ATIVIDADE SORTING

Comparar o tempo de execução (em segundos) dos três algoritmos de ordenação, considerando uma lista com N números aleatórios.

O que deverá ser entregue:

1) Comparação dos valores encontrados na execução do programa ilustrando essa comparação por meio de um único gráfico contendo os dados das três simulações, como mostra a Figura 1.
2) A Tabela 1 com os resultados obtidos pelo seu programa (4 casas decimais).
3) Comentários sobre os resultados encontrados.
4) O programa fonte em C usado para obter os resultados.

Tabela 1

	N = tamanho da lista				
Sorting	10000	20000	50000	100000	150000
SelectionSort					
InsertionSort					
BubbleSort					

INSTRUÇÕES

- Utilizar o programa **`LABORATORIO TempoSorting.c`** como base para o seu programa.
- Esse programa usa a biblioteca **`"TipoListaInteiro.h"`** onde se encontram as implementações dos algoritmos de ordenação. Não é necessário acrescentar outras funções à biblioteca.
- Nesse programa há duas funções implementadas (que devem ser mantidas e usadas) para a construção da lista de números aleatórios e para mostrar os valores armazenados.
- Mostrar a lista original e mostrar a lista ordenada em um teste com uma lista de 20 números, para cada um dos quatro algoritmos antes de executar o programa com os valores de N dados na tabela.
- ATENÇÃO – os três algoritmos devem ser executados com a mesma lista de números.
- Nome do arquivo: **`<seu nome com sobrenome>_Sorting.c`**

1) ALGORITMOS DE ORDENAÇÃO

Ordenação por Seleção Direta

```
SelectionSort(A)
  n ← nA;
  para j de 1 até (n-1) repita
      min ←j;
      para k de (j+1) até nA repita  se (A[k]< A[min]) então min ← k;
      aux ← A[j];  A[j] ← A[min];  A[min ← aux ;
```

$EC(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = n(n-1)/2 = O(n^2)$

Ordenação por Inserção Direta

```
InsertionSort(A)
  n ← nA;
  para j de 2 até n repita
      aux ← A[j];  k ← j-1;
      enquanto ((k>0) e (A[k] > aux)) faça  A[k+1] ← A[k];  k ← k-1;
      A[k+1] ← aux;
```

Complexidade

Pior caso – lista invertida

$wEC(n) = 2 + 3 + 4 + ... + n = (n-1)(n+2)/2 = O(n^2)$

Melhor Caso – lista ordenada

$bEC(n) = 1 + 1 + 1 + ... + 1 = n-1 = O(n)$

Caso médio - lista aleatória

$AEC(n) = (n – 1)(n+4)/4 = O(n^2)$

Ordenação pelo algoritmo Bolha

```
BubbleSort(A)
  n ← nA;  fim ← n-1;
  para j de 1 até (n-1) repita
      para k de 1 até fim repita
          se (A[k] > A[k+1]) então  aux ← A[k];  A[k] ← A[k+1];  A[k+1] ← aux  ;
      fim ← fim – 1;
```

$EC(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = n(n-1)/2 = O(n^2)$

2) IMPLEMENTAÇÕES DOS ALGORITMOS DE ORDENAÇÃO

O arquivo “TipoListaInteiro.h” contém as implementações dos três algoritmos de ordenação, SelectionSort, InsertionSort e BubbleSort, de mesma ordem de complexidade, cujas interfaces são relacionadas a seguir:

```
void ordenarListaCrescenteS(ListaInt *); // ordenar pelo SelectionSort
void ordenarListaCrescenteI(ListaInt *); // ordenar pelo InsertionSort
void ordenarListaCrescenteB(ListaInt *); // ordenar pelo BubbleSort
```

Efetuamos medidas de tempo de execução desses processos, com uma mesma lista aleatória, e obtivemos o seguinte gráfico que mostra uma pequena diferença entre os tempos de execução de cada um dos três processos.

Figura 1