PROVA 2007/02

Questão 1 - CLRS 4.3-2 - Um algoritmo ALG-A tem seu consumo de tempo descrito através da recorrência

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

Um outro algoritmo ALG-B tem seu consumo de tempo descrito através da recorrência $T'(n) = aT'(n/4) + n^2$ onde a é uma constante. Qual é o maior valor inteiro possível para a de tal forma que o algoritmo Alg-B seja assintoticamente mais eficiente que o algoritmo ALG-A. Justifique a sua resposta.

Para $T(n) = 7T(n/2) + n^2$, seja $a = 7$, $b = 2$ e $f(n) = n^2$ aplicando o teorema mestre temos:

Se $n^{\log_b a - \epsilon} = O(f(n))$ para um $\epsilon > 0$ então

$T(n) = \Theta(n^{\log_b a})$

Assim, $n^{\log_2 7 - 1} \leq 1 \cdot n^2$ para $n \geq 1$, portanto o algoritmo ALG-A consome $\Theta(n^{\log_2 7})$

Para $T(n) = aT(n/4) + n^2$, seja a, $b = 4$ e $f(n) = n^2$

Aplicando o teorema mestre para $a > 4^2$ temos

Lema 1: Se $a > 4^2$ então $\log_4 a - \epsilon = 2$

Prova: Seja $a > 4^2$ então $\log_4 a > \log_4 4^2$, ou ainda $\log_4 a - \epsilon = \log_4 4^2$ para $\epsilon > 0$ onde $\log_4 a - \epsilon = 2$ //

Assim pelo Teorema Mestre e pelo Lema 1, podemos afirmar que $n^{\log_4 a - \epsilon} = O(n^2)$ e portanto $T(n) = \Theta(n^{\log_4 a})$ O algoritmo Alg-B consome $\Theta(n^{\log_4 7})$

Como queremos saber o maior inteiro a de forma que Alg B seja mais eficiente que Alg A então podemos verificar simplesmente a afirmação $\log_4 a < \log_2 7$ onde:

$\frac{\log_2 a}{\log_2 4} < \log_2 7$, $\frac{\log_2 a}{2} < \log_2 7$, $\frac{\log_2 a}{\log_2 7} < 2$

$\log_7 a < 2$, assim $4^2 < a < 49$ então o maior inteiro possível para a é 48. //

Questão 2 - CLRS 2-4 - Dado um vetor $A[1..n]$ de inteiros, chamamos de uma inversão de A um par de índices (i, j) do vetor A tal que $i < j$ e $A[i] > A[j]$. Escreva um algoritmo CONTA-INVERSOES(A, n) que recebe um vetor $A[1..n]$ com inteiros todos distintos e devolve o número de inversões de A. Seu algoritmo deve consumir tempo $O(n \lg n)$. Explique sucintamente porque seu algoritmo está correto e tem o consumo de tempo pedido.

O Algoritmo CONTA-INVERSOES recebe um vetor $A[1..n]$ de inteiros e devolve o número de inversões de A sendo inversão de A um par (i, j) tal que $i < j$ e $A[i] > A[j]$.

```
CONTA-INVERSOES(A, n)
1  cont ← 0
2  MERGESORT2(A, 1, n, cont)
3  devolva cont
```

+INTERCALA 2

```
MERGE2 (A, p, q, r, cont)
1   n1 ← q - p + 1
2   n2 ← r - q
3   cria arrays L[1..n1+1] e R[1..n2+1]
4   para i ← 1 até n1
5           L[i] ← A[p + i - 1]
6   para j ← 1 até n2
7           R[j] ← A[q + j]
8   L[n1+1] ← +∞
9   R[n2+1] ← +∞
10  i ← 1
11  j ← 1
12  para k ← p até r
13          se L[i] ≤ R[j]
14                  então A[k] ← L[i]
15                        i ← i + 1
16                  senão A[k] ← R[j]
17                        j ← j + 1
18          * se L[i] ≠ +∞
19          *       então cont ← cont + n1 - i + 1
```

Correção:

O algoritmo CONTA-INVERSOES utiliza o algoritmo MERGESORT2 que consiste numa modificação do algoritmo já conhecido MERGESORT que possui consumo $O(n \lg n)$. O MERGESORT2 por sua vez utiliza a subrotina MERGE2 modificada nas linhas 18 e 19 em comparação à versão original. O MERGESORT2 recebe um vetor $A[1..n]$ e um contador cont a ser populado e devolvido. Na rotina MERGE2, ao passo que os elementos são comparados se o elemento da esquerda é maior que o elemento da direita não são feitas outras comparações com possíveis valores maiores que o elemento da esquerda, mas como trata-se de vetores crescentemente ordenados, podemos garantir a adição do contador para todos os possíveis valores do vetor da esquerda, que possui índice menor mas contém valor maior que o elemento em questão do vetor direita.

Consumo de tempo $t(n)$

Tamanho da instância n: $r - p + 1$

linha	$t(n)$
1	$O(1)$
2	$O(n \lg n)$
3	$O(1)$

Como o mergesort original consome $O(n \lg n)$ e nessa modificação tem consumo constante então MERGESORT2 também possui consumo $O(n \lg n)$ então o algoritmo CONTA-INVERSOES consome $O(n \lg n)$ //.

Questão 3 - ERS 9.3-5. Suponha que MEDIANA (A, p, r) é um algoritmo que rearranja os elementos de um dado vetor $A[p..r]$ de números inteiros e devolve um índice q, onde $p \leq q \leq r$, tal que $A[p..q-1] \leq A[q] \leq A[q+1..r]$ e $A[q]$ é uma mediana de $A[p..r]$. Suponha ainda que o consumo de tempo do algoritmo MEDIANA é linear. Escreva um algoritmo Select (A, p, r, i) que recebe um vetor $A[p..r]$ de números inteiros e um número inteiro i, onde $1 \leq i \leq r - p + 1$, e devolve o valor do i-ésimo menor elemento de $A[p..r]$. O seu algoritmo deve utilizar o algoritmo MEDIANA como subrotina e deve consumir tempo linear. Explique sucintamente porque seu algoritmo está correto e tem o consumo de tempo pedido.

i=3 1 2 3 7 9
10 11 12 15

O algoritmo SELECT recebe um vetor A[p..r] e um valor inteiro i e devolve o i-ésimo menor elemento de A.
$1 \le i \le r-p+1$

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

Correção: tamanho da instância: $n = r - p + 1$
Prova por indução em n
base: $n = 1$
Como existe apenas 1 elemento e $1 \le i \le r-p+1$ então A[p] é o i-ésimo elemento (linha 2)
passo: $n > 1$
Devido a especificação do algoritmo Mediana e $k = q-p+1$
Temos que:
1- se $k = i$, então o i-ésimo menor elemento do vetor A[p..q] é A[q].
2- se $k < i$ então o elemento procurado é o i-ésimo menor elemento de A[p..q-1]
3- se $k > i$ então o elemento procurado é o $(i-k)$-ésimo elemento de A[q+1..r]
A corretude do algoritmo SELECT é devida a implementação das condições acima.

Consumo de tempo:
Tamanho da instância: $n = r - p + 1$
Seja $T(n)$ o valor máximo de tempo consumido

Linha	Consumo $T(n)$
1-2	$O(1)$
3	$O(n)$
4-5	$O(1)$
6-8	$T(\lfloor n/2 \rfloor)$

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

Recorrência:
$T(n) = 1$ para $n = 1$
$T(n) = T(n/2) + n$ para $n > 2^1, 2^2, 2^3 \ldots$
desenrolando a recorrência temos:

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

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

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

$\sum_{k=0}^{i-1} \frac{1}{2^k} = -2 \cdot \left(\frac{1}{2^{i-1+1}} - 1\right)$

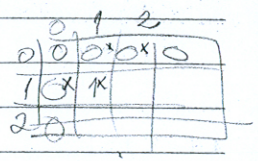
$= 1 + 2n - \left(2n \cdot \frac{1}{n}\right)$
$= 2n + 1 - 2$
$= 2n - 1$

Como $2n - 1 \le 2 \cdot n$ para $n \ge 1$ então concluímos que $T(n) = O(n)$ //.

Prova 2007/2

Questão 5 - Questão de programação dinâmica sobre Torres Gêmeas. Escrever um algoritmo TORRES-GEMEAS(A, n, B, m) que recebe dois vetores A[1..n] e B[1..m] de inteiros, com raio de cada uma das lajotas de duas torres na ordem em que elas aparecem nas torres, e devolve a altura (nº de lajotas) das maiores torres gêmeas que podem ser construídas a partir destas duas torres. Seu algoritmo deve ser o mais eficiente possível. Explique sucintamente porque seu algoritmo está correto e diga qual é o consumo de tempo dele.

O algoritmo TORRES-GEMEAS recebe dois vetores A[1..n] e B[1..m] onde cada elemento tem um raio da lajota e devolve o nº de lajotas máximo que pode deixar as duas torres exatamente iguais.




```
TORRES-GEMEAS(A, n, B, m)
1  para i ← 0 até n
2      Z[i, 0] ← 0
3  para j ← 0 até m
4      Z[0, j] ← 0
5  para i ← 1 até n
6      para j ← 1 até m
7          se A[i] = B[j]
8              então Z[i, j] ← Z[i-1, j-1] + 1
9              senão Z[i, j] ← max[Z[i-1, j], Z[i, j-1]]
10 devolva Z[n, m]
```

O algoritmo encontra a subsequência comum máxima entre as torres.

Subestrutura ótima:

Se Z é uma LSCO máxima de A e B de tamanho K então:

1- se A[n] = B[m] então Z[k] = A[n] = B[m] e Z[1..k-1] é LSCO de A[1..n-1] e B[1..m-1]

2- se A[n] ≠ B[m] então Z[k] ≠ A[n] e Z[k] = B[m] e Z[1..k] é LSCO de A[1..n-1] e B[1..m]

3- se A[n] ≠ B[m] então Z[k] = A[n] e Z[k] ≠ B[m] e Z[1..k] é LSCO de A[1..n] e B[1..m-1].

Prova.

1- Suponha Z^*_{k-1} LSC de A_{n-1} e B_{m-1} de tamanho k-1 existe um outro conjunto W que é LSCO de A_{n-1} e B_{m-1} de comprimento maior que k-1, assim acrescentando o último elemento n é produzida uma solução de comprimento maior que k, uma contradição.

2- Suponha Z^* LSC de A_{n-1} e B_m de comprimento k, existe um outro conjunto W que é LSCO de A_{n-1} e B_m de comprimento maior que k, uma contradição.

3- Semelhante a 2.

Recorrência

$Z[i,0] = Z[0,j] = 0$

$Z[i,j] = Z[i-1, j-1]$ se $A[i] = B[j]$

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

Consumo de tempo: tamanho da instância: (m, n)

linha	
1-2	$O(n)$
3-4	$O(m)$
5	$O(n)$
6-9	$O(n \cdot m)$
10	$O(1)$

$T(n) = O(n \cdot m)$ //

Pergunta tática 1 / 1

Questão 6 - TEM-CICLO(p)

```
1  p1 ← p    p2 ← p
2  enquanto p2 ≠ NIL faça
3      p2 ← prox(p2)
4      se p2 = p1
5          então devolva VERDADEIRO
6      se p2 ≠ NIL
7          então p2 ← prox(p2)
8      se p2 = p1
9          então devolva VERDADEIRO
10     p1 ← prox(p1)
11 devolva FALSO
```

a) Explique porque o algoritmo está correto

b) Denote por n o número de elementos na lista apontada por p. Quantas vezes a linha 2 é executada no pior caso? Não use notação assintótica. Dê a resposta mais precisa que você conseguir. Justifique lindamente a sua resposta.

b) O pior caso é quando ciclo é formado pelo 1º e o último nó. Neste caso a linha 2 é executada n-1 vezes.

a) invariante

Questão 7 Considere o seguinte algoritmo que recebe um vetor E[1..n] e devolve um vetor de números inteiros S[1..n]

```
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
```

Sim, está correto. Podemos fazer uma análise justa e observar que a linha 6 executa se t > 0 e os elementos forem diferentes.

Supondo E de elementos distintos, na 1ª iteração a linha 6 não executa pois t = 0, e após t é acrescentado de 1 na linha 7 mas na próxima iteração na linha 6 t recebe S[1] que vale zero, fazendo com que t oscile entre 1 e 0. Isso faz com que a linha 6 execute no máximo 1 vez. No caso de E com elementos iguais a linha 6 nunca vai ser executada

Assim concluímos que o loop das linhas 5-6 é O(1) consequentemente FAZ-ALGO tem um consumo de tempo de O(n)

1 - 3 - O(1)
4 - 11 - O(n)
12 - O(1)

Questão 8 - TEM-PARTIÇÃO (A, n)

```
1  s ← 0
2  para i ← 1 até n faça s ← s + A[i]
3  se s é ímpar então devolva FALSO
4  T[0,0] ← VERDADEIRO
5  para i ← 1 até n faça T[0,i] ← FALSO
6  para i ← 1 até n faça
7      T[i,0] ← VERDADEIRO
8      para j ← 1 até s faça
9          T[i,j] ← T[i-1,j]
10         se A[i] > j e T[i-1, j-A[i]] = VERDADEIRO
11             então T[i,j] ← VERDADEIRO
12 devolva T[n, s/2]
```

a) Qual o consumo de tempo do algoritmo TEM-PARTIÇÃO?

b) Sabe-se que o problema PARTIÇÃO é NP-completo. Isso contradiz a análise do item anterior? Justifique cuidadosamente a sua resposta.

a)

linha	$t(n)$
1	$O(1)$
2	$O(n)$
3-4	$O(1)$
5	$O(n)$
6-7	$O(n)$
8-11	$O(n \cdot s)$
12	$O(1)$

$t(n) = O(n.s)$ — tamanho da instância: $(n, \lceil \log s + 1 \rceil)$

É importante observar que s pode ser um valor muito grande, assim é correto representar s por $\log s$ assim $t(n, \lceil \log s + 1 \rceil) = O(n \cdot 2^{\log s})$

b) Não, pois, TEM-PARTIÇÃO é resolvido em tempo exponencial, não permitindo portanto uma solução polinomial.

Prova 2007/1

Questão 1 - Suponha que $f(n)$ e $g(n)$ são funções dos inteiros não-negativos nos inteiros não-negativos. Demonstre que as seguintes afirmações são equivalentes:

1 - existem números reais $a>1$, $b>1$, $c>1$ e existe um número inteiro $n_0>0$ tais que $a^{f(n)} \leq b^{g(n)} \leq c^{f(n)}$, para todo $n \geq n_0$.

2 - $g(n)$ é $\Theta(f(n))$.

Lema 1: Se existem números reais $a>1$, $b>1$, $c>1$ e existe um número inteiro $n_0>0$ tais que $a^{f(n)} \leq b^{g(n)} \leq c^{f(n)}$ para todo $n \geq n_0$ então $g(n) = \Theta(f(n))$.

Prova: Seja $a^{f(n)} \leq b^{g(n)} \leq c^{f(n)}$ para todo $n \geq n_0$ temos,

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

$\log_b a \cdot f(n) \leq \log_b b \cdot g(n) \leq \log_b c \cdot f(n)$, considerando:

$\log_b a = c_a$, $\log_b b = 1$ e $\log_b c = c_c$, temos:

$c_a f(n) \leq g(n) \leq c_c f(n)$, portanto:

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

Lema 2: Se $g(n)$ é $\Theta(f(n))$ então existem números reais $a>1$, $b>1$ e $c>1$ e existe um número $n_0>0$ tais que $a^{f(n)} \leq b^{g(n)} \leq c^{f(n)}$, para todo $n \geq n_0$

Prova: Seja $g(n) = \Theta(f(n))$ temos que $c_a \cdot f(n) \leq g(n) \leq c_c \cdot f(n)$ para todo $n \geq n_0$ e $n_0>0$.

Seja $c_a = \log_b a$, $1 = \log_b b$ e $c_c = \log_b c$, onde $a>1$, $b>1$ e $c>1$ podemos afirmar que: $\log_b a \cdot f(n) \leq \log_b b \cdot g(n) \leq \log_b c \cdot f(n)$

para $n \geq n_0$ e $n_0 > 0$.

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

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

Pelos lemas 1 e 2 podemos concluir que as duas afirmações são equivalentes.

Questão 2 - Considere os problemas a seguir.

Problema Partição (A, n): Dado um vetor $A[1..n]$ de números inteiros positivos, decidir se existe um subconjunto I de $\{1, \ldots, n\}$ tal que $\sum_{i \in I} A[i] = \sum_{i \notin I} A[i]$.

Problema TRI-PARTIÇÃO (A, n): Dado um vetor $A[1..n]$ de números inteiros positivos, decidir se existem subconjuntos disjuntos I e J de $\{1, \ldots, n\}$ tais que $\sum_{i \in I} A[i] = \sum_{i \in J} A[i] = \sum_{i \notin I \cup J} A[i]$.

Sabe-se que o problema PARTIÇÃO é NP-completo. Um aluno alega que o problema TRI-PARTIÇÃO também é NP-completo. O aluno está certo? Justifique cuidadosamente a sua resposta.

Sim, o aluno está certo.

Vou provar que o problema TRI-PARTIÇÃO é NP-completo reduzindo-o ao problema da PARTIÇÃO.

Seja o problema de decisão correspondente à PARTIÇÃO Π e o problema de decisão correspondente a TRIPARTIÇÃO Π' e I e I' instâncias de Π e Π' respectivamente uma resposta sim para I' também é resposta sim para I. PROVA: Π' está em NP: É fácil decidir em tempo polinomial que I' é uma tripartição.

Assim podemos reduzir polinomialmente: $\Pi \leq_p \Pi'$.

Seja $S = \sum_{i=1}^{n} A[i]$ sendo n o nº de elementos de A, crie um vetor B com $n+1$ elementos e copie todos os elementos de $A[1..n]$ para $B[1..n]$, sendo que na posição $n+1$ de vetor B inspira o valor $S/2$.

Dessa forma temos que $\sum_{i=1}^{n+1} B[i] = \sum_{i=1}^{n} A[i] + \sum_{i=1}^{n} A[i]$

Assim é fácil notar que se executarmos o algoritmo TEM-TRIPARTIÇÃO $(B, n+1)$ e dá resposta sim. Ao executar o algoritmo tem-partição (B, n) a resposta também será sim. ou seja, se Π' devolve sim para I então Π também devolve sim para a instância I' e vice-versa caso contrário.

Portanto como problema TRI-PARTIÇÃO é redutível ao problema NP-completo PARTIÇÃO então TRI-PARTIÇÃO também é um problema NP-completo.

Questão 3 - Suponha que, para instâncias de tamanho n, você tenha que escolher um dentre três algoritmos A, B e C.

a) Algoritmo A resolve problemas dividindo-os em lineo subproblemas de metade do tamanho, recursivamente resolve cada subproblema e então combina as soluções em tempo $O(n)$.

b) Algoritmo B resolve problemas dividindo-os em dois subproblemas de tamanho $n-1$, recursivamente resolve cada subproblema e então combina as soluções em tempo $O(1)$.

c) Algoritmo C resolve problemas dividindo-os em nove subproblemas de tamanho $n/3$, recursivamente resolve cada subproblema e então combina as soluções em tempo $O(n^2)$.

Qual o consumo de tempo de cada um desses algoritmos? Exprime as suas respostas em termos da notação O, mas procure dar as respostas mais justas possíveis. Qual algoritmo é assintoticamente mais eficiente no pior caso? Justifique as suas respostas.

a) $t(1) = 0$

$t(n) = 5t(n/2) + n$ para $n > 2, 2^2 \ldots$

b) $T(1) = 0$

$T(n) = 2 + (n-1) + 1$ para $n > 1$

c) $T(1) = 0$

$T(n) = 9 + (n/3) + n^2$ para $n > 3^1, 3^2 \ldots$

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

$= 5(5 \cdot T(n/4) + n/2) + n$

$= 5^2(5T(n/8) + n/4) + 5 \cdot \frac{n}{2} + n$

$= 5^3 T(n/8) + 5^2 \cdot n/4 + 5 \cdot n/2 + n$

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

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

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

$= \left(\frac{5^{\lg n}}{2^{\lg n}} - 1\right) \cdot \frac{2n}{3}$

$= \left(\frac{n^{\lg 5}}{n} - 1\right) \cdot \frac{2}{3} n$

$= \frac{n^{\lg 5}}{n} \cdot \frac{2}{3} n - \frac{2}{3} n$

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

$= \left(\left(\frac{5}{2}\right)^i - 1\right) \cdot 2/3$

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

$i = \lg n$

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

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

$\frac{1}{3} =$

b) $T(n) = 2T(n-1) + 1$

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

$= 2^2(2T(n-3) + 1) + 2 + 1$

$= 2^3 T(n-3) + 2^2 + 2^1 + 2^0$

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

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

$= \boxed{2^{n-1} - 1}$

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

$= 2^i - 1$

$n - i = 1$

$i = n - 1$

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

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

$= 9^2(9 + (n/27) + (n/9)^2) + 9 \cdot \left(\frac{n}{3}\right)^2 + n^2$

$= 9^3 + (n/27) + 9^2 \cdot \left(\frac{n}{9}\right)^2 + 9 \cdot (n/3)^2 + n^2$

$= 9^i + (n/3^i) + n^2 + n^2 + n^2$

$= 9^{\log_3 n} + (1) + \log_3 n \cdot n^2$

$= n^2 \log_3 n.$

$\frac{n}{3^i} = 1$

$n = 3^i$

$\log_3 n = \log_3 3^i$

$\log_3 n = i$

$n^2 \cdot \frac{\log_2 n}{\log_2 3} > 1$

$n^2 \cdot \lg n \; //$

Prova por indução em n

a) Base $n = 1$

$T(1) = 0 = \frac{1}{3}(1^{\lg 5} - 1) = \frac{2}{3} \cdot 0 = 0$

Passo: $n > 1, 2^2 \ldots$

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

H.I: $T(n/2) = \frac{2}{3} \cdot \left(\left(\frac{n}{2}\right)^{\lg 5} - \frac{n}{2}\right)$

$T(n) \overset{H.I}{=} 5\left(\frac{2}{3} \cdot \left(\frac{n^{\lg 5}}{2^{\lg 5}} - \frac{n}{2}\right)\right) + n$

$= 5\left(\frac{2}{3} \cdot \frac{n^{\lg 5}}{5} - \frac{2}{3} \cdot \frac{n}{2}\right) + n$

$= \frac{2}{3} n^{\lg 5} - \frac{5n}{3} + n$

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

b) Prova por indução em n

Base $n = 1$

$T(1) = 0 = 2^{1-1} - 1 = 1 - 1 = 0$

passo $n > 1$

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

H.I: $T(n-1) = 2^{n-1-1} - 1 = 2^{n-2} - 1$

$T(n) \overset{H.I}{=} 2 \cdot (2^{n-2} - 1) + 1$

$= 2^{n-1} - 2 + 1$

$= 2^{n-1} - 1$ //

$O(2^{n-1})$

Prova por indução em n.

C) Base $n = 1$

$T(1) = 0 = 1^2 \cdot \lg_3 1 = 1^2 \cdot 0 = 0$

Passo $n > 3^1, 3^2 \ldots$

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

H.I.: $T(n/3) = (n/3)^2 \cdot \lg_3(n/3)$

$T(n) \overset{H.I.}{=} 9\left[\frac{n^2}{3^2} \cdot (\lg_3 n - \lg_3 3)\right] + n^2$

$= 9\left(\frac{n^2}{3^2} \cdot \lg_3 n - \frac{n^2}{3^2}\right) + n^2$

$= \frac{9 \cdot n^2}{3^2} \cdot (\lg_3 n - 1) + n^2$

$= n^2 \lg_3 n - n^2 + n^2$

$= n^2 \lg_3 n$ // $\quad O(n^2 \lg_3 n)$

O algoritmo é Ω o mais eficiente no pior caso.

Questão 4

Considere o seguinte algoritmo que recebe como entrada um vetor $A[0..n-1]$ de números inteiros:

```
CAIXA-PRETA(A, n)
1  k ← 0  i ← 1  j ← 0                                   n-1 = 1+1
2  enquanto i < n e k+j+1 < n faça                          2
3      se A[k+j] = A[(i+j) mod n]
4          então j ← j+1
5          senão se A[k+j] < A[(i+j) mod n]
6              então i ← i+j+1  j ← 0
7          senão se A[k+j] > A[(i+j) mod n]
8              então k ← max{i, k+j+1}  k ← k  i ← k+1  j ← 0
9  devolva k
```

Qual o consumo de tempo desse algoritmo? Expresse a sua resposta em

Questão 4 - se linha 4 incrementa j de 1 em 1
k linha 6 incrementa de 1 ou mais
se linha 8 incrementa k no mínimo 1 vez
i de no mínimo 2
No pior caso a linha 6 poderá ser executada $n-1$ vezes e a linha 4 $n-1$ vezes $\quad n-1+n-1 \quad 2n-2$ vezes

NOME:
E-MAIL:
MATÉRIA:
PROFESSOR:
SALA:

HORÁRIOS

HORÁRIO	SEGUNDA	TERÇA	QUARTA	QUINTA	SEXTA	SÁBADO

ANOTAÇÕES

termos da notação O mas procure dar a resposta mais justa possível. Justifique a sua resposta.

Na linha 2 → $i < n$ | $k' + i' + j = 1$

$k + j + 1 < n$

na linha 4 (sendo k', i', j' valor no final da iteração)

$k' = k$, $i' = i$, $j' = j + 1$ } $k' + i' + j' = k + i + j + 1$

Na linha 7:

$k' = k$, $i' = i + j + 1$, $j' = 0$ } $k' + i' + j' = k + i + j + 1$

Na linha 8

$k' \geq k + j + 1$, $i' = k + j + 1 + 1$, $j' = 0$ } $k' + i' + j' = k + j + 1 + k + j + 2$

Como i voltar i começa em 1 e é acrescentado ao no mínimo 1 unidade, mesmo que ocorra a interpolação entre a linha 4 e 6 o máximo de comparações não passará de $2n$ assim o consumo de tempo do algoritmo CAIXA-PRETA é $O(n)$ //

ANÁLISE AMORTIZADA

Questão 5 - Considere uma sequência de n operações $O_1 O_2 \dots O_n$ - n operações VERMELHA e AZUL, O_1 = VERMELHA e O_2 é Azul. O tempo de execução de cada operação Azul é constante. O tempo de execução da primeira operação VERMELHA é constante, mas cada operação VERMELHA a seguir consome o dobro do tempo da operação VERMELHA anterior. Qual o consumo de tempo de uma sequência de operações VERMELHA e AZUL em cada uma das situações a seguir. Expresse as suas respostas em termos da notação O, mas procure dar as respostas mais justas possíveis. Justifique as suas respostas.

a) A sequência possui $\Theta(1)$ operações Azul entre operações Vermelha consecutivas. | $O_V\ O_A$ | $O_V\ O_A$ | $O_V\ O_A$ | $2 \cdot k \leq n \Rightarrow k \leq n/2$

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

$$= 2^{n/2+1} + n/2 = 2 \cdot 2^{n/2} + n/2 \leq 2 \cdot 2^{n/2} + 2^{n/2} = 3 \cdot 2^{n/2}$$

$T(n) \leq 3\, 2^{n/2}$ assim $\boxed{T(n) = O(2^{n/2})}$ $\sqrt{2^n}$

b) A sequência possui $\Theta(\sqrt{n})$ operações AZUL entre operações VERMELHA consecutivas.

$O_V\ \underbrace{O_A \dots}_{2^0 \dots (\sqrt{n})}\ |\ O_V\ O_A$ | $(1 + \sqrt{n}) \cdot k \leq n \Rightarrow \boxed{k \leq \frac{n}{1+\sqrt{n}}}$

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

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

$$\leq 2 \cdot 2^{\sqrt{n}} + 2^{\sqrt{n}} + 2^{\sqrt{n}} = 4 \cdot 2^{\sqrt{n}}$$

$T(n) \leq 4 \cdot 2^{\sqrt{n}}$. Portanto: $\boxed{T(n) = O(2^{\sqrt{n}})}$

c) Na sequência, se O_i, O_j e O_k são operações VERMELHA consecutivas, $i < j < k$, então o número de operações AZUL entre O_j e O_k é pelo menos duas vezes o número de operações AZUL entre O_i e O_j.

$O_V\ O_A$	$O_V\ O_A \dots O_A$	$O_V\ O_A \dots O_A$
1 2^0	1 2^1	1 2^2

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

$\lg 2^{k+1} \leq \lg(n+1) \Rightarrow k+1 \leq \lg(n+1) \Rightarrow k \leq \lg(n+1) - 1$

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

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

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

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

$$\leq 2^{\lg(n+1)+1}$$

$$\leq 2 \cdot 2^{\lg(n+1)}$$

$$= O(2^{\lg n})$$

$$= O(n) \text{ linear}$$

Questão 6 HINGAP (Perguntar a alguém)

Vou utilizar a árvore AVL como caixa preta.

Busca

Como preconiza-se a propriedade da árvore binária a busca consiste em se a chave do valor procurado for a própria raiz encerra-se, caso não busca-se no nó à esquerda caso contrário busca-se no nó à direita. Como é utilizado apenas um dos lados da árvore e trata-se de uma AVL então é percorrido no máximo a altura = $O(\lg n)$.

Inserção

É inserido um nó no lugar apropriado como a árvore binária e logo em seguida é executado um algoritmo de balanceamento que consiste em rotações para que a árvore permaneça balanceada. A inserção consome a altura da árvore $O(\lg n)$ e as rotações consomem $O(1)$ no máximo, resultando num tempo $O(\lg n)$.

Remoção

É removido um nó como na árvore binária são consideradas 3 casos quando o nó não tem filhos, quando tem apenas 1 ou quando tem 2 filhos, após essa remoção podem ser feitas rotações para que a árvore permaneça balanceada, a remoção é feita em tempo $O(\lg n)$ e as rotação em tempo $O(1)$ sendo portanto $O(\lg n)$ no pior caso

HINGAP

Como trata-se de uma árvore binária para descobrir

Análise Amortizada

[Exr 12.1] - Qual o consumo de tempo do algoritmo?

POWEROFTWO(n)

```
1 i ← 0                                   1
2 enquanto 2 · ⌊n/2⌋ = n faça             1 2 3 4 5 6 7 8
3     n ← ⌊n/2⌋                           4 2 1
4     i ← i + 1                           3 2 1
5 devolva 2^i                             1 1 1
```

4 - 2 lg(n)
6 - 1 < lg(n)
8 - 3 lg(n)

questão da lista de exercícios do Paulo Feofiloff.

Se n é par e não é potência de 2 $\lg(n)$
Se n é ímpar 0
Se n é par mas não é potência 1

O pior caso é quando n é potência de 2 então o consumo de tempo é $O(\lg(n))$

e) Mostre que um vetor k-ordenado pode ser ordenado em tempo $O(n \lg k)$

O algoritmo abaixo recebe um vetor $A[1..n]$ de inteiros e um inteiro k e devolve A k-ordenado.

ORDENA(A, n, k)

1	H ← BUILD-HEAP(A, 1, k)	$O(k)$	
2	para i ← 1 até n-k	$O(n)$	n-k+1 vezes
3	INSERT-HEAP(H, A[i+k])	$O(n) \cdot O(\lg k)$	
4	A[i] ← EXTRACT-HEAP(H)	$O(n) \cdot O(\lg k)$	
5	para i ← n-k+1 até n	$O(k)$	
6	A[i] ← EXTRACT-HEAP(H)	$O(k) \cdot O(\lg k)$	
7	devolva A		

questão da prova 2008.

consumo: $t(n)$

tamanho da instância: (n, k)

linha	consumo
3-4	$O(n \cdot \lg k)$
5-6	$O(k \cdot \lceil \lg \lceil n+1 \rceil \rceil)$
7	$O(k)$
8-9	$2O(1)$

consumo total: $O(n \cdot \lg \lceil n+1 \rceil) = O(n \lg n)$

O consumo esperado $= O(E[X] \lg n)$

Vamos agora definir o valor de $E[X]$.

Para $i = 1, 2, 3 \ldots$ X_i representa as execuções das linhas 3-7 ocorrem pelo menos i vezes.

$X_i \begin{cases} 1, \text{ se as linhas 3-7 executam } i \text{ vezes} \\ 0, \text{ caso contrário.} \end{cases}$

$X = X_1 + X_2 + X_3 \ldots$

$Pr\{X \geq i\}$ se e somente se $r > n$, ou seja, seja r_i, r na i-ésima iteração.

$r_1 > n$ e $r_2 > n$ e $\ldots$ $r_{i-1} > n$

Como o valor de r está entre 0 e $2^t - 1$

então $Pr\{r_i > k\} = \frac{1}{2^t}$ para k em $0 \ldots 2^t - 1$

e como $n = 2^t - 1$

$$Pr\{r > n\} = Pr\{r = n+1\} + Pr\{r = n+2\} + \ldots Pr\{r = 2^t - 1\}$$

$$= \frac{1}{2^t} + \frac{1}{2^t} + \frac{1}{2^t} \ldots = \sum_{2^t}^{2^t} \frac{1}{2^t}$$

$$< \frac{2 - 2^{t-1}}{2^t}$$

$$= \frac{2^t(1 - 2^{-1})}{2^t}$$

$$= \frac{1}{2} \quad //$$

Assim $Pr\{X \geq i\} = Pr\{X \geq 1\} + Pr\{X \geq i\}$ para $i > 2$

$Pr\{X \geq 1\} = 1$ e para $i \geq 2$ com certeza vai executar 1 vez

$Pr\{X \geq i\} = Pr\{r_1 > n, r_2 > n \ldots r_{i-1} > n\}$

$= Pr\{r_1 > n\} \times Pr\{r_2 > n\} \times \ldots \times Pr\{r_{i-1} > n\}$

↑ porque $i \geq 2$

PROVA 2007/2

Questão 4 = Random - Bit-Aleatório

Seja X uma variável aleatória que representa o número de execuções das linhas 3-7 logo o consumo do algoritmo

linha	consumo
1-2	$2O(1)$

$< \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \cdots = \frac{1}{2^{i-1}}$

Agora podemos determinar $E[X]$

$E[X] = E[X_1 + X_2 + \cdots]$

$= E[X_1] + E[X_2] + \cdots$

$= Pr\{X \geq 1\} + Pr\{X \geq 2\} + \cdots$

$= 1 + \frac{1}{2} + \frac{1}{2^2} + \cdots$

$= 1 + \sum_{k=1}^{\infty} \frac{1}{2^k}$

$= 1 + 1$

$= 2$ //

Assim $E[X] = \Theta(1)$

Então o tempo esperado do algoritmo RANDOM é $O(\lg n)$ //

Prova 2004-1

Questão 4 - Alice deseja fazer uma festa e está decidindo a quem convidar. Há n pessoas que são candidatas a serem convidadas. Ela preparou uma lista de pares de pessoas que se conhecem. Alice deseja convidar o maior número possível de pessoas de tal forma que na festa cada pessoa conheça pelo menos cinco pessoas e não conheça pelo menos cinco pessoas. Descreva um algoritmo eficiente que receba como entrada uma lista de n pessoas e uma lista dos pares que se conhecem e devolva uma melhor escolha de pessoas a serem convidadas. Qual o consumo de tempo de seu algoritmo? Uma resposta plenamente satisfatória deve ser um algoritmo que consome tempo $O(n^3)$. O seu algoritmo é plenamente satisfatório? Justifique as suas respostas.

Seja as n pessoas um conjunto de n vértices $\in V$ e os pares como arestas $\in E$ em um grafo $G(V, E)$ então o algoritmo CONVIDADOS recebe um grafo G e devolve uma lista contendo as pessoas convidadas.

```
CONVIDADOS (G)
1   Para cada v ∈ V                          ▷ inicializar Amigo e NãoAmigo com zero
2       Para cada u em Adv[v]                ▷ lista vértice adjacente
3           Amigo[v] ← Amigo[v] + 1
4           NaoAmigo[v] ← n - Amigo[v] - 1
5   A ← ORDENE DE FORMA CRESCENTE POR AMIGO (V)
6   Para cada v em Q
7       Se Amigo[v] < 5 ou NaoAmigo[v] < 5
8           então Para cada u em Adv[v]
9                 Amigo[u] ← Amigo[u] - 1
10            Para cada x em Adv[v]
11                NãoAmigo[x] ← NaoAmigo[x] - 1
12            Remova Q(v)
```

Descobra Q

Tamanho da instância $(n, |E|)$ para $n = |V|$

Linha	
1	$O(n)$
2-3	$O(E)$
4	$O(n)$
5	$O(n \lg n)$
6-7	$O(n)$
8-11	$O(n \cdot E)$
12	$O(1)$

Assim o pior caso tem consumo de tempo de $O(n^2) = O(n^3)$

$O(|n \cdot E|) + O(n \lg n) + O(E) = O(n \cdot |E|) = O(n \cdot n) = O(n^2)$

/ /

Escolha gulosa

Lema: Se a pessoa n não conhece pelo menos k pessoas e desconhece menos que k pessoas então certamente a solução ótima $S-\{n\}$ é ótima para a instância (n,k);

Prova

Vamos supor que a solução ótima seja S^* para a instância (n,k) contendo n

Seja h o habilitado para h sendo $h<k$ então a solução será ótima somente para (n,h) o que é uma contradição.

Subestrutura ótima

Lema: Suponha S^* uma solução ótima da instância (n,k)

(1) se n conhece k pessoas e desconhece pelo menos k pessoas então $S^*-\{n\}$ é ótimo para $(n-1,k)$

(2) se n não conhece pelo menos k pessoas e não desconhece pelo menos k pessoas então $S^*-\{n\}$ é ótimo para (n,k)

Prova S^* é ótimo para $(n-1,k)$

(1) Suponha S^+ uma solução ótima de (n,k) Suponha que S^+_{n-1} não é solução ótima para $(n-1,k)$ então existe uma outra solução melhor que sendo adicionada de n produz uma solução melhor que S^+, uma contradição.

(2) Suponha que S^* uma solução não ótima de $(n-1,k)$ Concluímos de forma direta que S^* não é solução ótima para (n,k) o que é uma contradição.

2001/1

Questão - Escreva um algoritmo PALINDROMO-MAX(Y,n) que recebe uma sequência $Y[1..n]$, e devolve o maior comprimento de uma subsequência palíndromo de Y. Seu algoritmo deve consumir tempo $O(n^2)$. Explique sucintamente porque seu algoritmo está correto e tem o consumo de tempo pedido.

6.12

152

/ /

O algoritmo PALINDROMO-MAX recebe uma sequência $Y[1..n]$ e devolve o maior comprimento de uma subsequência palíndromo de Y.

```
PALINDROMO-MAX(Y, n)
1  Para i ← n até 1
2      X[n-i+1] ← Y[i]
3  Para i ← 0 até n
4      C[0,i] ← 0
5  Para i ← 1 até n
6      C[i,0] ← 0
7  Para i ← 1 até n
8      Para j ← 1 até n
9          se Y[i] = X[j]
10             então C[i,j] ← C[i-1,j-1] + 1
11             senão se C[i-1,j] > C[i,j-1]
12                 então C[i,j] ← C[i-1,j]
13                 senão C[i,j] ← C[i,j-1]
14 devolva C[n,n]
```

tamanho da instância: n para n comprimento da sequência de entrada.

consumo no pior caso: $T(n)$

Linha	$T(n)$
1-2	$O(n)$
3-4	$O(n)$
5-6	$O(n)$
7	$O(n)$
8-13	$O(n^2)$
14	$O(1)$

Total de consumo: $T(n) = O(n^2)$

Corretude

(1) Seja X a sequência Y invertida, se existir uma lcs o.

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 $i_1 < i_2 < \dots < i_m$ tais que $X[j] = Y[i_j]$ para $j = 1, \dots, m$, como X é Y invertida, 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 $Y[1..m]$

(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 $Y[1..m-1]$

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

(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 $Y[1..m]$

Prova: (1) Suponha que Z^*_{k-1} (comprimento $k-1$) não é ssco máxima 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 $f_i(n)$ e $g_i(n)$ funções tais que $f_i(n) = O(g_i(n))$. 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 $n_0 > 0$ tal que $f_i(n) \leq c \cdot g_i(n)$ para $\forall n \geq n_0$

Como $F(n) = \sum_{i=1}^{n} f_i(n)$

$= f_1(n) + f_2(n) + f_3(n) + \dots f_n(n)$ e

$G(n) = \sum_{i=1}^{n} g_i(n)$

$= g_1(n) + g_2(n) + g_3(n) + \dots g_n(n)$

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

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

Como $f_1(n) \leq c \cdot g_1(n)$

$f_2(n) \leq c \cdot g_2(n)$

$\vdots$

$\max f_i(n) \leq \max g_i(n)$

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

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

$F(n) \leq 1 \cdot \max_{1 \leq i \leq n} \{ g_i(n) \}$ para $\forall n > n_0$

$F(n) \leq c \cdot G(n) \Rightarrow F(n) = O(G(n)) \; //$