Estruturas de Dados 2019/2020 – Época Normal (30 de Junho de 2020)

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) Extrair o algarismo dos milhares de um inteiro n:

b) Criar um matriz quadrada m com espaço para $k \times k$ caracteres:

c) Expressão booleana para: 1^o caracter da String s é um dígito?

d) Expressão booleana para: comprimento da String s é ímpar?

e) Ligar (colocar em 1) o penúltimo bit de um inteiro n:

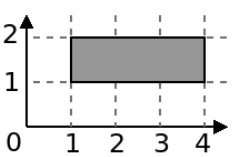
Grupo 2 - Implementação de uma classe simples *(11%)*

2.1. Escreva **código para definir uma classe** `Rectangle` que representa um rectângulo através das coordenadas do seu canto inferior esquerdo $(x1, y1)$ e canto superior direito $(x2, y2)$ (com $x1 \leq x2$ e $y1 \leq y2$). Os atributos da classe devem ser privados e serem inteiros representando as coordenadas pedidas. Adicione também um construtor que recebe 4 inteiros como argumentos e cria um retângulo com essas coordenadas dadas (pode assumir que são dadas corretamente).

2.2. Implemente um **método** `area()` da classe `Rectangle` que devolve a área do rêtangulo respetivo.

2.3. Implemente um **método** `origin()` da classe `Rectangle` que devolve um novo retângulo que tem a mesma altura e largura do retângulo original, continua a ser paralelo aos eixos, mas tem o canto inferior esquerdo na origem, ou seja, nas coordenadas $(0, 0)$.

2.4. Escreva um excerto de código que crie duas variáveis $r1$ e $r2$ tal que: $r1$ é um retângulo como indicado na figura da direita; $r2$ é um retângulo de dimensões iguais a $r1$ mas movido para a origem usando o método da pergunta anterior.



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

Interface Set<E>

Type Parameters: E - the type of elements maintained by this set

All Superinterfaces: Collection<E>, Iterable<E>

All Known Subinterfaces: NavigableSet<E>, SortedSet<E>

All Known Implementing Classes: AbstractSet, ConcurrentHashMap.KeySetView, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet, JobStateReasons, LinkedHashSet, TreeSet

3.1. A caixa de cima contém parte da documentação oficial do Java para o interface `Set` (um conjunto). Suponha que quer criar uma variável *s* para conter um conjunto de inteiros. Indique tudo o que vê de errado na declaração `Set<E> s = new Set<E>();` e como seria uma declaração correcta para criar *s* (implementada como uma árvore).

(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 *(23%)*

4.1. Para os pares de funções seguintes, escreva na caixa em branco **Θ**, **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.

$2^3 \in \square\ 24$ $\quad \log n \in \square\ (\sqrt{n})$ $\quad 42 \times n^5 \in \square\ (2^n)$ $\quad n^2 \in \square\ (n \log n)$ $\quad 2^{2n} \in \square\ (2^n)$

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

b) ☐

```
for (int i=n/2; i>0; i=i-2)
    for (int j=2; j<n-2; j++)
        for (int k=22; k<333; k++)
            count++;
```

c) ☐

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

d) ☐

```
int f1(int n) {
    if (n <= 2) return 2;
    return f1(n-2) + 2;
}
```

e) ☐

```
int f2(int n) {
    if (n <= 0) return 0;
    int count = 0;
    for (int i=0; i<42; i++) count++;
    return count + f2(n/2) + f2(n/2);
}
```

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. Nos tempos, use uma previsão baseada no tempo já preenchido, tendo em conta a proporção entre n_1 e n_2 (ms = milisegundos). Se não souber o valor exacto, pode deixar ficar a "fórmula" que usou (ex: $\sqrt{25}$ em vez de 5)

Programa	**Complexidade**	**Nome Comum**	**Tempo para $n_1 = 10$**	**Tempo para $n_2 = 30$**
A	$\Theta(1)$			30ms
B	$\Theta(n)$		30ms	
C		quadrática	10ms	
D		cúbica		100ms
E	$\Theta(3^n)$		5ms	

4.5. Suponha que tem uma implementação de conjuntos de inteiros que usa como base um array ordenado onde são guardadas os inteiros (uma por cada posição). Seja n o número de inteiros do conjunto. Indique a **melhor complexidade temporal** (notação $\mathcal{O}$) que conseguiria obter para os seguintes métodos (justificando sucintamente):

a) Devolver número de inteiros no conjunto:

b) Verificar se um inteiro está no conjunto:

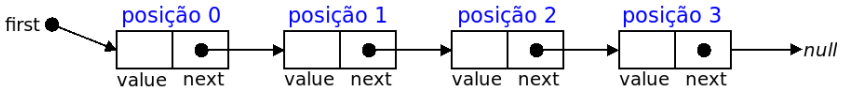
c) Remover um inteiro do conjunto:

4.6. Explique como poderia obter uma **implementação mais eficiente para a verificar se um inteiro está no conjunto**, indicando a nova complexidade e outras potenciais vantagens e desvantagens dessa outra implementação quando comparada com o array ordenado (ex: mais algum método é melhor ou pior?). Explique também se a implementação funcionaria para outros tipos que não fossem números inteiros.

(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 *(20%)*

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** `count(x)` da classe `SinglyLinkedList<T>` devolve o número de vezes que o elemento x aparece na lista.

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

b) Implemente um **método** `removeEven()` da classe `SinglyLinkedList<T>` que remove todos elementos da lista que estão em posições pares (mantendo os de posições ímpares). As posições começam em 0 (por exemplo, numa lista de quatro elementos como a da figura de cima, seriam removidos os elementos nas 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 usar a menor quantidade de memória possível

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

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

d) [ ] Quero poder guardar objectos de uma classe *Student* que representa um aluno

e) [ ] Quero poder aplicar pesquisa binária

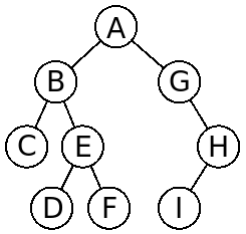
5.3. Suponha que tem apenas disponíveis duas pilhas $p1$ e $p2$ com as operações habituais (*push*, *pop* e *size*). Explique (por palavras), como poderia simular o funcionamento de uma **fila** usando as duas pilhas (e mais nenhuma outra estrutura de dados), e como ficariam os seus métodos *enqueue*, *dequeue* e *size*.

(pode usar esta página para a continuação de respostas do **grupo 5** *que necessitem de mais espaço)*

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

Grupo 6 - Árvores *(16%)*

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** `size()` da classe `BTree<T>` que devolve o número de nós da árvore.

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** `binary(a,b)` da classe `BTree<T>` que devolve *true* se todos os nós com profundidade entre a e b (inclusive, ou seja no intervalo $[a, b]$) têm exatamente dois filhos, ou *false* caso contrário (se não existir nenhum nó nessas profundidades deve devolver *true*).

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

Grupo 7 - Árvores Binárias de Pesquisa *(13 %)*

7.1. Suponha que quer inserir (por esta ordem) os seguintes 6 números numa árvore binária de pesquisa: $10, 3, 2, 4, 1, 14$.

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 10**?

7.2. Suponha que quer inserir as 14 letras maiúsculas de 'A' a 'N' 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 k níveis (de altura $k-1$ portanto). Indique:

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

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

7.4. Numa qualquer árvore binária de pesquisa, determinar se um elemento existe na árvore tem (no pior caso) complexidade temporal [ ] $(\log n)$

Complete a frase de cima, escrevendo na caixa em branco **Θ**, $\mathcal{O}$ ou **Ω** para tornar a afirmação verdadeira. Note que se a relação for Θ, não deverá escrever $\mathcal{O}$ ou Ω. Justifique sucintamente a sua resposta na caixa em baixo.

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

8.1. Imagine que tem uma **minHeap** descrita pelo seguinte array | 2 | 6 | 4 | 9 | 7 | 5 |

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 1**?

c) Como fica a heap (original) depois de **removermos o mínimo**?

8.2. Suponha que quer descobrir o k-**ésimo maior elemento de um conjunto de** n **números**. Explique como poderia usar uma *minHeap* para esse efeito e descreva a complexidade temporal e espacial da sua solução.

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