Departamento de Ciência de Computadores FCUP

Algoritmos (CC4010) 2019/20

Teste (12.11.2019) *duração: 3h*

N.º [ ] Nome [ ]

1. Usando a definição de $O(n^2)$, prove a veracidade ou falsidade da afirmação: "Se $f(n) = 10n^2 \log_2 n$, para $1 \leq n \leq 100$, e $f(n) = n^2 + 7\log_2 n$, para $n > 100$, então $f(n) \notin O(n^2)$".

2. Considere o problema de ordenar n inteiros $a_1, \ldots a_n$. Analise a veracidade da afirmação: "O tempo de execução de qualquer algoritmo de ordenação comparativo, no melhor caso e no pior caso, é $\Omega(n \log_2 n)$".

3. Diga que problema resolvem os algoritmos *quickselect* e *mediana das medianas de cinco*, descritos nas aulas. Indique a sua complexidade e as ideias principais.

4. Assuma que pretendemos calcular o produto de dois números X e Y, representados em binário com n bits. O produto será representado com $2n$ bits. Justifique a veracidade ou falsidade das afirmações seguintes.

a) O algoritmo que consiste em adicionar Y vezes o valor X a um acumulador (com $2n$ bits) tem complexidade exponencial em n, sendo a adição efetuada em $O(n)$;

b) O algoritmo de multiplicação usual (baseado em adição e *shifts* à esquerda) tem complexidade quadrática.

c) Como $X = 2^{n/2}A + B$ e $Y = 2^{n/2}C + D$, o algoritmo baseado em divisão-e-conquista e no facto de $XY = 2^n AC + 2^{n/2}(BC + AD) + BD$ tem complexidade sub-quadrática (sendo as multiplicações por 2^k efetuadas por k *shifts*).

Dos problemas 5-10, resolva apenas 4

5. Seja $T(n) = T(n/3) + T(n/5) + 2n$, para $n \geq 2$. Assuma que $T(n) \leq c$, para valores de n suficientemente pequenos. Baseando-se na análise da árvore de recursão, prove que existem constantes c_1, c_2 e n_0, com $c_1, c_2 \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}$, tais que $c_1 n \leq T(n) \leq c_2 n$ para todo $n > n_0$, com n inteiro. Em termos de ordem de grandeza, o que se pode concluir sobre $T(n)$?

6. Pretendemos ordenar n inteiros positivos, $a_1, \ldots, a_n$, por ordem crescente, dadas as suas representações base b, com k dígitos. Iremos aplicar *radix sort*, com *counting sort* como algoritmo auxiliar.

a) Que propriedade de *counting sort* é relevante para a correção de *radix sort*?

b) Escreva, em pseudocódigo, a função auxiliar *counting sort* e a função *radix sort*. Assuma que a_i está na linha i de uma matriz A, sendo $A[i][1]$ o dígito mais significativo de a_i.

c) Ilustre os passos principais da ordenação de 573, 568, 764, 034, 234, 008, 712, 549, 237, 222, 171, e 032, por ordem crescente (sendo $b = 10$, $k = 3$, $n = 12$). Apresente também os passos da **primeira** aplicação de *counting sort*.

d) Indique a complexidade da função *radix sort* que apresentou, usando b, k e n. Justifique sucintamente.

(Continua, v.p.f.

7. Considere a função PARTITION usada por QUICKSORT (sem randomização).

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x then
          i = i + 1
          Exchange A[i] with A[j]
  Exchange A[i + 1] with A[r]
  return i + 1
```

a) Apresente o invariante de ciclo que permite justificar formalmente a correção da função PARTITION.

b) Apresente o algoritmo RANDOMIZED_QUICKSORT(A, p, r) e indique os passos principais da análise da sua complexidade, assumindo que os valores são todos distintos.

8. Recorde a seguinte descrição do algoritmo HEAPSORT, com complexidade temporal $O(n \log n)$.

```
HEAPSORT(A):
  BUILD_MAX_HEAP(A)
  for i = A.length downto 2
      Exchange A[1] with A[i]
      A.heapsize = A.heapsize − 1
      MAXHEAPIFY(A, 1)
```

a) Suponha que na chamada da função $A = [1, 2, 3, 4, 5, 6, 7, 8]$ e $A[1] = 1$ e $A.length = n = 8$. Qual é o resultado da chamada de BUILD_MAX_HEAP(A)?

b) O que faz a função MAXHEAPIFY? Ilustre a sua aplicação na primeira iteração do ciclo **for**.

c) Que invariante é mantido no ciclo **for** para garantir a correção do algoritmo?

d) Apresente sucintamente as ideias principais da prova de que MAXHEAPIFY$(A, 1)$ é $O(\log_{3/2} A.heapsize)$ e de que HEAPSORT(A) é $O(n \log n)$, se o número de elementos a ordenar for n.

9. Considere a determinação do invólucro convexo *(convex hull)* de n pontos $p_1, p_2, p_3, \ldots, p_n$ no plano, pelo algoritmo de Graham (*Graham-scan*), usando varrimento rotacional. Suponha que p_1 é o ponto de menor ordenada e que os pontos estão já ordenados por ordem crescente de ângulo polar relativamente a p_1 (não existindo pontos com o mesmo ângulo polar nem a mesma ordenada).

Admita que cada operação de *push*, *pop*, e *teste de viragem* tem um custo real de 1 unidade. Usando o método contabilístico (*accounting*) mostre que se se definir os custos amortizados das operações de *push*, *pop*, e *teste de viragem* como 4, 0, e 0, então o crédito total acumulado nunca é negativo. Conclua que o custo amortizado da análise de cada p_i é $O(1)$.

10. Considere um dos problemas "unit task scheduling" e "interval scheduling". Apresente-o, indique a estratégia *greedy* que o resolve e a as ideias centrais da prova de que é correta.

(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:

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

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

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