MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Alfenas . UNIFAL-MG
Rua Gabriel Monteiro da Silva, 714 . Alfenas/MG . CEP 37130-000
Fone: (35) 3299-1000 . Fax: (35) 3299-1063

Prova 01 de Análise e Projeto de Algoritmos - Prof. Humberto César Brandão de Oliveira
Semestre: 2009/1 - Data: 14/04/2008 - Valor: 15,0 pontos – Duração: 100 minutos

Aluno: _____ Matrícula: _____ Nota:_____

Questão	Valor	Nota
01	5,0	
02	3,0	
03	2,0	
04	5,0	
Total	**15,0**	

1. Indique, para cada par de expressões (A, B) na tabela abaixo, se A é O, o, Ω, ω ou θ de B. Assuma que $k > 1$, $c > 1$, $m > 1$ são constantes quaisquer e n é o tamanho da entrada do problema. Sua resposta deve ser na forma SIM ou NÃO acompanhada da justificativa matemática.

	A	B	O	o	Ω	ω	Θ
(a)	n^k	c^n	Sim	Sim	Não	Não	Não
(b)	2^n	$2^{n/2}$	Não	Não	Sim	Sim	Não
(c)	3^n	2^n	Não	Não	Sim	Sim	Não
(d)	$c \cdot n^{\log_2 m}$	$k \cdot n^{\log_2 m}$	Não	Não	Sim	Sim	Sim
(e)	k^c	$\log_2 n$	Sim	Sim	Não	Não	Não

(a)

$$\lim_{n\to\infty} \frac{n^k}{c^n} = \frac{polinômio}{\exp onencial} = 0$$

Portanto, $n^k = o(c^n)$

$n^k = o(c^n)$ implica em:

$n^k = O(c^n)$

$n^k \notin \omega(c^n)$

$n^k \notin \Omega(c^n)$

$n^k = O(c^n)$ e $n^k \notin \Omega(c^n)$ implica em:

$n^k \notin \Theta(c^n)$

(b)

<u>1ª suposição:</u> $2^n = \omega(2^{n/2})$

Se $2^n = \omega(2^{n/2})$ então $\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = \infty$

$$\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = \lim_{n\to\infty} 2^n \times 2^{-n/2} = \lim_{n\to\infty} 2^{n/2} = \infty$$

Portanto, a suposição é verdadeira: $2^n = \omega(2^{n/2})$.

$2^n = \omega(2^{n/2})$ implica em:

$$2^n \notin o(2^{n/2})$$
$$2^n \notin O(2^{n/2})$$
$$2^n = \Omega(2^{n/2})$$

$2^n \notin O(2^{n/2})$ e $2^n = \Omega(2^{n/2})$ implica em:

$$2^n \notin \Theta(2^{n/2})$$

(c)

<u>1ª suposição:</u> $3^n = \omega(2^n)$

$$\lim_{n\to\infty} \frac{3^n}{2^n} = \lim_{n\to\infty} \left(\frac{3}{2}\right)^n$$
$$\frac{3}{2} > 1, \textit{entao}$$
$$\lim_{n\to\infty} \left(\frac{3}{2}\right)^n = \infty$$

Portanto, a suposição é verdadeira: $3^n = \omega(2^n)$.

$3^n = \omega(2^n)$ implica em:

$$3^n \notin o(2^n)$$
$$3^n \notin \mathrm{O}(2^n)$$
$$3^n \in \Omega(2^n)$$

$3^n \notin \mathrm{O}(2^n)$ e $3^n \in \Omega(2^n)$ implica em:

$$3^n \notin \Omega(2^n)$$

(d)

1ª suposição: $c \cdot n^{\log_2 m} \in \mathrm{O}(k \cdot n^{\log_2 m})$

c e k são constantes. Portanto: $n^{\log_2 m} \in \mathrm{O}(n^{\log_2 m})$

Re *gra do* O

$f(n) \in \mathrm{O}(g(n)), se$

$\exists c, n_0$

$\forall n \geq n_0$

$f(n) \leq c \cdot g(n)$

$c = 1, n_0 = 1$

$n^{\log_2 m} \leq 1 \cdot n^{\log_2 m}$

Portanto: $c \cdot n^{\log_2 m} \in \mathrm{O}(k \cdot n^{\log_2 m})$

2ª suposição: $c \cdot n^{\log_2 m} \in \Omega(k \cdot n^{\log_2 m})$

c e k são constantes. Portanto: $n^{\log_2 m} \in \Omega(n^{\log_2 m})$

Re *gra do* Ω

$\Omega(g(n)) = \{f(n):\ \exists\ c\ e\ n_0 > 0$

$\quad | \ 0 \leq c \cdot g(n) \leq f(n) \quad \forall \quad n \geq n_0 \}$

$c = 1, n_0 = 1$

$n^{\log_2 m} \leq 1 \cdot n^{\log_2 m}$

Portanto: $c \cdot n^{\log_2 m} \in \Omega(k \cdot n^{\log_2 m})$

Se $c \cdot n^{\log_2 m} \in \Omega(k \cdot n^{\log_2 m})$ e $c \cdot n^{\log_2 m} \in \mathrm{O}(k \cdot n^{\log_2 m})$ então

$c \cdot n^{\log_2 m} \in \Theta(k \cdot n^{\log_2 m})$

Se os limite é assintoticamente restrito, então

$c \cdot n^{\log_2 m} \notin o(k \cdot n^{\log_2 m})$

$c \cdot n^{\log_2 m} \notin \omega(k \cdot n^{\log_2 m})$

(e)

1ª suposição: $k^c \in o(\log_2 n)$

k e c são constantes positivas maiores que 1.

Portanto, k^c produz outra constante qualquer x que não depende do tamanho da entrada.

$x \in o(\log_2 n)$

$$\lim_{n\to\infty} \frac{x}{\log_2 n} = \frac{cons\tan te}{funcao\ \ monotonicamente\ \ crescente} = 0$$

Portanto, $k^c \in o(\log_2 n)$

$k^c \in o(\log_2 n)$ implica em $k^c \in \mathrm{O}(\log_2 n)$

$k^c \in o(\log_2 n)$ implica em $k^c \notin \omega(\log_2 n)$

$k^c \in o(\log_2 n)$ implica em $k^c \notin \Omega(\log_2 n)$

$k^c \in o(\log_2 n)$ implica em $k^c \notin \Theta(\log_2 n)$

2. Apresente os limites assintóticos superior e inferior para *T(n)* em cada uma das recorrências abaixo. Assuma que *T(n)* é constante para n =< 2. Tente achar cada limite tão firme quanto possível e justifique a sua resposta.

a. $T(n) = 16T(n/4) + n^2$
b. $T(n) = 7T(n/2) + n^2$
c. $T(n) = 2T(n/4) + \sqrt{n}$
d. $T(n) = T(n-1) + n$

(a) $T(n) = 16T(n/4) + n^2$

$a = 16; b = 4; f(n) = n^2$

Testando 1º caso do teorema mestre:

$Se\ T(n) = O(n^{\log_b a - \varepsilon})\ \ para\ um\ \varepsilon > 0$

$\log_2 4 = 2; \log_2 4 - \varepsilon < 2$

$n^2 = O(n^{\log_b a - \varepsilon}) = O(n^{1,9999})$?

Falso.

Testando 2º caso do teorema mestre:

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

$\log_2 4 = 2;$

$n^2 = \Theta(n^{\log_b a}) = \Theta(n^2)$?

Verdadeiro.

Portanto,

$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^2 \log n)$

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

$a = 7; b = 2; f(n) = n^2$

Testando 1º caso do teorema mestre:

$Se\ T(n) = O(n^{\log_b a - \varepsilon})\ \ para\ um\ \varepsilon > 0$

$\log_2 7 = 2; 3 > \log_2 7 - \varepsilon > 2$

$n^2 = O(n^{\log_b a - \varepsilon}) = O(n^{2,8})$?

Verdadeiro.

Portanto:

$T(n) = \Theta(n^{\log_2 7})$

(c) $T(n) = 2T(n/4) + \sqrt{n}$

$a = 2; b = 4; f(n) = n^{1/2}$

Testando 2º caso do teorema mestre:

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

$\log_4 2 = 1/2;$

$n^{1/2} = \Theta(n^{\log_b a}) = \Theta(n^{1/2})$?

Verdadeiro.

Portanto:

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

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

$T(n) = T(n-1) + n$
$T(n-1) = T(n-2) + n - 1$
$T(n-2) = T(n-3) + n - 2$
... ...
$T(1) = c$

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

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

3. Apresente a análise detalhada de complexidade do algoritmo ***main1***.

```
public void main1(ArrayList args){

     subAlgoritmo01();

     double x, y, z;
     for(int i = (1000+args.size()); i >= 0; i--){
         x=(double)i/2; y=(double)x/2; z=(double)x+y;
         System.out.print(z);
     }

     subAlgoritmo02(args);
}

public void subAlgoritmo01(){
     double x, y, z;
     for(int i = 1000; i >= 0; i--){
         x=i/2; y=x/2; z=x+y;
         System.out.print(z);
     }
}

public void subAlgoritmo02(ArrayList args){
     for(int i = 0; i < args.size(); i++){
         System.out.print(args.get(i));
     }
     for(int i = 0; i < Math.pow(args.size(),10); i++){
         System.out.println("aloAlo");
     }
     double x, y, z;
     for(int i = 1000+Math.pow(args.size(),5); i >= 0; i--){
         x=(double)i/2; y=(double)x/2; z=(double)x+y;
         System.out.print(z);
     }
}
```

Complexidade do algoritmo *subAlgoritmo01*:
$f(n) = 1000 \in \Theta(1)$
Complexidade do algoritmo *subAlgoritmo02*:
$g(n) = n + n^{10} + n^5 \in \Theta(n^{10})$
Complexidade do algoritmo *main1*:
$h(n) = f(n) + 1000 + n + g(n) \in \Theta(n^{10})$

4. Apresente a análise detalhada de complexidade do algoritmo ***main2***.

```
public void main2(ArrayList args){
     subAlgoritmo01();
     subAlgoritmo02(args);
     subAlgoritmo03(args);
}

public void subAlgoritmo01(){
     double x, y, z;
     for(int i = 1000; i >= 0; i--){
         x=i/2; y=x/2; z=x+y;
         System.out.print(z);
     }
}

public void subAlgoritmo02(ArrayList args){
     for(int i = 0; i < args.size(); i++){
         System.out.print(args.get(i));
     }
     for(int i = 0; i < args.size(); i++){
         System.out.print(args.get(i));
     }
     double x, y, z;
     for(int i = 1000; i >= 0; i--){
         x=(double)i/2; y=(double)x/2; z=(double)x+y;
         System.out.print(z);
     }
}

public void subAlgoritmo03(ArrayList args){
     for(int i = 0; i < Math.pow(args.size(),2); i++){
        System.out.print("Alo mundo " + i);
     }
     subAlgoritmo03(args.subArray(0, (int)(args.size()/2)));
     subAlgoritmo03(args.subArray((int)(args.size()/2))+1, args.size()-1);
     subAlgoritmo03(args.subArray(0, (int)(args.size()/2)));
     subAlgoritmo03(args.subArray((int)(args.size()/2))+1, args.size()-1);
}
```

Complexidade do algoritmo *subAlgoritmo01*:

$f(n) = 1000 \in \Theta(1)$

Complexidade do algoritmo *subAlgoritmo02*:

$g(n) = n + n + 1000 \in \Theta(n)$

Complexidade do algoritmo *subAlgoritmo02*:

$T(n) = 4(T(n/2)) + n^2 \in n^2 \log n$ -- aplicando teorema mestre (caso 2).

Complexidade do algoritmo *main1*:

$f(n) = 1000 + n + n + 1000 + n^2 \log n$

$f(n) = 2n + 2000 + n^2 \log n \in \Theta(n^2 \log n)$