Exercícios de Revisão

1. Considere o algoritmo abaixo. O que ele faz? Qual é a função de complexidade do número de comparações de elementos do vetor no melhor caso e no pior caso? Que configuração do vetor de entrada A leva a essas duas situações? Explique / Demonstre como você chegou a esses resultados. (Dica: analise para cada valor de i quantas vezes o while é executado no melhor e no pior caso, e monte um somatório...)

```
typedef int Vetor[MAX];

void EX1(Vetor A, int n){
  int i, j, x;

  for(i=1; i<n; i++) {
     x = A[i];
     j = i - 1;
     while ( (j >= 0) && (x < A[j]) ) {
        A[j + 1] = A[j];
        j = j - 1;
     }
     A[j + 1] = x;
  }
}
```

2. O **Casamento de Padrõe**s é um problema clássico em ciência da computação e é aplicado em áreas diversas como pesquisa genética, editoração de textos, buscas na internet, etc. Basicamente, ele consiste em encontrar as ocorrências de um padrão P de tamanho m em um texto T de tamanho n. Por exemplo, no texto T = “PROVA DE AEDSII” o padrão P = “OVA” é encontrado na posição 3 enquanto o padrão P = “OVO” não é encontrado. O algoritmo mais simples para o casamento de padrões é o algoritmo da “Força Bruta”, mostrado abaixo. Analise esse algoritmo e responda: Qual é a função de complexidade do número de comparações de caracteres efetuadas no melhor caso e no pior caso. Dê exemplos de entradas que levam a esses dois casos. Explique sua resposta!

```
typedef char TipoTexto[MaxTamTexto];
typedef char TipoPadrao[MaxTamPadrao];

void ForcaBruta(TipoTexto T, int n, TipoPadrao P, int m) {
  int i, j, k;
  for (i = 1; i <= (n - m + 1); i++) {
      k = i;
      j = 1;
      while (T[k-1] == P[j-1] && j <= m) {
         j++;
         k++;
      }
      if (j > m)
         printf(" Casamento na posição %3d\n", i);
    }
}
```

3. Sejam f(n), g(n) duas funções assintóticas positivas e a e b. Prove que as afirmativas abaixo são verdadeiras ou falsas, usando para isso as definições das notações assintóticas ou contraexemplos.

a) $2^{n+1} = O(2^n)$

b) $2^{2n} = O(2^n)$

c) $f(n) + g(n) = O(Max(f(n), g(n))$

d) A notação θ é simétrica, ou seja, $f(n) = \theta(g(n))$ se e somente se $g(n) = \theta(f(n))$

4. Implemente uma função recursiva para computar o valor de 2^n. Determine a sua equação de recorrência e a resolva para determinar a complexidade do algoritmo.

5. O que faz a função abaixo? Explique o seu funcionamento.

```
void f(int a, int b) { // considere a > b
    if (b == 0)
       return a;
    else
       return f(b, a % b); //o operador % fornece o resto da divisão
}
```

6. Vários algoritmos em computação usam a técnica de "Dividir para Conquistar": basicamente eles fazem alguma operação sobre todos os dados, e depois dividem o problema em sub-problemas menores, repetindo a operação. Uma equação de recorrência típica para esse tipo de algoritmo é mostrada abaixo. Resolva essa equação de recorrência usando o Teorema Mestre.

```
T(n) = 2T(n/2) + n;

T(1) = 1;
```

Além destes exercícios sugiro os seguintes exercícios do livro Texto [Ziviani,2.a Edição]:

Cap. 1: 2, 5, 7, 17, 18.

Cap. 2: 5, 6