DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2018-2019 - 2º Semestre
Algoritmos e Estruturas de Dados

2º Exame, 4 de Julho de 2019, 15h00m Duração: 3 horas

Prova escrita, individual e sem consulta

NOME:_____ NÚMERO:_____

PARTE I - Questões de Escolha Múltipla (A/B/C/D)

Preencha as respostas na tabela (usando apenas letras maiúsculas). Se nenhuma opção servir, escreva **NENHUMA**. Se pretender alterar a sua resposta, risque e escreva ao lado a sua nova opção. Todas as questões de escolha múltipla seguintes valem 0.75 valores. Estas questões de escolha múltipla não respondidas são cotadas com 0 valores, mas por cada resposta errada são descontados 0.75/4 valores.

Questão	1	2	3	4	5	6
Resposta						

1. No contexto do problema da conectividade, uma das tabelas abaixo não pode ter sido produzida por nenhum dos algoritmos que estudou nas aulas. Qual?

A.

0	12	4	0	4	0	4	12	6	5	10	6	12

B.

9	11	2	0	1	5	5	5	0	9	11	11	10

C.

3	7	7	5	1	0	6	7	8	6	0	8	8

D.

5	1	4	9	0	5	1	8	8	4	8	1	7

2. Considere a tabela de inteiros (na primeira linha da matriz abaixo) sobre a qual se aplicaram algumas iterações de um dos algoritmos de ordenação estudados (nas linhas seguintes). Qual o algoritmo usado?

10	18	19	16	19	14	12	14	21	13
10	18	19	16	19	14	12	14	21	13
10	12	19	16	19	14	18	14	21	13
10	12	14	16	19	14	18	19	21	13
10	12	14	16	19	14	18	19	21	13
10	12	14	16	13	14	18	19	21	19

A. Bubble
B. Inserção
C. Seleção
D. *Shellsort* ($h = 5, 3, 1$)

3. Ao aplicar o algoritmo de Dijkstra a um grafo ponderado, não direccionado, possuindo 15 vértices numerados de 0 a 14, obteve-se a sua árvore de caminhos mais curtos representada no vector `st` abaixo.

`st =`

14	2	10	13	10	13	12	14	2	13	-1	10	4	11	9

Qual das seguintes afirmações é falsa?

A. O vértice 9 está no caminho mais curto iniciado em 13 e terminado em 7.
B. O caminho mais curto entre os vértices 11 e 14 possui 4 passos.
C. O caminho mais curto entre os vértices 6 e 10 passa pelo vértice 4.
D. O vértice 12 não está no caminho mais curto entre os vértices 10 e 5.

4. Suponha que os seguintes números são as chaves de objectos introduzidos numa tabela de dispersão: 1139, 4314, 2043, 194, 1311, 763, 4314, 74. Assuma que a função de dispersão é $f(x) = x\%20$ e que as colisões são resolvidas por dispersão em lista, com inserção no final. Indique qual a afirmação verdadeira (contabilize comparações exclusivamente entre números).

A.	Ocorrem 3 colisões e 5 comparações	B.	Ocorrem 3 colisões e 6 comparações
C.	Ocorrem 4 colisões e 6 comparações	D.	Ocorrem 4 colisões e 5 comparações

5. Considere a seguinte recorrência

$$C_N = 3C_{N/3} + 2N$$

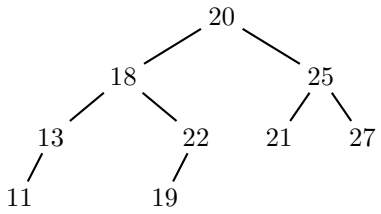
Qual dos seguintes conjuntos representa a complexidade associada com a recorrência indicada?

A.	$\mathcal{O}(2N + 3\lg(N))$	B.	$\mathcal{O}(N\lg(N))$	C.	$\mathcal{O}(N^2)$	D.	$\mathcal{O}(N^2\lg(N^3))$

6. Considere o código à esquerda, utilizado para o varrimento da árvore binária à direita, no qual a função `visit()` imprime o valor do nó visitado.

```
void traverse(link *h, void (*visit)(link))  {
  STACKinit(max);
  STACKpush(h);

  while (!STACKempty())  {
    (*visit)(h=STACKpop());
    if (h->r != NULL) STACKpush(h->r);
    if (h->l != NULL) STACKpush(h->l);
  }
}
```



Indique qual o resultado que se obtém no final da execução da função quando chamada com a raíz da árvore.

A.	11 13 19 22 18 21 27 25 20	B.	20 18 13 11 22 19 25 21 27
C.	11 13 18 19 22 20 21 25 27	D.	20 18 25 13 22 21 27 11 19

PARTE II - Questões de Escolha Binária (V/F)

Preencha as respostas na tabela (usando apenas letras maiúsculas – **V**(erdadeira) ou **F**(alsa)). Todas as questões de escolha múltipla seguintes valem 0.50 valores. Estas questões de escolha múltipla não respondidas ou erradas são cotadas com 0 valores.

Questão	7	8	9	10	11	12	13	14
Resposta								

7. As seguintes funções estão listadas por ordem crescente quanto ao seu crescimento assimptótico: $2\log N$, $N\log N$, 2^5, 1.1^N.

8. Numa máquina de 64 bits, em que os inteiros ocupam 32 bits, após a alocação dinâmica de uma matriz de 20×30 inteiros existe um total de 1344 bits ocupados com ponteiros.

9. Quando os dados numa tabela estão já ordenados ou quase ordenados, o algoritmo de ordenação *insertion* é mais eficiente que o algoritmo *selection*.

10. Ao aplicar DFS num dado grafo não direccionado, com implementação usando matriz de adjacências, produziu-se o seguinte caminho: `6-2-5-0-3-8`. Pode concluir-se que não existe a aresta $(3, 5)$.

11. Fazer N inserções ordenadas numa lista simplesmente ligada tem, em média, complexidade $\mathcal{O}(N^2)$, mas é $\mathcal{O}(N \log N)$ se se tratar de um lista duplamente ligada.

12. Num acervo, para qualquer vértice que tenha dois filhos, o filho esquerdo, como nas árvores ordenadas, tem menor prioridade que o filho direito.

13. *Divide-and-conquer* é a abordagem indicada para problemas em que a decomposição produz problemas independentes, enquanto a programação dinâmica se aplica a casos em que os sub-problemas possuem interdependências.

14. Em árvores ordenadas, balanceadas AVL, a complexidade das operações de rotação simples é sempre $\mathcal{O}(1)$, mas a complexidade das operações de rotação dupla depende da altura do vértice onde se efectua a rotação principal.

PARTE III - Questões de Desenvolvimento

Responda a cada uma das questões de desenvolvimento em **folhas de exame separadas** e devidamente identificadas com nome e número.

[4.5] 15. Considere um grafo ponderado não direccionado com 13 vértices, identificados de 0 a 12, representado pela matriz de adjacências indicada abaixo. A ausência de valor para um par linha e coluna significa ausência de aresta entre esses dois vértices, i.e., custo ∞.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0				15				12			8	
1		0		7		20				3			2
2			0				5		9		10	10	
3		7		0		7				14			9
4	15				0		17		22			6	
5		20		7		0				13			6
6			5		17		0	11			9		
7							11	0	2			14	
8	12		9		22			2	0		23		
9		3		14		13				0			4
10			10				9		23		0	5	
11	8		10		6			14			5	0	
12		2		9		6				4			0

[2.5] a) Construa a árvore mínima de suporte deste grafo fazendo uso do algoritmo de Kruskal. Para facilitar a compreensão da sua resolução, sugere-se que comece por apresentar o vector de arestas deste grafo, aplicando depois o algoritmo sobre esse vector de arestas. Na apresentação do vector de arestas, para além dos vértices que une, não se esqueça de indicar o peso de cada aresta. Explique as razões de inserção e/ou exclusão de cada uma das arestas que tiver de analisar para a construção da árvore mínima de suporte.

[1.0] b) Trace a árvore obtida na alínea anterior, indicando o seu custo total. Descreva e explique a natureza dos resultados obtidos.

[1.0] c) Indique, de forma justificada, qual o valor lógico da seguinte afirmação: o algoritmo de Prim, para cálculo de árvores mínimas de suporte, e o algoritmo de Kruskal, para o mesmo problema, podendo produzir árvores diferentes ao escolherem conjuntos diferentes de arestas, obtêm sempre árvores com o mesmo custo total.

Responda em folhas separadas. Não se esqueça de mudar de folha

[4.5] 16. Suponha que se registam troços de um dado sinal, colocando cada troço numa linha de uma matriz bi-dimensional. Cada linha corresponde a um registo do sinal em intervalos de tempo de igual comprimento, para diferentes instantes iniciais. O sinal amostrado e registado na matriz é tal que, para L linhas todas com C amostras, em cada linha i existe uma coluna $c(i)$, $0 \leq c(i) < C$ tal que o sinal é crescente até $c(i)$ sendo decrescente a partir daí até ao final da linha. Ou seja, cada linha contém um troço de um dente de serra. Pretende-se desenvolver um algoritmo eficiente para determinar o pico mais alto em toda a matriz. Ou seja, sendo A a matriz, pretende-se calcular $\max_i\{A[i][c(i)]\}$ (note que o máximo em cada linha pode estar num dos extremos da linha). Para resolver este problema foi proposto o seguinte par de funções:

```
float max_in_matrix(float ** A, int L, int C)
{
 int l;
 float Max_in_line, Max;
 Max = max_in_vector(A[0], C);
 for (l=1, l < L; l++) {
  Max_in_line = max_in_vector(A[l], C);
  if (Max < Max_in_line) Max = Max_in_line;
 }
 return Max;
}
```

```
float max_in_vector(float * A, int C)
{
 int c;
 float Max = A[0];
 for (c = 1; c < C; c++) {
  if (Max > A[c]) break;
  Max = A[c];
 }
 return Max;
}
```

[1.0] a) Indique, como função de L e C, qual a complexidade computacional da função `max_in_matrix(.)`, tal como está proposta acima. Para ser considerada completa, a sua análise deverá estar devidamente fundamentada e justificada.

[2.5] b) Proponha e implemente em C uma versão mais eficiente para resolver o problema proposto. Deverá começar por explicar como vai alterar a abordagem ao problema, apresentando de seguida o código que concretiza as ideias por si propostas.

Nota: Quando se pede uma solução mais eficiente não se está à espera de uma simples redução da constante multiplicativa.

[1.0] c) Discuta a complexidade da solução por si proposta, mesmo que não a tenha implementado. Tal como na alínea a), a sua análise de complexidade computacional deverá ser acompanhada da devida justificação.

Responda em folhas separadas. Não se esqueça de mudar de folha

[2.5] 17. Neste exercício pretende-se construir uma árvore ordenada. Assuma uma árvore inicialmente vazia onde se inserem os valores seguintes, pela ordem apresentada: 74, 92, 120, 82, 79, 42, 87, 20 e 90.

Após cada inserção a árvore deverá ser balanceada AVL. Assim, para cada nova inserção deverá indicar se a árvore permanece balanceada ou não. Nos casos em que não fique balanceada, deverá indicar qual a rotação a fazer – se simples, se dupla, se à direita, se à esquerda e em que vértice se realiza –, e executar a rotação por si indicada.

Na sua resolução deverá apresentar a árvore imediatamente após cada inserção e, quando tal for o caso, apresentar a árvore que resulta da rotação executada. Só depois deverá prosseguir com as restantes inserções.