Lab. 13

Ordenação

ECT2303 - T01 - 19.1

13.1 Exercícios de Fixação

1. Implemente o método de ordenação de vetores *bubble sort* para classificar os elementos de um vetor, de tamanho `n`, em ordem ascendente.

```
Exemplo de execução:
Informe o tamanho do vetor: 10
Informe os elementos do vetor: 2 6 4 8 10 12 89 68 45 37

Passo 1:   2   4   6   8  10  12  68  45  37  89
Passo 2:   2   4   6   8  10  12  45  37  68  89
Passo 3:   2   4   6   8  10  12  37  45  68  89
Passo 4:   2   4   6   8  10  12  37  45  68  89
Passo 5:   2   4   6   8  10  12  37  45  68  89
Passo 6:   2   4   6   8  10  12  37  45  68  89
Passo 7:   2   4   6   8  10  12  37  45  68  89
Passo 8:   2   4   6   8  10  12  37  45  68  89
Passo 9:   2   4   6   8  10  12  37  45  68  89

Dados em ordem ascendente:
2   4   6   8  10  12  37  45  68  89
```

2. A classificação com o *bubble sort* é ineficiente para vetores grandes. Faça as modificações simples a seguir para melhorar o desempenho do *bubble sort*.

   (a) Depois da primeira passagem, garante-se que o maior número esteja no elemento de maior número do vetor; depois da segunda passagem, os dois maiores números estão em "seus lugares", e assim por diante. Em vez de fazer `n-1` comparações em cada passagem, modifique o *bubble sort* para fazer `n-2` comparações na segunda passagem, `n-3` na terceira passagem e assim por diante.

```
Exemplo de execução:
Informe o tamanho do vetor: 10
Informe os elementos do vetor: 2 6 4 8 10 12 89 68 45 37

Passo 1:   2  4  6  8 10 12 68 45 37 89
Passo 2:   2  4  6  8 10 12 45 37 68
Passo 3:   2  4  6  8 10 12 37 45
Passo 4:   2  4  6  8 10 12 37
Passo 5:   2  4  6  8 10 12
Passo 6:   2  4  6  8 10
Passo 7:   2  4  6  8
Passo 8:   2  4  6
Passo 9:   2  4

Dados em ordem ascendente:
2   4   6   8  10  12  37  45  68  89
```

(b) Os dados do vetor podem estar na ordem adequada ou quase ordenados completamente; assim, por que fazer `n-1` passagens se um número menor seria suficiente? Modifique a classificação para verificar se alguma permuta foi feita ao final de cada passagem. Se nenhuma permuta foi realizada, os dados já devem estar na ordem adequada e o programa deve ser encerrado. Se foram feita permutas, é necessário pelo menos mais uma passagem.

```
Exemplo de execução:
Informe o tamanho do vetor: 10
Informe os elementos do vetor: 2 6 4 8 10 12 89 68 45 37

Passo 1:   2  4  6  8 10 12 68 45 37 89
Passo 2:   2  4  6  8 10 12 45 37 68
Passo 3:   2  4  6  8 10 12 37 45
Passo 4:   2  4  6  8 10 12 37

Dados em ordem ascendente:
2   4   6   8  10  12  37  45  68  89
```

3. Uma classificação por seleção (*Selection Sort*) pesquisa um vetor à procura do seu menor elemento. Quando o menor elemento for encontrado, ele é permutado com o primeiro elemento do vetor. O processo é então repetido para o subvetor que se inicia com o segundo elemento do vetor. Cada passada do vetor resulta na colocação de um elemento em seu local apropriado. Essa classificação exige capacidade de processamento idêntica à da classificação com o *Buble Sort* - para um vetor de `n` elementos, deve ser feitas `n - 1` passagens, e para cada subvetor devem ser feitas `n - 1` comparações para encontrar o menor valor. Quando o subvetor a ser processado possuir um elemento, o vetor estará ordenado. Escreva uma função `selectionSort` para implementar esse algoritmo.

```
Exemplo de execução:
Informe o tamanho do vetor: 10
Informe os elementos do vetor: 12 68 4 89 10 2 8 6 45 37
```

```
Passo 0:  12  68   4  89  10   2   8   6  45  37
Passo 1:   2  68   4  89  10  12   8   6  45  37
Passo 2:   2   4  68  89  10  12   8   6  45  37
Passo 3:   2   4   6  89  10  12   8  68  45  37
Passo 4:   2   4   6   8  10  12  89  68  45  37
Passo 5:   2   4   6   8  10  12  89  68  45  37
Passo 6:   2   4   6   8  10  12  89  68  45  37
Passo 7:   2   4   6   8  10  12  37  68  45  89
Passo 8:   2   4   6   8  10  12  37  45  68  89
Passo 9:   2   4   6   8  10  12  37  45  68  89

Dados em ordem ascendente:
2   4   6   8  10  12  37  45  68  89
```

4. Na classificação por inserção (*Insertion Sort*) a primeira iteração do algoritmo seleciona o segundo elemento no vetor e, se for menor que o primeiro elemento, permuta-o pelo primeiro elemento. A segunda iteração examina o terceiro elemento e o insere na posição correta com relação aos dois primeiros elementos de modo que todos os três elementos estejam na ordem. Na i-ésima iteração desse algoritmo, os primeiros i elementos no vetor original estarão classificados. Escreva uma função `insertionSort` para implementar esse algoritmo.

```
Exemplo de execução:
Informe o tamanho do vetor: 10
Informe os elementos do vetor: 56 34 4 10 77 51 93 30 5 52
Passo 0:  56  34   4  10  77  51  93  30   5  52
Passo 1:  34  56   4  10  77  51  93  30   5  52
Passo 2:   4  34  56  10  77  51  93  30   5  52
Passo 3:   4  10  34  56  77  51  93  30   5  52
Passo 4:   4  10  34  56  77  51  93  30   5  52
Passo 5:   4  10  34  51  56  77  93  30   5  52
Passo 6:   4  10  34  51  56  77  93  30   5  52
Passo 7:   4  10  30  34  51  56  77  93   5  52
Passo 8:   4   5  10  30  34  51  56  77  93  52
Passo 9:   4   5  10  30  34  51  52  56  77  93

Dados em ordem ascendente:
4   5  10  30  34  51  52  56  77  93
```

13.2 Referências Bibliográficas

1. ASCENCIO, A.F.G.; CAMPOS, E.A.V. **Fundamentos da Programação de Computadores - Algoritmos, Pascal e C/C++**. 3ed. Editora Pearson.

2. DEITEL, H. M.; DEITEL, P. J. **C++ Como Programar**. 3ed. Editora Bookman.