UNIVERSIDADE FEDERAL DO MARANHÃO
DEPARTAMENTO DE INFORMÁTICA

Estruturas de Dados II (DEIN0083) 2024.1
Curso de Ciência da Computação
1ª avaliação

Prof. João Dallyson Sousa de Almeida — **Data:** 22/04/2024

Aluno: Luis Eduardo D. C. Martins **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. (1.0pt) Dê um exemplo de um vetor de comprimento 7 no qual o InsertionSort faz um total de 21 trocas. Explique sua resposta.

II. (2.0pt) Considere a seguinte variante do algoritmo de ordenação por seleção. Dado um vetor A de tamanho n, ele funciona da seguinte forma:

```
Seja M[0...n-1] um vetor de tamanho n
    Para i=0 até n-1
        Seja j o índice do menor elemento em A[i...n-1]
        Troca (A[i], A[j])
        M[i] = j
```

No final da função, o vetor A está ordenado, e o vetor M contém os índices dos elementos mínimos em cada iteração.
a) Demonstre a execução no seguinte vetor A = [6,9,7,2,8]
b) Escreva uma função que execute a "engenharia reversa" do algoritmo. A função recebe o vetor ordenado A de tamanho n e o vetor M de tamanho n. A função retorna o vetor A ao seu estado inicial antes da ordenação por seleção ser aplicada. Descreva a ideia antes de escrever o algoritmo.

III. (2.0pt) Considere os seguintes algoritmos recursivos que resolvem o mesmo problema em uma entrada de tamanho n:
Algoritmo 1: Divide o problema em 5 partes de tamanho n/5 e gasta um tempo adicional n/5
Algoritmo 2: Divide o problema em 6 partes de tamanho n/6 e gasta um tempo adicional de $\sqrt{n}$
Algoritmo 3: Divide o problema em 3 partes de tamanho n/2 e gasta um tempo adicional de O(1)
Apresente a complexidade dos algoritmos 1, 2 e 3. Descreva a solução e mostre a ordem crescente de complexidade.

IV. (1.0pt) Apresente o limite superior para as funções abaixo. Descreva a solução.

a) $7n^4 + n^2 + 100$ c) $nlog_2n + log_3n$
b) $3n^3/100 + 50$ d) $nlog_2n + m * n * o$

V. (3.0pt) Considere a seguinte sequência de chaves do vetor [7, 4, 9, 5, 1, 3]. OBS: Todos os itens a seguir devem considerar o vetor original.
a) Mostre o vetor resultante após a segunda iteração do método "Particiona" do algoritmo do QuickSort para ordenar em ordem crescente. Utilize o elemento da direita como o pivô.
b) Mostre o passo a passo da construção do Heap e o vetor resultante após 3 execuções completas do HeapSort para ordenar em ordem crescente.

n^d $a < b^d$
$n^d \log n$ $=$
$n^{\log_b a}$ $>$

c) Mostre o passo a passo da ordenação utilizando o MergeSort. Mostre o número de comparações e trocas realizadas.

VI. (1.0pt) Apresente o custo de execução e a análise assintótica do algoritmo abaixo. Descreva a solução.

```
a)

int t1, t2 = 0;

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j*=3)
        t1++;

print(t1);

for (int i = 0; i < n; i++)
    for (int j = 0; j <= i; j++)
        t2++;

print(t2);
```

```
b)

int aux1, aux2 = 0

for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
        for (int k = T; k >= 0; k=k/2)
            aux1++;
print(aux1);

for (int i = 0; i < N * M; i++)
    aux2++;
print(aux2)
```

1
2
3
4

$1+2+3+n$

I) ~~o vetor seria [9, 8, 7, 1, 2, 3, 4, 5 6], pois se observarmos~~
~~te.~~

0 7

o vetor [7, 6, 5, 4, 3, 2, 1] pois cada elemento ~~se~~ mover em 6, 5, 4, 3, 2, 1 ~~Ainda~~ casas sendo assim somando esses números de trocas temos que fará 21 trocas

Faltou descrever a melhor solução.

1) II)

a)

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ \end{array}$$

[6, 9, 7, 2, 8] → [2, 9, 7, 6, 8] → [2, 6, 7, 9, 8] → [2, 6, 7, 9, 8]

i i i i

M = [3] M = [3, 3] M = [3, 3, 2]

→ [2, 6, 7, 8, 9] → [2, 6, 7, 8, 9]

i

M = [3, 3, 2, 4] M = [3, 3, 2, 4, 4]

b) ~~faria um loop de i = n-2 até 0 e trocaria o elemento da posição i com o elemento da posição M[i] e faria isso até~~

faria um loop de i = n-1 até 0 e trocaria o elemento da posição i com o elemento da posição M[j],

```
função(A, M, n) {
 for(int i=n-1; i>=0; i--){
      indice = M[i];
      troca(A[i], A[indice]);
  }
}
```

III) 2.0

algoritmo 1: $5T(n/5) + n/5$
algoritmo 2: $6T(n/6) + \sqrt{n}$
algoritmo 3: $3T(n/2) + O(1)$

usando o teorema mestre temos que
algoritmo 1: $a=5$, $b=5$, $d=1$, $\log 5 = 5^1$ assim $\theta(n \log n)$
algoritmo 2: $a=6$, $b=6$, $d=1/2$, $\log 6 > \sqrt{6}$ assim $\cancel{\theta(\sqrt{n})}$ $\theta(n^{\log_6 6}) = \theta(n^1)$
algoritmo 3: $a=3$, $b=2$, $d=0$, $\log 3 > 2^0$ assim $\theta(n^{\log_2 3})$

assim a ordem é:

$\theta(n)$, $\theta(n \log n)$, $\theta(n^{\log_2 3})$

isso pois $\log_2 3 > 1$.

__/__/__

1, 0

IV)

a) $7n^4 + n^2 + 100$

usando limites (~~termos~~) com n^4 termos:

$$\lim_{n\to\infty} \frac{7n^4+n^2+100}{n^4} = \lim_{n\to\infty} 7 + \frac{1}{n^2} + \frac{100}{n^4} = 7$$, logo elas tem a mesma orden de crescimento sendo assim $7n^4+n^2+100 = \theta(n^4)$ ou seja $O(n^4)$

b) $\frac{3n^3}{100} + 50$

usando limites com n^3 termos:

$$\lim_{n\to\infty} \frac{\frac{3n^3}{100}+50}{n^3} = \lim_{n\to\infty} \frac{3}{100} + \frac{50}{n^3} = \frac{3}{100}$$, logo $\frac{3n^3}{100}+50 = \theta(n^3)$, ou seja $O(n^3)$

c) $n\log_2 n + \log_3 n$

temos que $O(g(n)) = f(n)$ se para um ponto $n \leq M$ e um c constantes é verdade $f(n) \leq c \cdot g(n)$, sendo assim

$n\log_2 n + \log_3 n \leq c \cdot n\log_2 n$, assumindo $n = 2$ temos

$2 + \log_3 2 \leq c \cdot 2$, (~~temos que se~~ $\cancel{c = \frac{2+\log_3 2}{2}}$ ~~isto sera verdade~~

~~Logo podemos~~) se assumirmos $c = 2$ teremos que isto será verdade logo podemos deduzir que $O(n\log_2 n)$ da complexidade

22 / 04 /

Luis Eduardo S. C. Martins

IV)

1) $n \log_2 n + m * n * a$

Temos que ~~$O(g(n)) = f(n)$~~ $O(g(n, m, a)) = f(n, m, a)$ se para um ponto $n_1 \leq n$ e $m_1 \leq m$ e $m_2 \leq a$ e para um c constantes ~~temos~~ isso for verdade $f(n, m, a) \leq c . g(n, m, a)$:

$n \log_2 n + m * n * a \leq c . (m * n . a$, assumindo $n = 2$, $m = 1$ e $a = 1$

$2 + 2 \leq c . 2 \rightarrow 2 \leq c$, assim temos que isso será verdade para $c = 2$, logo para $n \log_2 n + m . n . a = O(m . n . a)$

0

1 II)

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array}$$

1) $[6, 9, 7, 2, 8] \rightarrow [2, 9, 7, 6, 8] \rightarrow [2, 6, 7, 9, 8] \rightarrow [2, 6, 7, 9, 8]$

22/04/

Luis ~~Eduardo~~ Eduarda S. C. Martins

10,0

V) [7,4,9,5,1,3] -> [1,3,9,5,7,4]

1,0 a) na primeira iteração do método particiona temos
[1,3,9,5,7,4]
como só há 1 elemento na esquerda não é executado o método particiona no lado esquerdo, mas é executado no lado direito assim temos
[1,3,4,5,7,9]

1,0 b) Como é em ordem crescente usamos um maxheap, para construirmos um heap com o vetor [7,4,9,5,1,3], começamos do meio ajeitamos os filhos da posição i*2 e i*2+1, para que sejam menores que o pai dessa forma:

[7,4,9,5,1,3] -> [7,4,9,5,1,3] -> [7,5,9,4,1,3] -> [7,5,9,4,1,3]

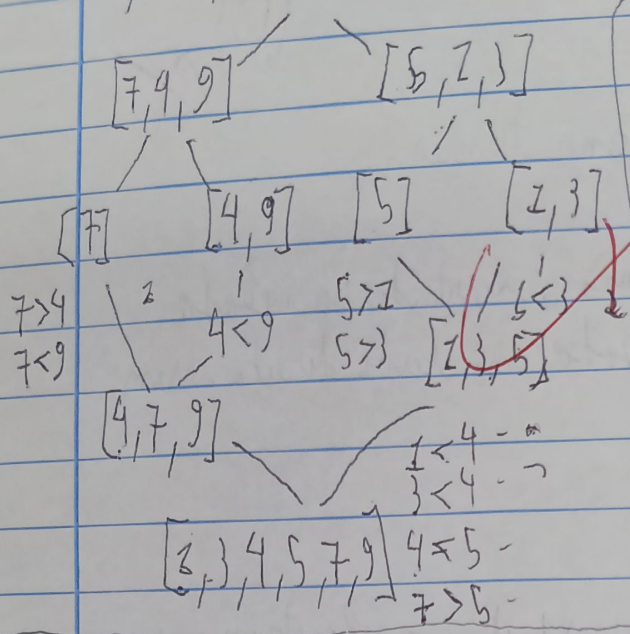
-> [9,5,7,4,1,3] -> [9,5,7,4,1,3]

Com a maxheap feita o resultado do vetor após 3 execuções completas do heapsort:

[4,1,3,5,7,9]

c) ~~[7,4,9,5,1,3] -> [7,4,9][5,1,3] -> [7][4,9][5,1,3]~~ -> 4<9 -> .

c) [7, 4, 9, 5, 1, 3] 0,8 (Houve) ocorreram 6 trocas e 10 comparações



VI)

a) No primeiro for irá ter n repetições e no for interno terá $\log_3 n$ repetições pois irá incrementar j multiplicando por 3, logo esse primeiro laço é $n \log_3 n$

No segundo for vamos ter uma relação do loop interno com o externo tal que $n=3$, terá 6 repetições, para $n=4$, terá 10 repetições, se observarmos veremos que a quantidade de repetições é uma PA logo o segundo loop é:

1,0

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

Dessa forma a complexidade do código é:

$n\log_3 n + \frac{n^2}{2} + \frac{n}{2}$, temos que quando n tender ao infinito somente a variável de maior valor fará diferença logo O do código é $O(n^2)$, pois é o maior entre as variáveis

b) No primeiro for teremos N repetições e no primeiro for interno teremos M repetições e no ~~terceiro for~~ segundo for interno irá ocorrer $(\log_2 T)+1$ repetições pois ele sempre irá dividir o T por 2 até chegar em 0, como exemplo $T=2$, 2 repetições, $T=4$, 3 repetições, $T=8$, 4 repetições e assim por diante, logo esse primeiro loop é:

$$N \cdot M \cdot ((\log_2 T)+1)$$

no segundo for irá ocorrer N·M repetições, logo o segundo for é:

$N \cdot M$

para definir a complexidade nos somamos os resultados:

$N \cdot M \cdot \log_2 T + 2 \cdot N \cdot M$, com isso temos que quando os valores tenderem ao infinito somente as varianeis de maior valor vão fazer diferença logo O é $O(N \cdot M \cdot \log_2 T)$