PROGRAMAÇÃO DE COMPUTADORES em C

Prof. Geraldo Pereira de Souza (geraldo@mlink.com.br)

Roteiro Prático

Objetivo: praticar o uso de algoritmos de ordenação para tipos simples e objetos em C++

Parte 1 – Revisão dos conceitos sobre ponteiros e alocação dinâmica

Etapa 1 (5 minutos): baixe os slides (do moodle) da aula teórica e reveja os conceitos relacionados a esse tópico.

Etapa 2 (20 minutos): Responda as questões abaixo:

a) O quê você entende por algoritmos de ordenação e por quê eles são importantes?
b) Quais requisitos são importantes na escolha de um algoritmo de ordenação?
c) Quando dizemos que um algoritmo tem comportamento conforme notação O(n), O(n^2) ou O(log n) o que isso representa? Explique e desenhe as respectivas curvas.

Etapa 3 (10 minutos): Crie um projeto com nome TesteOrdenacao e faça o seguinte:

a) O programa deve declarar um vetor de 10 posições de inteiros;
b) Solicite que o usuário digite 10 números de modo aleatório e guarde em cada posição do vetor;
c) Dentro do programa invoque o método de ordenação bolha fornecido abaixo:

```
void bolha (int v[], int n){
  for (int i=n-1; i>=1; i--){
    for (int j=0; j<i; j++){
      if (v[j]>v[j+1]) { /* troca */
        int temp = v[j];
        v[j] = v[j+1];
        v[j+1] = temp;
      }
    }
  }
}
```

d) Imprima e conteúdo do vetor antes e depois de ordená-lo;
e) Mude o programa para considerar um vetor de tamanho 10, 50, 100, 500 e 1000. Para isso crie uma função chamada carregaVetor que gera números aleatórios para cada posição do vetor. A função abaixo gera um número aleatório entre “de” e “ate”. Veja:

```
int getRandom(int de, int ate)
{
   int random;
   ate -= de;
   random = rand() % (ate + 1) + de;
   return random;
}
```

A função carregaVetor deve ter a seguinte assinatura:

void carregarVetor (int v[], int n){} // v é o vetor a ser carregado com números aleatórios e n é número de elementos a serem considerados no vetor.

Etapa 4: Veja uma variação do método bubble:

```
void bubbleSort(int a[], int n) {

        for (int i = n; --i>=0; ) {
           bool troca = false;
           for (int j = 0; j<i; j++) {

        if (a[j] > a[j+1]) {
            int T = a[j];
            a[j] = a[j+1];
            a[j+1] = T;
            troca = true;
        }
           }
           if (!troca)
        return;
        }
}
```

Etapa 5: Mude o main do seu programa para usar a implementação bubbleSort com uma melhoria implementada na etapa 4:

Etapa 6: Veja a implementação do método de ordenação quickSort. Primeiro crie um projeto com nome ProjetoOrdenacaoQuickSort e incorpore o código fonte abaixo no main.cpp

```
#include <iostream>

using namespace std;

void swap(int a[], int i, int j)
{
      int T;
      T = a[i];
      a[i] = a[j];
      a[j] = T;

   }
```

```
void QuickSort(int a[], int indiceEsquerdo, int indiceDireito)
  {
     int i = indiceEsquerdo;
     int j = indiceDireito;
     int mid;

     if ( indiceDireito > indiceEsquerdo)
     {

        /* Pega o pivô.
         */
        mid = a[ ( indiceEsquerdo + indiceDireito ) / 2 ];

        // repete até que as extremidades se cruzem
        while( i <= j )
        {
           /* procura o primeiro elemento que seja maior ou igual
            * ao pivê começando do lado esquerdo.
            */

            while( ( i < indiceDireito ) && ( a[i] < mid ))
               ++i;

           /* procura o primeiro elemento que seja menor ou igual
            * ao pivô começando do lado direito.
            */
            while( ( j > indiceEsquerdo ) && ( a[j] > mid ))
               --j;

           // se os índices não se cruzaram, efetua a troca
           if( i <= j )
           {
              swap(a, i, j);
              ++i;
              --j;
           }
        }

        /* Ordenar a parte indiceEsquerdo.
         */
        if( indiceEsquerdo < j )
           QuickSort( a, indiceEsquerdo, j );

        /* Se os índices não se cruzaram,
         * ordenar a parte indiceDireito.
         */
        if( i < indiceDireito )
            QuickSort( a, i, indiceDireito );
     }
  }

 void sort(int a[], int n)
  {
     QuickSort(a, 0, n - 1);
  }

int main()
```

```
{
   int v[8] = {25,48,37,12,57,86,33,92};
   cout << "\nVetor desordenado: \n";
   for (int i=0; i<8; i++){
      cout << v[i] << " ";
   }
   sort(v, 8);
   cout << "\n\nVetor ordenado: \n";
   for (int i=0; i<8; i++){
      cout << v[i] << " ";
   }
   cout << "\nFim da impressão do vetor!" << endl;
    return 0;
}
```

Etapa 8: Rode e execute o código.

Etapa 9: Na função main do seu programa, simule a execução dos métodos de ordenação para vetores de tamanho 100, 200, 500, 1000, 10.000, etc; Faça medições de tempo para as execuções para cada tamanho e para cada método de ordenação. Pesquise na internet sobre “Como medir tempo de execução de uma função em C++”.