MAC 5711 - Análise de Algoritmos

Departamento de Ciência da Computação
Segundo semestre de 2005

Lista 4

Devolução dos exercícios 2, 11 e 13: **22 de setembro**

1. Escreva em pseudocódigo a rotina que denominamos na sala de SOBE_HEAP(A, n, i), que restaura um heap que teve a chave da posição i aumentada. Use-a para implementar as seguintes rotinas: INSERE_HEAP(A, n, x), que insere um elemento com chave x no heap armazenado no vetor $A[1..n]$, e ALTERA_CHAVE(A, n, i, x), que altera para x a chave do elemento $A[i]$ do heap $A[1..n]$. As suas três rotinas devem consumir no pior caso tempo $O(\lg n)$.

2. Escreva em pseudocódigo a rotina REMOVE_HEAP(A, n, i), que remove o elemento que está armazenado na posição i do heap, restaurando o heap resultante. Sua rotina deve consumir tempo $O(\lg n)$.

3. Adapte todas as rotinas que você escreveu para heaps de máximo para que funcionem para armazenar um heap de mínimo.

4. Considere um heap $A[1..m]$; a raiz do heap é o elemento de índice 1. Seja m' o número de elementos do "sub-heap esquerdo", cuja raiz é o elemento de índice 2. Seja m'' o número de elementos do "sub-heap direito", cuja raiz é o elemento de índice 3. Mostre que $m'' \leq m' < 2m/3$.

5. Mostre que a solução da recorrência $T(k) = T(2k/3) + \Theta(1)$ é $O(\log k)$. (Curiosidade: Essa é a recorrência do DESCE-HEAP (A, m, i) se interpretarmos k como sendo o número de nós na subárvore com raiz i).

6. Quanto vale S no fim do seguinte algoritmo?

   $S \leftarrow 0$
   para $i \leftarrow 1$ **até** n faca
       para $j \leftarrow 1$ **até** i **faça**
           $S \leftarrow S + 1$

7. Quanto vale S no fim do seguinte fragmento de código?

   $S \leftarrow 0$
   $i \leftarrow n$
   enquanto $i > 0$ **faça**
       para $j \leftarrow 1$ **até** i **faça**
           $S \leftarrow S + 1$
       $i \leftarrow \lfloor i/2 \rfloor$

8. Construa a árvore de decisão para o algoritmo MERGESORT para $n = 4$. Quais as permutações de pior caso?

9. Construa a árvore de decisão do algoritmo QUICKSORT (versão não-probabilística) para $n = 3$. Quais as permutações de pior caso?

10. Por que a hipótese de que os elementos são todos distintos na análise acima não é restritiva? (Ou seja, por que ela não impede que concluamos que o algoritmo, para instâncias arbitrárias, consome tempo de pior caso $\Omega(n \log n)$?)

11. Qual é a menor profundidade possível de uma folha em uma árvore de decisão para um algoritmo de ordenação? Qual é a informação que você conclui sobre algoritmo relacionada a este número?

12. Professor Ludovico afirma que o limite inferior de $\Omega(n \log n)$ para ordenação de n números não se aplica ao seu ambiente de programação. No seu ambiente, o fluxo de controle do programa pode dividir-se em três depois de uma única comparação $a_i : a_j$, dependendo de qual dos três seguintes casos ocorre: $a_i < a_j, a_i = a_j$ ou $a_i > a_j$. Mostre que o professor está errado, provando que o número de "super" comparações requeridas para ordenar n elementos ainda é $\Omega(n \log n)$.

13. Mostre que qualquer algoritmo de busca baseado em comparações faz no pior caso pelo menos $\lceil \log n \rceil$ comparações. Um algoritmo de busca é um algoritmo que determina uma posição em que um elemento x aparece em um dado vetor $v[0..n-1]$. Um algoritmo de busca baseia-se em comparações se o seu fluxo de controle depende apenas de comparações envolvendo elementos do vetor e x.