DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2016-2017 - 2º Semestre
Algoritmos e Estruturas de Dados

1º Exame, 22 de Junho de 2017, 18h30m 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. Considere o acervo abaixo, em que menor valor corresponde a maior prioridade.

17	23	37	31	43	59	67	53	71	89	97

Qual das seguintes tabelas representa o acervo que resulta da aplicação das seguintes operações: inserir um novo elemento de prioridade 26; remover o elemento de maior prioridade.

A.

23	31	26	53	43	37	67	59	71	89	97

B.

23	31	26	53	43	37	67	71	59	89	97

C.

23	31	26	53	43	37	67	59	71	97	89

D.

23	31	26	59	43	37	67	53	71	89	97

2. Considere uma tabela de dispersão, inicialmente vazia, para armazenar objectos de chave inteira. A função de dispersão usada é o resto da divisão por 100 e as colisões resolvem-se por dispersão em lista, com inserção no início. Indique quantas colisões e quantas comparações se realizam para a seguinte sequência de entrada: 327, 414, 927, 633, 1654, 3814, 327, 833, 414, 1127.

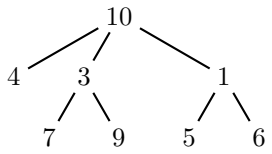
A. 5 colisões e 9 comparações
B. 6 colisões e 10 comparações
C. 7 colisões e 9 comparações
D. 6 colisões e 9 comparações

3. Para a função recursiva apresentada à direita, assumindo que a tabela passada como argumento está ordenada de forma crescente, qual das recorrências melhor descreve a complexidade temporal da função?

```
int walse(float * v, int N)
{
 if (N == 0) return 1;
 if (v[0] > v[N-1])
  return 1 + walse(v, N/2);
 else
  return N + walse(v, N/2)*walse(v, N/4);
}
```

A. $C_N = 3C_{N/4} + 1$
B. $C_N = C_{N/2} * C_{N/4} + N$
C. $C_N = C_{N/2} + C_{N/4} + 1$
D. $C_N = 3C_{N/4} + N$

4. Na árvore à esquerda, indique qual das opções é verdadeira:



A.	A procura DFS, na árvore segue: 10, 4, 3, 1, 7, 9, 5, e 6;
B.	A procura BFS, na árvore segue: 10, 4, 3, 1, 7, 9, 5, e 6;
C.	A procura BFS, na árvore segue: 10, 4, 3, 7, 9, 1, 5, e 6;
D.	A procura DFS, na árvore segue: 4, 7, 9, 3, 5, 6, 1, e 10.

5. Considere a seguinte tabela de conectividade, obtida através do algoritmo de procura rápida:

$id[.] =$

0	1	2	9	9	2	0	7	8	9

ao qual se pretendem adicionar as ligações em baixo, pela ordem indicada:

$$(6,7), \quad (1,0), \quad (3,7), \quad \text{e} \quad (1,5).$$

Suponha que só tem permissões para realizar 6 alterações na tabela de conexões, isto é, só consegue alterar 6 valores da tabela. Indique quantas ligações consegue incluir:

A. 1	B. 2	**C.** 3	D. 4

6. A função, à direita, representa um algoritmo de ordenação de um vector `Item a[]`, com limites à esquerda e direita identificados pelas variáveis `int l` e `int r`, respectivamente. Analise o código e identifique qual das seguintes afirmações é verdadeira. Considere para os devidos efeitos $N = R - L + 1$:

```
void sort(Item a[], int l, int r)
{
  int j;
  for (j = r; j > l; j--)
    if (less(a[j], a[j-1])
      exch(a[j-1], a[j]);
  if (++l < r) sort(a, l, r);
}
```

A.	Representa o algoritmo *Bubble Sort*.	B.	Em média, possui complexidade $\mathcal{O}(N\lg N)$.
C.	Representa o algoritmo *Insertion Sort*.	D.	Implica uma recorrência de $C_N = C_{N-1} + 1$.

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								

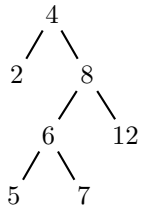
7. Seja a função `heap_search` com dois argumentos: um ponteiro para um acervo de tamanho N e um inteiro, k. A função devolve a posição em que fica o primeiro elemento encontrado que possua prioridade igual a k, ou -1 se nenhum dos elementos possuir essa prioridade. A complexidade desta função é $\mathcal{O}(\lg N)$. (**F**)

8. Determinar se dois vértices são adjacentes, em pior caso, possui complexidade $\mathcal{O}(1)$ quando o grafo é representado por matriz de adjacências. (**V**)

9. Numa tabela de dispersão de tamanho M, contendo N elementos, em que $N > M$ e as colisões se resolvem por dispersão em lista, o tempo médio de procura por um elementos é $\mathcal{O}(\log N)$. (**F**)

10. $N^{5/3} \notin \mathcal{O}(N^2 \lg N)$. (**F**)

11. A árvore binária AVL à esquerda, está ordenada. (**F**)



12. No pior dos casos, o número de operações necessárias para procura, usando o método *Quick Union*, e para a união, usando o método *Quick Find*, é igual e tem o valor de $\mathcal{O}(N^2)$, para N pares de novas ligações. (**V**)

13. Quando, à partida, são conhecidas as dimensões do problema deve usar-se tabelas porque, apesar destas necessitarem de mais memória que as listas, o seu acesso é mais rápido. (**F**)

14. O cabeçalho da função, em baixo, representa um método para inserção de elementos em pilha, do tipo genérico, que permite a possibilidade de um programa cliente possuir mais que uma pilha em simultâneo. (**V**)

```
pilha* Push(pilha* gp, int item);
```

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.

[5.0] 15. Considere um grafo ponderado não direccionado com 15 vértices, identificados de 0 a 14, 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	13	14
0	0		14		15				12				8		
1		0		7		20				3				16	
2	14		0		24		5				18	10			9
3		7		0		28				11				2	
4	15		24		0		6	26				23	4		
5		20		28		0				21				1	
6			5		6		0								
7					26			0					34		
8	12								0		31		33		
9		3		11		21				0				35	
10			18						31		0				17
11			10		23							0			36
12	8				4			34	33				0		
13		16		2		1				35				0	
14			9								17	36			0

[2.00] a) Determine a árvore de mínima de suporte (MST) usando o algoritmo de Prim e tomando o vértice **4** como fonte. Apresente os seus cálculos de forma clara, detalhada e completa para cada iteração do algoritmo. Por exemplo, mas sem se restringir a estes aspectos, identifique a franja da procura e pesos, assim como deverá indicar por que ordem entra cada vértice na árvore e o estado da franja em cada momento.
Resolução:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	15/4	∞	24/4	∞	-	∞	6/4	26/4	∞	∞	∞	23/4	**4/4**	∞	∞
12	8/12	∞	24/4	∞	-	∞	**6/4**	26/4	33/12	∞	∞	23/4	-	∞	∞
6	8/12	∞	**5/6**	∞	-	∞	-	26/4	33/12	∞	∞	23/4	-	∞	∞
2	**8/12**	∞	-	∞	-	∞	-	26/4	33/12	∞	18/2	10/2	-	∞	9/2
0	-	∞	-	∞	-	∞	-	26/4	12/0	∞	18/2	10/2	-	∞	**9/2**
14	-	∞	-	∞	-	∞	-	26/4	12/0	∞	17/14	**10/2**	-	∞	-
11	-	∞	-	∞	-	∞	-	26/4	**12/0**	∞	17/14	-	-	∞	-
8	-	∞	-	∞	-	∞	-	26/4	-	∞	**17/14**	-	-	∞	-
10	-	∞	-	∞	-	∞	-	**26/4**	-	∞	-	-	-	∞	-
7	-	∞	-	∞	-	∞	-	-	-	∞	-	-	-	∞	-

Justificação dos cálculos:

Como o vértice de partida é o 4, a primeira linha da tabela apresenta as entradas da matriz de adjacências associadas com o vértice 4 como ponto de partida. O segundo vértice a entrar na MST é o 12 por ser, de todos os adjacentes do vértice 4, aquele que fica mais próximo.

A segunda linha apresenta a actualização da franja de procura decorrente da entrada do vértice 12. Concretamente, o vértice 0 está mais próximo do vértice 12 do que do vértice 4 e o vértice 8, que não é visível a partir do vértice 4, é adjacente do vértice 12. As restantes linhas da tabela repetem este tipo de procedimento de escolha para entrar do vértice que esteja mais próximo da MST já construída e consequente actualização da franja.

O algoritmo pára após a entrada do vértice 7, dado que nenhum dos restantes vértices que ainda estão fora da MST possui qualquer aresta para nenhum dos 10 vértices que estão na MST.

[2.00] b) Fazendo agora uso do algoritmo de Kruskal, volte a determinar a MST do grafo acima. Seja claro e preciso na explicitação dos cálculos e decisões tomadas ao longo da aplicação do algoritmo. Se entender útil, comece por apresentar as arestas por ordem crescente do seu peso, no formato $[v, w, w_t]$, e sem repetições. No final do exercício deverá ser claro, para o docente avaliador, quais as arestas que fazem parte da MST final e porquê.

Resolução:

A lista de arestas ordenada é: [5, 13, 1], [3, 13, 2], [1, 9, 3], [4, 12, 4], [2, 6, 5], [4, 6, 6], [1, 3, 7], [0, 12, 8], [2, 14, 9], [2, 11, 10], [3, 9 , 11], [0, 8, 12], [0, 2, 14], [0, 4, 15], [1, 3, 16], [10, 14, 17], [2, 10, 18], [1, 5, 20], [5, 9, 21], [4, 11, 23], [2, 4, 24], [4, 7, 26], [3, 5, 28], [8, 10, 31], [8, 12, 33], [7, 12, 34], [9, 13, 35], [11, 14, 36].

Vou representar os conjuntos de vértices ligados, omitindo os conjuntos que apenas contêm um vértice.

A primeia aresta faz parte da MST, pelo que passa a haver o conjunto {5, 13}. A segunda aresta liga 3 e 13 que fazem parte de conjuntos diferentes, por isso a aresta faz parte da MST, passando a haver o conjunto {3, 5, 13}.

A terceira aresta liga elementos de conjuntos diferentes, pelo que entra. Após a sua entrada temos: {1, 9} e {3, 5, 13}.

A quarta aresta liga os vértices 4 e 12, que ainda estão em conjuntos diferentes. Passamos a ter três conjuntos com mais que um elemento: {1, 9}, {4, 12} e {3, 5, 13}.

Este procedimento prossegue de modo similar, sendo que as arestas, que após a quarta acima referida, entram são: quinta, sexta, sétima, oitava, nona e décima. A décima primeira aresta não entra, dado que liga os vértices 3 e 9 e estes estão no mesmo conjunto desde a entrada da sétima aresta.

As arestas seguintes entram pela ordem seguinte: [0, 8, 12], [10, 14, 17] e [4, 7, 26].

Após a inserção da aresta [4, 7, 26], apesar se ainda não terem entrado 14 arestas (o grafo possui 15 vértices), todas as restantes ligam vértices que já estão no mesmo conjunto.

Os dois conjuntos finais são: {1, 3, 5, 9, 13} e {0, 2, 4, 6, 7, 8, 10, 11, 12, 14}.

Ou seja, o algoritmo de Kruskal produziu duas MST's.

[1.00] c) Explique as razões que, no seu entender, justificam eventuais diferenças entre as duas árvores produzidas nas alíneas anteriores. Retire, e explicite, conclusões desta sua análise.

Resolução:

O algoritmo de Prim calcula uma MST usando um vértice de partida. Se o grafo for ligado, o algoritmo de Prim produz a MST desse grafo. Se o grafo não for ligado, o algoritmo de Prim apenas produz a MST associada ao sub-grafo a que o vértice de partida pertence. Já o algoritmo de Kruskal determina florestas mínimas de suporte. Essa é a razão que justifica que a primeira alínea tenha produzido uma MST e a segunda alínea tenha produzido duas MST's.

[2.0] 16. Considere a árvore representada por:

```
23
  10
    1
    20
      15
      22
  40
    35
    43
      42
```

obtida através do seu varrimento *Pré-Fixado*, em que os espaços (*tabs*) representam os diferentes níveis da árvore. Pretende-se introduzir uma valor inteiro na árvore indicada. Para isso, responda às seguintes questões:

[0.75] a) Indique qual a árvore que originou o varrimento apresentado em cima.

[0.5] b) Indique, justificando, se a árvore está ordenada e/ou balanceada.

[0.75] c) Supondo que se procede à inserção do número 41 na árvore, indique uma possível solução para a mesma, em que esta esteja balanceada. Caso sejam necessárias transformações na árvore inicial, indique, justificando, quais as respectivas alterações.

[4.5] 17. Suponha que se pretende resolver o seguinte problema numérico em tabelas: dadas duas tabelas, contendo números reais, possivelmente de diferentes tamanhos, pretende-se saber quantos quadrados de números da primeira tabela estão contidos na segunda. Por exemplo, se a primeira tabela for $[2.5, 3.0, 1.1, -1.5]$ e a segunda tabela for $[1.0, 9.2, 1.21, 2.25, 0.04]$, a resposta deverá ser 2, dado que o número $1.21 = 1.1^2$ e $2.25 = (-1.5)^2$. O objectivo deste exercício é que implemente esta funcionalidade. Para isso, deverá seguir os passos que abaixo se indicam. Pode assumir que não existem repetições na primeira tabela.

(1.5) (a) Proponha um algoritmo para resolver este problema, indicando se é ou não necessário executar algum pré-processamento de alguma ou ambas as tabelas. A sua resposta deve ser clara e sem ambiguidades. Sugere-se que recorra ao traçado de um flusograma e/ou que apresente pseudo-código para ilustrar como deverá funcionar o algoritmo.

Nota: A cotação será atribuída em função da eficiência da solução proposta, pelo que deverá procurar a mais eficiente solução para este problema.

(1.0) (b) Assumindo que a primeira tabela tem tamanho N_1 e a segunda tabela tem tamanho N_2, discuta de forma fundamentada a complexidade temporal da solução proposta por si. Naturalmente que a sua resposta deverá ser dada como uma função de ambos os tamanhos, não podendo assumir, na sua análise, qualquer espécie de relação entre N_1 e N_2.

(2.0) (c) Escreva em linguagem C a função `int count_squares(double * v1, int n1, double * v2, n2);`, em que o argumento `n1` explicita o tamanho da tabela `v1` e o argumento `n2` o tamanho da tabela `v2`. Confirme que o código por si desenvolvido, possui de facto a complexidade demonstrada anteriormente e ilustre o seu funcionamento através de um exemplo.

Nota: se necessitar alguma funcionalidade acessória para tratamento de dados pode assumir que essas funções estão disponíveis, bastando apenas indicar o seu nome e o que é suposto fazerem. Por exemplo (silly example, for illustrative purposes only), se achar que precisa aplicar o algoritmo de Prim em algum passo da sua função, pode invocar a função *prim*(....), assumindo que ela existe e definindo os parâmetros que seria necessário passar-lhe para produzir o resultado desejado e qual o formato em que esse resultado seria devolvido e para onde.