Rio de Janeiro, 11 de Setembro de 2013.

PROVA 1 DE PROJETO E ANÁLISE DE ALGORITMOS

PROFESSOR: EDUARDO SANY LABER

DURAÇÃO: 1:50h

1. (2.5pt) Seja $T(n)$ a complexidade de pior caso de um algoritmo.

   a) Assuma que $T(n) = O(n^2)$. Podemos afirmar que **existe** alguma entrada suficientemente grande para a qual o algoritmo realiza pelo menos $100n$ operações? Por que?

   solução. Não. Porque o fato do algoritmo ter complexidadede $T(n) = O(n^2)$ não descarta a possibilidade dele realizar a$O(1)$ para cada entrada. A afirmativa seria verdadeira se $T(n)$ fosse $\Theta(n^2)$

   b) Assuma que $T(n) = \Omega(n^2)$. Podemos afirmar que **existe** um inteiro n_0 tal que **para toda** entrada de tamanho $n \geq n_0$, o algoritmo vai realizar pelo menos $n^{1/2}$ operações? Por que?

   solução. Não. Isso implica apenas que existe **pelo menos** uma entrada de tamanho $n \geq n_0$, o algoritmo vai realizar pelo menos $n^{1/2}$ operações.

   c) Assuma que $T(n) = \theta(n^2)$. Podemos afirmar que **para toda** entrada de tamanho n o algoritmo vai realizar no máximo $10n^3$ operações? Por que?

   solução. Não, para valores pequenos de n não podemos afirmar isso.

2. (2.0pt) Considere o pseudo-código abaixo.

   Proc1(L: lista de inteiros)

   Se $|L| = 1$ Return valor do único elemento de L

   Divida L em 4 listas, L_1, L_2, L_3, L_4, cada uma com $n/4$ elementos

   $S_1 \leftarrow$ soma dos elementos de L_1

   $S_2 \leftarrow$ soma dos elementos de L_2

   $S_3 \leftarrow$ soma dos elementos de L_3

   $S_4 \leftarrow$ soma dos elementos de L_4

   Return $S_1 + S_2 + S_3 + S_4$+Proc1(L_1)+Proc1(L_2)+Proc1(L_3)+Proc1(L_4)

   Fim Proc1

   a) Seja $T(n)$ a complexidade de pior caso do pseudo-código em função de n. Ache uma equação de recorrência para $T(n)$

   solução. $T(n) = 4T(n/2) + n$, se $n > 1$e $T(1) = 1$, caso contrário.

   b) Encontre uma função $f(n)$ tal que $T(n) = \Theta(f(n))$

   solução. Resolvendo encontramos $f(n) = n^2$.

3. (2.0pt) Seja $T(n)$ a complexidade de pior caso do pseudo-código abaixo em função de n. Encontre uma função $f(n)$ tal que $T(n) = \Theta(f(n))$. Assuma que a linha $(*)$ é executada em $O(1)$

    $loop \leftarrow n^2$

    Enquanto $loop > 1$

    $j \leftarrow 1$

    For $i = 1 \dots n$

    $c \leftarrow$ número aleatório no conjunto $\{2, 3, 4\}$ (*)

    $j \leftarrow c \times j$

    For $k = 1 \dots j$

    $cont++$

    Fim For

    Fim For

    $loop \leftarrow loop/2$

    Fim Enquanto

    solução. O algoritmo gasta tempo proporcional a $\log(n^2) \times \sum_{i=1}^{i} 4^i$ que é $O(4^n \log n)$.

4. (2.5pt) Explique com palavras como seria um algoritmo eficiente que recebe n palavras da língua portuguesa $p_1, \dots, p_n$, cada uma com no máximo k letras, e devolve as palavras em ordem alfabética. Analise a complexidade de pior caso do algoritmo proposto em função de n e k. Quanto mais eficiente o algoritmo proposto maior a pontuação. Lembre que o alfabeto português tem 26 letras distintas.

    Podemos pensar cada palavra tendo k dígitos (letras) e aplicar o RADIX-Sort. A complexidade seria proporcional a $k(26 + n)$. Utilizando um algoritmo de ordenação como o mergesor teriamos $O(kn \log n)$.

5. (2.0pt) Seja uma sequência de números reais $A = (a_1, a_2, \dots, a_n)$ em que existe um inteiro $j \in \{1, \dots, n\}$ tal que $a_i < a_{i+1}$ para todo $i < j$ e $a_i > a_{i+1}$ para todo $i \geq j$. Em palavras, a sequência é crescente até o índice j e depois passa a ser decrescente. Escreva o pseudo-código de um algoritmo `Search(A)` que recebe como entrada a sequência A e devolve o inteiro j. O algoritmo deve executar em $O(\log n)$.

    solução. Basta fazer uma busca binaria e ir descartando uma das partes do vetor.