Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

Ordenação – Algoritmos Simples

Prof. Mateus Ferreira Satler

Índice

1. Introdução

- Ordenação é o processo de rearranjo de um certo conjunto de objetos (elementos) de acordo com um critério (ordem) específico.
- O objetivo da ordenação é facilitar a localização dos membros de um conjunto de objetos.
- Assim sendo, é uma atividade fundamental e universalmente utilizada para a elaboração de algoritmos mais complexos.
  - Exemplos de casos em que os objetos estão ordenados podem ser encontrados em listas telefônicas, índices, dicionários, etc. e em quase todos os casos em que estejam colecionados objetos sujeitos à procura e alteração.

1. Introdução

- **Aplicações de Ordenação**:
  - **Busca**: a busca binária permite testar se um item está em um arquivo em um tempo **O (log n)** se as chaves do arquivo estiverem ordenadas. Busca é uma das mais importantes aplicações de ordenação.
  - **Par mais próximo**: dado um conjunto de **n** números, como encontrar o par de números que tem a menor diferença entre eles? Depois dos números estarem ordenados, o par mais próximo estará disposto próximo um do outro de forma ordenada.

1. Introdução

- **Aplicações de Ordenação:**
  - **Elementos duplicados:** existem elementos duplicados em um conjunto de **n** itens? O algoritmo mais eficiente consiste em ordená-los e então fazer uma busca para verificar os pares adjacentes.
  - **Frequência de distribuição:** dado um conjunto de **n** itens, qual o elemento que ocorre mais vezes? Se os itens estiverem ordenados, pode-se percorrer da esquerda para a direita e contá-los, uma vez que todos os elementos idênticos ficarão juntos durante a ordenação.

1. Introdução

- A ordenação é baseada em uma **chave.**
  - A chave de ordenação é o campo do item utilizado para comparação. Por exemplo:
    - Valor armazenado em um vetor de inteiros.
    - Campo nome de uma struct.
    - Etc.
  - É por meio dela que sabemos se um determinado elemento está a frente ou não de outros no conjunto.

1. Introdução

- Podemos usar qualquer tipo de chave.
  - Deve existir uma regra de ordenação bem-definida.
- Alguns tipos de ordenação:
  - **Numérica**
    - 1, 2, 3, 4, 5
  - **Lexicográfica (ordem alfabética)**
    - Ana, André, Bianca, Ricardo

1. Introdução

- Exemplo de chave:

```
typedef int TChave;

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

1. Introdução

- Independente do tipo, a ordenação pode ser:
  - **Crescente**
    - 1, 2, 3, 4, 5
    - Ana, André, Bianca, Ricardo
  - **Decrescente**
    - 5, 4, 3, 2, 1
    - Ricardo, Bianca, André, Ana

1. Introdução

- Os algoritmos de ordenação podem ser classificados como de:
  - **Ordenação interna**
    - O conjunto de dados a ser ordenado cabe todo na memória principal (RAM).
    - Qualquer elemento pode ser imediatamente acessado.
  - **Ordenação externa**
    - O conjunto de dados a ser ordenado não cabe na memória principal (estão armazenados em memória secundária, por exemplo, um arquivo).
    - Os elementos são acessados sequencialmente ou em grandes blocos.

1. Introdução

- Além disso, a ordenação pode ser **estável** ou **instável (não estável).**
  - Um método de ordenação é denominado **estável** se a ordem relativa dos elementos que exibam a mesma chave permanecer **inalterada** ao longo de todo o processo de ordenação.
  - Caso contrário, ele é denominado **instável**.
- Em geral, a estabilidade da ordenação é desejável, especialmente quando os elementos já estiverem ordenados em relação a uma ou mais chaves secundárias.

1. Introdução

- Exemplo de ordenação **estável** e **não estável**.

Original	
Adão	1
Beto	2
Bruno	4
João	2
José	4
Sonia	1
Thiago	4
Vanda	2

Não Estável	
Adão	1
Sonia	1
Vanda	2
João	2
Beto	2
Thiago	4
Bruno	4
José	4

Estável	
Adão	1
Sonia	1
Beto	2
João	2
Vanda	2
Bruno	4
José	4
Thiago	4

1. Introdução

- Os métodos de ordenação estudados podem ser divididos em:
  - **Básicos**
    - Fácil implementação.
    - Auxiliam o entendimento de algoritmos complexos.
  - **Sofisticados**
    - Em geral, melhor desempenho.

2. Algoritmo BubbleSort

- Também conhecido como ordenação por bolha.
- É um dos algoritmos de ordenação mais conhecidos que existem.
- Remete a ideia de bolhas flutuando em um tanque de água em direção ao topo até encontrarem o seu próprio nível (ordenação crescente).

2. Algoritmo BubbleSort

- **Funcionamento:**
  - Compara pares de valores adjacentes e os troca de lugar se estiverem na ordem errada.
    - Trabalha de forma a movimentar, uma posição por vez, o maior valor existente na porção não ordenada de um vetor para a sua respectiva posição no vetor ordenado.
  - Esse processo se repete até que mais nenhuma troca seja necessária.
    - Elementos já ordenados.

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	j-1
	9	10	j

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	j-1
	8	9	j
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	j-1
	7	1	j
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	1	j-1
	7	5	j
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	j-1
	6	1	j
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	1	j-1
	6	3	j
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	j-1
	5	1	j
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	1	j-1
	5	6	j
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	j-1
	4	1	j
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	1	j-1
	4	7	j
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	j-1
	3	1	j
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	1	j-1
	3	2	j
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	8	j-1
	2	1	j
	3	2	
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	
	1	1	j-1
	2	8	j
	3	2	
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	4	j-1
	1	1	j
	2	8	
	3	2	
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
i	0	1	j-1
	1	4	j
	2	8	
	3	2	
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	7	
	5	6	
	6	3	
	7	5	
	8	9	j-1
	9	10	j

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	7	
	5	6	
	6	3	
	7	5	j-1
	8	9	j
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	7	
	5	6	
	6	3	j-1
	7	5	j
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	7	
	5	6	j-1
	6	3	j
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	7	
	5	3	j-1
	6	6	j
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	7	j-1
	5	3	j
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	
	4	**3**	j-1
	5	**7**	j
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	
	3	2	j-1
	4	3	j
	5	7	
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	8	j-1
	3	2	j
	4	3	
	5	7	
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	
	2	2	j-1
	3	8	j
	4	3	
	5	7	
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	4	j-1
	2	2	j
	3	8	
	4	3	
	5	7	
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
i	1	2	j-1
	2	4	j
	3	8	
	4	3	
	5	7	
	6	6	
	7	5	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	
	5	7	
	6	6	
	7	5	
	8	9	j-1
	9	10	j

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	
	5	7	
	6	6	
	7	5	j-1
	8	9	j
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	
	5	7	
	6	6	j-1
	7	5	j
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	
	5	7	
	6	**5**	j-1
	7	**6**	j
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	
	5	7	j-1
	6	5	j
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	
	5	**5**	j-1
	6	**7**	j
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	
	4	3	j-1
	5	5	j
	6	7	
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	8	j-1
	4	3	j
	5	5	
	6	7	
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	
	3	3	j-1
	4	8	j
	5	5	
	6	7	
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	4	j-1
	3	3	j
	4	8	
	5	5	
	6	7	
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
i	2	3	j-1
	3	4	j
	4	8	
	5	5	
	6	7	
	7	6	
	8	9	
	9	10	

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
	2	3	
i	3	4	
	4	8	
	5	5	
	6	7	
	7	6	
	8	9	j-1
	9	10	j

2. Algoritmo BubbleSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	1	
	1	2	
	2	3	
	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
i	8	9	j-1
	9	10	j

2. Algoritmo BubbleSort

- **Melhoria:**
  - Se não acontecer nenhuma troca, é possível parar o algoritmo.

```
void bubblesort (Item *v, int n) {
  int i, j, troca;
  Item aux;
  for (i = 0; i < n - 1; i++) {
    troca = 0;
    for (j = n - 1; j > i; j--)
      if (v[j].chave < v[j-1].chave) {
        aux = v[j];
        v[j] = v[j-1];
        v[j-1] = aux;
        troca = 1; }
    if (troca == 0)
      break;
  }
}
```

2. Algoritmo BubbleSort

- **Análise**
  - Comparações – $C(n)$:
    - $C(n) = \sum_{i=0}^{n-2}(n-i-1) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1$
    - $C(n) = n(n-1) - \frac{(0+n-2)(n-1)}{2} - (n-1)$
    - $C(n) = \frac{n^2-n}{2} = O(n^2)$
  - Movimentações – $M(n)$:
    - $M(n) = 3C(n) = O(n^2)$

2. Algoritmo BubbleSort

- **Vantagens:**
  - Algoritmo estável.
  - Simples e de fácil entendimento e implementação.
  - Está entre os métodos de ordenação mais difundidos existentes.

- **Desvantagens:**
  - Não é um algoritmo eficiente.
    - Sua eficiência diminui drasticamente a medida que o número de elementos no vetor aumenta.
  - O fato de o arquivo já estar ordenado não ajuda reduzir o número de comparações (o custo continua quadrático), porém o número de movimentação cai a zero.

3. Algoritmo SelectionSort

- Também conhecido como ordenação por seleção.
- É outro algoritmo de ordenação bastante simples.
  - A cada passo ele seleciona o melhor elemento para ocupar aquela posição do vetor.
    - Maior ou menor, dependendo do tipo de ordenação.
  - Na prática, possui um desempenho quase sempre superior quando comparado com o BubbleSort.

3. Algoritmo SelectionSort

- **Funcionamento:**
  - A cada passo, procura o menor valor do vetor e o coloca na primeira posição do vetor.
    - Divide o vetor em duas partes: a parte ordenada, a esquerda do elemento analisado, e a parte que ainda não foi ordenada, a direita do elemento.
  - Descarta-se a primeira posição do vetor e repete-se o processo para a segunda posição.
  - Isso é feito para todas as posições do vetor.

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
min, i	0	4	
	1	8	j
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
min, i	0	4	
	1	8	
	2	2	j
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
min	2	2	j
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
min	2	2	
	3	7	j
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
min	2	2	
	3	7	
	4	6	j
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	j
	6	5	
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	
	6	5	j
	7	1	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	j
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
min	7	1	j
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
min	7	1	
	8	9	j
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	4	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
min	7	1	
	8	9	
	9	10	j

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
i	0	1	
	1	8	
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
min	7	4	
	8	9	
	9	10	j

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
min, i	1	8	
	2	2	j
	3	7	
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	j
	3	7	
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	j
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	
	4	6	j
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	j
	6	5	
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	
	6	5	j
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	4	j
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	j
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	8	
min	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	j

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
i	1	2	
min	2	8	
	3	7	
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	j

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
	1	2	
min, i	2	8	
	3	7	j
	4	6	
	5	3	
	6	5	
	7	4	
	8	9	
	9	10	

3. Algoritmo SelectionSort

- Algoritmo:

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    aux = v[min];
    v[min] = v[i];
    v[i] = aux;
  }
}
```

n = 10	Pos.	Chave	
	0	1	
	1	2	
	2	3	
	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
min, i	8	9	
	9	10	j

3. Algoritmo SelectionSort

- **Melhoria:**

```
void selectionsort (Item *v, int n) {
  int i, j, min;
  Item aux;
  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;
    if (min != i) {
    aux = v[min];
    v[min] = v[i];
    v[i] = aux; }
  }
}
```

3. Algoritmo SelectionSort

- **Análise**
  - Comparações – $C(n)$:
    - $C(n) = \sum_{i=0}^{n-2}(n-i-1) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1$
    - $C(n) = n(n-1) - \frac{(0+n-2)(n-1)}{2} - (n-1)$
    - $C(n) = \frac{n^2-n}{2} = O(n^2)$
  - Movimentações – $M(n)$:
    - $M(n) = 3(n-1) = O(n)$

3. Algoritmo SelectionSort

- **Vantagens:**
  - Algoritmo estável.
  - Custo linear no tamanho da entrada para o número de movimentos de registros.
  - É o algoritmo a ser utilizado para arquivos com registros muito grandes.
  - É muito interessante para arquivos pequenos.

- **Desvantagens:**
  - O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.
  - Sua eficiência diminui drasticamente a medida que o número de elementos no vetor aumenta.

4. Algoritmo InsertionSort

- Também conhecido como ordenação por inserção.
- Similar a ordenação de cartas de baralho com as mãos:
  - Pegue uma carta de cada vez e a insira em seu devido lugar, sempre deixando as cartas da mão em ordem.

4. Algoritmo InsertionSort

- **Funcionamento:**
  - O algoritmo percorre o vetor e para cada posição **X** verifica se o seu valor está na posição correta.
  - Isso é feito andando para o começo do vetor a partir da posição **X** e movimentando uma posição para frente os valores que são maiores do que o valor da posição **X**.
  - Desse modo, teremos uma posição livre para inserir o valor da posição **X** em seu devido lugar.

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave
	0	4
	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 8
	0	4	j-1
i	1	8	j
	2	2	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave
	0	4
i	1	8
	2	2
	3	7
	4	6
	5	3
	6	5
	7	1
	8	9
	9	10

aux = 8

j

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	4	aux = 2
	1	8	j-1
i	2	2	j
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	4	aux = 2
	1	8	j-1
i	2	8	j
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 2
	0	4	j-1
	1	8	j
i	2	8	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	4	j-1
	1	4	j
i	2	8	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

aux = 2

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 2
	0	4	j
	1	4	
i	2	8	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	j
	1	4	
i	2	8	
	3	7	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

aux = 2

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	aux = 7
	1	4	
	2	8	j-1
i	3	7	j
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	aux = 7
	1	4	
	2	8	j-1
i	3	8	j
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	aux = 7
	1	4	j-1
	2	8	j
i	3	8	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 7
	0	2	
	1	4	
	2	7	j
i	3	8	
	4	6	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	aux = 6
	1	4	
	2	7	
	3	8	j-1
i	4	6	j
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	aux = 6
	1	4	
	2	7	
	3	8	j-1
i	4	8	j
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 6
	0	2	
	1	4	
	2	7	j-1
	3	8	j
i	4	8	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	aux = 6
	1	4	
	2	7	j-1
	3	7	j
i	4	8	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 6
	0	2	
	1	4	j-1
	2	7	j
	3	7	
i	4	8	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	
	0	2	
	1	4	
	2	6	j
	3	7	
i	4	8	
	5	3	
	6	5	
	7	1	
	8	9	
	9	10	

aux = 6

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 3
	0	2	
	1	4	
	2	6	
	3	7	
	4	8	j-1
i	5	3	j
	6	5	
	7	1	
	8	9	
	9	10	

4. Algoritmo InsertionSort

- Algoritmo:

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

n = 10	Pos.	Chave	aux = 10
	0	1	
	1	2	
	2	3	
	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
	8	9	
i	9	10	j

4. Algoritmo InsertionSort

- **Análise**
  - Comparações – $C(n)$:
    - No anel mais interno, na i-ésima iteração, o valor de C_i é:
      - Melhor caso: $C_i(n) = 1$
      - Pior caso: $C_i(n) = i$
    - Assumindo que todas as permutações de **n** são igualmente prováveis, temos:
      - Melhor caso: $C(n) = (1 + 1 + \cdots + 1) = n - 1$
      - Pior caso: $C(n) = (1 + 2 + \cdots + n - 1) = \frac{n^2 - n}{2} = O(n^2)$

4. Algoritmo InsertionSort

- **Análise**
  - Movimentações – $M(n)$:
    - No anel mais interno, na i-ésima iteração, o valor de M_i é:
      - Melhor caso: $M_i(n) = 1$
      - Pior caso: $M_i(n) = i$
    - Assumindo que todas as permutações de **n** são igualmente prováveis, temos:
      - Melhor caso: $M(n) = (1 + 1 + \cdots + 1) = n - 1$
      - Pior caso: $M(n) = (1 + 2 + \cdots + n - 1) = \frac{n^2 - n}{2} = O(n^2)$

4. Algoritmo InsertionSort

- **Vantagens:**
  - Algoritmo Estável.
  - Fácil implementação.
  - Na prática, é mais eficiente que a maioria dos algoritmos de ordem quadrática.
    - Como o SelectionSort e o BubbleSort.

- **Desvantagens:**
  - Sua eficiência diminui drasticamente a medida que o número de elementos no vetor aumenta.

5. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf
- Horowitz, E. & Sahni, S.; Fundamentos de Estruturas de Dados, Editora Campus, 1984.
- Wirth, N.; Algoritmos e Estruturas de Dados, Prentice/Hall do Brasil, 1989.
- Material de aula do Prof. José Augusto Baranauskas, da USP: https://dcm.ffclrp.usp.br/~augusto/teaching.htm
- Material de aula do Prof. Rafael C. S. Schouery, da Unicamp: https://www.ic.unicamp.br/~rafael/cursos/2s2019/mc202/index.html