IME-USP – Cursos de Verão 2010
Tópicos de Programação – 2ª Lista de Exercícios

Exercício B1 – Para cada uma das afirmações abaixo, responda *verdadeiro* ou *falso* (prove suas respostas):

a) Se $f(n)$ no pertence a $O(g(n))$, então $g(n) = O(f(n))$;

b) $50n^2 + n + 300 = O(n^5)$;

c) $50n^2 + n + 300 = \Omega(n)$;

d) $50n^2 + n + 300 = \Theta(n)$;

e) $50n^2 + n + 300 = \Theta(n^2)$;

f) Se $f(n) = O(g(n))$ e $g(n) = \Omega(f(n))$, então $f(n) = \Theta(g(n))$.

Exercício B2 – Escreva um algoritmo que recebe um inteiro $n \geq 1$ e devolve o n-ésimo número de Fibonacci. O consumo de tempo do seu algoritmo deve ser $\Theta(n)$.

Exercício B3 – Escreva um algoritmo que recebe um inteiro $n \geq 1$ e devolve o n-ésimo número de Fibonacci. O consumo de tempo do seu algoritmo deve ser $\Theta(\lg n)$ (sendo que lg é $\log_2$). **Dica:** Seja M a matriz abaixo:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

O n-ésimo número de Fibonacci está em uma das entradas da matriz M^n.

Exercício B4 – Seja $v[0 \mathrel{..} n-1]$ um vetor de n números inteiros distintos. Se $i < j$ e $v[i] > v[j]$, então o par (i, j) chamado uma *inversão* de v.

a) Qual é a relação entre o tempo de execução do algoritmo de ordenação por inserção (`InsertionSort`) e o número de inversões no vetor de entrada? Justifique sua resposta.

b) Escreva um algoritmo que determine o número de inversões em qualquer permutação sobre n elementos. Seu algoritmo deve consumir tempo $\Theta(n \lg n)$. **Dica:** modifique o algoritmo `MergeSort`.

Exercício B5 – Considere o seguinte algoritmo:

```
    int potencia_rec(int n, int k) {
       int p;

(1)    if(k == 0) {
(2)       return 1;
       } else {
(3)        p = potencia_rec(n, k/2);
(4)        return (k%2 == 0)? (p*p) : (p*p*n);
       }
    }
```

a) prove que o algoritmo acima resolve o seguinte problema: *dados dois números inteiros n e k, tal que $k \geq 0$, calcular n^k.*

b) seja $T(k)$ o consumo de tempo do algoritmo `potencia_rec`. Definimos $T(k)$ através da seguinte recorrência:

$$T(k) = \begin{cases} 1, & \text{se } k = 0 \\ T(\lfloor k/2 \rfloor) + 1, & \text{se } k > 0. \end{cases}$$

Prove que $T(k) = \Theta(\lg k)$.

Exercício B6 – Escreva uma função em C que, dado um vetor $v[1 \ .\ .\ n]$ tal que v é um max-heap, remove um elemento que está na posição i de v. Ou seja, após a execução do seu algoritmo deve valer que $v[1 \ .\ .\ n-1]$ é um max-heap que contém os elementos que estavam originalmente em v, com exceção do elemento que estava originalmente em $v[i]$.

Exercício B7 – Escreva um algoritmo para resolver o seguinte problema: dado um vetor $v[0 \ .\ .\ n-1]$, devolver o i-ésimo menor elemento do vetor, sendo que i é um parâmetro do algoritmo, tal que $0 \leq i \leq n-1$. Seu algoritmo não pode utilizar-se de nenhum algoritmo de ordenação (na média seu algoritmo deve ser linear - não precisa provar isso). Dica: utilize a função `partition` vista em aula.

Exercício B8 – Seja $n := r - p + 1$ e considere as seguintes questões:

a) Descreva um cenário no qual o tamanho da pilha de chamadas recursivas do Quicksort é $\Theta(n)$.

b) Modifique o código de Quicksort de tal modo que o tamanho da pilha de chamadas recursivas seja $\Theta(\lg n)$ no pior caso. Justifique sua resposta. Mantenha o tempo de execução esperado $O(n \lg n)$ do algoritmo (não precisa provar esta parte). Dica: modifique a seguinte versão do Quicksort:

```
void quicksort (int p, int r, int v[]) {
   int j;

   while(p < r) {
      j = partition(p, r, v);
      quicksort(p, j-1, v);
      p = q + 1;
   }
}
```