

Estrutura de Dados

Ordenação: Introdução

Professores: Anisio Lacerda
Wagner Meira Jr.

Ordenação

■ Objetivo:

- ❑ Rearranjar os itens de um vetor ou lista de modo que suas chaves estejam ordenadas de acordo com alguma regra.

■ Estrutura:

- ❑ um vetor `v` vai ser um `Item *v` ou `Item v[max]`

```
typedef int TipoChave;  
typedef struct {  
    TipoChave chave;  
    /* outros componentes */  
} Item;
```



Critérios de Classificação

- Localização dos dados
- Estabilidade
- Adaptabilidade
- Uso da memória
- Movimentação dos dados
- Estratégia de ordenação: Comparação de Chaves x Outros

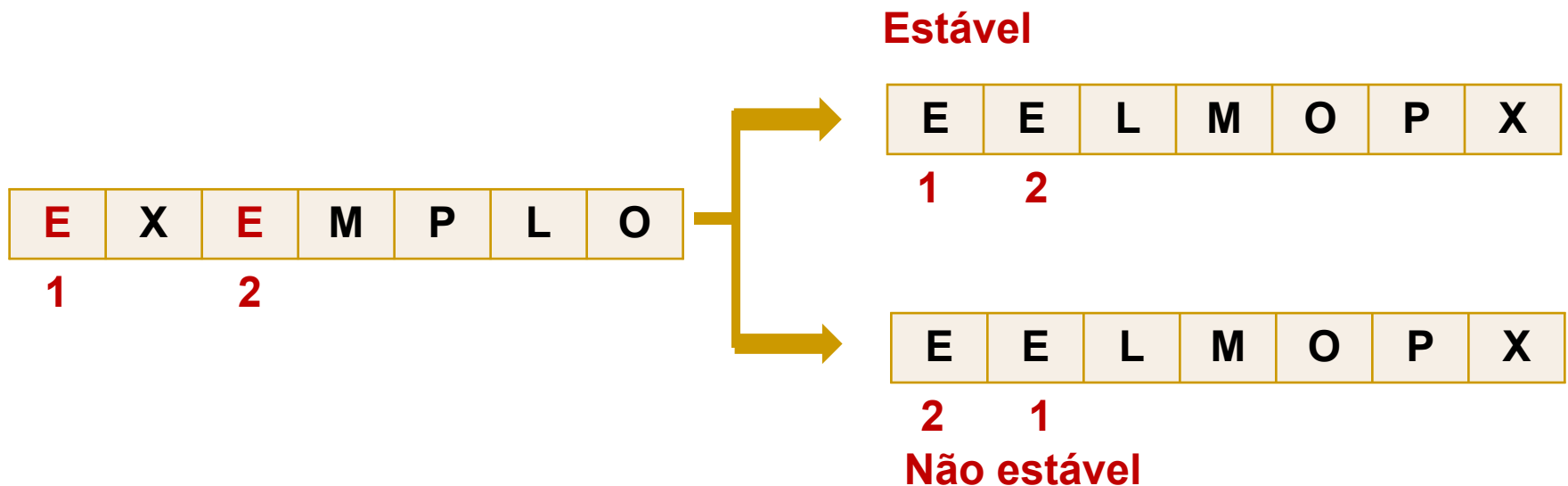
Critérios de Classificação

- Localização dos dados:
 - ❑ Ordenação interna: todas as chaves estão na memória principal.
 - ❑ Ordenação externa: chaves na memória principal e na memória secundária.

Critérios de Classificação

■ Estabilidade:

- ❑ Relacionado com o manutenção da ordem relativa entre chaves de mesmo valor.
- ❑ Método é estável se a ordem relativa dos registros com a mesma chave não se altera após a ordenação.



Critérios de Classificação

- Adaptabilidade:
 - Sequência de operações executadas conforme a entrada.
 - Não adaptável: operações executadas independente da entrada.
- Uso da memória:
 - ***In place***: transforma os dados de entrada utilizando apenas um espaço extra de tamanho constante.
- Movimentação dos dados:
 - Direta: registro todo é acessado e deve ser movido
 - Indireta: apenas as chaves são acessadas e ponteiros são rearranjados e não o registro todo.
- **Comparação de Chaves x Outros**

Critérios de Avaliação

- Seja n o número de registros em um vetor, considera-se duas medidas de complexidade:
 - Número de **comparações** $C(n)$ entre as chaves;
 - Número de **trocas ou movimentações** $M(n)$ de itens;

```
#define Troca(A, B) {Item c = A; A = B; B = c; }
```

```
void Ordena(Item *v, int n) {  
    int i, j;  
    for (i = 0; i < n-1; i++) {  
        for (j = n-1; j > i; j--) {  
            if (v[j-1].chave > v[j].chave) /* comparações */  
                Troca(v[j-1], v[j]); /* trocas */  
        }  
    }  
}
```

Métodos de Ordenação

- Métodos simples:
 - ❑ Bolha
 - ❑ Seleção
 - ❑ Inserção
- Métodos eficientes:
 - ❑ Quicksort
 - ❑ Mergesort
 - ❑ Heapsort
- Métodos lineares:
 - ❑ Bucketsort
 - ❑ Radixsort

Estrutura de Dados

Ordenação: Métodos Simples

Professores: Anisio Lacerda
Wagner Meira Jr.

MÉTODO DA BOLHA EXPLICAÇÃO

Método Bolha

- Ideia:
 - Passa no arquivo e troca elementos adjacentes que estão fora de ordem, até os registros ficarem ordenados.

Método Bolha

■ Algoritmo

- ❑ Supondo movimentação da esquerda para direita no vetor;
- ❑ Cada elemento é comparado com o seguinte. Se a ordem estiver invertida, a posição dos dois é trocada;
- ❑ Quando o maior elemento do vetor for encontrado, ele será trocado até ocupar a última posição;
- ❑ Na segunda passada, o segundo maior será movido para a penúltima posição do vetor.
- ❑ E assim por diante durante $n-1$ passadas...

MÉTODO DA BOLHA

CÓDIGO

Método Bolha

```
void Bolha (Item *v, int n)
{
    int i, j;

    for (i = 0; i < n-1; i++)
        for (j = 1; j < n-i; j++)
            if (v[j].chave < v[j-1].chave)
                Troca(v[j-1], v[j]);
}
```

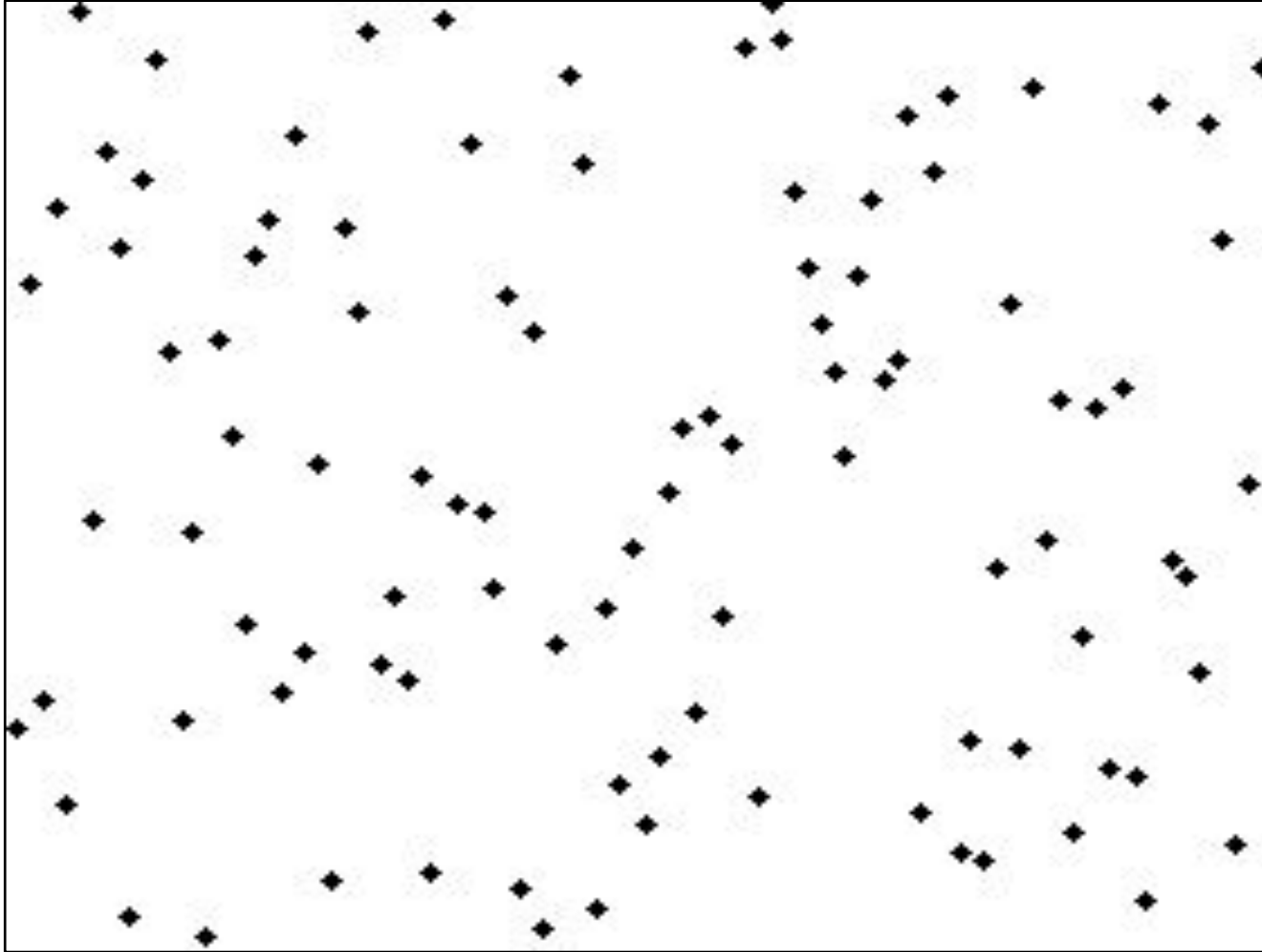
Método Bolha

```
void Bolha (Item *v, int n)
{
    int i, j;
    for(i = 0; i < n-1; i++)
        for(j = 1; j < n-i; j++)
            if (v[j].chave < v[j-1].chave)
                Troca(v[j-1], v[j]);
}
```

Para cada elemento do vetor

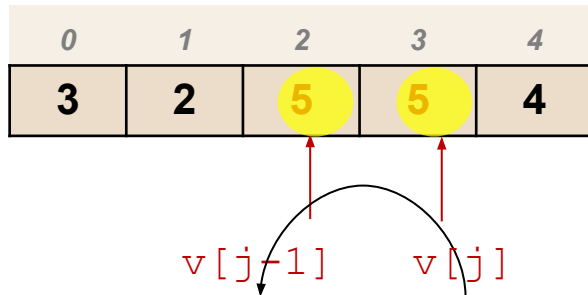
**Compara os
elementos 2 a 2,
e move o maior
para a direita**

Método Bolha

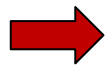


MÉTODO DA BOLHA ANÁLISE

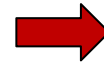
Método é estável?



menor?



NÃO!



Não troca



É ESTÁVEL!

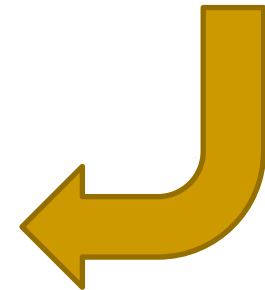
Método Bolha: Complexidade

■ Comparações – C(n):

```
void Bolha (Item *v, int n) {  
    int i, j;  
  
    for(i = 0; i < n-1; i++)  
        for(j = 1; j < n-i; j++)  
            if (v[j].chave < v[j-1].chave)  
                Troca(v[j-1], v[j]);  
}
```

i	comparações
0	n-1
1	n-2
2	n-3
...	...
n-2	1

$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$



Método Bolha: Complexidade

Movimentações – $M(n)$

```
void Bolha (Item *v, int n) {  
    int i, j;  
  
    for(i = 0; i < n-1; i++)  
        for(j = 1; j < n-i; j++)  
            if (v[j].chave < v[j-1].chave)  
                Troca(v[j-1], v[j]); // 3 movimentações  
}
```

Método Bolha: Complexidade

Movimentações – $M(n)$

```
void Bolha (Item *v, int n) {  
    int i, j;  
  
    for(i = 0; i < n-1; i++)  
        for(j = 1; j < n-i; j++)  
            if (v[j].chave < v[j-1].chave)  
                Troca(v[j-1], v[j]); // 3 movimentações  
}
```

□ Pior Caso:

□ Melhor Caso:

Método Bolha: Complexidade

Movimentações – $M(n)$

```
void Bolha (Item *v, int n) {  
    int i, j;  
  
    for(i = 0; i < n-1; i++)  
        for(j = 1; j < n-i; j++)  
            if (v[j].chave < v[j-1].chave)  
                Troca(v[j-1], v[j]); // 3 movimentações  
}
```

- Pior Caso: $M(n) = 3nC(n)$
 - Vetor inversamente Ordenado
- Melhor Caso: $M(n) = 0$
 - Vetor Ordenado

Análise de Complexidade

■ Comparações $C(n)$

$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

■ Movimentações $M(n)$

- ❑ Pior caso: $M(n) = 3 \times C(n) \rightarrow O(n^2)$
- ❑ Melhor caso: $M(n) = 0 \rightarrow O(1)$

Método Bolha

■ Vantagens

- ❑ Algoritmo simples
- ❑ Algoritmo estável

■ Desvantagens

- ❑ Não adaptável em termos de comparações
- ❑ Muitas trocas de itens

❑ Possível Melhoria

- ❑ Parar o algoritmo quando não forem efetuadas trocas em uma passagem (menos comparações)

```
void Bolha (Item *v, int n) {  
    int i, j, trocou;
```

```
    for(i = 0; i < n-1; i++){  
        trocou = 0;  
        for(j = 1; j < n-i; j++){  
            if (v[j].chave <  
v[j-1].chave){  
                Troca(v[j-1], v[j]);  
                trocou = 1;  
            }  
            if (!trocou) break;  
        }  
    }  
}
```


MÉTODO DE SELEÇÃO EXPLICAÇÃO

Método Seleção

- Seleção do n -ésimo menor (ou maior) elemento da lista
- Troca do n -ésimo menor (ou maior) elemento com o n -ésimo elemento da lista
- Uma única troca por vez é realizada

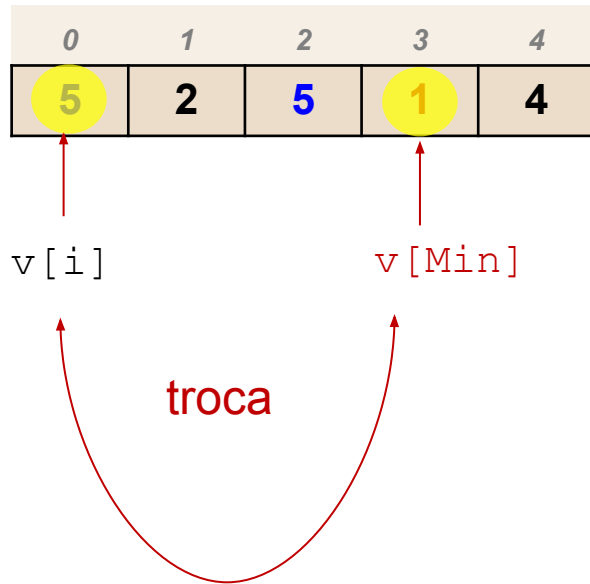
MÉTODO DE SELEÇÃO CÓDIGO

Método Seleção

```
void Selecao (Item *v, int n) {  
    int i, j, Min;  
  
    for (i = 0; i < n - 1; i++) {  
        Min = i;  
        for (j = i + 1 ; j < n; j++) {  
            if (v[j].chave < v[Min].chave)  
                Min = j;  
        }  
        Troca(v[i], v[Min]);  
    }  
}
```

MÉTODO DE SELEÇÃO ANÁLISE

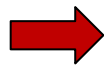
Método é estável?



Método é estável?

0	1	2	3	4
1	2	5	5	4

0	1	2	3	4
1	2	5	5	4



Inverteu a posição dos '5's



NÃO É ESTÁVEL!

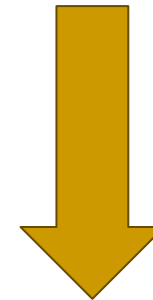
Método Seleção - Complexidade

■ Comparações – C(n):

```
void Selecao (Item *v, int n)
{
    int i, j, Min;

    for (i = 0; i < n - 1; i++)
    {
        Min = i;
        for (j = i + 1 ; j < n; j++)
        {
            if (v[j].chave < v[Min].chave)
                Min = j;
        }
        Troca(v[i], v[Min]);
    }
}
```

i	comparações
0	n-1
1	n-2
2	n-3
...	...
n-2	1




$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

Método Seleção - Complexidade

■ Movimentações – $M(n)$:

```
void Selecao (Item *v, int n)
{
    int i, j, Min;

    for (i = 0; i < n - 1; i++)
    {
        Min = i;
        for (j = i + 1 ; j < n; j++)
        {
            if (v[j].chave < v[Min].chave)
                Min = j;
        }
        Troca(v[i], v[Min]);
    }
}
```



→ $M(n) = 3 \times (n-1)$

Método Seleção: Complexidade

- Comparações – $C(n)$:

$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

- Movimentações – $M(n)$:

$$M(n) = 3(n-1) = O(n)$$

Método Seleção

- Vantagens:

- Custo linear no tamanho da entrada para o número de movimentos de registros – a ser utilizado quando há registros muito grandes;

- Desvantagens:

- Não adaptável (não importa se o arquivo está parcialmente ordenado);
- Algoritmo não é estável;

MÉTODO DE INSERÇÃO EXPLICAÇÃO

Método Inserção

- Algoritmo utilizado pelo jogador de cartas
 - ❑ As cartas são ordenadas da esquerda para direita uma a uma.
 - ❑ O jogador escolhe a segunda carta e verifica se ela deve ficar antes ou na posição que está.
 - ❑ Depois a terceira carta é classificada, deslocando-a até sua correta posição.
 - ❑ O jogador realiza esse procedimento até ordenar todas as cartas.

MÉTODO DE INSERÇÃO

CÓDIGO

Método Inserção

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;  
    for (i = 1; i < n; i++) {  
        aux = v[i];  
        j = i - 1;  
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {  
            v[j + 1] = v[j];  
            j--;  
        }  
        v[j + 1] = aux;  
    }  
}
```

Método Inserção

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;
```

```
    for (i = 1; i < n; i++) {
```

— Para cada elemento do vetor

```
        aux = v[i];
```

```
        j = i - 1;
```

```
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {
```

```
            v[j + 1] = v[j];
```

```
            j--;
```

```
        }
```

```
        v[j + 1] = aux;
```

```
    }
```

```
}
```

Acha a posição
do elemento

Método Inserção

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;
```

```
    for (i = 1; i < n; i++) {
```

— Para cada elemento do vetor

```
        aux = v[i];
```

```
        j = i - 1;
```

```
        while (j >= 0) && (aux.Chave < v[j].Chave) {
```

```
            v[j + 1] = v[j];
```

```
            j--;
```

```
        }
```

```
        v[j + 1] = aux;
```

```
    }
```

```
}
```

Não é o 1º.
elemento do
vetor

Elemento da
esquerda for
maior

Acha a posição
do elemento

Método Inserção

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;
```

```
    for (i = 1; i < n; i++) {
```

Para cada elemento do vetor

```
        aux = v[i];
```

```
        j = i - 1;
```

```
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {
```

```
            v[j + 1] = v[j];
```

```
            j--;
```

```
        }
```

```
        v[j + 1] = aux;
```

```
    }
```

```
}
```

Muda a
posição do
elemento
anterior

Acha a posição
do elemento

Método Inserção

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;
```

```
    for (i = 1; i < n; i++) {
```

— Para cada elemento do vetor

```
        aux = v[i];
```

```
        j = i - 1;
```

```
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {
```

```
            v[j + 1] = v[j];
```

```
            j--;
```

```
        }
```

```
        v[j + 1] = aux;
```

Acha a posição
do elemento

```
    }
```

```
}
```

Colocar o elemento
na posição correta

MÉTODO DE INSERÇÃO ANÁLISE

Método é estável?

0	1	2	3	4
3	5	2	5	4

0	1	2	3	4
2	3	5	5	4

Método é estável?

0	1	2	3	4
3	5	2	5	4

0	1	2	3	4
2	3	5	5	4

aux
5



Método Inserção: Exemplos

■ Melhor Caso:

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

■ Pior Caso:

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

Método Inserção - Número de Comparações

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;  
    for (i = 1; i < n; i++) {  
        aux = v[i];  
        j = i - 1;  
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {  
            v[j + 1] = v[j];  
            j--;  
        }  
        v[j + 1] = aux;  
    }  
}
```

Melhor caso: 1
Pior caso: i

Método Inserção: Complexidade

■ Comparações – $C(n)$:

□ Loop interno: i -ésima iteração, valor de C_i :

■ melhor caso: $C_i = 1$

■ pior caso: $C_i = i$

□ Loop externo: $\sum_{i=1}^{n-1} C_i$

□ Complexidade total:

■ Melhor caso (itens já estão ordenados)

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 = O(n)$$

■ Pior caso (itens em ordem reversa):

$$C(n) = \sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

Método Inserção - Número de Movimentações

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;  
    for (i = 1; i < n; i++) {  
        aux = v[i];  
        j = i - 1;  
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {  
            v[j + 1] = v[j];  
            j--;  
        }  
        v[j + 1] = aux;  
    }  
}
```

**Para cada iteração do
loop externo são $2 \cdot (n-1)$**

Método Inserção - Número de Movimentações

```
void Insercao(Item *v, int n) {  
    int i,j;  
    Item aux;  
    for (i = 1; i < n; i++) {  
        aux = v[i];  
        j = i - 1;  
        while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {  
            v[j + 1] = v[j];  
            j--;  
        }  
        v[j + 1] = aux;  
    }  
}
```

Melhor caso: 0
Pior caso: $i-1$

Para cada iteração do loop externo são $2 = 2(n-1)$

Método Inserção: Complexidade

- Movimentações - $M(n)$:
 - 2 movimentações no loop externo + 1 no loop interno
 - **Melhor caso:** $2(n-1)$ – nunca entra no loop interno ➤ $O(n)$
 - **Pior caso:** $2(n-1) + n(n-1)/2$ – sempre entra no loop interno: pior caso das comparações ➤ $O(n^2)$

Método Inserção: Complexidade

- Comparações - $C(n)$:
 - ❑ Melhor caso: $O(n)$
 - ❑ Pior caso: $O(n^2)$
- Movimentações - $M(n)$:
 - ❑ Melhor caso: $O(n)$
 - ❑ Pior caso: $O(n^2)$

Método Inserção – Melhoria

■ Uso de um sentinela

```
void Insercao(Item *v, Indice n){
    Indice i, j;
    Item aux;
    for (i = 2; i <= n; i++){
        aux = v[i];
        j = i - 1;
        v[0] = aux; /* sentinela */
        while (aux.Chave < v[j].Chave){
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = aux;
    }
}
```

- Primeira posição válida do vetor é 1
- Posição 0: guarda sentinela (valor sendo testado)
- Evita uma comparação de índice a cada iteração ao custo de uma eventual comparação de chaves

Método Inserção

■ Vantagens:

- ❑ É o método a ser utilizado quando o arquivo está “quase” ordenado.
- ❑ É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado, pois o custo é linear.
- ❑ O algoritmo de ordenação por inserção é **estável**.

■ Desvantagens:

- ❑ Alto custo de movimentação de elementos no vetor.

