Projeto e Analise de Algoritmos
Prova 1 – 08/09/2005

NOME: _____
MATRICULA: _____ [ ] Regular [ ] Especial
PROFESSORA: Diane Castonguay

JUSTIFIQUE TODAS AS SUAS RESPOSTAS

Questão 1 [1pt]

Usando a definição básica da notação O, mostre que $f(n) = 5n^3+n^2+7n+3$ é $O(n^3)$.

Questão 2 [2pt]

Considere o seguinte algoritmo ALGO de dividir e conquistar:

```
ALGO(A, p, r)
1. se p < r
2.       então  d ← DIVIDIR(A, p, r)
3.              ALGO(A, p, p + d – 1)
4.              ALGO(A, p + 1, p + d)
5.              ALGO(A, p + d, p + 2d – 1)
6.              ALGO(A, p + d + 1 , p + 2d)
7.              ALGO(A, p + d + 2, p + 2d + 1)
8.              ALGO(A, r – d, r – 1)
9.              ALGO(A, r – d + 1, r)
10.             COMBINAR(A, p, q, d)
11. fim-se
```

(a) Exiba a equação de recorrências do tempo de execução de ALGO.

(b) Usando o teorema mestre, resolva a recorrência obtida em (a) sabendo que:
- d é sempre igual a n/3
- o tempo de execução de DIVIDIR é O(n)
- o tempo de execução de COMBINAR é O(lg n)

onde n = r – p + 1

Questão 3 [3pt]

Considere o vetor A dado por < 32, 15, 8, 16, 9, 10, 5, 20, 2, 9, 5, 3 >.

(a) Represente este vetor sobre forma de heap (arvore). Ele satisfaz a propriedade do heap máximo? Se não, para qual ou quais nó(s), ele não satisfaz a propriedade do heap máximo?

(b) Enumere o mínimo de chamadas a MAX-HEAPIFY que é preciso para que o heap dado no item (a) satisfaz a propriedade do heap máximo. Estas chamadas produzem o mesmo heap que a chamada a BUILD-MAX-HEAP?

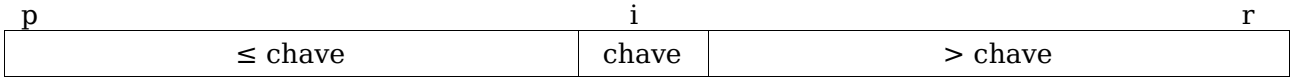
(c) Ilustre o heap obtido antes e depois da primeira chamada do loop para de HEAPSORT(A).

Questão 4 [4pt]

Considere o seguinte algoritmo:

```
/*
 * p ≤ r
 */
SEPARA(A, p, r)
1.      chave ← A[p]
2.      i ← p
3.      j ← r
4.      para k ← p + 1 até r faça
5.              se A[k] ≤ chave
6.                      então B[i] ← A[k]
7.                              i ← i + 1
8.                      senão B[j] ← A[k]
9.                              j ← j – 1
10.             fim-se
11.     fim-para
12.     B[i] ←  chave
13.     para k ← p até r faça
14.             A[k]  ← B[k]
15.     fim-para
16.     retorna i
```

Observação: No final do algoritmo, i = j e o vetor A é da forma:



(a) Qual é o loop invariante do primeiro loop para do algoritmo SEPARA?

(b) Prove que o loop invariante do item (a) é valido.

(c) Use os item (a) e (b) para concluir sobre a correteza do algoritmo SEPARA.

(d) Compare a eficiência do algoritmo SEPARA ao algoritmo PARTITION visto em classe. Qual você escolheria e por quê?

Questão 5 [2pt]

Considere o seguinte algoritmo:

```
/*
 * As entradas do vetores são inteiras. Os valores podem variar de -k até k.
 */
TEMPO(A)
1.      para i ← 1 até comprimento[A] faça
2.              se A[i]  ≥ 0
3.                      então  aux  ← 1
4.                             para j  ← 2 até A[i] faça
5.                                     aux ← aux * j
6.                             fim-para
7.                             A[i] ← aux
8.              fim-se
9.      fim-para
```

(a) Exiba quantas vezes cada linha do algoritmo TEMPO será efetuada.

(b) Use o item (a), para analisar o tempo de execução do algoritmo no melhor e no pior caso sabendo que k é O(n) onde n = comprimento[A].

Questão 2

```
ALGO(A, p, r)
1. se p < r
2.     então      d ← DIVIDIR(A, p, r)
3.           ALGO(A, p, p + d)
4.           ALGO(A, p + 1, p + 1 + d)
5.           ALGO(A, p – 1 + d, p – 1 + 2d)
6.           ALGO(A, p+d, p+2d)
7.           ALGO(A, p + 1 + d, p + 1 + 2d)
8.           ALGO(A, r – 1 – d, r – 1)
9.           ALGO(A, r – d, r)
10.          COMBINAR(A, p, q, d)
11. fim-se
```

Questão 4

```
SEPARA(A, p, r)
1.    chave ← A[p]
2.    i ← p
3.    j ← r
4.    para k ← p + 1 até r faça
5.          se A[k] ≤ chave
6.                then B[i] ← A[k]
7.                     i ← i + 1
8.                senão B[j] ← A[k]
9.                     j ← j – 1
10.         fim-se
11.   fim-para
12.   B[i] ←  chave
13.   para k ← p até r faça
14.         A[k]  ← B[k]
15.   fim-para
16.   retorna i
```

Questão 5

```
/*
 * As entradas do vetores são inteiras. Os valores podem variar de -k até k.
 */
TEMPO(A)
1.    para i ← 1 até comprimento[A] faça
2.          se A[i]  ≥ 0
3.                então      aux  ← 1
4.                     para j  ← 2 até A[i] faça
5.                          aux ← aux * j
6.                     fim-para
7.                     A[i] ← aux
8.          fim-se
9.    fim-para
```

ALGORITMOS VISTO EM SALA DE AULA

```
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

HEAPSORT(A)
1. BUILD-MAX-HEAP(A)
2. para i ← comprimento[A] até 2 passo -1 faça
3.                A[1] ↔ A[i]
4.                tamanho-do-heap[A] ← tamanho-do-heap[A] – 1
5.                MAX-HEAPIFY(A, 1)
6. fim-para
```