

MAC0323 Algoritmos de Estruturas de Dados II

Primeira Prova – 02 de abril de 2019

Nome: _____

Assinatura: _____

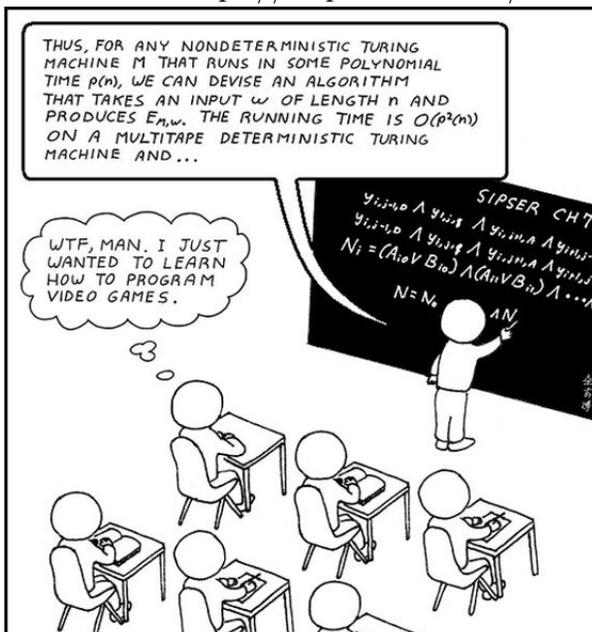
Nº USP: _____

Instruções:

1. Não destaque as folhas deste caderno. A prova pode ser feita a lápis.
2. A prova consta de 10 questões; Verifique antes de começar a prova se o seu caderno está completo.
3. Cuidado com a legibilidade.
4. Não é permitido o uso de folhas avulsas para rascunho, a consulta a livros, apontamentos, colegas ou equipamentos eletrônicos. Desligue o seu celular e qualquer equipamento que possa perturbar o andamento da prova.

DURAÇÃO DA PROVA: 100 minutos

Fonte: <https://br.pinterest.com/>



Questão	Valor	Nota
1	1,0	
2	1,0	
3	1,0	
4	1,0	
5	1,0	
6	1,0	
7	1,0	
8	1,0	
9	1,0	
10	1,0	
Total	10,0	

1. Estruturas de dados (vale 1,0 ponto)

A classe `StringBuilder` do Java representa uma sequência de caracteres mutáveis. Suponha que a implementação dessa classe utilize um **vetor redimensionável** (dobra o vetor quando esta cheio e diminui pela metade quando menos de um quarto está ocupado). A classe mantém uma variável de instância `n` para contar o número de caracteres na string e uma outra variável de instância `a[]` para armazenar a sequência de caracteres.

```
public class StringBuilder {
    private int n; // número de caracteres na sequência
    private char[] a; { // caracteres da sequência a[0], a[1], ...
        [...]
    }
```

(a) Sabendo que tipicamente um `int` ocupa 4 bytes e um `char` ocupa 2 bytes, qual é a quantidade de memória ocupada por um objeto `StringBuilder` para armazenar uma sequência de `n` caracteres?

melhor caso: _____ bytes

pior caso: _____ bytes

(b) Qual o consumo de tempo **amortizado** de cada uma das operação a seguir?

Escreva a melhor resposta no espaço fornecido, usando uma das seguintes possibilidades.

1 $\lg n$ \sqrt{n} n $n \log n$ n^2

<code>charAt(int i)</code>	retorna o <i>i</i> -ésimo caractere na sequência	
<code>deleteCharAt(int i)</code>	remove o <i>i</i> -ésimo caractere da sequência	
<code>append(char c)</code>	coloca <i>c</i> no final da sequência	
<code>set(int i, char c)</code>	troca o <i>i</i> -ésimo caractere por <i>c</i>	

2. Union find (vale 1,0 ponto)

Circule as letras correspondente a configuração de vetores `id[]` (`=pai[]`) que **não podem** ocorrer durante a execução de uma sequência de operações `find()` e `union()` que utilizam o algoritmo *weighted quick union*.

	i:	0	1	2	3	4	5	6	7	8	9

A.	id[i]:	3	3	6	9	3	6	3	4	1	9
B.	id[i]:	9	0	0	0	0	0	9	9	9	9
C.	id[i]:	2	1	1	1	1	1	1	2	1	7
D.	id[i]:	8	0	4	0	0	4	0	4	2	0
E.	id[i]:	4	1	8	2	1	5	1	1	4	5

3. Análise de algoritmos (vale 1,0 ponto)

Para cada função abaixo à esquerda, forneça a opção que melhor corresponde ao consumo de tempo da função em notação assintótica, segundo a lista fornecida à direita.

- | | | |
|---------|--|--------------------|
| [B] | public static int f1(int N) {
int x = 0;
for (int i; i < N; i++)
x += 1;
return x;
} | A. $O(R)$ |
| | | B. $O(N)$ |
| | | C. $O(N + R)$ |
| | | D. $O(N \log R)$ |
| [___] | public static int f2(int N, int R) {
int x = 0;
for (int i = 0; i < R; i++)
x += f1(i);
return x;
} | E. $O(R \log N)$ |
| | | F. $O(N R)$ |
| | | G. $O(R^2)$ |
| [___] | public static int f3(int N, int R) {
int x = 0;
for (int i = 0; i < R; i++) {
for (int j = 0; j < N; j++)
x += f1(j);
}
return x;
} | H. $O(N^2)$ |
| | | I. $O(N R \log N)$ |
| | | J. $O(N R \log R)$ |
| | | K. $O(N R^2)$ |
| | | L. $O(R N^2)$ |
| [___] | public static int f4(int N, int R) {
int x = 0;
for (int i = 0; i < N; i++) {
int j = 1;
while (j <= R) {
x += 1;
j += j;
}
}
return x;
} | M. $O(R^3)$ |
| | | N. $O(N^3)$ |
| [___] | public static int f5 (int N, int R) {
int x = 0;
for (int i = 0; i < N; i++) {
int j = 1;
while (j <= R) {
x += f1(j);
j += j;
}
}
return x;
} | |

4. Construção de heaps (vale 1,0 ponto)

(a) No preprocessamento do **Heapsort** uma lista dada é transformada em um **max-heap**. Considere a seguinte lista em que a parte relevante é a partir do índice 1.

—	-1	77	99	33	66	88	55	22	11	44
0	1	2	3	4	5	6	7	8	9	10

Mostre a seguir o resultado desse preprocessamento aplicado na lista acima.

—										
0	1	2	3	4	5	6	7	8	9	10

(b) Qual o consumo de tempo desse preprocessamento para construir um max-heap com n itens? Na sua resposta utilize a notação assintótica $O(\dots)$.

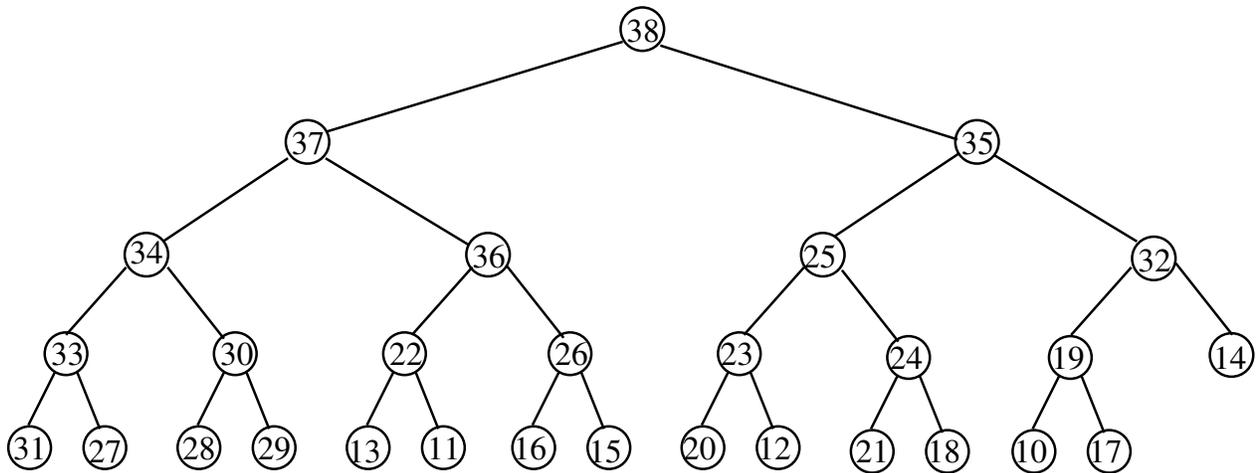
Resposta: _____

(c) Considere o algoritmo *online* para construir um max-heap, ou seja, os itens são fornecidos ao algoritmo um após o outro e a operação básica para construir-se o max-heap é `insert()`. Qual o consumo de tempo desse algoritmo para construir um max-heap com n itens? Na sua resposta utilize a notação assintótica $O(\dots)$.

Resposta: _____

5. Heaps binarios (vale 1,0 ponto)

Considere o seguinte max-heap



(a) Suponha que a última operação executada no max-heap acima foi `insert(x)`.

Circle todos os possíveis valores para `x`.

- | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

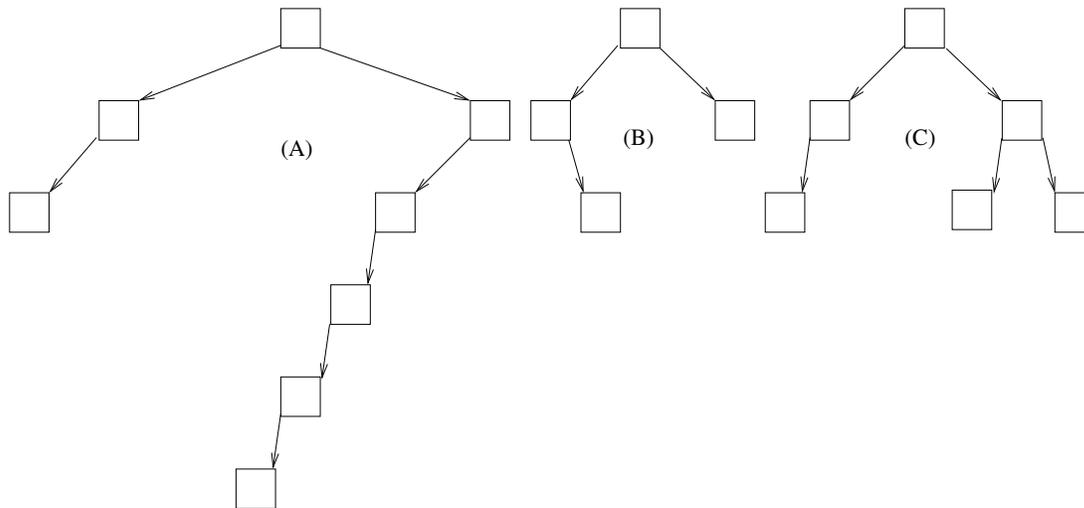
(b) Considere a execução da operação `delMax()` no max-heap acima.

Circle todos os valores envolvidos em alguma comparação.

- | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

6. Leftist heaps (vale 1,0 ponto)

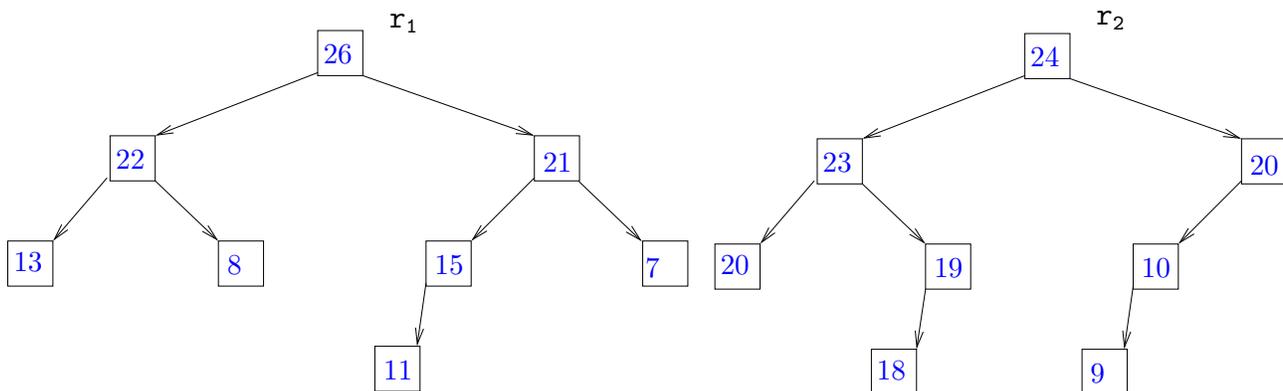
(a) Indique quais das árvores abaixo são esquerdista (*leftist*)? Justifique a sua resposta. Você pode rabiscar sobre as figura para justificar.



(b) Em um **max-heap esquerdista** com n itens, qual é o consumo de tempo de `delMax()`? E o consumo de tempo de `insert()`? Nas respostas utilize notação assintótica.

`delmax()`: _____ `insert()` _____

(c) Desenhe o heap esquerdista resultante da **união** (=merge()) do dois heaps esquerdistas abaixo. Os valores nos nós representam as chaves dos itens.



8. Skip lists (vale 1,0 ponto)

Desenhe a skip list resultante da inclusão das chaves

9 3 6 7 19 12 17 21 26 25

em uma skip list inicialmente vazia. Suponha que a sequência de níveis devolvidos pelas chamadas sucessivas do método `randLevel()` seja

1 4 1 2 1 2 1 1 3 1

Na sua ilustração destaque com **linhas grossas** o caminho feito pela operação `get(21)`.

9. Árvores binárias (vale 1,0 ponto)

A classe **BT** a seguir que representa uma **árvore binária**.

```
public class BT<Item> {
    private Node root; // raiz da árvore
    private int n; // número de nós na árvore
    private class Node {
        private Item item; // o item
        private Node left; // subárvore esquerda
        private Node right; // subárvore direita
        public class Node(Item item) {
            this.item = item;
        }
    }
    [...]
}
```

(a) Escreva um método `altura()` que calcule a altura de uma **BT**.

(b) Escreva um método `posOrdem()` que imprima (`StdOut.println(item)`) os itens de uma **BT** em pós-ordem.

10. Análise de desempenho (vale 1,0 ponto)

Suponha que experimentalmente você observou os seguintes tempos de execução de três funções usando entradas com n itens.

n	tempo
1024	0.03 segundos
2048	0.09 segundos
4096	0.37 segundos
8192	1.70 segundos
16384	7.08 segundos
32768	28.54 segundos
65536	113.55 segundos
131072	493.08 segundos

n	tempo
1000000	0.06 segundos
2000000	0.11 segundos
3000000	0.16 segundos
6000000	0.32 segundos
8000000	0.43 segundos
10000000	0.55 segundos
15000000	0.82 segundos
20000000	1.08 segundos
25000000	1.37 segundos
30000000	1.70 segundos

n	tempo
10.000	1,2 segundos
40.000	2,1 segundos
160.000	3,9 segundos
640.000	7,9 segundos
2.560.000	16,0 segundos

Qual o consumo de tempo das funções (em segundos) em função de n ?
Nas respostas a seguir utilize a notação assintótica $O(\dots)$.

Qual o consumo de tempo da função a()? _____

Qual o consumo de tempo da função b()? _____

Qual o consumo de tempo da função c()? _____

