# ALGORITMOS DE ORDENAÇÃO

**PROBLEMA –** Rearranjar os registros de uma lista em uma ordem determinada (ordem numérica ou alfabética). A ordenação é considerada em relação a um determinado campo denominado chave da ordenação. Se os registros da lista contêm vários campos, um único campo deve ser escolhido como chave da ordenação.

**Métodos de ordenação –** Se a lista é inteiramente carregada na memória principal, os métodos de ordenação são denominados métodos de ordenação interna. Se a lista fica armazenada em dispositivos externos os métodos de ordenação são denominados métodos de ordenação externa. A diferença básica está no acesso aos registros da lista, que no segundo caso, deve ser feito em blocos ou seqüencialmente.

**Algoritmos de ordenação** – Um algoritmo de ordenação pode ser **Local**, em que os registros são rearranjados fisicamente ou **Indireto**, em que a ordenação é indicada por meio de apontadores (indicadores de posição).

**Algoritmo de ordenação estável** – Um algoritmo de ordenação é estável se preserva a ordem original para os registros que possuem valores iguais no campo chave.

**Estratégias para ordenação** – Os principais métodos de ordenação são baseados nas seguintes estratégias:

- **Ordenação por trocas** – Cada dois registros da lista são trocados de posição até que a ordem desejada seja atingida. Os algoritmos BubbleSort e QuickSort pertencem a esse grupo.
- **Ordenação por seleção** – A cada etapa, o maior (ou menor) item é selecionado e posicionado no lugar definitivo. Os algoritmos SelectionSort e HeapSort pertencem a esse grupo.
- **Ordenação por inserção** – A cada etapa, um registro é inserido em uma parte da lista já ordenada. O algoritmo InsertionSort é baseado nessa idéia.

**Complexidade do algoritmo** – A complexidade de algoritmos de ordenação é medida em função do número de comparações e cópias de registros. Nos métodos de ordenação elementares, como o BubbleSort, o SelectionSort e o InsertionSort, esse número é proporcional a $n^2$ (em que n é o número de registros da lista).

Vamos considerar nesse texto o problema de ordenar uma lista de números inteiros em ordem crescente. Admitimos que a lista não contém duplicatas, não é vazia e encontra-se armazenada em uma estrutura de armazenamento de dados A(lista de inteiro),nA(inteiro).Seja a lista de inteiros: 31, 41, 59, 26, 53, 58, 97, 93

## Ordenação por seleção direta

O algoritmo SelectionSort é um algoritmo de ordenação por seleção direta, em que a cada etapa, o maior elemento é localizado e posicionado no sua posição definitiva. Assim, na etapa 1 o maior item é colocado na última posição, na etapa 2, considera-se a sublista contendo do primeiro até o penúltimo registro, e o maior item é localizado e colocado no final da sublista, e assim por diante até que na última etapa (são n-1 etapas) o maior dos dois itens restantes é colocado na segunda posição.

SelectionSort(A)

```
n ← nA;  fim ← n;
para j de 1 até (n-1) repita
    pos ← 1;
    para k de 2 até fim repita  se (A[k] > A[pos]) então pos ← k;
    se (pos ≠ fim) então [ aux ← A[fim];  A[fim] ← A[pos];  A[pos] ← aux ] ;
    fim ← fim – 1;
```

Simulação do algoritmo

| ETAPA | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | pos | fim |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 31 | 41 | 59 | 26 | 53 | 58 | **97** | 93 | 7 | 8 |
| | 31 | 41 | 59 | 26 | 53 | 58 | 93 | 97 | Troca | A[7] e A[8] |
| 2 | 31 | 41 | 59 | 26 | 53 | 58 | **93** | 97 | 7 | 7 |
| 3 | 31 | 41 | **59** | 26 | 53 | 58 | 93 | 97 | 3 | 6 |
| | 31 | 41 | 58 | 26 | 53 | 59 | 93 | 97 | Troca | A[3] e A[6] |
| 4 | 31 | 41 | **58** | 26 | 53 | 59 | 93 | 97 | 3 | 5 |
| | 31 | 41 | 53 | 26 | 58 | 59 | 93 | 97 | Troca | A[3] e A[5] |
| 5 | 31 | 41 | **53** | 26 | 58 | 59 | 93 | 97 | 3 | 4 |
| | 31 | 41 | 26 | 53 | 58 | 59 | 93 | 97 | Troca | A[3] e A[4] |
| 6 | 31 | **41** | 26 | 53 | 58 | 59 | 93 | 97 | 2 | 3 |
| | 31 | 26 | 41 | 53 | 58 | 59 | 93 | 97 | Troca | A[2] e A[3] |
| 7 | **31** | 26 | 41 | 53 | 58 | 59 | 93 | 97 | 1 | |
| | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | Troca | A[1] e A[2] |
| | **26** | **31** | **41** | **53** | **58** | **59** | **93** | **97** | FIM | |

Complexidade (Pior caso = Melhor caso = Caso médio):

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

**Ordenação por trocas - BubbleSort**

O algoritmo BubbleSort é um algoritmo de ordenação por trocas, em que a cada etapa, itens consecutivos são comparados e, se necessário, são trocados de posição. Como resultado da primeira etapa, o maior elemento é levado mediante sucessivas trocas até a sua posição definitiva. Na etapa 2, considera-se a sublista contendo do primeiro até o penúltimo registro, e a varredura dessa sublista tem por finalidade comparar (e trocar) pares de itens consecutivos de forma que o maior item é levado para o final da sublista, e assim por diante até que na última etapa (são n-1 etapas) os dois primeiros itens são comparados (trocados, se necessário) de modo que o maior dos dois ocupe a segunda posição.

BubbleSort(A)

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

Simulação do algoritmo

| ETAPA | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | fim |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 31 | 41 | 59 | 26 | 53 | 58 | 97 | 93 | | 8 |
| | 31 | 41 | 26 | 53 | 58 | 59 | 93 | 97 | Trocas | 3,4,5,7 |
| 2 | 31 | 41 | 26 | 53 | 58 | 59 | 93 | 97 | | 7 |
| | 31 | 26 | 41 | 53 | 58 | 59 | 93 | 97 | Trocas | 2 |
| 3 | 31 | 26 | 41 | 53 | 58 | 59 | 93 | 97 | | 6 |
| | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | Trocas | 1 |
| 4 | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | | 5 |
| 5 | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | | 4 |
| 6 | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | | 3 |
| 7 | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | | 2 |
| | **26** | **31** | **41** | **53** | **58** | **59** | **93** | **97** | FIM | |

Complexidade (Pior caso = Melhor caso):

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

**Ordenação por inserção direta - InsertionSort**

Nesse processo, consideramos a lista formada por duas sublistas: uma sublista esquerda, já ordenada e uma sublista direita, não ordenada. A cada etapa, um item da sublista direita é inserido na sublista esquerda. Na primeira etapa, a sublista esquerda é unitária, contendo somente o primeiro registro e a sublista direita contém os n-1 registros restantes. Como resultado da primeira etapa, a sublista esquerda passa a conter os dois primeiros registros, já ordenados. Na etapa 2, o terceiro registro é inserido na sublista esquerda. Na última etapa (são n-1 etapas) a sublista esquerda contém (n-1) registros ordenados e o último registro é finalmente inserido. A cada etapa, é necessário efetuar uma busca (busca em lista ordenada) seguida de uma inserção. O processo de busca e o de inserção podem ser conjugados de forma a compartilhar o mesmo loop.

InsertionSort(A)

```
n ← nA;  fim ← n;
para j de 2 até n repita
    A[0] ← A[j];  k ← j-1;
    enquanto (A[0] < A[k]) faça [A[k+1] ← A[k];  k ← k-1] ;
    A[k+1] ← A[0];
```

Simulação do algoritmo

| Etapa | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | **j** |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 31 | 41 | 59 | 26 | 53 | 58 | 97 | 93 | | 2 |
| 2 | 31 | 41 | 59 | 26 | 53 | 58 | 97 | 93 | | 3 |
| 3 | 31 | 41 | 59 | 26 | 53 | 58 | 97 | 93 | | 4 |
| | 26 | 31 | 41 | 59 | 53 | 58 | 97 | 93 | inserção | A[1] |
| 4 | 26 | 31 | 41 | 59 | 53 | 58 | 97 | 93 | | 5 |
| | 26 | 31 | 41 | 53 | 59 | 58 | 97 | 93 | inserção | A[4] |
| 5 | 26 | 31 | 41 | 53 | 59 | 58 | 97 | 93 | | 6 |
| | 26 | 31 | 41 | 53 | 58 | 59 | 97 | 93 | inserção | A[5] |
| 6 | 26 | 31 | 41 | 53 | 58 | 59 | 97 | 93 | | 7 |
| 7 | 26 | 31 | 41 | 53 | 58 | 59 | 97 | 93 | | 8 |
| | 26 | 31 | 41 | 53 | 58 | 59 | 93 | 97 | inserção | A[7] |
| | **26** | **31** | **41** | **53** | **58** | **59** | **93** | **97** | FIM | |

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

$AEC(n) = (n - 1)(n+4)/4 = O(n^2)$