DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2019-2020 - 1º Semestre
Algoritmos e Estruturas de Dados

2^o Exame, 3 de Fevereiro de 2020, 11h30m 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. Qual das seguintes tabelas não resulta de uma execução do algoritmo da união rápida equilibrada.

A.	| 0 | 1 | 2 | 2 | 2 | 2 | 4 | 5 | 0 | 0 |	B.	| 7 | 5 | 3 | 3 | 4 | 3 | 6 | 7 | 8 | 9 |
C.	| 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |	D.	| 1 | 2 | 9 | 9 | 4 | 5 | 6 | 7 | 4 | 9 |

A.	| 1 | 1 | 2 | 3 | 3 | 3 | 3 | 5 | 6 | 1 |	B.	| 0 | 8 | 6 | 4 | 4 | 5 | 4 | 7 | 8 | 9 |
C.	| 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |	D.	| 0 | 2 | 3 | 0 | 0 | 5 | 6 | 7 | 8 | 5 |

A.	| 2 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 6 | 7 |	B.	| 0 | 1 | 9 | 7 | 5 | 5 | 6 | 5 | 8 | 9 |
C.	| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |	D.	| 6 | 1 | 3 | 4 | 1 | 1 | 6 | 7 | 8 | 9 |

A.	| 4 | 2 | 2 | 3 | 2 | 5 | 6 | 7 | 8 | 6 |	B.	| 0 | 1 | 1 | 1 | 1 | 3 | 4 | 9 | 9 | 9 |
C.	| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |	D.	| 1 | 8 | 8 | 3 | 4 | 5 | 6 | 3 | 8 | 0 |

2. Dada uma tabela com N inteiros, se pretendermos determinar se há elementos repetidos na tabela, qual das afirmações abaixo está correta?

A. A forma mais eficiente de descobrir se há números repetidos é ordenar a tabela, com custo $\mathcal{O}(N^2)$, e percorrê-la, procurando números idênticos em posições consecutivas.
B. É possível fazer a operação sugerida em A em tempo $\mathcal{O}(N)$ mas requer memória adicional.
C. É possível obter o resultado indicado em B, sem memória adicional.
D. Se transformar a tabela num heap (`heapify()`) podemos retirar os elementos do *heap* um a um, por ordem, e ver se há repetidos, com custo total $\mathcal{O}(N)$.

A. Se transformar a tabela num heap (`heapify()`) podemos retirar os elementos do *heap* um a um, por ordem, e ver se há repetidos, com custo total $\mathcal{O}(N)$.
B. A forma mais eficiente de descobrir se há números repetidos é ordenar a tabela, com custo $\mathcal{O}(N^2)$, e percorrê-la, procurando números idênticos em posições consecutivas.
C. É possível fazer a operação sugerida em B em tempo $\mathcal{O}(N)$ sem memória adicional.
D. É possível obter o mesmo resultado que indicado em C mas requer memória adicional.

A. A forma mais eficiente de descobrir se há números repetidos é ordenar a tabela, com custo $\mathcal{O}(N^2)$, e percorrê-la, procurando números idênticos em posições consecutivas.
B. Se transformar a tabela num heap (`heapify()`) podemos retirar os elementos do *heap* um a um, por ordem, e ver se há repetidos, com custo total $\mathcal{O}(N)$.
C. É possível fazer a operação sugerida em A em tempo $\mathcal{O}(N)$ mas requer memória adicional.
D. É possível obter o resultado indicado em C, sem memória adicional.

A. Se transformar a tabela num heap (`heapify()`) podemos retirar os elementos do *heap* um a um, por ordem, e ver se há repetidos, com custo total $\mathcal{O}(N)$.
B. A forma mais eficiente de descobrir se há números repetidos é ordenar a tabela, com custo $\mathcal{O}(N^2)$, e percorrê-la, procurando números idênticos em posições consecutivas.
C. É possível fazer a operação sugerida em B, sem memória adicional, em tempo $\mathcal{O}(N)$.
D. É necessário usar memória adicional para obter o resultado indicado em C.

3. Considere a função à esquerda e indique qual dos conjuntos indicados à direita não majora a complexidade temporal da função.

```
void f(int N)
{
 for (int j=0; j<N; j++)
   ;
}
```

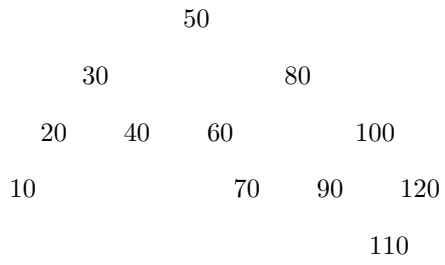
A. $\mathcal{O}(N^2)$
B. $\mathcal{O}(2^N)$
C. $\mathcal{O}(\log N)$
D. $\mathcal{O}(N)$

4. Considere a função à esquerda e indique qual dos conjuntos indicados à direita não majora a complexidade temporal da função.

```
void f(int N)
{
 for (int j=0; j<N; j++)
   ;
}
```

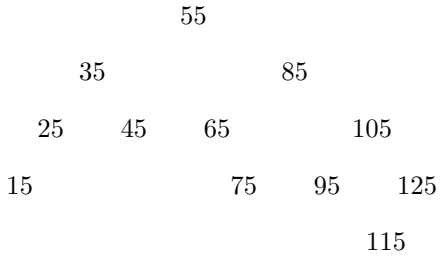
A. $\mathcal{O}(N^{\frac{1}{2}})$
B. $\mathcal{O}(N)$
C. $\mathcal{O}(N^2)$
D. $\mathcal{O}(2^N)$

5. Dada a árvore binária que abaixo se apresenta, qual das seguintes afirmações é falsa?



A. O varrimento in-fixado produz output crescente.
B. A árvore é ordenada mas não é balanceada AVL.
C. A árvore é ordenada.
D. A inserção de um elemento de chave 85 preserva o balanceamento.

6. Dada a árvore binária que abaixo se apresenta, qual das seguintes afirmações é falsa?



A.	O varrimento in-fixado produz output crescente.
B.	A árvore é ordenada mas não é balanceada AVL.
C.	A árvore é ordenada.
D.	A inserção de um elemento de chave 85 preserva o balanceamento.

7. Sejam f e g duas funções de variável natural, $N \geq 0$. Diga qual a opção correcta.

A.	$f \in \mathcal{O}(g) \Rightarrow g \in \Theta(f)$	B.	$f \in \Omega(g) \Rightarrow f \in \mathcal{O}(g)$
C.	$f \in \Theta(g) \Rightarrow g \in \Theta(f)$	D.	$f \in \mathcal{O}(g) \Rightarrow f \in \Omega(g)$

A.	$f \in \mathcal{O}(g) \Rightarrow f \in \Omega(g)$	B.	$f \in \mathcal{O}(g) \Rightarrow g \in \Theta(f)$
C.	$f \in \Omega(g) \Rightarrow f \in \mathcal{O}(g)$	D.	$f \in \Theta(g) \Rightarrow g \in \Theta(f)$

A.	$f \in \Theta(g) \Rightarrow g \in \Theta(f)$	B.	$f \in \mathcal{O}(g) \Rightarrow f \in \Omega(g)$
C.	$f \in \mathcal{O}(g) \Rightarrow g \in \Theta(f)$	D.	$f \in \Omega(g) \Rightarrow f \in \mathcal{O}(g)$

A.	$f \in \mathcal{O}(g) \Rightarrow f \in \Omega(g)$	B.	$f \in \mathcal{O}(g) \Rightarrow g \in \Theta(f)$
C.	$f \in \Omega(g) \Rightarrow f \in \mathcal{O}(g)$	D.	$f \in \Theta(g) \Rightarrow g \in \Theta(f)$

8. Aplicou-se o algoritmo de ordenação "shellsort" à tabela seguinte: | 19 | 12 | 13 | 18 | 20 | 11 | 17 | 15 | 14 | 16 |. Qual das sequências de h é tal que para o segundo valor de h não há qualquer troca?

A.	$h = 6, 4, 1$	B.	$h = 5, 3, 1$	C.	$h = 5, 4, 1$	D.	$h = 6, 3, 1$

9. Aplicou-se o algoritmo de ordenação "shellsort" à tabela seguinte: | 29 | 22 | 23 | 28 | 30 | 21 | 27 | 25 | 24 | 26 |. Qual das sequências de h é tal que para o segundo valor de h não há qualquer troca?

A.	$h = 5, 3, 1$	B.	$h = 5, 4, 1$	C.	$h = 6, 3, 1$	D.	$h = 6, 4, 1$

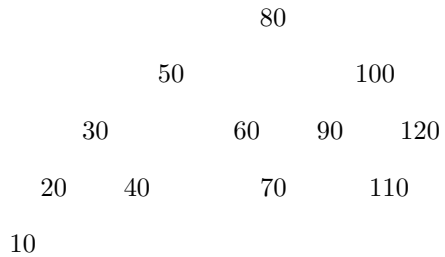
10. Aplicou-se o algoritmo de ordenação "shellsort" à tabela seguinte: | 39 | 32 | 33 | 38 | 40 | 31 | 37 | 35 | 34 | 36 |. Qual das sequências de h é tal que para o segundo valor de h não há qualquer troca?

A.	$h = 5, 4, 1$	B.	$h = 6, 3, 1$	C.	$h = 6, 4, 1$	D.	$h = 5, 3, 1$

11. Aplicou-se o algoritmo de ordenação "shellsort" à tabela seguinte: | 49 | 42 | 43 | 48 | 50 | 41 | 47 | 45 | 44 | 46 |. Qual das sequências de h é tal que para o segundo valor de h não há qualquer troca?

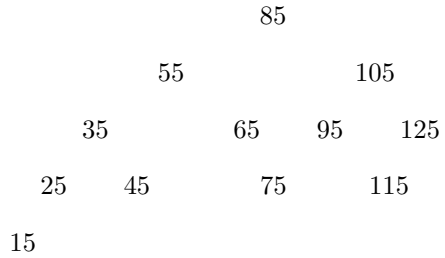
A.	$h = 6, 3, 1$	B.	$h = 6, 4, 1$	C.	$h = 5, 3, 1$	D.	$h = 5, 4, 1$

12. Dada a árvore binária que abaixo se apresenta, qual das seguintes afirmações é falsa?



A.	O varrimento in-fixado produz output crescente.
B.	A árvore é ordenada mas não é balanceada AVL.
C.	A árvore é ordenada.
D.	A inserção de um elemento de chave 85 preserva o balanceamento.

13. Dada a árvore binária que abaixo se apresenta, qual das seguintes afirmações é falsa?



A.	O varrimento in-fixado produz output crescente.
B.	A árvore é ordenada mas não é balanceada AVL.
C.	A árvore é ordenada.
D.	A inserção de um elemento de chave 85 preserva o balanceamento.

14. Considere a função à esquerda e indique qual dos conjuntos indicados à direita não majora a complexidade temporal da função.

```
void f(int N)
{
 for (int j=0; j<N; j++)
   ;
}
```

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

15. Considere a função à esquerda e indique qual dos conjuntos indicados à direita não majora a complexidade temporal da função.

```
void f(int N)
{
 for (int j=0; j<N; j++)
   ;
}
```

A.	$\mathcal{O}(N)$
B.	$\mathcal{O}(N^2)$
C.	$\mathcal{O}(2^N)$
D.	$\mathcal{O}(N^{\frac{1}{2}})$

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

Preencha as respostas na tabela (usando apenas letras maiúsculas – **V**(erdadeira) ou **F**(alsa)). Cada questão de escolha binária vale 0.50 valores. As questões não respondidas ou erradas são cotadas com 0 valores.

Questão	7	8	9	10	11	12	13	14
Resposta								

16. A complexidade da operação de união do algoritmo da união rápida não é pior que a complexidade da operação de procura no algoritmo da procura rápida.

17. Com um tipo de dados abstracto, se for alterada apenas a implementação interna será necessário actualizar a definição da respectiva interface para o cliente.

18. Considere dois algoritmos que resolvem o mesmo problema. O algoritmo que corre em $\mathcal{O}(N^2)$ é sempre mais rápido do que o que corre em $\mathcal{O}(N^3)$.

19. Numa árvore binária, se todos os elementos da sub-árvore esquerda forem menores que a raíz e todos os elementos da sub-árvore direita forem maiores que a raíz, então a árvore diz-se ordenada.

20. Seja G um grafo ponderado não direccionado e seja T o conjunto de arestas que constitui a sua árvore mínima de suporte. Seja agora G' um grafo com os mesmos vértices e arestas que G, mas todas as arestas de G' são $c > 0$ mais caras que as arestas de G. T também é a árvore mínima de suporte de G'.

21. Se no algoritmo de Kruskal para determinação da árvore mínima de suporte de um grafo, não ordenarmos as arestas mas fizermos todos os restantes passos, obtemos ainda assim uma árvore de suporte.

22. No algoritmo da união rápida ponderada com compressão de caminho, se p e q estiverem no mesmo conjunto, então os seus `id[.]` são iguais.

23. Considerando a complexidade temporal, o caso médio do algoritmo de ordenação "selection" é idêntico ao pior caso.

24. No algoritmo da união rápida ponderada com compressão de caminho, se p e q estiverem no mesmo conjunto, então os seus `id[.]` são iguais.

25. Considerando a complexidade temporal, o caso médio do algoritmo de ordenação "selection" é idêntico ao pior caso.

26. A complexidade da operação de união do algoritmo da união rápida é melhor que a complexidade da operação de procura no algoritmo da procura rápida.

27. Com um tipo de dados abstracto, se for alterada apenas a implementação interna não será necessário actualizar a definição da respectiva interface para o cliente.

28. Considerando a complexidade temporal, o caso médio do algoritmo de ordenação "selection" é idêntico ao pior caso.

29. Considere dois algoritmos que resolvem o mesmo problema. O algoritmo que corre em $\mathcal{O}(N^2)$ pode ser mais lento que o que corre em $\mathcal{O}(N^3)$.

30. Numa árvore binária ordenada o elemento que possua a maior chave não pode ter filhos à sua esquerda.

31. A execução do passo inicial do algoritmo de Kruskal, a ordenação das arestas por ponderação crescente, não é essencial para que a árvore de suporte a que se chega seja mínima.

32. Considerando a complexidade temporal, o caso médio do algoritmo de ordenação "selection" é idêntico ao pior caso.

33. Considere dois algoritmos que resolvem o mesmo problema. O algoritmo que corre em $\mathcal{O}(N^2)$ é sempre mais rápido do que o que corre em $\mathcal{O}(N^3)$.

34. A complexidade da operação de união do algoritmo da união rápida não é pior que a complexidade da operação de procura no algoritmo da procura rápida.

35. Com um tipo de dados abstracto, se for alterada apenas a implementação interna será necessário actualizar a definição da respectiva interface para o cliente.

36. Seja G um grafo ponderado não direccionado e seja T o conjunto de arestas que constitui a sua árvore mínima de suporte. Seja agora G' um grafo com os mesmos vértices e arestas que G, mas todas as arestas de G' são $c > 0$ mais caras que as arestas de G. T também é a árvore mínima de suporte de G'.

37. No algoritmo da união rápida ponderada com compressão de caminho, se p e q estiverem no mesmo conjunto, então os seus `id[.]` são iguais.

38. Numa árvore binária, se todos os elementos da sub-árvore esquerda forem menores que a raíz e todos os elementos da sub-árvore direita forem maiores que a raíz, então a árvore diz-se ordenada.

39. Se no algoritmo de Kruskal para determinação da árvore de suporte mínimo de um grafo, não ordenarmos as arestas mas fizermos todos os restantes passos, obtemos ainda assim uma árvore de suporte.

40. Considerando a complexidade temporal, o caso médio do algoritmo de ordenação "selection" é idêntico ao pior caso.

41. A complexidade da operação de união do algoritmo da união rápida é melhor que a complexidade da operação de procura no algoritmo da procura rápida.

42. Considere dois algoritmos que resolvem o mesmo problema. O algoritmo que corre em $\mathcal{O}(N^2)$ pode ser mais lento que o que corre em $\mathcal{O}(N^3)$.

43. No algoritmo da união rápida ponderada com compressão de caminho, se p e q estiverem no mesmo conjunto, então os seus `id[.]` são iguais.

44. Com um tipo de dados abstracto, se for alterada apenas a implementação interna não será necessário actualizar a definição da respectiva interface para o cliente.

45. Considerando a complexidade temporal, o caso médio do algoritmo de ordenação "selection" é idêntico ao pior caso.

46. A execução do passo inicial do algoritmo de Kruskal, a ordenação das arestas por ponderação crescente, não é essencial para que a árvore de suporte a que se chega seja mínima.

47. Numa árvore binária ordenada o elemento que possua a maior chave não pode ter filhos à sua esquerda.

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] 48. Considere o grafo G cuja descrição se apresenta abaixo, através de listas de adjacências. Em cada caixa apresenta-se o par constituído pelo vértice (a negrito) e o valor da aresta, separados pelo carácter '/'. Por exemplo, o vértice **00** possui uma aresta de peso 2 para o vértice **07**.

00 : [**07**/02] → [**11**/03] → [**13**/05] → [**06**/08]
01 : [**00**/09] → [**02**/01] → [**12**/05]
02 : [**09**/07] → [**03**/01] → [**12**/03]
03 : [**00**/04] → [**04**/01] → [**05**/08]
04 : [**12**/04] → [**05**/01] → [**08**/06]
05 : [**02**/02] → [**10**/01] → [**08**/02] → [**07**/05]
06 : [**13**/11] → [**01**/01] → [**11**/03] → [**09**/07]
07 : [**03**/04] → [**10**/03] → [**12**/05] → [**06**/04]
08 : [**11**/09] → [**00**/02]
09 : [**04**/05] → [**01**/10] → [**07**/08] → [**05**/03]
10 : [**13**/06] → [**12**/01]
11 : [**00**/03] → [**03**/02] → [**09**/01]
12 : [**08**/07] → [**13**/01] → [**06**/07]
13 : [**04**/01] → [**01**/02]

[1.5] a) Pretende-se aplicar procura em largura ("Breadth First Search" ou BFS), tomando o vértice **06** como ponto de partida e terminando apenas após estarem visitados todos os vértices do grafo, se tal for possível. Durante os seus cálculos, que deverá justificar de forma clara, represente a fila de procura através de uma lista de vértices. Na sua resolução deverá ser claro no final qual a ordem pela qual os vértices são visitados. Assuma que em cada nó, a ordem pela qual os seus adjacentes são visitados é dada pela posição na lista de adjacências respectiva.

[2.0] b) Tomando de novo o vértice **06** como ponto de partida aplique o algoritmo de Dijkstra, justificando os seus cálculos.

[1.0] c) Trace a árvore que obteve na alínea anterior.

[0.5] d) Comente a seguinte afirmação: A BFS é usada como forma de obter solução para problemas de caminhos mais curtos.

[4.0] 49. O número 24 possui exactamente $D(24) = 8$ divisores, a saber: $1, 2, 3, 4, 6, 8, 12$ e 24. Qualquer inteiro, excepto a unidade, possui sempre, pelo menos, dois divisores. Um número é primo se e só se possuir exactamente 2 divisores: 1 e ele próprio. Por exemplo, $D(17) = 2$. É objectivo deste exercício escrever uma função que receba um número inteiro como argumento e calcule quantos números inteiros entre 1 e o número dado, inclusive, possuem exactamente 8 divisores.

Pretende-se, portanto, calcular $\sum_{i=1}^{N} \mathbf{1}\{D(i) = 8\}$, em que $\mathbf{1}\{.\}$ vale 1 quando o argumento é verdade e vale 0 no caso contrário. Ou seja, $\mathbf{1}\{D(24) = 8\} = 1$, mas $\mathbf{1}\{D(17) = 8\} = 0$.

[2.0] a) Proponha um algoritmo o mais eficiente possível para resolver este problema, fazendo uso de uma descrição por texto e/ou socorrendo-se de um fluxograma suficientemente claro e devidamente explicado. Seja preciso e conciso na sua descrição.

[1.5] b) Escreva a função `int all_with_8_divisors(int N)` que implementa o algoritmo por si proposto, não se esquecendo de a comentar devidamente, para que seja claro o que pretende que cada troço do seu código faça.

[0.5] c) De forma devida e adequadamente detalhada e justificada, determine a complexidade temporal do seu algoritmo como função do parâmetro de entrada.

Nota: É possível resolver este problema em $\mathcal{O}(N^{\alpha})$ com $\alpha \leq 2$. Para obter a cotação completa neste exercício, a função, além de estar correcta, tem de ser o mais eficiente possível.

[2.5] 50. Pensando em árvores genealógicas como contexto, o problema da ancestralidade resume-se a perguntar se numa árvore M-ária, um dado vértice v é ou não um antepassado de outro vértice w.

Neste problema vamos discutir formas alternativas de resolver computacionalmente o problema da ancestralidade e discutir como o uso de memória adicional pode afectar a complexidade temporal.

[1.0] a) Assuma que se pretende escrever uma função que, recebendo como argumentos dois ponteiros para vértices da árvore, indique se o segundo argumento corresponde a um vértice descendente do primeiro.

Sem escrever código, e fazendo uso do que aprendeu sobre árvores com raíz, descreva um algoritmo que permita resolver o problema da ancestralidade, dados dois vértices v e w, em tempo $\mathcal{O}(N)$, em que N é o número total de vértices na árvore.

[1.5] b) **Também sem escrever código** e inspirado no que aprendeu sobre árvores binárias e acervos, assuma que M pode tomar qualquer valor, mas que é uma constante conhecida, e indique de que forma seria possível resolver o mesmo problema em $\mathcal{O}(\log N)$.

Sugestão: Considere a possibilidade de adicionar alguma informação a cada vértice. Explicite qual a informação que adicionaria e de que modo essa informação seria usada.