

Estruturas de Dados II (DEIN0083) 2023.1  
Curso de Ciência da Computação  
1<sup>a</sup> avaliação

Prof. João Dallyson Sousa de Almeida

Data: 24/04/2023

Aluno: [REDACTED]

Matrícula: [REDACTED]

Regras durante a prova:

- É vetada: a consulta a material de apoio, conversa com colega e a utilização de dispositivos eletrônicos. A não observância de algum dos itens acima acarretará a anulação da prova.

I. (2.0pt) Indique se as afirmativas a seguir são verdadeiras e justifique sua resposta:

- (a)  $n^2 \log n / 3 - 5n \in \theta(n^2 + n)$    (b)  $4500n \in \Omega(n)$    (c)  $7000n^2 \in \Omega(n^3)$    (d)  $n^{3n} \in O(6^{2n})$

II. (2.0pt) Apresente o custo de execução e a análise assintótica do algoritmo abaixo em função de N.

```

1 void teste () {
2     for (int i = 0; i < N; i = i+1) {
3         for (int j = 0; j < N; j = j+1) {
4             for (int k = N; k > 0; k = k/2) {
5                 imprimirAlgo();
6             }
7         }
8     }
9 }
10
11    for (int j = 0; j < N; j = j+1) {
12        for (int z = 0; z < N; z = z*3) {
13            imprimirOutravez();
14        }
15    }
16
17    int i = 0;
18    while( i < N ) {
19        i = i + 1;
20    }
21
22 }
```

Annotations on the left side of the code:

- Line 1:  $z = 1$
- Line 2:  $\frac{N}{2}$
- Line 3:  $\frac{N}{2} \times N$
- Line 4:  $\frac{N}{2} \times N \times \frac{N}{2}$
- Line 5:  $\frac{N}{2} \times N \times \frac{N}{2} \times \text{imprimirAlgo}()$
- Line 6:  $\frac{N}{2} \times N \times \frac{N}{2}$
- Line 7:  $\frac{N}{2} \times N$
- Line 8:  $\frac{N}{2}$
- Line 9:  $N$
- Line 10:  $\log_3 N$
- Line 11:  $\frac{N}{2}$
- Line 12:  $\frac{N}{2} \times N$
- Line 13:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 14:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 15:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 16:  $\frac{N}{2}$
- Line 17:  $\frac{N}{2}$
- Line 18:  $\frac{N}{2}$
- Line 19:  $\frac{N}{2}$
- Line 20:  $\frac{N}{2}$
- Line 21:  $\frac{N}{2}$
- Line 22:  $\frac{N}{2}$
- Line 23:  $\frac{N}{2}$

Annotations on the right side of the code:

- Line 1:  $\lambda$
- Line 2:  $\frac{N}{2}$
- Line 3:  $\frac{N}{2} \times \frac{N}{2}$
- Line 4:  $\frac{N}{2} \times \frac{N}{2} \times \frac{N}{2}$
- Line 5:  $\frac{N}{2} \times \frac{N}{2} \times \frac{N}{2} \times \text{imprimirAlgo}()$
- Line 6:  $\frac{N}{2} \times \frac{N}{2} \times \frac{N}{2}$
- Line 7:  $\frac{N}{2} \times \frac{N}{2}$
- Line 8:  $\frac{N}{2}$
- Line 9:  $N$
- Line 10:  $\lambda$
- Line 11:  $\frac{N}{2}$
- Line 12:  $\frac{N}{2} \times N$
- Line 13:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 14:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 15:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 16:  $\frac{N}{2} \times N \times \frac{N}{3}$
- Line 17:  $\frac{N}{2}$
- Line 18:  $\frac{N}{2}$
- Line 19:  $\frac{N}{2}$
- Line 20:  $\frac{N}{2}$
- Line 21:  $\frac{N}{2}$
- Line 22:  $\frac{N}{2}$
- Line 23:  $\frac{N}{2}$

$\lambda = 2$

$\log_2 N$

III. (4.0pt) Considere a seguinte sequência de chaves do vetor [PP, MM, JJ, GG, AA, BB]. OBS: Todos os itens a seguir devem considerar o vetor original.

- Apresente o passo a passo da ordenação em ordem crescente utilizando o algoritmo InsertSort.
- Apresente o passo a passo da ordenação em ordem crescente utilizando o HeapSort.
- Elabore uma versão denominada BuildHeapInsertSort, na qual o algoritmo constrói o MaxHeap e depois utiliza o InsertSort para ordenar as chaves no Heap Construído. Apresente o algoritmo e o passo a passo no vetor dado na questão.
- Discuta a análise de complexidade em função da quantidade de comparações e trocas realizadas pelos algoritmos InsertSort, HeapSort e BuildHeapInsertSort na instância fornecida como entrada.

IV. (1.0pt) Dada a seguinte lista de números [9, 2, 5, 4, 1, 3, 8] qual será o conteúdo da lista após a terceira partição do algoritmo QuickSort? Utilize o último elemento como pivô (direita) e apresente o estado do vetor após cada partição.

V. (10pt) Utilize o teorema Mestre para analisar assintoticamente as recorrências a seguir:

a)  $T(n) = 64T(n/8) + n$    b)  $T(n) = 3T(n/3) + n/3$

$\frac{1}{2} - f(n) = O(n^{\log_2 64}) \rightarrow O(n^6)$

$\frac{1}{2} - f(n) = O(n^{\log_3 3}) \rightarrow O(n^{\log_3 3} \ln n)$

$\frac{1}{2} - f(n) = \Omega(n^{\log_3 3})$ ,  $f(n/b) \leq c \cdot f(n) \rightarrow O(f(n))$

① a) I) Para provar  $\Theta$ , precisamos encontrar duas constantes  $c_1, c_2$  tal que  $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

2)  $c_1(n^2+n) \leq \frac{n^2 \log n}{3} - 5n \leq c_2(n^2+n) \quad (\div n^2+n)$

$c_1(\cancel{n^2+n}) \leq \frac{n^2 \log n}{3} - 5n \leq c_2$   
 $3(n^2+n) \quad (n^2+n)$

$c_1 \leq \frac{n^2 \log n}{3n^2(1+\epsilon)} - \frac{5}{n+1} \leq c_2$

$c_1 \leq \frac{\log n}{3+n} - \frac{5}{n+1} \leq c_2$

Para  $n$  muito grande, não existe constante  $c_2$  que seja maior que  $\frac{\log n - 5}{3+n}$ ,

logo:

FALSO

b) Para provar  $\Omega$ , deve existir uma constante  $c$  tal que  $f(n) \geq c \cdot g(n)$

$4500n \geq c \cdot n \quad (\div n)$

$4500 \geq c$

Para  $c = 4500$ , ~~a afirmção~~ a afirmação é VERDADEIRA

a) Repetindo o passo a passo da letra anterior

$$7000n^2 \geq c \cdot n^3 \left(\div n^3\right)$$

$$\frac{7000}{n} \geq c$$

Para qualquer  $n$  grande, a constante  $c$  é menor que  $\frac{7000}{n}$ ,

logo:

FALSO

d) Para provar 0, deve existir uma constante  $c$ , tal que  $f(n) \leq c \cdot g(n)$

$$n^{3n} \leq 6^{2n} \cdot c \quad \left(\div 6^{2n}\right)$$

$$\frac{n^{3n}}{6^{2n}} \leq c$$

$$\left(\frac{n^3}{6^2}\right)^n \leq c$$

Para qualquer  $n$  grande, a constante  $c$  é menor que  $\left(\frac{n^3}{6^2}\right)^n$ ,

logo:

FALSO

④ ~~0~~ 0 1 9  
~~9, 2, 5, 1, 3, 18~~ & pivo  
~~9, 2, 5, 1, 3, 18~~  
~~2, 9, 5, 1, 3, 18~~  
~~2, 5, 9, 1, 3, 18~~  
~~2, 5, 1, 3, 9, 18~~  
~~2, 5, 1, 3, 18, 9~~ ✓ → pivo chegou ao final, então troca vitor[direita] e vitor[x+1]

1º partição 1 9  
~~2, 5, 1, 3~~ ✓  
~~2, 5, 1, 3~~  
~~2, 5, 1, 3~~  
~~2, 2, 5, 1, 3~~  
~~2, 2, 3, 1, 5~~ ✓  
 2, 1 5  
 2, 1  
 1, 2

Após o 3º partição o vetor é:

1, 2, 3, 5, 8, 9

Q 5) a)  $\begin{cases} a = 64 \\ b = 8 \\ f(n) = n \end{cases}$   $n^{\log_2 64} \rightarrow n^6$   $n^{\log_8 64} \rightarrow n^2$

Como  $f(n) = \Theta(n^{\log_2 64})$  é verdadeiro para  $\epsilon = 1$  ( $f(n) = \Theta(n^{2-\epsilon})$ ), vamos no caso 2 do método master, logo:

$$T(n) = \Theta(n^{\log_2 64}) \rightarrow T(n) = \Theta(n^2)$$

b)  $\begin{cases} a = 3 \\ b = 3 \\ f(n) = n/3 \end{cases}$

$$n^{\log_3 3} \rightarrow n^1 \rightarrow n$$

Como  $f(n) = \Theta(n^{\log_3 3})$  é verdadeiro  $n/3 = \Theta(n)$ , vamos no caso 2 do método master, logo:

$$T(n) = \Theta(n^{\log_3 3} \cdot \lg n) \rightarrow T(n) = \Theta(n \cdot \lg n)$$

A, B, G, f, M, P

word bank PB

Markham

④ a)

PP, MM, JJ, GG, AA, BB

elute = MM

PP, PP, JJ, GG, AA, BB

MM, PP, JJ, GG, AA, BB

elute = JJ

MM, PP, PP, GG, AA, BB

MM, MM, PP, GG, AA, BB

JJ, MM, PP, GG, AA, BB

elute = GG

JJ, MM, PP, PP, AA, BB

JJ, MM, MM, PP, AA, BB

JJ, JJ, MM, PP, AA, BB

GG, JJ, MM, PP, AA, BB

elute = AA

GG, JJ, MM, PP, PP, BB

GG, JJ, MM, MM, PP, BB

GG, JJ, JJ, MM, PP, BB

GG, GG, JJ, MM, PP, BB

AA, GG, JJ, MM, PP, BB

elute = BB

AA, GG, JJ, MM, PP, PP

AA, GG, JJ, MM, MM, PP

AA, GG, GG, JJ, MM, PP

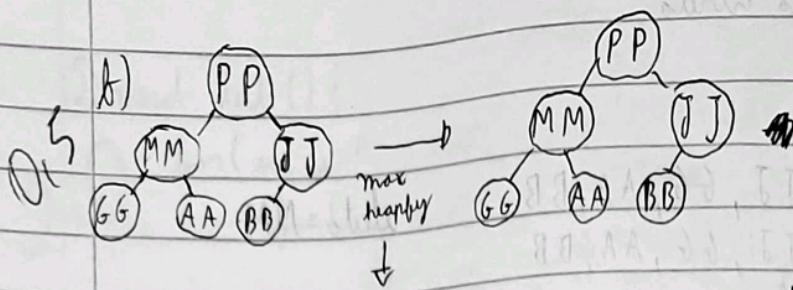
AA, BB, GG, JJ, MM, PP

(A) took twice

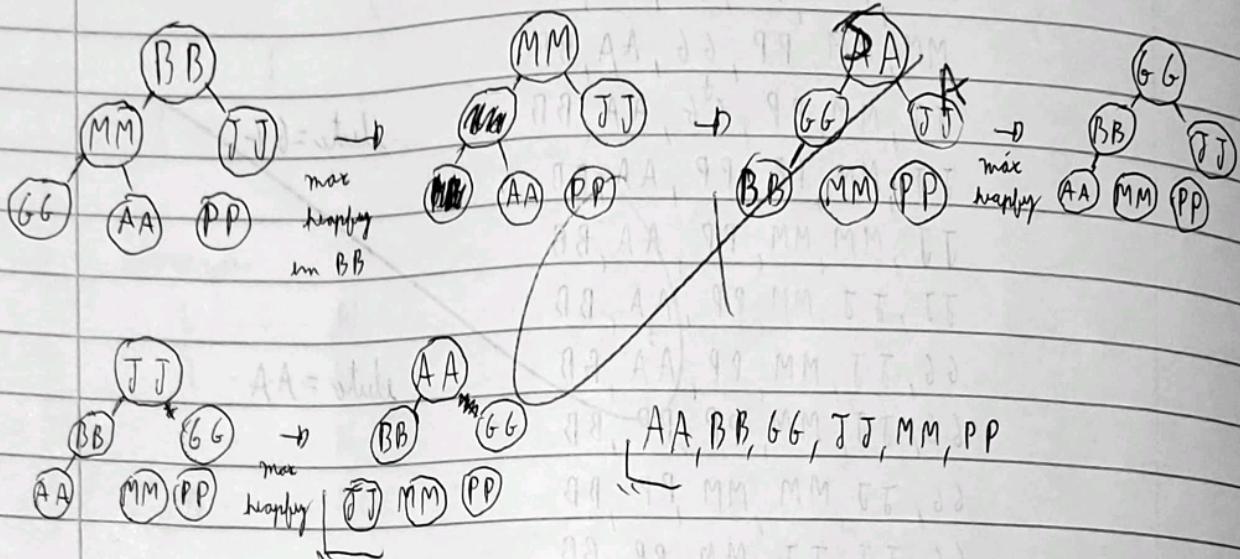
when I was a child, I used to play with my word bank wall. I would make up words and see what ratio I could get.

when I was a child, I used to play with my word bank wall. I would make up words and see what ratio I could get.

when I was a child, I used to play with my word bank wall. I would make up words and see what ratio I could get.



para JJ, MM e PP não mudar mais o heap já é máximo



D) void BuildHeap ~~InsertSort~~(int[] A){

for (int i = ~~0~~ A.length / 2; i > 0; i--) { int n = A.length;  
heappy(A, n, i); }

insertSort(A);

}

Como o vetor dado já é um max heap, basta repetir o passo a passo da letra a.

D) Nesse caso, o heap sort não já receber um heap máximo de entrada ainda por realizar poucas comparações e somente trocas na ordenação em si, logo  
heapsort: comparações  $O(n)$ , trocas  $O(n)$

Insertion sort acaba por realizar muitas comparações e trocas para o vetor inteiro ficar na ordem desejada, logo:

Invert sort: comparações  $O(n^2)$ , trocas  $O(n^2)$

Build heap Invert sort para esse caso acaba performando igual ao invert sort comum, principalmente por o heap máximo  $\overset{m}{\rightarrow}$  quase ordenado na ordem contraria à desejada, logo:

Build heap Invert sort: comparações  $O(n^2)$ , trocas  $O(n^2)$

