Departamento de Ciência de Computadores FCUP

# Algoritmos (CC4010)

2019/20

Teste (21.01.2020) *duração: 3h (+30')*

N.º Nome

**1.** [2.0] Relacione os conjuntos indicados usando o símbolo $\subset$, $\supset$, $\subseteq$, $\supseteq$, $=$, e $\neq$ que for **mais adequado**.

**a)** $\Omega(n^2 + 3n) \square O(n^2)$ **b)** $\Theta(n^2) \square \Omega(n \log n)$ **c)** $\Theta(\log_2 n) \square \Theta(\log_{10} n)$

Justifique a alínea **(a)**, usando diretamente a definição matemática das ordens de grandeza.

**2.** Seja $\mathcal{T}$ um conjunto de $n$ tarefas que tem de realizar (se puder). A tarefa $i$ teria início no instante $t_i$ e duração $d_i$ (ambos são inteiros positivos). Em cada instante só pode estar a realizar uma tarefa, mas pode começar uma nova tarefa no instante em que outra termina. Deve realizar **o número máximo de tarefas**.

**a)** [0.5] Prove que a estratégia que escolhe sempre a tarefa *com menor duração* que é compatível com as escolhidas anteriormente pelo mesmo algoritmo, pode produzir uma **solução não ótima**.

**b)** [1.0] Prove que a estratégia *"latest start first"*, que escolhe sempre a tarefa que *começa mais tarde* e é compatível com as escolhidas anteriormente pelo mesmo algoritmo, determina uma **solução ótima**.

**3.** Suponha que se alterou a chave do elemento `k` numa heap de máximo `q`, com $n$ elementos, e é necessário verificar (e, talvez repor) a propriedade de *heap*, usando `heapify(k,q)`.

**a)** [0.3] Em que consiste a "propriedade de *heap*" (para *heap de máximo*)?

**b)** [1.7] Apresente em pseudocódigo a função `heapify(k,q)`, supondo definidas funções `troca(i,j,q)` e `compara(i,j,q)` para trocar os elementos `i` e `j`, entre si, e para comparar dois elementos (retornando 0 se forem iguais, um inteiro negativo se o primeiro for menor que o segundo, e positivo, caso contrário). Justifique a sua correção assumindo que as sub-árvores com raíz no filho esquerdo e direito do nó `k`, se existirem, satisfazem a propriedade de *heap*.

**c)** [1.0] Na análise da complexidade assintótica de `heapify(1,q)`, usou-se $T(n) \leq T(2n/3) + c$, com $c$ constante. Explique para quê e porquê. O que se conclui? Porque é que no **pior caso** $T(n) \in \Omega(\log_2 n)$.

**4.** [1.0] Explique sucintamente de que modo o algoritmo de Strassen, cuja complexidade é [ ], melhora a complexidade assintótica do produto de duas matrizes quadradas $n \times n$, face ao algoritmo trivial, cuja complexidade é [ ].

N.º | Nome

**5.** Considere as funções MYPARTITION e SELECT assim definidas, sendo $x$ é um *array* de $n$ inteiros **distintos** $x[1], \ldots, x[n]$, e $a$ e $b$ inteiros tais que $1 \leq a \leq b \leq n$.

MYPARTITION$(x, a, b)$

```
1. z = x[a]
2. j = a
3. for i = a + 1 to b do
4.     if x[i] < z then
5.         Exchange x[i] with x[j + 1]
6.         j = j + 1
7. Exchange x[a] with x[j]
8. return j
```

SELECT$(x, a, b, k)$

```
1. if a > b ∨ b − a + 1 < k then
2.     return −1
3. t = MYPARTITION(x, a, b)
4. p = t − a + 1
5. if p = k then return x[t]
6. if p < k then
7.     return SELECT(x, t + 1, b, k − p)
8. return SELECT(x, a, t − 1, k)
```

**a)** [1.0] Indique o invariante de ciclo da função MYPARTITION, quando está a testar a condição de paragem para $i$ fixo. O que se deduz sobre o resultado de MYPARTITION$(x, a, b)$?

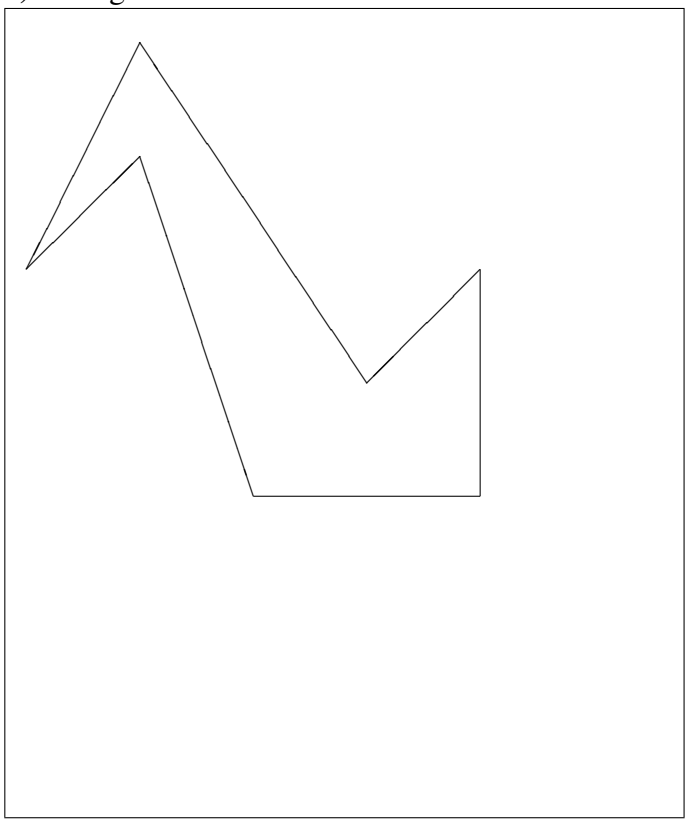
**b)** [1.0] Justifique a correção de SELECT$(x, a, b, k)$, i.e., que obtém o $k$-ésimo menor elemento de $x[a], \ldots, x[b]$, se existir.

**c)** [1.5] Indique a complexidade temporal assintótica de SELECT$(x, a, b, k)$ no melhor caso ____, no pior caso ____ e no caso médio ____. Para o caso médio, admita uma distribuição uniforme. Justifique a resposta que deu para o **pior caso**, começando por o descrever.

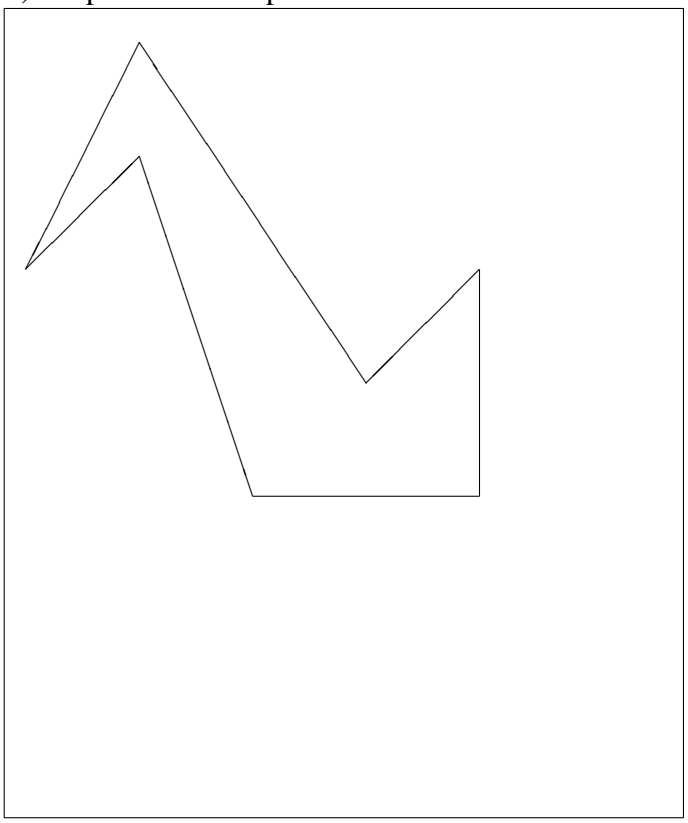
**d)** [1.0] Explique que vantagem teriam implementações baseadas em *quickselect* (com randomização) e *mediana das medianas de 5*, relativamente à indicada.

**6.** [2.0] Considere o polígono com vértices $(2,0),(4,0),(4,2),(3,1),(1,4),(0,2),(1,3)$, numerados de 1 a 7. Apresente, **explicando**, um posicionamento de guardas que possa resultar da aplicação:

**a)** do algoritmo de Ghosh.



**b)** da prova de Fisk para o teorema de Chvátal.



**7.** [1.5] Sejam $X$ e $Y$ duas sequências de carateres, com comprimento $m$ e $n$, respetivamente. Considere o problema *Minimum Edit Distance*, aplicado a $X$ e $Y$, sendo o custo da operação de inserção de 2, remoção 1 e substituição 2. Apresente a recorrência que define o custo da transformação de $X_i$ em $Y_j$, sendo $X_i$ e $Y_j$ os prefixos de formados pelos $i$ e $j$ primeiros carateres, com $i \geq 0$ e $j \geq 0$. **Explique.**

**8.** [1.0] Explique sucintamente porque é que o problema de decisão associado a *minimum cardinality vertex cover* (em grafos não dirigidos) é da classe NP e porque é que o problema de otimização é da classe APX.

N.º | Nome

**9.** Considere uma *stack* suportada por um *array* $V$ e uma variável $top$ que designa o índice da primeira posição livre de $V$, com as operações usuais PUSH$(x)$ e POP$()$ assim definidas:

```
PUSH(x)
    V[top] = x
    top = top + 1
```

```
POP()
    top = top − 1
    return V[top + 1]
```

Suponha que inicialmente $V$ tem $M$ posições e considere a possibilidade de **a operação de PUSH$(x)$ ser substituída para aumentar dinamicamente a capacidade da stack** quando está cheia e é necessário inserir um novo valor. Nessa operação, começa por realocar espaço (um novo *array* com mais $M$ posições do que o anterior, sendo $M$ a constante inicial), copia os elementos que estavam na *stack* para o novo *array* e só depois insere o novo elemento. Considere um modelo de custos em que a inserção de um elemento e a remoção de um elemento têm custo 1 e o custo da expansão da estrutura de dados é igual ao número de elementos transferidos.

**a)** [1.0] Suponha que $M = 30$ e efetua uma sequência de 135 operações de PUSH, partindo da *stack vazia*. Apresente a expressão que define o custo total. Qual é o custo de uma operação de PUSH no pior caso e no melhor caso? Qual é o custo amortizado de cada operação?

**b)** [1.0] Considere o caso geral (em que apenas se sabe que $M$ é uma constante). Suponha que efetua uma sequência de $kM$ operações de PUSH, sendo $k$ inteiro positivo. Compare o custo amortizado de cada operação nesta abordagem com o custo amortizado se, em cada expansão, se *duplicasse* o tamanho do *array*.

**10.** No problema BIN PACKING são dados $n$ itens com pesos $p_1, \ldots, p_n$, tais que $0 < p_i \leq 1$, para todo $i$, e há que os distribuir por latas de capacidade **unitária**, usando o menor número de latas possível.

No problema PARTITION são dados $n$ inteiros positivos $a_1, a_2, \ldots, a_n$, e há que decidir se existe uma partição $\{S, T\}$ do conjunto de índices $\{1, 2, \ldots, n\}$ tal que $\sum_{i \in S} a_i = \sum_{i \in T} a_i$. Sabe-se que PARTITION é um problema **NP-completo**.

**a)** [0.2] Prove que as instâncias de PARTITION com $a_j > \sum_{j \neq i} a_i$, para algum $j$, são trivialmente decidíveis.

**b)** [1.0] Dada uma instância de PARTITION, com $a_j \leq \sum_{j \neq i} a_i$, para todo $j$, definimos uma instância de BIN PACKING com $p_j = 2a_j / \sum_{i=1}^{n} a_i$, para todo $j$. Justifique que se trata de uma redução polinomial que permite decidir PARTITION em tempo polinomial se algum dos algoritmos seguintes existir e conclua que não podem existir a menos que P=NP: (i) um algoritmo polinomial que calcule uma solução ótima para BIN PACKING; (ii) um algoritmo de aproximação polinomial de razão $c$ para BIN PACKING, com $c < 3/2$.

**c)** [0.3] Sabendo que estratégia "first fit decreasing" obtém em tempo polinomial uma aproximação de razão 3/2, conclua que BINPACKING é um problema APX-completo.

(Fim)

---

**Master theorem:**

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we interpret $n/b$ to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$, for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

**Stirling's approximation:**

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + \Theta(1/n)) = \sqrt{2\pi n}\left(\frac{n}{e}\right)^{\alpha_n}, \quad \text{with } 1/(12n+1) < \alpha_n < 1/(12n)$$

**Some useful results:**

$$\log(\prod_{k=1}^{n} a_k) = \sum_{k=1}^{n} \log a_k \qquad \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}, \quad \text{for } |x| < 1$$

If $(u_k)_k$ is an arithmetic progression (i.e., $u_{k+1} = r + u_k$, for some constant $r \neq 0$), then $\sum_{k=1}^{n} u_k = \frac{(u_1+u_n)n}{2}$.

If $(u_k)_k$ is a geometric progression (i.e., $u_{k+1} = ru_k$, for some constant $r \neq 1$), then $\sum_{k=1}^{n} u_k = \frac{u_{n+1}-u_1}{r-1}$.

If $f \geq 0$ is continuous and a monotonically increasing function, then

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$