Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Disciplina: BCC202 - Estruturas de Dados I
Professor: Túlio Toffolo (www.toffolo.com.br)

Lista de Exercícios 02

Ordem de Complexidade

1) O que significa dizer que uma função *g(n) é O(f(n))*?

2) Indique se as afirmativas a seguir são verdadeiras ou falsas e justifique a sua resposta:
   a. $2^{n+1} = O(2^n)$.
   b. $2^{2n} = O(2^n)$.
   c. É melhor um algoritmo que requer 2^n passos do que um que requer $10n^5$ passos.
   d. f(n) = O(u(n)) e g(n) = O(v(n)) => f(n) + g(n) = O(u(n) + v(n))
   e. f(n) = O(u(n)) e g(n) = O(v(n)) => f(n) - g(n) = O(u(n) - v(n))

3) O **Casamento de Padrões** 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!

```
#define MaxTexto 100
#define MaxPadrao 10

/* Pesquisa o padrao P[1..m] no texto  T[1..n] */
void ForcaBruta(      char T[MaxTexto], int n,
                      char T[MaxPadrao], int m)
{
   int i,j,k;
   for( i = 0 ; i < n - m + 1 ; i++ )
   {
          k = i;
          j = 0;
          while ( ( j <= m ) && ( T[k] == P[j] ) )
          {
                 j = j + 1;
                 k = k + 1;
          }
          if (j > m)
          {
                 printf("Casamento na posicao %d",i);
                 break;
          }
   }
}
```

4) Suponha um algoritmo A e um algoritmo B com funções de complexidade de tempo $a(n) = n^2 - n + 549$ e $b(n) = 49n + 49$, respectivamente. Determine quais são os valores de n pertencentes ao conjunto dos números naturais para os quais A leva menos tempo para executar do que B.

5) Defina um Tipo Abstrato de Dados `TMatriz`, para representar matrizes quadradas de tamanho n. Implemente as operações para somar e multiplicar 2 matrizes. Explique qual é a ordem de complexidade dessas duas operações. Se você tivesse a opção de utilizar um algoritmo exponencial $O(2^n)$ para multiplicar duas matrizes, qual algoritmo você iria preferir? Justifique. Qual seria a modificação necessária em seu tipo abstrato de dados para representar matrizes genéricas com dimensões (m,n)? Nesse caso, qual seria a ordem de complexidade para multiplicar 2 matrizes: (m,n) * (n, k)?

6) Considere que você tenha um problema para resolver e duas opções de algoritmos. O primeiro algoritmo é quadrático tanto no pior caso quanto no melhor caso. Já o segundo algoritmo, é linear no melhor caso e cúbico no pior caso. Considerando que o melhor caso ocorre 90% das vezes que você executa o programa enquanto o pior caso ocorre apenas 10% das vezes, qual algoritmo você escolheria? Justifique a sua resposta em função do tamanho da entrada.

7) Apresente a função de complexidade (no pior e melhor caso e no caso médio) para os programas abaixo, fazendo as considerações que considerar pertinente. Lembre-se que a função de complexidade quando não mencionado o caso refere-se ao pior caso.

a)

```
int Max(int A[n])
{
  int i, Temp;

  Temp = A[0];
  for (i = 1; i < n; i++)
    if (Temp < A[i])
      Temp = A[i];
  return Temp;
}
```

b)

```
void MaxMin1(int A[n], int* pMax, int* pMin)
{
  int i;

  *pMax = A[0];
  *pMin = A[0];
  for (i = 1; i < n; i++) {
    if (A[i] > *pMax) *pMax = A[i];
    if (A[i] < *pMin) *pMin = A[i];
  }
}
```

c)

```
void MaxMin2(int A[n], int* pMax, int* pMin)
{
  *pMax = A[0];
  *pMin = A[0];
  for (i = 1; i < n; i++){
    if (A[i] > *pMax) *pMax = A[i];
    else if (A[i] < *pMin) *pMin = A[i];
  }
}
```

d)

```
void MaxMin3(Vetor A, int* pMax, int* pMin)
{
  int i, FimDoAnel;

  if ((n % 2) > 0) {
    A[n] = A[n - 1];
    FimDoAnel = n;
  }
  else FimDoAnel = n - 1;

  if (A[0] > A[1]) { *pMax = A[0]; *pMin = A[1]; }
  else             { *pMax = A[1]; *pMin = A[0]; }

  i = 3;
  while (i <= FimDoAnel) {
    if (A[i - 1] > A[i]) {
      if (A[i - 1] > *pMax) *pMax = A[i - 1];
      if (A[i]     < *pMin) *pMin = A[i];
    }
    else {
      if (A[i - 1] < *pMin) *pMin = A[i - 1];
      if (A[i]     > *pMax) *pMax = A[i];
    }
    i += 2;
  }
}
```

e)

```
void bubblesort( int A[n], int n)
{
  int i,j;
  int aux;

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

f)

```
void bubblesort2( int A[n], int n)
{
  int i,troca;
  int aux;

  do {
    troca = 0;
    for ( i = 0 ; i < n-1 ; i++ ) {
      if ( A[i] > A[i+1] ) {
        aux    = A[i];
        A[i]   = A[i+1];
        A[i+1] = aux;
        troca = 1;
      }
    }
  } while (troca);
}
```

g)

```
void selectsort ( int A[n] , int n)
{
  int i,j,min;
  int aux;

  for ( i = 0 ; i < n - 1 ; i++ ) {
    min = i;
    for ( j = i + 1 ; j < n ; j++ )
      if ( A[j] < A[min] )
        min = j;

    aux = A[min];
    A[min] = A[i];
    A[i] = aux;
  }
}
```

h)

```
void insertsort(int A[n], int n )
{
  int j;
  for (int i = 1; i < n; i++) {
    aux  = A[i];
    j = i - 1;

    while ( ( j >= 0 ) && ( aux < v[j] ) ) {
      v[j + 1] = v[j];
      j--;
    }
    v[j + 1] = aux;
  }
}
```

i)

```
void FazAlgo( int n )
{
  int i,j,k;

  x = 0;
  for( i = 1 ; i <= n - 1 ; i++ ) {
    for( j = i + 1 ; j <= n ; j++ ) {
      for( k = 1 ; k <= j ; k++ )
        x = x + 1;
    }
}
```

j)

```
void FazAlgo2( int n)
{
  int i,j,k,x;
  x = 0;
  for( i = 1 ; i <= n ; i ++)
    for( j = i + 1 ; j <= n - 1 ; j++ )
      for( k = 1 ; k <= j ; k++ )
        x = x + 1;
}
```