Estruturas dado efolio global 10.6/12

## 1. Resposta Múltipla

Considere:

$f(n) = n^2 \log n + 4n$

$g(n) = n^3$

Qual das seguintes afirmações é verdadeira?

| | | |
|---|---|---|
| A | $f(n) = \Theta(g(n))$ | |
| B | $f(n) = O(g(n))$, mas $f(n) \neq \Theta(g(n))$ | ✔ |
| C | $f(n) = \Omega(g(n))$, mas $f(n) \neq \Theta(g(n))$ | |
| D | Nenhuma das anteriores | |

Considere o seguinte segmento de código:

```
for (int i = 1; i <= n; i *= 2)
   a++;
```

Qual é a ordem de complexidade temporal em função do número de incrementos de a?

| | | |
|---|---|---|
| A | *O(1)* | |
| B | *O(log n)* | ✔ |
| C | *O(n)* | |
| D | *O(n log n)* | |

Numa lista simplesmente ligada, qual é a principal vantagem de manter um apontador para o último nó da lista, além do apontador para o primeiro nó?

| | | |
|---|---|---|
| A | Permite pesquisar qualquer elemento em tempo constante. | |
| B | Permite inserir no fim da lista em tempo constante, desde que não seja necessário pesquisar previamente. | ✔ |
| C | Elimina a necessidade de usar apontadores nos nós. | |
| D | Permite remover qualquer nó da lista em tempo constante. | |

Qual das seguintes afirmações está correta?

| | | |
|---|---|---|
| A | Uma pilha segue a política FIFO e uma fila segue a política LIFO. | |
| B | Uma pilha segue a política LIFO e uma fila segue a política FIFO. | ✔ |
| C | Tanto pilhas como filas seguem a política FIFO. | |
| D | Tanto pilhas como filas seguem a política LIFO. | |

Numa árvore de pesquisa binária que não admite chaves repetidas, que propriedade tem a travessia em-ordem?

| | | |
|---|---|---|
| A | Visita as chaves por ordem decrescente. | |
| B | Visita primeiro sempre todos os nós folha. | |
| C | Visita as chaves por ordem crescente. | ✔ |
| D | Visita primeiro a raiz, depois a sub-árvore esquerda e depois a sub-árvore direita. | |

Considere uma árvore de pesquisa binária inicialmente vazia. São inseridas, por esta ordem, as chaves:

5, 3, 8, 4

Qual das seguintes representações por níveis corresponde à árvore obtida? Use - para representar ausência de nó.

| | | |
|---|---|---|
| A | [5, 3, 8, -, 4, -, -] | ✔ |
| B | [5, 8, 3, -, -, 4, -] | |
| C | [3, -, 5, 4, 8, -, -] | |
| D | [5, 3, 8, 4, -, -, -] | |

Na remoção por cópia de um nó com dois filhos numa BST, usando o predecessor, qual é o procedimento correto?

| | | |
|---|---|---|
| A | Substituir a chave do nó pela menor chave da sub-árvore direita. | |
| B | Substituir a chave do nó pela maior chave da sub-árvore esquerda. | ✔ |
| C | Substituir diretamente o nó pela sua sub-árvore direita, ignorando a sub-árvore esquerda. | |
| D | Remover sempre a raiz da árvore e reconstruir a árvore de novo. | |

Considere uma B-Tree de ordem 3, em que cada nó pode conter no máximo 2 chaves.

Se, durante uma inserção, um nó passa temporariamente a conter 3 chaves, o que deve acontecer?

| | | |
|---|---|---|
| A | O nó é eliminado. | |
| B | A chave intermédia sobe para o nó pai e o nó é dividido. | ✔ |
| C | Todas as chaves são movidas para a raiz. | |
| D | A árvore deixa de ser uma B-Tree e deve ser convertida numa BST. | |

Considere uma tabela de dispersão de dimensão 11, com função de hash:

$h(x) = x \bmod 11$

As colisões são resolvidas por sondagem linear.

A tabela está inicialmente vazia.

Após inserir as chaves 14, 4 e 15, por esta ordem, em que posições ficam colocadas as três chaves?

| | | |
|---|---|---|
| A | $14 \rightarrow 3, 4 \rightarrow 4, 15 \rightarrow 5$ | ✔ |
| B | $14 \rightarrow 4, 4 \rightarrow 3, 15 \rightarrow 5$ | |
| C | $14 \rightarrow 3, 4 \rightarrow 4, 15 \rightarrow 4$ | |
| D | $14 \rightarrow 2, 4 \rightarrow 4, 15 \rightarrow 6$ | |

Qual das seguintes afirmações sobre o Merge Sort é correta?

| | | |
|---|---|---|
| A | Tem complexidade média $O(n^2)$ e não usa recursividade. | |
| B | Tem complexidade $O(n \log n)$ no pior caso e divide recursivamente o vetor em sub-vetores. | ✔ |
| C | Tem complexidade $O(\log n)$, porque divide sempre o vetor ao meio. | |
| D | É sempre mais eficiente em memória do que o Insertion Sort, pois não necessita de memória auxiliar. | |

**Secção 2** 2 de 3

3 perguntas 0 anotação

| | |
|---|---|
| Pergunta 1 | 0 de 1 |
| Pergunta 2 | 1 de 1 |
| Pergunta 3 | 1 de 1 |

## 2. Resposta curta e preenchimento

**2.1 - (1 valor)**

Considere o seguinte segmento de código:

```
int a = 0;
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= i; j++)
      a++;
```

Determine a expressão exata de *f(n)*, correspondente ao número de incrementos realizados sobre a variável a, e indique a respetiva complexidade na notação *O(.)*.

Justifique a sua resposta.

1. ¾ºTemos no:

- ciclo externo : o indice i comeca em 1 e incrementa de uma unidade ate chegar a n;$
- ciclo interno: para cada valor fixo de i, o indice j comeca em 1 e incrementa de uma em uma unidade ate chegar a i.

o numero de vezes que a operacao a++ é executada em cada iteracao do ciclo externo é determinado pelo limite superior do ciclo externo, entao somamos o numero de execucoes do ciclo interno para todos os valore de i de 1 ate n:

89 / 10000 Word Limit

**2.2 - (1 valor)**

Considere uma tabela de dispersão de dimensão 11, com índices de 0 a 10, inicialmente vazia.

A função de hash é: *h(x) = x mod 11*, e as colisões são resolvidas por sondagem linear.

Insira as seguintes chaves, pela ordem apresentada:
22, 1, 13, 24, 35, 2, 12

Preencha a tabela final e justifique os casos em que ocorreu colisão.

Vamos comecar vamos calcular a posicao de cada chave utilizando a funcao de dispersao base h(x) = x (mod11) e aplicando a sondagem linear em caso de colisao.

i 22: 22(mod11) = posicao 0 livre por isso vamos colocar la o chave 22

i1: 1(mod11) = 1 --> posicao 1 livre --> colocamos chave 1

i13: 13 (mod13)=2 --> livre

i24: 24(mod11) = 2 --> colisao, entao vamos sondagem linear com (2+1)(mod11)=3 --> posicao livre

i35: 35 mod 11 = 2 --> colisao, sondagem linear --> 2+1 mod 11= 3 --> ocupado --> entao 2+2 mod 11= 4 --> posicao livre

i3 : 2 mod 11 = 2 --> colisao --> temos de ir 2+3 mod 11= 5 --> posicao livre

i12: 12 mod 11= 1--> colisao --> temos de ir ate 1+5 mod 11 = 6 --> possicao livre

| indice | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| chave | 22 | 1 | 13 | 24 | 35 | 2 | 12 | - | - | - | - |

152 / 10000 Word Limit

**2.3 - (1 valor)**

Considere o vetor: [7, 3, 5, 2, 6]

Ordene o vetor por ordem crescente utilizando o algoritmo Insertion Sort.

Explique brevemente o funcionamento geral do algoritmo e indique o conteúdo do vetor após cada iteração principal do algoritmo, isto é, após inserir corretamente o elemento da posição atual na parte já ordenada do vetor.

Em cada iteracao i, o insertion sort, o elemento na posicao i é inseriro na posicao correta dentro do subvector 0..i-1 , ja ordenado e deslocando para a direita os elementos mais grandes.

comecemos para iteracao 1 - chave = 3 (i=1)

- 3<7--> entao desloca 7 e insere 3 na poicao 0 --> ficando assim [3,7,5,2,6]

iteracao 2 com chave 5 (i=2)

- 5<7 --> entao desloca 7; 5>3 --> entao vamos inserir 5 na posicao 1 ---> ficando assim [3,5,7,2,6]

iteracao 3 com a chave 2 (i=3)

- 2<7 --> desloca ; 2<5 --> desloca; 2<3 --> desloca e insere 2 na posicao 0 ---> ficando assim [2,3,5,7,6]

iteracao 4 com a chave 6 (i=4)

- 6<7 --> desloca ; 6 > 5 --> é inserido 6 na posicao 3 --> ficando assim [2,3,5,6,7]

127 / 10000 Word Limit

**Secção 3** 2.6 de 3

4 perguntas 0 anotação

| | |
|---|---|
| Pergunta 1 | 0.6 de 0.6 |
| Pergunta 2 | 0.9 de 0.9 |
| Pergunta 3 | 0.7 de 1.1 |
| Pergunta 4 | 0.4 de 0.4 |

# 3. Árvores e estruturas hierárquicas

**3.1**

Considere uma árvore de pesquisa binária inicialmente vazia.

Insira as seguintes chaves, pela ordem indicada: 10, 5, 15, 3, 7, 12, 18, 6, 8

**3.1.1 (0.6 valores) -** Desenhe a árvore obtida após todas as inserções. Justifique o raciocínio usado nas duas últimas inserções.

**Para desenhar as árvores, pode usar uma representação textual como a que se segue:**

```
  1
 / \
2   3
```

temos a árvore final assim:

```
          10
        /    \
       5      15
     /  \    /  \
    3    7  12  18
        / \
       6   8
```

Nas ultimas duas insercoes:

i 6: 6<10 --> vai para esq de 10 // 6>5 --> direita de 5 // 6<7 --> filho esq de 7

i 8: 8<10 --> vai para esq de 10 // 8>5 --> dir de 5 // 8>7 --> filho direito de 7

58 / 10000 Word Limit

**3.1.2 (0.9 valores) -** A partir da árvore obtida na alínea anterior, remova a chave 5 utilizando o algoritmo de **remoção por cópia**, recorrendo ao predecessor. Desenhe a árvore resultante e justifique os passos principais da remoção.

O no 5 tem dois filhos, pelo que se aplica remocao por copia.

A saber, que o predecessor é o maior elemento da subarvore esquerda de 5, sendo assim, é o no mais à direita a partir do filho esquerdo nomeadamente o 3.

Assim sendo:

- filho esquerdo de 5 --> no 3

- No de 3 nao tem filho direito --> o predecessor é o proprio 3

Entao vamos:

- copiar o valor do predecessor 3 para o no a remover, e sendo assim o no que era 5 passa a conter o valor 3.

- agora falta remover o no predecessor original (no 3), que é folha --> e assim basta desligar o ponteiro do pai.

Sendo assim, a arvore fica final assim:


129 / 10000 Word Limit

**3.2 -** Considere uma B-Tree de ordem 3 inicialmente vazia.

Admita que cada nó pode conter no máximo 2 chaves.

Insira as seguintes chaves, pela ordem apresentada: 10, 20, 5, 6, 12, 30, 17

**3.2.1 - (1,1 valores)**

Indique a árvore final após todas as inserções.

Sempre que ocorrer divisão de um nó, indique qual a chave que sobe e em que momento ocorre a divisão.

**Pode desenhar a árvore ou usar uma representação textual por níveis, por exemplo:**

**Raiz: [10]**

**Filhos: [5, 6] [12, 20]**

Numa arvore b-tree de ordem m=3, aplica-se a capacidade maxima de chaves por no: m-1=2haves ; e a capacidade minima por no com excepacao da raiz : m/2-1=1 chave.

entao temos a raiz 10:

[10]

inserir 20 : no com raiz de 2 chaves sem overflow

[10 | 20 ]

inserir 5: aqui vamos ter overflow, pois temos 3 chaves. [5 |10 | 20 ] , entao vamos fazer mediana --> 10 sobe para nova raiz

```
      [10]
     /    \
   [5]    [20]
```

inserir 6 : como 6<10 --> vai para 5, e nao existe overflow

```
      [10]
     /    \
  [5|6]    [20]
```

inserir 12 : como 12>10 --> vais para 20 e nao existe overflow

```
      [10]
     /    \
  [5|6]    [12|20]
```

inserir 30: como 30 > 10 --> vai para [12|20|30] --> temos overflow; entao vamos fazer medianda = 20 sobe para a raiz e fica [10|20]. As chaves 12 e 30 isolam-se em nos independentes.

```
      [10|20]
     /   |   \
  [5|6]  [12]  [30]
```

inserir 17: o 17 deve ser inserido no no do meio pois esta entre 10 e 20. O no do meio passa a ter as chaves [12|17], sendo assim nao ha divisoes.

```
      [10|20]
     /    |    \
  [5|6]  [12|17]  [30]
```

175 / 10000 Word Limit

**3.2.2 - (0,4 valores)**

A partir da árvore obtida na alínea anterior, remova a chave 17.

Indique a árvore resultante e justifique resumidamente a operação realizada.

A chave 17 é folha no no [12|17], entao a partir daqui devo verificar se a remocao viola ou nao o maximo.

Assim sendo, apos remover a chave 17 o no fica somente com uma chave, nomeadamente [12], pois preserva o numero minimo obrigatorio de elementos para a arvore que tinha sido pedido na questao anterior