoritmo que resolva esse problema usa $\Omega(n^2)$ chamadas de compara, no pior caso.

Descreva um algoritmo que faz $O(n^2)$ chamadas de compara.

2) REORDENA



```
Para i <- 1 até n-1
    Para j <- i até n
        Se compara(A[i], A[j]) = '<'
            Se i > j
                A[i] <-> A[j]
        Se = '>'
            se i < j
                A[i] <-> A[j]
    A[i] <- A[n]
```

melhorar.

1- Porque os algoritmos de comparação $n \lg n$ não fazem comparação 2 a 2 e para fazer este tipo de comparação são necessários no mínimo n^2 chamadas de compara que supomos tem consumo O(1).

Consumo:

Linha — Tamanho da instância: n — / /

1 $O(n)$

2-8 $O(n^2)$

9 $O(1)$

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

Corretude:

Invariante no início da linha 1 $A[1..i]$ está ordenado

Início: Com $i = 1$ o vetor $A[1..0]$ está ordenado de forma trivial ↑ na realidade vazio

Manutenção: A manutenção é feita entre as linhas 4-8, ou seja, caso na chamada do compara indique < ou > é feita a mudança de valores para que se mantenha ordenado caso sejam incomparáveis não se modifica nada mantendo a ordenação para a próxima iteração. O segundo loop percorre todos os elementos que ainda não foram comparados com $A[i]$ os de $A[i+1]$ até $A[n]$.

Término: Com $i=n$ significa que $A[1..n-1]$ está totalmente ordenado e que só falta a última troca: de $A[n-1] \leftrightarrow A[n]$ que ocorre na linha 9 finalizando a ordenação.

10

Questão 9 - Dado um vetor $A[1..n]$ de strings distintas, definimos, para $i=1,\dots,n$

$p(i)$ = o nº de índices j tais que $A[j] \leq A[i]$ (na ordem lexicográfica) — $p(i)$ quantos são menores ou iguais que i

$\delta(i) = i - p(i)$ — $5 - 4 = 1$

Dê algoritmos baseados em comparações para, dado A, calcular:

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

2- $\bar{\delta}_A = \sum_{i=1}^{n} |\delta(i)|$ — Os algoritmos devem ser $o(n^2)$, e um deles deve ser muito mais rápido que o outro.

1ª tentativa — / /

~~1- Algoritmo1 (A)~~

~~1 Para $i \leftarrow 1$ até n~~ — $O(n)$

~~2 $q \leftarrow$ particione(A, p, r, i)~~ — $O(n) \cdot O(n) = O(n^2)$

~~3 $\delta \leftarrow i - q$~~

~~4 Soma ← Soma + δ~~

1- Se somarmos todos $\delta(i)$ $1 \leq i \leq n$ teremos sempre $S_A = n$, podemos adquirir em tempo $O(1)$

2- Algoritmo $(A, 1, n, B)$

1 se $p < r$

2 $q \leftarrow \lfloor (r+p)/2 \rfloor$

3 $x \leftarrow$ SELECT.BFPRT(A, p, r, q) ↑ pivô

4 PARTICIONE(A, B, p, r, x)

5 Algoritmo$(A, p, q-1)$

6 Algoritmo$(A, q+1, r)$

7 devolva B

O algoritmo Particione acompanhado da troca de valores $(A[i] \leftrightarrow A[j])$ deve executar sempre em seguida $(B[i] \leftrightarrow B[j])$ que contém os índices originais.

Algoritmo2 (A, n)

1 Soma ← 0

2 Para $i \leftarrow 1$ até n

3 faça $B[i] \leftarrow i$

4 Algoritmo$(A, 1, n, B)$

5 Para $i \leftarrow 1$ até n

6 faça Soma ← Soma + $|B[i] - i + 1|$

7 devolva Soma

O algoritmo2 tem tamanho de instância n e $T(n)$ como consumo de tempo total

Linha

1 $O(1)$ — 4- $O(n \lg n)$

2-3 $O(n)$ — 5-6 - $O(n)$

7 - $O(1)$

o algoritmo Algoritmo executa basicamente o Quick sort mas sempre tendo como pivô a mediana, que produz o seu melhor caso, ou seja $(n \lg n)$. A modificação no particione é de uma linha para troca de elementos em B que leva tempo constante. Ele também utiliza a rotina SELECT BFPRT que como já é sabido tem consumo linear.

No algoritmo 2 nas linhas 5-6 são calculadas as diferenças entre as posições originais e as posições ordenadas com acréscimo de 1 posição já que os índices são a partir de 1. Como queremos avaliar o complemento é feito o módulo deste valor, onde a soma total será S_A.

Prova 2006/1

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

Vou mostrar que $\frac{1}{2} n^3 \leq \sum_{i=1}^{n} i^2 \leq 2 \cdot n^3$

Teorema 1 $\sum_{i=1}^{n} i^2 \leq 2 \cdot n^3$ para $n \geq 1$

Prova:

$$\sum_{i=1}^{n} i^2 \leq \frac{n \cdot (1+n^2)}{2} = \frac{n^3+n}{2} \leq n^3+n \leq n^3+n^3 = 2n^3$$

$$1^2+2^2+3^2+4^2+\dots \leq n^2+n^2+n^2 = n \cdot n^2 = n^3$$

Então $\sum_{i=1}^{n} i^2 \leq 2n^3$ para $n \geq 1$ portanto $\sum_{i=1}^{n} i^2 = O(n^3)$

$\frac{117}{9} \; 26$

Teorema 2 $\sum_{i=1}^{n} i^2 \geq \frac{1}{2} n^3$ para $n \geq 1$

$$1^2+2^2+3^2+4^2 \dots + n^2$$

$$\frac{1}{2} \; \frac{n}{2} \cdot \frac{1}{2} n^2 + \left(\frac{n+1}{2}\right)^2 + \dots + n^2$$

$$\frac{1}{4} n^3 + \sum_{i=\frac{n+1}{2}}^{n/2} i^2$$

Prova:

$$\sum_{i=1}^{n} i^2 \overset{errado}{=} \frac{n \cdot (1+n^2)}{2} = \frac{1}{2} n^3 + \frac{1}{2} \cdot n \geq \frac{1}{2} n^3$$

Então $\sum_{i=1}^{n} i^2 \geq \frac{1}{2} n^3$ para $n \geq 1$ e portanto $\sum_{i=1}^{n} i^2 = \Omega(n^3)$

Corolário: $\sum_{i=1}^{n} i^2 = \Theta n^3$

Prova

Pelos teoremas 1 e 2 provamos diretamente que $\frac{1}{2} n^3 \leq \sum_{i=1}^{n} i^2 \leq 2 \cdot n^3$ e portanto $\sum_{i=1}^{n} i^2 = \Theta n^3$ //

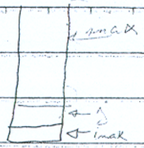
Questão 3 - Em uma lanchonete o cozinheiro era um fanático por algoritmos de ordenação que não arrumou emprego e acabou tendo que fazer panquecas. Mesmo longe de seu computador ele não consegue se desligar dos algoritmos. Todos os dias, depois de fritar as panquecas, ele as ordena em ordem decrescente de diâmetro, ou seja, as maiores embaixo e as menores em cima. A única operação de que ele dispõe é a operação de enfiar a espátula em uma posição qq da pilha de panquecas e inverter toda a pilha dali pra cima. Com isso, as panquecas abaixo da espátula permanecem em suas posições, a panqueca que estava imediatamente em cima da espátula vai parar no topo, a que estava a seguir vai parar embaixo dessa, e assim por diante, até que a panqueca que estava no topo. Suponha que são dados os diâmetros das n panquecas em um vetor $v[1..n]$ (a panqueca na 1ª posição do vetor está no fundo da pilha) e suponha que o único método disponível é FLIP(v, k, n) que inverte a ordem das panquecas como descrito acima a partir da posição k (assim, FLIP(v, 1, n) inverteria toda a pilha). Escreva um algoritmo para ordenar as panquecas que faça $O(n)$ chamadas ao método FLIP.

(não precisa do K)

① / /

PANQUECA(v, n)

```
1  i ← 1    k ← 1
2  enquanto i < n e k < n
3      imax ← k
4      Para j ← k+1 até n
5          faça se v[imax] < v[j]
6              então imax ← j
7      Se imax = n
8          então FLIP(v, i, n)
9          senão se imax ≠ k
10             então FLIP(v, imax, n)
11                   FLIP(v, k, n)
12     i ← i+1
13     k ← k+1
```



Consumo: tamanho da instância: n

$t(n)$	Linha
1	$O(1)$
2	$O(n)$
3	$O(n)$
4-6	$O(n^2)$
7-11	$O(n) \cdot O(2\,FLIP)$
12-13	$O(n)$

Consumo total $T(n) = O(n) \cdot O(2\,FLIP)$

ou seja para cada elemento é feito no máximo 2 chamadas a flip que corresponde a $2n \leq O(n)$

Corretude

O algoritmo panqueca a partir da base escolhe a panqueca de maior diâmetro e faz um flip para que ela fique no topo e logo em seguida faz outro flip para que ela vá para seu local certo na pilha.

/ /

A invariante é que $v[1..k]$ já é uma pilha ordenada na linha 2

Questão 5 - Resolva as seguintes recorrências. Você pode assumir que n é potência de 2.

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

Vamos considerar: $T(1) = 0$

$T(n) = 4T(n/2) + n^2$ para $n \geq 2^1, 2^2$

Pelo teorema mestre temos que

$f(n) = n^2$, $a = 4$ e $b = 2$

Como $f(n) = \Theta(n^{\log_2 4}) = \Theta\, n^2$ então

$T(n) = \Theta\, n^{\log_2 4} \lg n = \Theta(n^2 \lg n)$ //

Ou mais detalhadamente:

$T(n) = 4(T(n/2) + n^2$ $\qquad \frac{n}{2^i} = i$

$= 4(4T(n/4) + (\frac{n}{2})^2) + n^2$ $\qquad \lg n = i$

$= 4^2(4T(n/8) + (\frac{n}{4})^2) + 4(\frac{n}{2})^2 + n^2$

$= 4^i T(\frac{n}{2^i}) + n^2 + n^2 + n^2$

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

$= 0 + n^2 \cdot \lg n$ //

Fórmula fechada $= n^2 \lg n$

Vou provar que $T(n) = n^2 \lg n$ para $n \geq 2^1$ sendo n potência de 2

Prova por indução em n

base: $n = 1$

$T(1) = 1^2 \lg 1 = 0$ //

passo: $n \geq 2^1, 2^2 \ldots$

H.I: $T(n/2) = (\frac{n}{2})^2 \lg(n/2)$

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

$\overset{H.I}{\leq} 4\left(\left(\frac{n}{2}\right)^2 \lg(n/2)\right) + n^2$

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

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

$= n^2 \lg n$ //

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

Consideremos: $T(1) = a$

$T(n) = 4T(n/2) + n^2 \lg n$ para $n \geq 2^1, 2^2, \ldots$

Pelo teorema mestre temos:

$f(n) = n^2 \lg n$, $a = 4$ e $b = 2$

Como $f(n) = \Omega(n^{\log_2 4 + \epsilon})$ para $\epsilon > 0$ e $f(n/2) = \left(\frac{n}{2}\right)^2 \lg n/2 \leq c \cdot n^2 \lg n$

par $c < 1$ $\quad 4 \cdot \frac{n^2}{4}(\lg n - 1) \leq c\, n^2 \lg n$

$n^2 \lg n - n^2 \leq \frac{1}{4} n^2 \lg n$ Não

Então vamos para o desenvolvimento:

$T(n) = 4T(n/2) + n^2 \lg n$	$\frac{n}{2^i} = 1$
$= 4\left(4T(n/4) + \left(\frac{n}{2}\right)^2 \lg n/2\right) + n^2 \lg n$	
$= 4^2\left(4T(n/8) + \left(\frac{n}{4}\right)^2 \lg n/4\right) + n^2 \lg n/2 + n^2 \lg n$	$i = \lg n$
$= 4^3 T(n/2^3) + n^2 \lg n/4 + n^2 \lg n/2 + n^2 \lg n$	
$= 4^i T(n/2^i) + \sum_{k=0}^{i-1} n^2 \lg \frac{n}{2^i}$	

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

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

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

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

Solução fechada: $a n^2 + \frac{1}{2} n^2 \lg^2 n + \frac{1}{2} n^2 \lg n$

Vou provar que a solução está correta

Prova por indução em n:

base: $n = 1$	$T(n) = 4T(n/2) + n^2 \lg n$
$T(1) = a = a \cdot 1 + \frac{1}{2} 1^2 \lg^2 1 + \frac{1}{2} 1^2 \lg 1 = a$	$\overset{H.I}{\leq} 4\left(a \frac{n^2}{4} + \frac{1}{2} \frac{n^2}{4} \lg^2(n/2) + \frac{1}{2} \frac{n^2}{4} \lg(n/2)\right) + n^2 \lg n$
passo: $n \geq 2^1, 2^2, 2^3 \ldots$	
H.I $T(n/2) =$	$= a n^2 + \frac{1}{2} n^2 (\lg n - 1)^2 + \frac{1}{2} n^2 (\lg n - 1) + n^2 \lg n$
$a \frac{n^2}{2^2} + \frac{1}{2}\left(\frac{n}{2}\right)^2 \lg^2(n/2) + \frac{1}{2}\left(\frac{n}{2}\right)^2 \lg(n/2)$	

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

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

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

Questão 6 - Considere esses dois problemas em que é dado um 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) Determinar se existe um caminho de s a t que passa por no mínimo $n/2$ vértices.

(a) Não existe algoritmo polinomial para nenhum desses problemas

(b) Existe algoritmo polinomial para pelo menos um desses problemas.

(c) Existe algoritmo polinomial para exatamente um desses problemas.

(d) Existe algoritmo polinomial para ambos problemas.

Qual dessas afirmativas são verdadeiras ou falsas com certeza? Se o problema P=NP for resolvido, no que isso afeta a resposta anterior?

Considerando que $P \neq NP$

a - F b - V c - V d - F

O algoritmo P2 pode ser resolvido por um algoritmo de caminho mais curto de origem s de tempo polinomial.

Considerando $P = NP$

a - F b - V c - F d - V

Os dois problemas com certeza teriam solução polinomial.

Questão 7 - Você deve cortar uma tora de madeira em várias pedaços. Cortadora de Pedaços Inteiros (CPI). Ex: 10 metros (corte 2,4,7) $= 10+8+6 = 24$

Escreva um algoritmo que dado o comprimento l da tora, um inteiro não-negativo k e um vetor $p[1..k]$ de inteiros tal que $0 < p[1] < p[2] < \dots < p[k] < l$, com os pontos onde a tora deve ser cortada, encontre o custo mínimo para executar esses cortes na CPI. Seu algoritmo deve consumir tempo $O(k^3)$

p(2 4 7) k=3 l=10 min custo = 20

Considere $p[0]=0$ e $p[k+1]=l$ e i e j entre 0 e 4

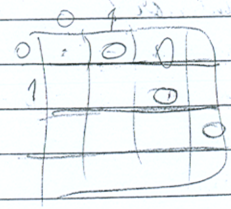
$c[i,j]$ - custo mínimo de efetuar cortes dados em $p[i+1..j-1]$ no pedaço da tora original que vai de $p[i]$ a $p[j]$

Recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } j-i=1 \\ \min\limits_{i<t<j} \{c[i,t]+c[t,j]\} + p[j]-p[i] & \text{se } j-i>1 \end{cases}$$

Queremos um algoritmo que calcule o valor de $c(0,k+1)$.

```
CORTADORA(p, k, W)
1 p[0] ← 0     p[k+1] ← W
  para i ← 0 até k faça
     c[i,i+1] ← 0
  para l ← 2 até k+1
     para i ← 0 até k-l+1
        j ← i+l
        min ← c[i+1, j]
        para t ← i+2 até j-1
           se c[i,t] + c[t,j] < min
              min ← c[i,t] + c[t,j]
        c[i,j] ← min + p[j] - p[i]
  devolva c[0,k+1]
```



Consumo

Linha 4, 5 e 8 executam cada uma no máximo k vezes totalizando $O(k^3)$

Subestrutura ótima

Se S é uma solução ótima para $\langle i,j \rangle$ então S_{ik} e S_{kj} também são soluções ótimas.

Prova: Suponha que S_{ik} não seja solução ótima, e que exista outra solução que substitua S_{ik}, adicionando S_{kj}. Temos uma outra solução para a instância $\langle i,j \rangle$ o que é uma contradição.

Questão 8 - Uma matriz $m \times n$ semi-ordenada é uma matriz $m \times n$ tal que as entradas de cada linha estão em ordem não-decrescente da esquerda para a direita, e as entradas de cada coluna estão em ordem não-decrescente de cima para baixo. Algumas entradas de uma matriz semi-ordenada podem valer ∞, e são tratadas como posições vagas da matriz. Assim uma matriz semi-ordenada pode ser usada para armazenar um conjunto de $r \le m.n$ números. Dizemos então que uma matriz semi-ordenada $Y_{m\times n}$ está vazia se $Y[1,1] = \infty$ e que está cheia se $Y[m,n] < \infty$.

a) Desenhe uma matriz semi-ordenada 3×3 com números do conjunto {9, 16, 3, 2, 4, 8, 5, 14, 12}

	1	2	3
1	2	5	12
2	3	8	14
3	4	9	16

b) Escreva um algoritmo EXTRAI-MIN que recebe como parâmetro uma matriz semi-ordenada $A_{m \times n}$ e extrai da matriz

A um elemento de valor mínimo armazenado na matriz e o devolve. (Devolve ∞ se a matriz está vazia). Após a extração desse elemento, a matriz A deve continuar sendo uma matriz semi-ordenada. Seu algoritmo deve consumir tempo $O(m+n)$. Dica: Pense no algoritmo de extração de máximo de um heap.

EXTRAT-MIN(A)

$A[1,1] \leftrightarrow A[m,n]$
$d \leftarrow A[m,n]$
$A[m,n] \leftarrow \infty$
$i \leftarrow 1$ $j \leftarrow 1$
enquanto $i < m$ ou $j < n$
se $i<m$ e $j<n$
se $A[i,j+1] < A[i+1,j]$
então $A[i,j] \leftrightarrow A[i,j+1]$
$j \leftarrow j+1$
senão se $A[i,j+1] > A[i+1,j]$
então $A[i,j] \leftrightarrow A[i+1,j]$
$i \leftarrow i+1$
senão $i \leftarrow m$
$j \leftarrow n$
senão se $i<m$
se $A[i,j] > A[i+1,j]$
$A[i,j] \leftarrow A[i+1,j]$
$i \leftarrow i+1$
senão se $j<m$
se $A[i,j] > A[i,j+1]$
$A[i,j] \leftarrow A[i,j+1]$
$j \leftarrow j+1$

~~2~~3	6	7	9
5	6		
7	8		
9	10		

Como algoritmo caminhará no máximo em L ou na diagonal, tendo no máximo comprimento de caminhada largura + altura $= O(m+n)$

c) INSERE(A,x)
se $A[m,n] \neq \infty$
devolve "cheia"
$A[m,n] \leftarrow x$
$i \leftarrow m$
$j \leftarrow n$
enquanto $i > 1$ ou $j > 1$
se $i > 1$ e $j > 1$
se $A[i,j-1] > A[i+1,j]$
$A[i,j] \leftrightarrow A[i,j-1]$
$j \leftarrow j-1$
senão se $A[i,j-1] < A[i-1,j]$
$A[i,j] \leftrightarrow A[i-1,j]$
$i \leftarrow i-1$
senão se $i > 1$
então se $A[i,j] < A[i-1,j]$
$A[i,j] \leftrightarrow A[i-1,j]$
$i \leftarrow i-1$
senão se $A[i,j] < A[i,j-1]$
$A[i,j] \leftrightarrow A[i,j-1]$
$j \leftarrow j-1$

d) Como cada linha é ordenada crescentemente então pode-se fazer um merge acumulativo de todas as linhas. Como cada merge possui tempo $O(n)$ e cada uma das n linhas possui n elementos então custará no máximo $O(n^3)$