Estruturas de Dados 2020/2021 – Época Normal (30 de Junho de 2021)

Época Normal - Versão A Duração: 2h

Número mecanográfico: | | | | | | | | | |

Nome completo: _____

Grupo 1 - Fundamentos de Java *(5%)*

1.1. Escreva pequenos **excertos de código** para cada uma das seguintes alíneas:

a) Criar um array v com espaço para k caracteres:

b) Devolver o último elemento de um array v:

c) Expressão booleana para: conteúdo da String a é igual ao da String b?

d) Expressão booleana para: algarismo das dezenas de um inteiro n é par?

e) Expressão booleana para: todos os bits de um inteiro n estão ligados?

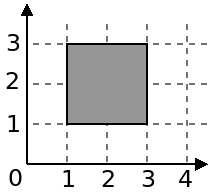
Grupo 2 - Implementação de uma classe simples *(12%)*

2.1. Escreva **código para definir uma classe** `Square` que representa um quadrado paralelo aos eixos através das coordenadas do seu canto inferior esquerdo (x, y) e do tamanho s do seu lado. Os atributos da classe devem ser públicos e serem inteiros representando as coordenadas e o tamanho. Adicione também um construtor que recebe 3 inteiros como argumentos e cria um quadro com esse canto e esse tamanho.

2.2. Implemente um **método** `inside(x,y)` da classe `Square` que devolva *true* se um **ponto de coordenadas** (x, y) **está dentro do quadrado respetivo** e *false* caso contrário.

2.3. Implemente um **método** `double()` da classe `Square` que devolve um novo quadrado que tem o mesmo canto inferior esquerdo, mas **uma área que é o dobro da área do quadrado original**.

2.4. Escreva um excerto de código que crie duas variáveis $s1$ e $s2$ tal que: $s1$ é um quadrado como indicado na figura da direita e $s2$ é um quadrado com o dobro da área de $s1$ criado usado o método da pergunta anterior.



Grupo 3 - Tipos Abstractos de Dados *(4%)*

3.1. Explique o que é um **interface** e quais são as diferenças em relação a uma **classe**.

(pode usar o resto desta página para a continuação de respostas dos grupos 1, 2 e 3 que necessitem de mais espaço)

Num. Mec. | | | | | | | | | | Nome:

Grupo 4 - Complexidade Algorítmica *(22%)*

4.1. Para os pares de funções seguintes, escreva na caixa em branco **Θ**, $\mathcal{O}$ ou **Ω** para tornar a afirmação verdadeira. Note que se as funções em causa tiverem uma relação Θ, não deverá escrever $\mathcal{O}$ ou Ω.

$n^2 \in$ [ ] $(2n)$ $\quad \log n + n \in$ [ ] (n) $\quad 99 \times n^3 \in$ [ ] $(0.1 \times n^4)$ $\quad \log_2 n \in$ [ ] $(\log_3 n)$ $\quad 2^n \in$ [ ] (3^n)

4.2. Descreva o pior caso no tempo de execução para os seguintes pedaços de código, usando notação $\mathcal{O}$ para a variável n. Use o limite mais "apertado" possível (ex: indicar todos como sendo $\mathcal{O}(n!)$ resultará em 0 pontos).

a) [ ]

```
for (int i=0; i<n; i++)
    for (int j=0; j<42; j++)
        count++;
```

b) [ ]

```
for (int i=n; i>0; i=i-2)
    for (int j=3; j<n-3; j=j+3)
        count++;
for (int i=1; i<n; i++)
    count++;
```

c) [ ]

```
for (int i=1; i<n; i=i*2)
    for (int j=n; j>0; j=j/2)
        count++;
```

d) [ ]

```
int f1(int n) {
    if (n <= 0) return 1;
    return f1(n/2) + 1;
}
```

e) [ ]

```
int f2(int n, int[] v) {
    if (n==0) return v[0];
    int k = 0;
    for (int i=0; i<n; i++) k += v[i];
    return k + f2(n/2, v) + f2(n/2, v);
}
```

4.3. Justifique a sua resposta para a alínea (e) da pergunta anterior, **escrevendo a recorrência e desenhando a respectiva árvore de recorrência**, explicando porque dá origem à complexidade indicada.

4.4. Previsão de Tempo de Execução.
Complete a tabela seguinte usando a informação já preenchida em cada linha. Use uma previsão baseada no tempo já preenchido, tendo em conta a proporção entre n_1, n_2 e n_3 (ms = milisegundos). Se não souber o valor exacto, pode deixar ficar a "fórmula" que usou (ex: $\sqrt{25}$ em vez de 5)

Programa	**Complexidade**	**Tempo para** $n_1 = 10$	**Tempo para** $n_2 = 20$	**Tempo para** $n_3 = 30$
A	$\Theta(1)$		100ms	
B	$\Theta(n)$		100ms	
C	$\Theta(n^2)$		100ms	
D	$\Theta(n^3)$		100ms	
E	$\Theta(2^n)$		100ms	

4.5. Suponha que está a criar uma estrutura de dados para representar **conjuntos de números inteiros**. Indique a **complexidade** (notação $\mathcal{O}$ do pior caso) do **melhor algoritmo que consegue imaginar** para a implementação de alguns métodos. Suponha que o conjunto tem n inteiros entre 1 e k e **justifique sucintamente** as suas respostas.

a) Complexidade temporal para **saber se o conjunto contém um elemento** x usando:

Array desordenado:	
Array ordenado:	
Array de booleanos:	

b) Complexidade temporal para **inserir um elemento no conjunto** usando:

Array desordenado:	
Array ordenado:	
Array de booleanos:	

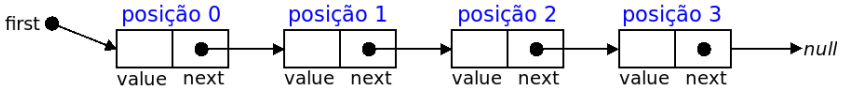
c) Complexidade espacial para armazenar o conjunto usando:

Array desordenado:	
Array ordenado:	
Array de booleanos:	

(pode usar o resto desta página para a continuação de respostas do grupo 4 que necessitem de mais espaço)

Grupo 5 - Listas, Pilhas e Filas *(18%)*

5.1. Considere uma classe `SinglyLinkedList<T>` representando uma lista ligada simples genérica tal como dada nas aulas com atributos `size` e first, e uma classe `Node<T>` representando um nó com atributos `value` e `next`.



a) Implemente um **método** `contains(x)` da classe `SinglyLinkedList<T>` devolve *true* se o elemento x está na lista e *false* caso contrário.

Não pode usar qualquer outro método já existente da classe. Indique a **complexidade temporal** do método que descreveu.

b) Implemente um **método** `duplicateEven()` da classe `SinglyLinkedList<T>` que duplica (repete 2x) todos os elementos que estavam em posições pares. Por exemplo, uma chamada a `duplicateEven()` da lista $2 \to 4 \to 6 \to 8$ modificaria a lista para $2 \to 2 \to 4 \to 6 \to 6 \to 8$ (duplica os elementos das posições 0 e 2).

Não pode usar qualquer outro método já existente da classe. Indique a **complexidade temporal** do método que descreveu.

5.2. Para cada um dos casos seguintes escreva `lista`, `array` ou `ambos` se para o cenário descrito for melhor usar, respetivamente, uma lista ligada, um array ou for indiferente (ambos dariam e nenhum apresenta vantagem).

a) [ ] Quero ter os elementos guardados em posições contíguas de memória (otimizando a *cache*)

b) [ ] Não sei à partida quantos elementos vou querer armazenar

c) [ ] Quero poder guardar objectos de uma classe *Person* que representa uma pessoa

d) [ ] Quero pode aceder rapidamente a um elemento na i-ésima posição

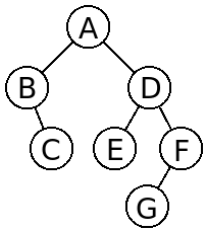
e) [ ] Quero pode inserir rapidamente um elemento na primeira posição

5.3. Justifique sucintamente a sua resposta para a alínea *(e)* da pergunta anterior

(pode usar o resto desta página para a continuação de respostas do grupo 5 que necessitem de mais espaço)

Grupo 6 - Árvores *(17%)*

6.1. Considerando a **árvore da figura** seguinte, responda às seguintes alíneas:



a) Os nós da árvore em **preorder:**

b) Os nós da árvore em **inorder:**

c) Os nós numa **pesquisa em largura:**

6.2. Considere uma classe `BTree<T>` de uma árvore binária genérica tal como dada nas aulas com um atributo `root` a apontar para a raíz que é um `BTNode<T>` representando um nó com atributos `value`, `left` e `right`.

a) Implemente um **método** `levels()` da classe `BTree<T>` que devolve o número de níveis de profundidade da árvore (por exemplo, a árvore da pergunta 6.1 tem 4 níveis).

Não pode usar qualquer outro método já existente da classe, mas pode usar métodos auxiliares. Indique a **complexidade temporal** do método que descreveu.

b) Implemente **um método estático** `mins(BTree<Integer> t)` que devolve um array com o menor elemento de cada um dos níveis da árvore contendo em cada posição i o mínimo desse nível. Por exemplo, para a árvore da pergunta 6.1, deveria ser devolvido um array contendo $[A, min(B, C), min(D, E, F), G]$.

Não pode usar qualquer outro método já existente da classe, excepto o método `levels()` da pergunta anterior, mas pode criar outros métodos auxiliares. Indique a **complexidade temporal** do método que descreveu.

Grupo 7 - Árvores Binárias de Pesquisa *(10%)*

7.1. Suponha que quer inserir (por esta ordem) os seguintes 6 números numa árvore binária de pesquisa: $4, 1, 2, 6, 5, 7$.

Para as duas alíneas ao lado, **desenhe a árvore** (depois de completa a operação respetiva).

a) Como fica a **árvore original** depois de inseridos os 6 números?

b) Como fica a árvore (original) depois de **removermos o 4**?

7.2. Suponha que quer inserir as 12 letras maiúsculas de 'A' a 'L' numa árvore binária de pesquisa. Qual a **mínima altura possível** da árvore? Indique por **qual ordem deve inserir os elementos** para garantir essa altura mínima.

Altura mínima: | | Ordem de Inserção: | | | | | | | | | | | | |

7.3. Imagine que tem uma árvore binária de pesquisa com altura 42 (com $42 + 1 = 43$ níveis portanto). Indique:

a) Qual o **maior** número possível de nós que pode ter?

b) Qual o **menor** número possível de nós que pode ter?

Grupo 8 - Filas de Prioridade e Heaps *(12%)*

8.1. Imagine que tem uma **maxHeap** descrita pelo seguinte array

8	5	7	3	2	4

Para as 3 alíneas de baixo **desenhe a árvore** (a heap) correspondente (sempre já com o invariante reposto):

a) Qual a heap representada pelo **array original**?

b) Como fica a heap (original) depois de **adicionarmos um 9**?

c) Como fica a heap (original) depois de **remover o máximo**?

8.2. Explique como poderia usar uma *maxHeap* para **ordenar um conjunto de n números por ordem crescente**, indicando que operações usava e qual seria a **complexidade temporal e espacial** do seu algoritmo.

(pode usar o resto desta página para a continuação de respostas dos grupo 6, 7 e 8 que necessitem de mais espaço)

8.3. (pergunta de valorização) Explique como poderia obter o k**-ésimo maior elemento de uma maxHeap** em tempo $\mathcal{O}(k \log k)$. Note que a solução trivial de chamar k vezes a remoção do máximo teria complexidade $\mathcal{O}(k \log n)$, que não é a pedida, e pode ser muito pior se k for mais pequeno que n.

(pode usar o resto desta página para continuar a resposta à pergunta 8.3, se necessitar de mais espaço para além do que foi dado)