

Estruturas de Dados II (DEIN0083) 2023.1
Curso de Ciência da Computação
1^a avaliação

Prof. João Dallyson Sousa de Almeida

Data: 24/04/2023

Aluno: Marcos Vinícius Mergulho Estrito

Matrícula:

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
3     for (int i = 0; i < N ; i = i+1) {
4         for (int j = 0 ; j < N ; j = j+1) {
5             for (int k = N ; k > 0 ; k = k/2) {
6                 imprimirAlgo();
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 }
23

```

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:

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

$$L - f(n) = O(n^{\log_8 6}) \rightarrow \Theta(n^{\log_8 6})$$

$$L - f(n) = \Theta(n^{\log_3 3}) \rightarrow \Theta(n^{\log_3 3} \ln n)$$

$$L - f(n) = \Omega(n^{\log_2 6}), \text{ se } f(n/b) \leq c \cdot f(n) \rightarrow \Theta(f(n))$$

Marcos Vinícius Marqueso Estrada

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

2)

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

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

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

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

ORJATI

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

Logo:

FALSO

b) Para provar Ω , 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 afirmacão~~ a afirmacão 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

Marcos Vinicius Mequita Estrela

Questão 2 da 2ª PGE

Markan

9) a)

PP, MM, JJ, GG, AA, BB

eluto = MM

PP, PP, JJ, GG, AA, BB

MM, PP, JJ, GG, AA, BB

eluto = JJ

MM, PP, PP, GG, AA, BB

MM, MM, PP, GG, AA, BB

JJ, MM, PP, GG, AA, BB

eluto = 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

eluto = 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

eluto = BB

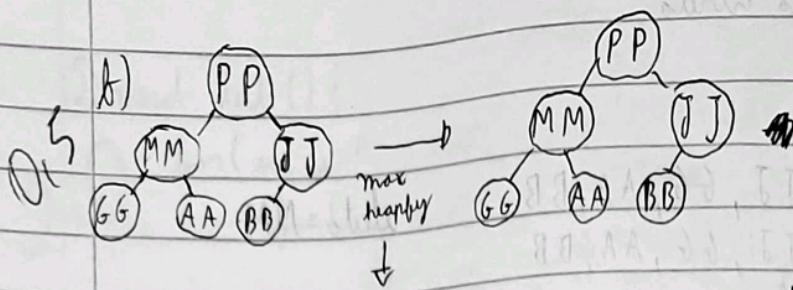
AA, GG, JJ, MM, PP, PP

AA, GG, JJ, MM, MM, PP

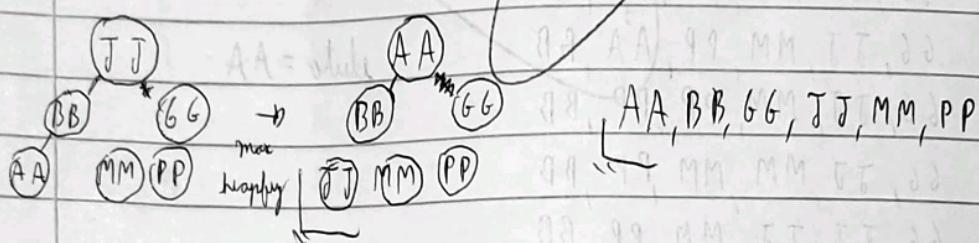
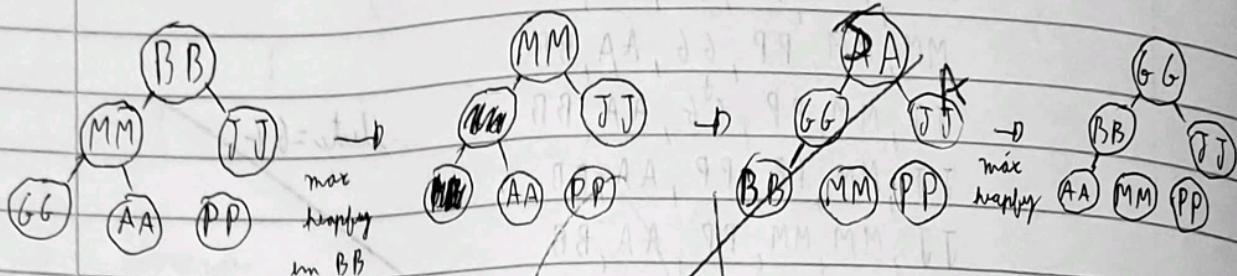
AA, GG, GG, JJ, MM, PP

AA, BB, GG, JJ, MM, PP

(A) traktor



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



AA, BB, GG, JJ, MM, PP

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 máx 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 não realiza muitas comparações e trocas para o vetor inteiro quase 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 $O(n^2)$

