

Métodos Simples de Ordenação de Vetores (*Sorting*)

1. Definição de Ordenação

Classificação ou ordenação, em computação, é o processo de rearranjar um dado conjunto de objetos em uma ordem específica. O propósito da ordenação é facilitar a busca futura de elementos no conjunto classificado.

Dado um conjunto de n elementos numerados

$$a_1, a_2, \dots, a_n$$

Ordenar consiste em permutar os elementos

$$a_{k1}, a_{k2}, \dots, a_{kn}$$

Através de uma função f de ordenamento, tal que:

$$f(a_{k1}) \leq f(a_{k2}) \leq \dots \leq f(a_{kn})$$



➔ Existem vários métodos de ordenação interna (estruturas em memória principal - vetores) e externas (arquivos). A escolha do método de ordenação interna leva em consideração um compromisso entre eficiência do método e esforço de implementação.

2. Eficiência dos métodos de ordenação: ordem de complexidade

O critério mais importante na medida de eficiência de um método de ordenação, que é independente da máquina, é o número médio de comparações entre os elementos do conjunto. O número de trocas necessárias para se realizar a ordenação também pode ser considerado.

Pode-se definir a expressão matemática que avalia o número de comparações que um método de ordenação faz para ordenar um certo conjunto de dados de entrada. Se um método é submetido a todos os conjuntos de dados de entrada possíveis e o número de comparações é computado, podem ser definidas a *complexidade de pior caso*, a *complexidade de melhor caso* e a *complexidade do caso médio*. As complexidades de um método têm por objetivo avaliar a eficiência de seu tempo de execução. Assim, a complexidade de tempo de pior caso corresponde ao número de comparações que o algoritmo efetua no seu pior caso de execução, isto é, para a entrada de dados mais desfavorável (que pode ser um conjunto de dados em ordem decrescente). Este valor fornece um limite superior para o número de comparações que o algoritmo pode efetuar em qualquer caso.

Para expressar a complexidade de um algoritmo é utilizada a notação O, que indica a ordem de grandeza da complexidade. Dentre os diversos métodos de ordenação de vetores, há aqueles considerados de desempenho mais simples (bolha, seleção e inserção), cuja ordem de complexidade é quadrática ou $O(n^2)$, e os de melhor desempenho (quicksort, mergesort, árvore binária de ordenação), com ordem de complexidade logarítmica ou $O(\log n)$.

Neste capítulo serão apresentados os métodos de ordenação mais simples, de ordem de complexidade $O(n^2)$, ou seja, o número de comparações que eles realizam é proporcional ao quadrado do tamanho do conjunto de dados de entrada.

Para que se possa ter uma idéia mais precisa da diferença de desempenho entre os métodos de ordenação de ordem quadrática e logarítmica, veja a tabela 1.

Seja n o número de elementos do conjunto a ser ordenado, observe o número de comparações que 2 diferentes métodos de ordenação realizam. O primeiro método faz $(n^2 - n) / 2$ comparações, ou seja, é de ordem de complexidade $O(n^2)$, e o segundo método faz $\text{trunc}(2n \log_2 n)$ comparações, ou seja, é de ordem logarítmica ou $O(n \log_2 n)$.

Tabela 1
Comparação de métodos de ordenação de ordem de complexidade $O(n^2)$ x $O(n \log_2 n)$

n	$(n^2 - n) / 2$	$\text{trunc}(2n \log_2 n)$
5	10	23
10	45	66
20	190	172
100	4950	1300
1000	499500	19000

A tabela 1 mostra que para pequenas entradas de dados, ou seja, vetores com até 20 elementos, o número de comparações feitas pelos métodos é equivalente. Mas quando se aumenta o número de elementos para 100 e depois para 1000, a eficiência do método de ordem logarítmica fica muito superior.

O mais conhecido desses métodos é o **método da bolha ou bubble sort**. Os outros dois são o **insertion sort ou método de inserção** e o **selection sort ou método de seleção**.

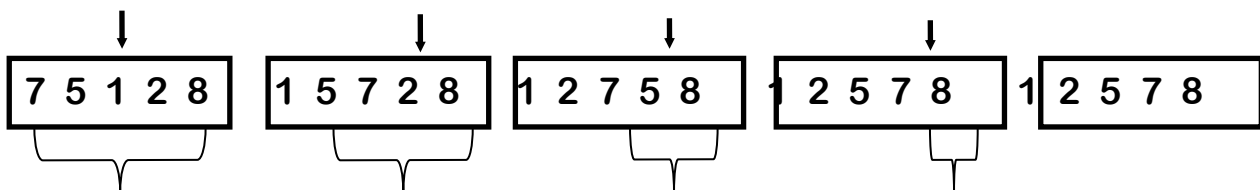
No método **insertion sort**, ou *ordenação por inserção*, começa-se a percorrer o vetor na 2ª posição em direção ao seu início, comparando os elementos da direita para a esquerda e, à medida que se avança, o método vai deixando os elementos mais à esquerda ordenados. O algoritmo funciona da mesma maneira com que muitas pessoas ordenam cartas em um jogo de baralho, como o pôquer.

O método da bolha *bubble sort*, ou ordenação por flutuação, é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vetor diversas vezes e, a cada passagem, ir fazendo o maior elemento flutuar para o final do vetor, onde o maior elemento da sequência deve estar.

O método de ordenação *selection sort* (ordenação por seleção) é um algoritmo que realiza sempre um número fixo de comparações e um número fixo de passos. O método seleciona o menor valor do vetor e o coloca na primeira posição, depois seleciona o segundo menor valor e o coloca na segunda posição, e assim sucessivamente com os $(n-1)$ elementos restantes.

3. Método de ordenação por seleção (*selection sort*)

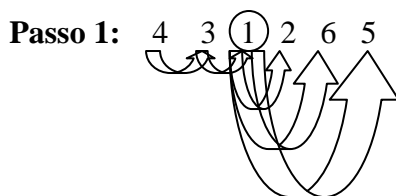
Percorre-se o vetor e seleciona-se a menor chave, esta chave é trocada de posição com o 1º elemento, no primeiro passo; seleciona-se a menor chave, ignorando-se a 1ª posição que já está ordenada, e troca-se esta chave com o 2º elemento no segundo passo; e assim sucessivamente.



- **Exemplo passo a passo:**

Seja o vetor inicial:

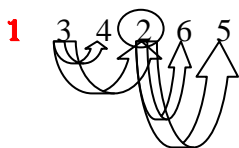
1	2	3	4	5	6
4	3	1	2	6	5



O menor elemento é o número 1 que está na posição 3 do vetor. Trocam-se os elementos da posição 1 e 3, ou seja, o menor elemento deve ficar na posição 1 no final do passo 1. Foram feitas 5 comparações (conte o número de setas) e 1 troca no passo 1.

1 3 4 2 6 5

Passo 2:



No 2º passo, a comparação começa da posição 2 do vetor, pois a posição 1 já está ordenada. O menor elemento é o número 2 que está na posição 4 do vetor. Trocam-se os elementos da posição 2 e 4, ou seja, o 2º menor elemento deve ficar na posição 2 no final do passo 2. Foram feitas 4 comparações (conte o número de setas) e 1 troca no passo 2.

1 2 4 3 6 5

Passo 3:



No 3º passo, a comparação começa da posição 3 do vetor, pois as posições 1 e 2 já estão ordenadas. O menor elemento é o número 3 que está na posição 4 do vetor. Trocam-se os elementos da posição 3 e 4, ou seja, o 3º menor elemento deve ficar na posição 3 no final do passo 3. Foram feitas 3 comparações (conte o número de setas) e 1 troca no passo 3.

1 2 3 4 6 5

Passo 4:



No 4º passo, a comparação começa da posição 4 do vetor, pois as posições 1, 2 e 3 já estão ordenadas. O menor elemento é o número 4 que está na posição 4 do vetor. Trocam-se os elementos da posição 4 e 4, ou seja, o 4º menor elemento deve ficar na posição 4 no final do passo 4. Foram feitas 2 comparações (conte o número de setas) e 1 troca no passo 4.

1 2 3 4 6 5

Passo 5:



No 5º e último passo, a comparação começa da posição 5 do vetor, pois as posições 1, 2, 3 e 4 já estão ordenadas. O menor elemento é o número 5 que está na posição 6 do vetor. Trocam-se os elementos da posição 5 e 6, ou seja, o 5º menor elemento deve ficar na posição 5 no final do passo 5. Foi feita 1 comparação (conte o número de setas) e 1 troca no passo 5.

1 2 3 4 5 6

Resultado do método de seleção: Foram feitas 15 comparações e 5 trocas

```
/* selecao.c: implementa o metodo de ordenacao por selecao - selection
sort */

#include <stdio.h>
#include <conio.h>

#define N 10

int i, j, posmenor, menor, aux;
int vet[N+1];
int cont=0; // conta o nro de comparacoes

int main()
{
    printf("\nMetodo de ordenacao por SELECAO \n");
    printf("\n\nForneca os elementos do vetor a ser ordenado \n");
    for (i=1; i<=N; i++)
    { printf("vet[%d]= ",i);
      scanf("%d",&vet[i]);
    }

    for (j=1; j <= N-1; j++)
    {   menor = vet[j];
        posmenor = j;
        for (i=j+1; i <= N; i++)
        {   if (vet[i] < menor)
            {   menor = vet[i];
                posmenor = i;
            }
            ++cont;
        }

        /* vet[posmenor] eh o registro com a menor chave:
           troca vet[j] com vet[posmenor] */

        aux = vet[posmenor];
        vet[posmenor] = vet[j];
        vet[j] = aux;
    }

    printf("\n Vetor ordenado \n");
    for (i=1; i<=N; i++)
        printf("vet[%d]= %d \n",i,vet[i]);

    printf("\n\n Numero de comparacoes na SELECAO: %d ", cont);
    printf("\n\nFim do programa");
    getch();
    return 0;
}
```

3.1. Esforço do Algoritmo de Seleção (Selection Sort)

Para determinar o menor elemento que é colocado na posição 1, são necessárias $n-1$ comparações. Como a posição 1 é ignorada a partir desse momento, o número de comparações para ordenar o elemento 2 é somente $n-2$. Para ordenar a lista inteira o número total de comparações é:

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

Para determinar o valor dessa soma em função de n , a série será escrita 2 vezes, em ordem crescente e decrescente:

$$\begin{array}{c} (n-1) + (n-2) + (n-3) + \dots + 2 + 1 \\ 1 + 2 + \dots + (n-3) + (n-2) + (n-1) \end{array}$$

Observe que $(n-1)+1 = n$, $(n-2)+2 = n$, Adicionando-se as 2 séries obtém-se $n-1$ somas, cada uma com valor n :

$$2 * [(n-1) + (n-2) + (n-3) + \dots + 2 + 1] = (n-1) * n$$

Dessa equação obtemos:

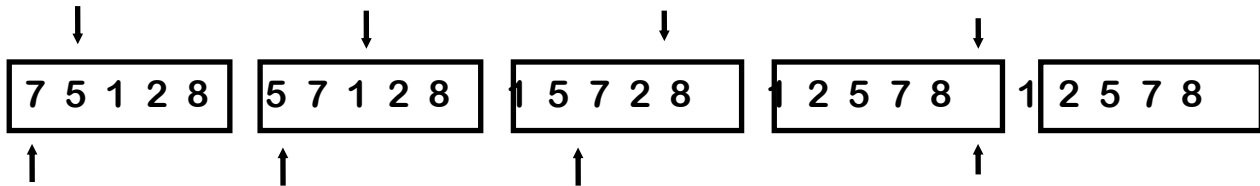
$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1 = \frac{n^2 - n}{2}$$

O que significa que o algoritmo é $O(n^2)$.

O método de Seleção é de ordem de complexidade quadrática ou $O(n^2)$, sendo n o número de elementos do vetor. No passo J sempre ocorrem $N-J$ comparações para localizar o menor elemento, então o número de comparações total para ordenar o vetor não depende de como o vetor está inicializado. Este método realiza $n-1$ passos para ordenar o vetor.

4. Método de ordenação por inserção (*insertion sort*)

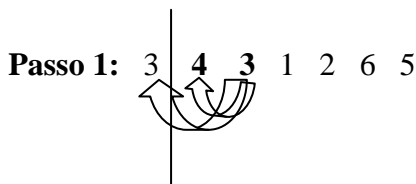
Para um vetor com n elementos, inicia-se a partir do 2º elemento. O método garante que, em cada passo, todos os elementos anteriores ao elemento sendo ordenado, já estão ordenados. Assim, supondo que os elementos $\text{vet}[1] \dots \text{vet}[j-1]$ já estão ordenados, para a ordenação de $\text{vet}[j]$, compara-se $\text{vet}[j]$ com $\text{vet}[j-1]$, com $\text{vet}[j-2]$, ..., até descobrir entre quais elementos este deve ser inserido, ou seja, a comparação com os elementos pára quando um elemento menor ou igual é encontrado ou quando se chega ao início do vetor.



- **Exemplo passo a passo:**

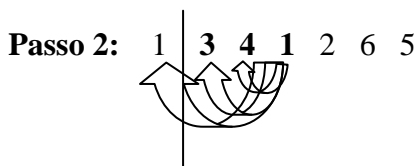
Seja o vetor inicial:

1	2	3	4	5	6
4	3	1	2	6	5



A posição 0 do vetor é utilizada apenas para guardar uma cópia do elemento a ser ordenado, que é o elemento da posição 2 do vetor (elemento 3). A comparação começa da posição 2 do vetor. Compara-se com os elementos anteriores à posição 2 até que se encontre um elemento menor ou igual ao elemento da posição 2, que neste caso, é o número 3. Quando um elemento menor ou igual é encontrado, o passo finaliza e o elemento sendo ordenado (o elemento 3 que está na posição 2) é encaixado. Ou seja, o método procura localizar a posição onde o elemento 3 devia estar. Neste caso, o passo finalizou apenas quando foi feita a comparação com o elemento 3 que está na posição 0 do vetor. O elemento 3, é, então, encaixado na posição 1. Foram feitas 2 comparações (conte o número de setas) e nenhuma troca é realizada.

3 4 1 2 6 5

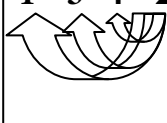


A posição 0 do vetor é utilizada apenas para guardar uma cópia do elemento a ser ordenado, que é o elemento da posição 3 do vetor (elemento 1). A comparação começa da posição 3 do vetor. Compara-se com os elementos anteriores à posição 3 até que se encontre um elemento menor ou igual ao elemento da posição 3, que neste caso, é o número 1. Quando um elemento menor ou igual é encontrado, o passo finaliza e o elemento sendo ordenado (o elemento 1 que está na posição 3) é encaixado. Ou seja, o método

procura localizar a posição onde o elemento 1 devia estar. Neste caso, o passo finalizou apenas quando foi feita a comparação com o elemento 1 que está na posição 0 do vetor. O elemento 1, é, então, encaixado na posição 1. Foram feitas 3 comparações (conte o número de setas) e nenhuma troca é realizada.

1 3 4 2 6 5


Passo 3: 2 | **1 3 4** 2 6 5



A posição 0 do vetor é utilizada apenas para guardar uma cópia do elemento a ser ordenado, que é o elemento da posição 4 do vetor (elemento 2). A comparação começa da posição 4 do vetor. Compara-se com os elementos anteriores à posição 4 até que se encontre um elemento menor ou igual ao elemento da posição 4, que neste caso, é o número 2. Quando um elemento menor ou igual é encontrado, o passo finaliza e o elemento sendo ordenado (o elemento 2 que está na posição 4) é encaixado. Ou seja, o método procura localizar a posição onde o elemento 2 devia estar. Neste caso, o passo finalizou apenas quando foi feita a comparação com o elemento 1 que está na posição 1 do vetor. O elemento 2, é, então, encaixado na posição 2. Foram feitas 3 comparações (conte o número de setas) e nenhuma troca é realizada.

1 2 3 4 6 5

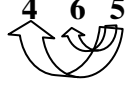
Passo 4: 6 | **1 2 3 4** 6 5



A posição 0 do vetor é utilizada apenas para guardar uma cópia do elemento a ser ordenado, que é o elemento da posição 5 do vetor (elemento 6). A comparação começa da posição 5 do vetor. Compara-se com os elementos anteriores à posição 5 até que se encontre um elemento menor ou igual ao elemento da posição 5, que neste caso, é o número 6. Como o primeiro elemento à esquerda do elemento 6 (elemento 4) já era menor, o passo finaliza e o elemento sendo ordenado (o elemento 6 que está na posição 5) já está na posição onde deveria estar. Foi feita 1 comparação (conte o número de setas) e nenhuma troca é realizada.

1 2 3 4 6 5

Passo 5: 5 | **1 2 3 4** 6 5



A posição 0 do vetor é utilizada apenas para guardar uma cópia do elemento a ser ordenado, que é o elemento da posição 6 do vetor (elemento 5). A comparação começa da posição 6 do vetor. Compara-se com os elementos anteriores à posição 6 até que se encontre um elemento menor ou igual ao elemento da posição 6, que neste caso, é o número 5. Como o segundo elemento à esquerda do elemento 6 (elemento

4) é menor, o passo finaliza e o elemento sendo ordenado (o elemento 5 que está na posição 6) é encaixado na posição onde deveria estar. Foram feitas 2 comparações (conte o número de setas) e nenhuma troca é realizada.

1 2 3 4 5 6

Resultado do método de inserção: Foram feitas 9 comparações e nenhuma troca

```
/* insercao.c: implementa o metodo de ordenacao por insercao - insertion
sort*/

#include <stdio.h>
#include <conio.h>

#define N 10

int vet[N+1];
int aux, j, i;
int cont =0;    // numero de comparações

int main()
{
    printf("\nMetodo de ordenacao por INSERCAO");

    printf("\n\n\nForneca os elementos do vetor a ser ordenado \n");
    for (i=1; i<=N; i++)
    { printf("vet[%d]= ",i);
      scanf("%d",&vet[i]);
    }

    for (j=2; j<=N; j++)
    { aux = vet[j];
      vet[0] = aux;
      i = j-1;
      while (aux < vet[i])
      { vet[i+1] = vet[i];
        --i;
        ++cont;
      }
      vet[i+1] = aux;
      ++cont;
    }

    printf("\n Vetor ordenado pelo INSERCAO \n");
    for (i=1; i<=N; i++)
        printf("vet[%d]= %d \n",i, vet[i]);

    printf("\n Numero de comparacoes na INSERCAO: %d ", cont);
    printf("\n\n\nFim do programa");
    getch();
    return 0;
}
```

4.1. Esforço do Algoritmo de Inserção (Insertion Sort):

No j -ésimo passo, é necessário fazer uma pesquisa sequencial para um vetor de $j-1$ posições. Considerando NC^{\min}_j , NC^{\max}_j , NC^{med}_j como os números mínimo, máximo e médio de comparações no j -ésimo passo, temos:

$$\begin{aligned} NC^{\min}_j &= 1 \\ NC^{\max}_j &= j \\ NC^{\text{med}}_j &= j / 2 \end{aligned}$$

Como j varia de 2 até N , temos:

$$\begin{aligned} NC^{\min} &= \sum_{j=2}^N NC^{\min}_j = N-1 \\ NC^{\max} &= \sum_{j=2}^N NC^{\max}_j = 2 + 3 + \dots + N = \frac{N(N-1) - 1}{2} \\ NC^{\text{med}} &= \sum_{j=2}^N NC^{\text{med}}_j = \frac{2 + 3 + \dots + N}{2} = \frac{N(N-1)}{4} - \frac{1}{2} \end{aligned}$$

O método de Inserção é de ordem de complexidade quadrática ou $O(n^2)$, sendo N o número de elementos do vetor. Como no passo j podem acontecer 1, j ou $j/2$ comparações, o número de comparações total para ordenar o vetor, depende de como o vetor está inicializado. Este método realiza $N-1$ passos para ordenar o vetor.

5. Comparação entre os métodos simples de ordenação

O método da bolha e o método de seleção são ambos $O(N^2)$. Mas o número de comparações no método da bolha, para casos em que o vetor está em ordem crescente ou ordenado, é menor que o método de seleção, embora o bolha apresente uma enorme desvantagem, que o torne o de pior desempenho de todos os três métodos estudados: ele exige um grande número de trocas.

Entre o método de inserção e o de seleção, o de inserção é superior em relação ao número médio de comparações, mas o número de trocas na seleção cresce linearmente com N , enquanto na inserção cresce com o quadrado de N .

Portanto, não existe entre esses três métodos um que seja bastante superior, mas apenas um que seja mais lento (o da bolha). Além disso os três algoritmos são de ordem $O(N^2)$.