Instituto de Informática UFG

Projeto e Analise de Algoritmos
Prova 1 – 12/02/2007

PROFESSORA: Diane Castonguay

JUSTIFIQUE TODAS AS SUAS RESPOSTAS

Questão 1 [3pt]

(a) Considere o algoritmo ALGO de dividir e conquistar em anexo. Exiba a equação de recorrências do tempo de execução de ALGO.

(b) Usando o teorema mestre, resolva a recorrência seguinte

$$T(n) = 4 * T(n/5) + f(n)$$

onde $f(n) = n/5 + 3 \lg(n)$

Questão 2 [1pt]

Considere o vetor A dado por < 5, 8, 3, 5, 6, 4, 4, 2, 3, 7>.
Ilustre o vetor em forma de heap antes e depois da chamada de BUILD-MAX-HEAP(A).
Explicita cada chamada de MAX-HEAPIFY no decorrer de BUILD-MAX-HEAP(A).

Questão 3 [3pt]

Considere o algoritmo SORT em anexo.
(a) Qual é o loop invariante do segundo loop para (l.2 a l.6) do algoritmo SORT?

(b) Quais são as diferências e as similitudes, do ponto de vista da eficiência, entre o algoritmo SORT e o algoritmo MERGE-SORT. O algoritmo SORT é estável? Explique.

Questão 4 [3pt]

Considere o algoritmo HOARE-PARTITION em anexo.

LOOP	*LOOP INVARIANTE*
Primeiro enquanto (l.5 a l.18)	Quando o teste é verdade, os valores de A[p .. i] são menores ou iguais a chave e os valores de A[j .. r] são maiores ou iguais a chave.
Segundo enquanto (l.7 a l.9)	$\max\{i, p\} \leq j \leq r$ e os valores de A[j + 1 .. r] são maiores ou iguais a chave.
Terceiro enquanto (l.11 a l.13)	$p \leq i \leq \min\{j+1, r\}$ e os valores de A[p .. i – 1] são menores ou iguais a chave.

(a) Prove que o “loop invariante” do segundo enquanto do algoritmo é valido.

(b) Supondo que os “loop invariante” do segundo e do terceiro enquanto do algoritmo são validos, mostre a correteza do “loop invariante” do primeiro enquanto do algoritmo.

(c) Use os itens (a) e (b) para concluir sobre a correteza do algoritmo HOARE-PARTITION, ou seja, retorna um índice j tal que os valores de A[p .. j] são menores ou iguais a chave. e os valores de A[j + 1 .. r] são maiores ou iguais a chave.

ALGORITMOS

Questão 1

```
ALGO(A, n)
1. se n > 1
2.      então  d ← DIVIDIR(A, n)
3.             r ← n
4.             enquanto r > 1 faça
5.                    ALGO(A, d)
6.                    r ← |_ r/2 _|
7.             fim-enquanto
8.             COMBINAR(A, n, d)
9. fim-se
```

Questão 2

```
MAX-HEAPIFY(A, i)
1.     l ← LEFT(i)
2.     r ← RIGHT(i)
3.     se l ≤ tamanho-do-heap[A] e A[l] > A[i]
4.            então maior ← l
5.            senão maior ← i
6.     fim-se
7.     se r ≤ tamanho-do-heap[A] e A[r] > A[maior]
8.            então maior ← r
9.     fim-se
10.    se maior ≠ i
11.           então A[i] ↔ A[maior]
12.                  MAX-HEAPIFY(A, maior)
13.    fim-se

BUILD-MAX-HEAP(A)
1. tamanho-do-heap[A] ← comprimento[A]
2. para i ← ⌊comprimento[A]/2⌋ até 1 passo -1 faça
3.            MAX-HEAPIFY(A, i)
4. fim-para
```

Questão 3

```
SORT(A)
1. para i ← 1 até comprimento[A] faça
2.     para j ← comprimento[A] até i + 1 passo – 1 faça
3.            se A[j] ≤ A[j – 1]
4.                   então A[j] ↔ A[j – 1]
5.            fim-se
6.     fim-para
7. fim-para
```

Questão 4

```
HOARE-PARTITION(A, p, r)
1. chave ← A[p]
2. i ← p - 1
3. j ← r + 1
4. teste ← VERDADE
5. enquanto teste faça
6.         j ← j – 1
7.         enquanto A[j] > chave faça
8.                 j ← j – 1
9.         fim-enquanto
10.        i ← i + 1
11.        enquanto A[i] < chave faça
12.                i ← i + 1
13.        fim-enquanto
14.        se i < j
15.                então A[i] ↔ A[j]
16.                senão teste ← FALSO
17.        fim-se
18. fim-enquanto
19. retorna j
```