Algoritmos e Estruturas de Dados II

2º Período Engenharia da Computação

Prof. Edwaldo Soares Rodrigues
Email: edwaldo.rodrigues@uemg.br

UNIVERSIDADE
DO ESTADO DE MINAS GERAIS | UEMG
UNIDADE DIVINÓPOLIS

Métodos de Ordenação

Ordenação de Arranjos

- Como sabemos, buscas binárias em arranjos são mais eficientes que buscas sequenciais

- Porém, estas buscas só podem ser feitas em arranjos ordenados

Ordenação de Arranjos

- É comum em programação a necessidade de se ordenar um arranjo

- Um motivo razoável para isto pode ser inclusive ordenação dos dados para facilitar sua visualização

Ordenação de Arranjos

- Imagine um catálogo telefônico onde os nomes das pessoas não estão em ordem

Ordenação de Arranjos – Conceitos de ordenação

- As estratégias para se ordenar um arranjo são diversas
- Neste curso, estudaremos algumas destas estratégias
- O processo de ordenação pode ser para colocar os itens em ordem crescente ou decrescente

Ordenação de Arranjos – Conceitos de ordenação

- Como no caso da busca binária, os algoritmos são estendidos em situações práticas para ordenar structs(registros) por suas chaves

- Porém, para fins didáticos, os algoritmos de ordenação serão apresentados para ordenação de arranjos de tipos de dados fundamentais

Ordenação de Arranjos – Ordenação estável

- Uma ordenação estável é aquela que mantém a ordem relativa dos elementos antes da ordenação

- Ou seja, se dois elementos são iguais, o elemento que aparece antes no arranjo desordenado deve ainda aparecer antes no arranjo ordenado

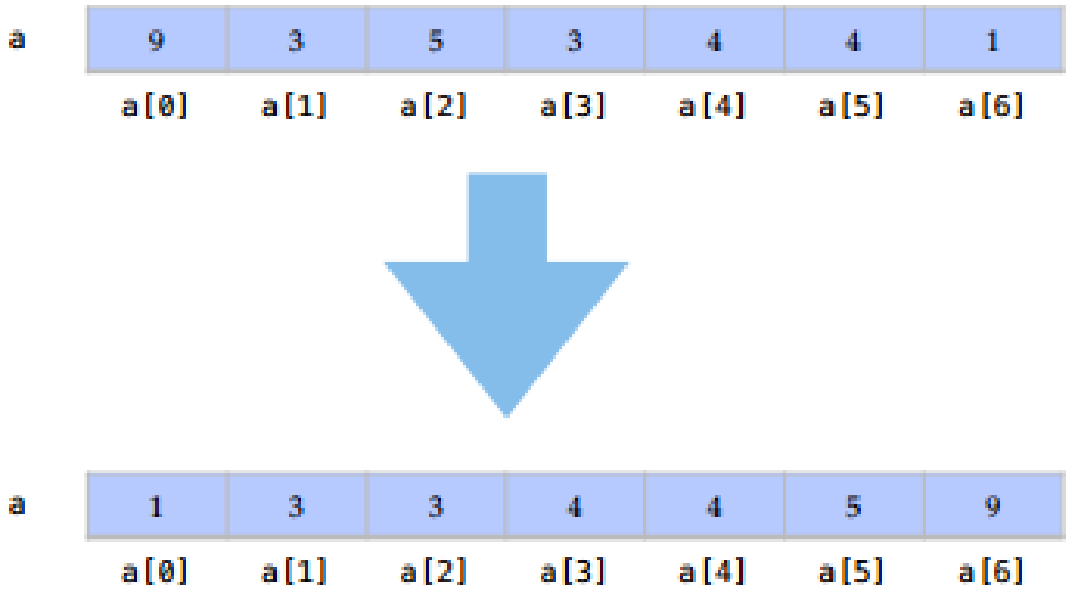
Ordenação de Arranjos – Ordenação estável

- Suponha um arranjo com os seguintes elementos:

a	9	3	5	3	4	4	1
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

Ordenação de Arranjos – Ordenação estável

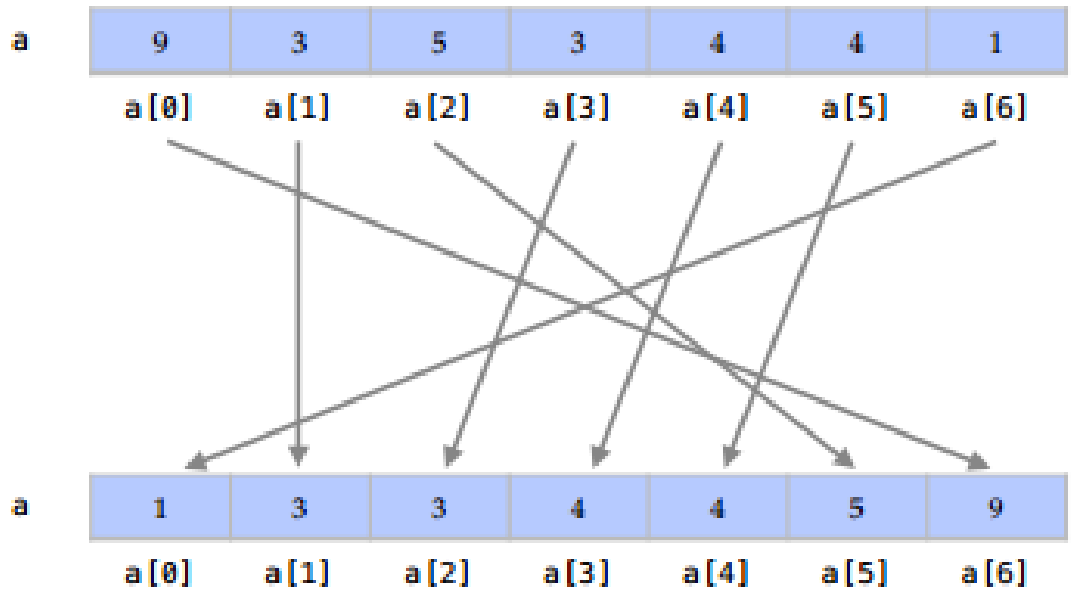
- Suponha um arranjo com os seguintes elementos:



- Este é o mesmo arranjo após a aplicação de um algoritmo de ordenação

Ordenação de Arranjos – Ordenação estável

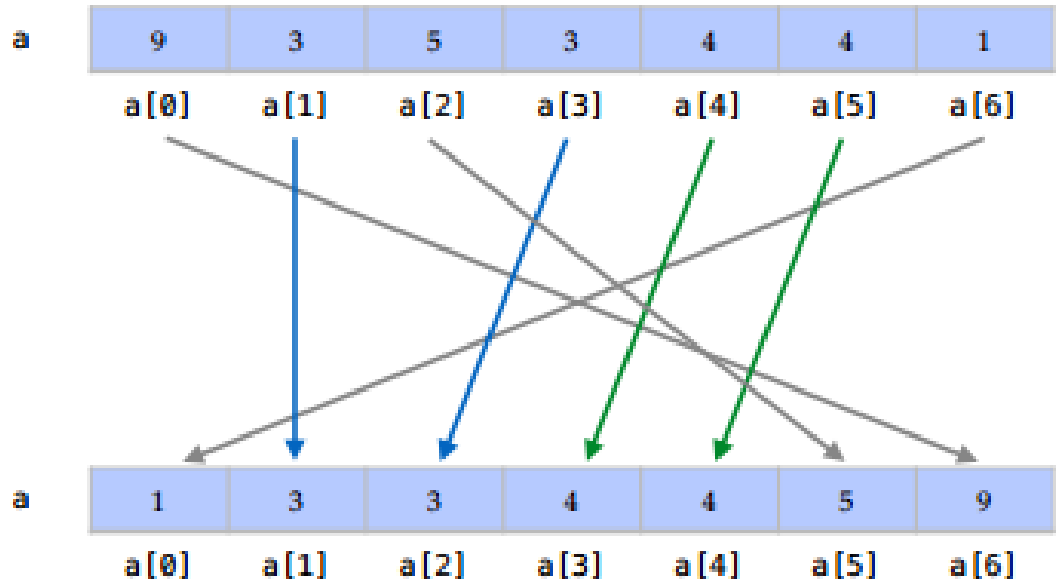
- Se o resultado final ocorreu da seguinte maneira...



- Nosso algoritmo de ordenação teve comportamento estável

Ordenação de Arranjos – Ordenação estável

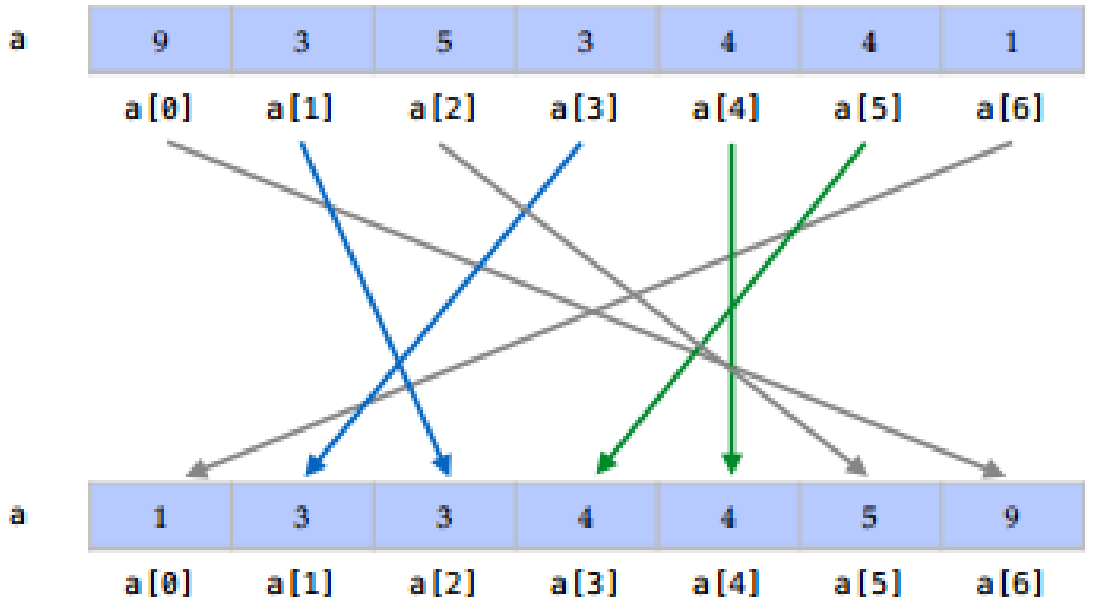
- Note a precedência dos elementos 3 e 4



- Perceba como mantiveram sua ordem relativa

Ordenação de Arranjos – Ordenação estável

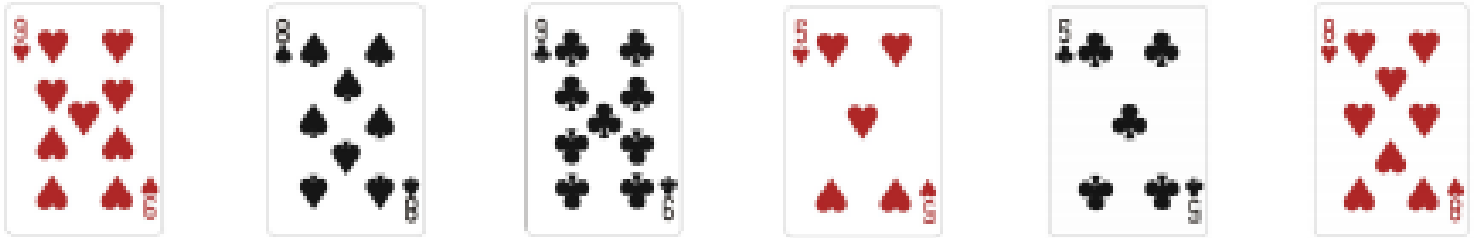
- Já este, poderia ser o resultado de um algoritmo instável



- Pois, apesar do resultado ser o mesmo, eles não se preocupam em manter a ordem relativa dos elementos

Ordenação de Arranjos – Ordenação estável

- Mas então, qual a desvantagem de algoritmos instáveis?

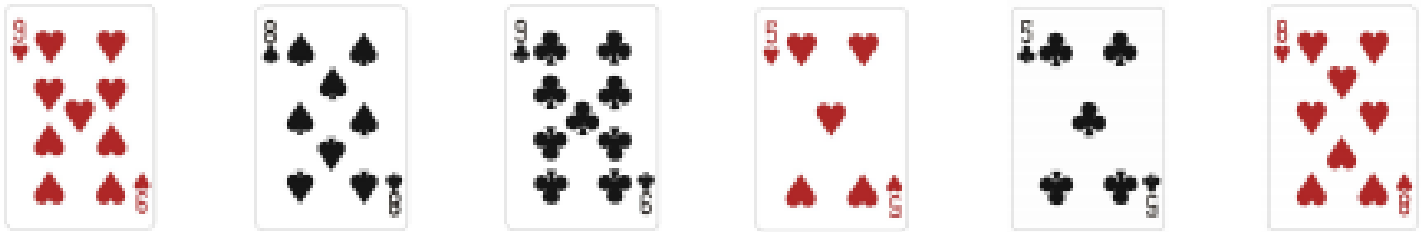


- Imagine que queremos ordenar um arranjo com as cartas acima

Ordenação de Arranjos – Ordenação estável

- Estas cartas poderiam ser facilmente representadas pela seguinte estrutura:

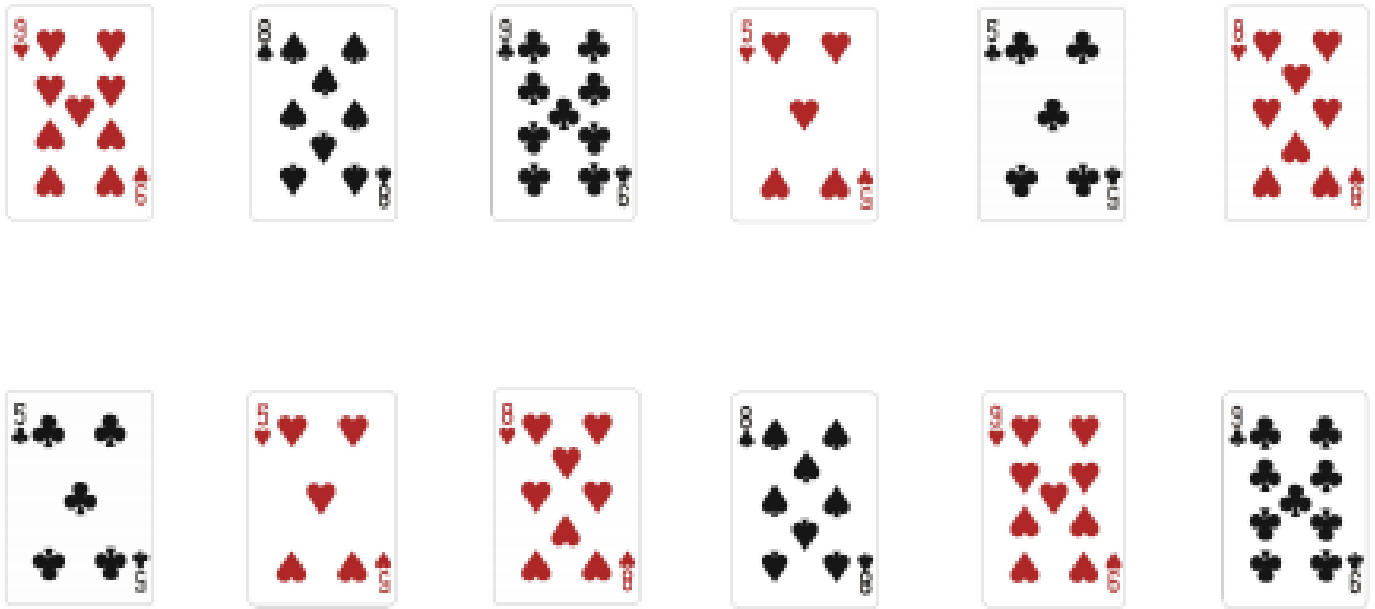
```
struct Carta{
    int valor;
    int naipe;
}
```



- Nesta estrutura, valor seria um número de 1 a 13 e naipe seria um número de 1 a 4

Ordenação de Arranjos – Ordenação estável

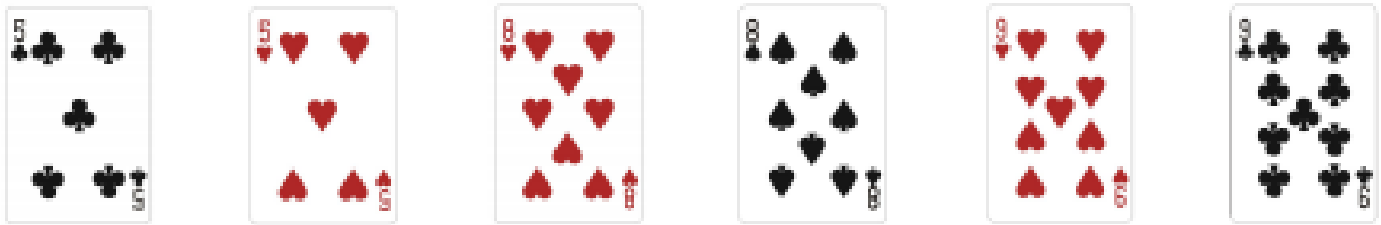
- Utilizando uma ordenação qualquer, podemos ter um resultado como o abaixo:



- As cartas estão ordenadas agora pelos números mas a ordem dos naipes é arbitrária

Ordenação de Arranjos – Ordenação estável

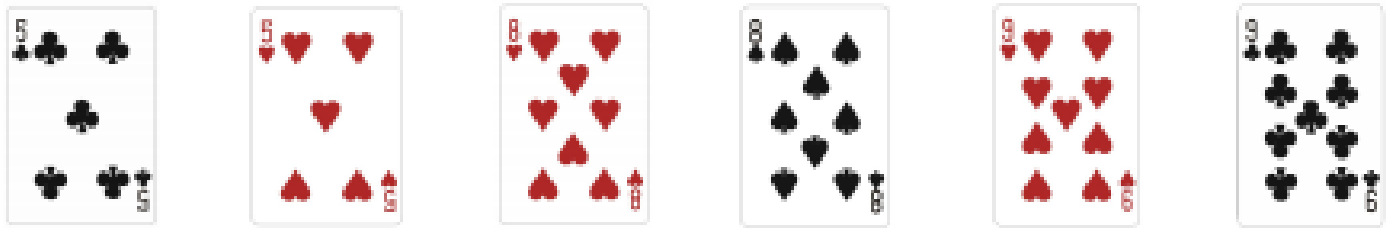
- Suponha que agora queremos que as cartas estejam ordenadas de acordo com seu naipe



- Para isto, precisamos de uma segunda ordenação, das cartas

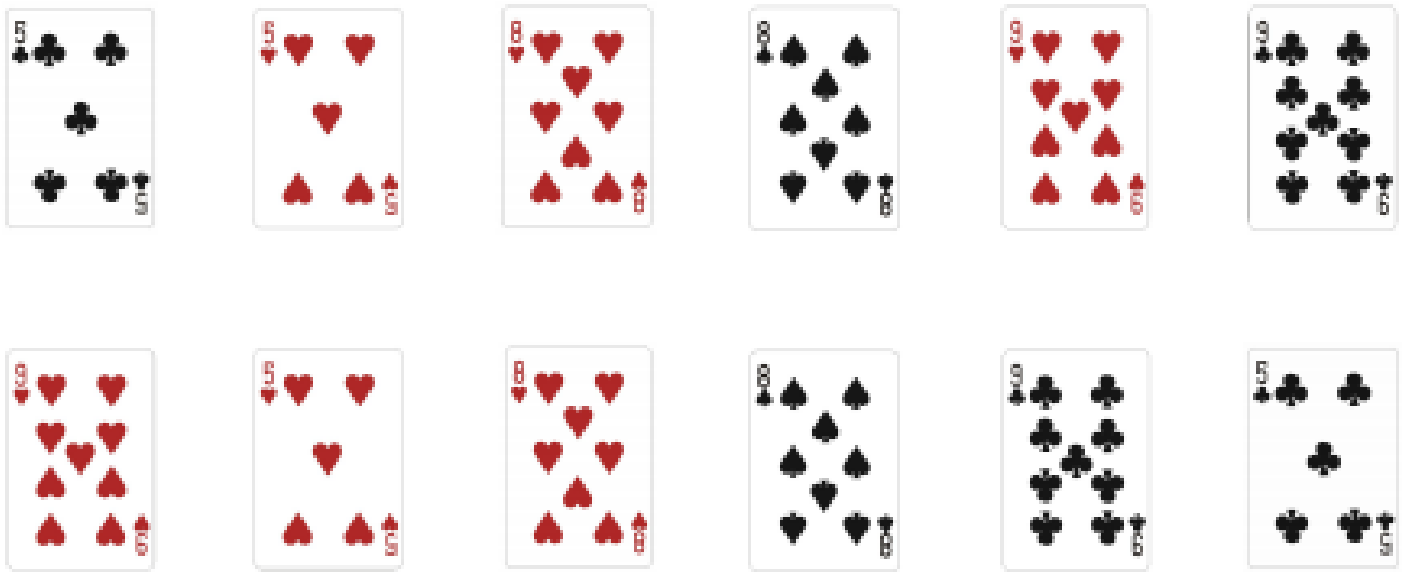
Ordenação de Arranjos – Ordenação estável

- Para esta segunda ordenação, podemos utilizar um algoritmo estável ou instável



Ordenação de Arranjos – Ordenação estável

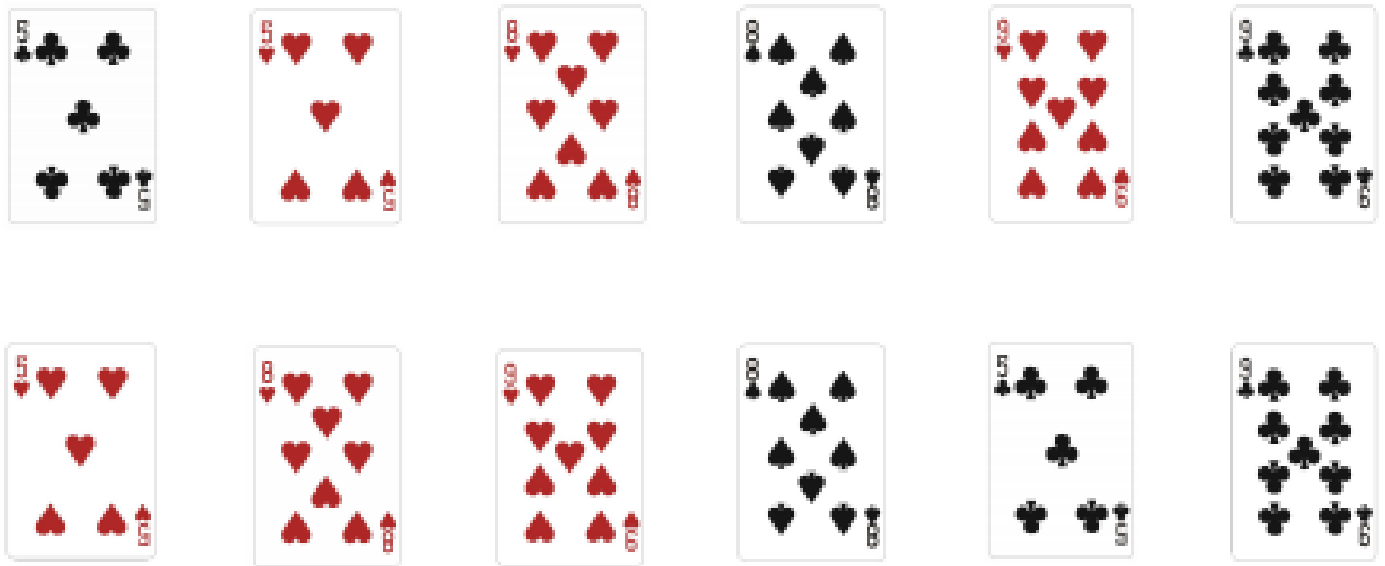
- Veja o que acontece quando ordenamos as cartas com um algoritmo instável



- Perdemos a ordem de valor das cartas ao ordenar pelos naipes. Claramente, isto não é o que queríamos

Ordenação de Arranjos – Ordenação estável

- Veja o que aconteceria se houvéssemos ordenado as cartas com um algoritmo estável



- Agora temos as cartas ordenadas por seus naipes mas também mantivemos a ordenação anterior por seus valores

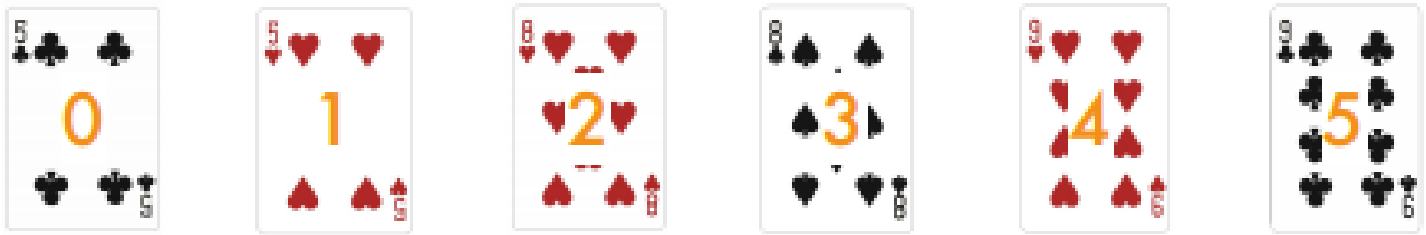
Ordenação de Arranjos – Ordenação estável

- Alguns métodos mais eficientes não são estáveis;

- Porém, podemos forçar um método não estável a se comportar de forma estável;

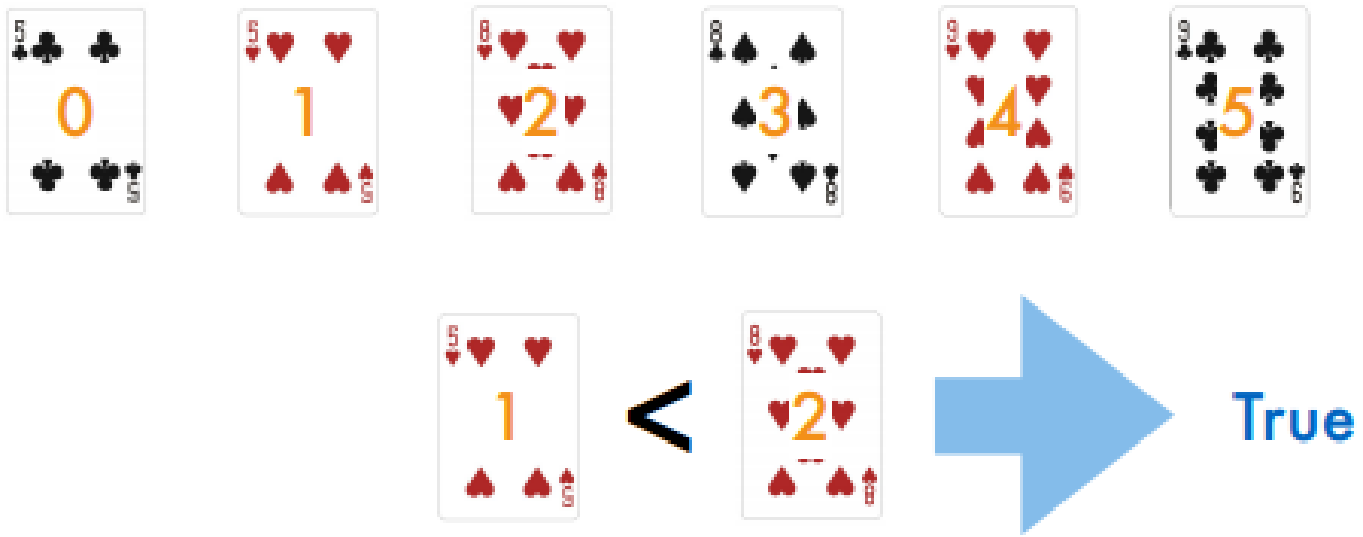
Ordenação de Arranjos – Ordenação estável

- Para forçar a estabilidade, adicionamos um índice a cada elemento do arranjo



Ordenação de Arranjos – Ordenação estável

- Para forçar a estabilidade, adicionamos um índice a cada elemento do arranjo



- Agora, podemos utilizar o índice como fator de desempate caso os elementos tenham chave igual (ou mesmo naipe em nosso exemplo)

Ordenação de Arranjos – Ordenação estável

- Apesar de possível, forçar um método não estável a se tornar estável diminui a eficiência dos algoritmos e faz com que o algoritmo gaste memória extra para armazenar todos os índices

- Como há um índice para cada elemento, este custo extra de memória é *O(n)*

Ordenação de Arranjos – Ordenação estável – Complexidade de tempo

- Outro fator a se considerar é a complexidade de tempo dos algoritmos de ordenação
- A medida mais relevante de tempo é o número de comparações *C(n)* feitas por um algoritmo para ordenar o vetor
- Uma medida secundária é o número *M(n)* de movimentações, ou operações de atribuição

Ordenação de Arranjos – Ordenação estável – Complexidade de tempo

- Em geral os métodos de ordenação simples:
  - Requerem $O(n^2)$ comparações
  - São apropriados para arranjos pequenos
- Já os métodos eficientes:
  - Requerem O(n log n) comparações
  - Adequados para arranjos grandes

Ordenação de Arranjos – Ordenação estável – Complexidade de Memória

- Mais um fator relevante é a memória extra que estes algoritmos utilizam para conseguir ordenar o arranjo

- Os métodos de ordenação que não precisam de memória extra são chamados de métodos **in place** (ou *in situ*)

Ordenação pelo método da bolha - BubbleSort

Ordenação pelo método da bolha

- Os elementos vão “borbulhando” a cada iteração do método até a posição correta para ordenação da lista;

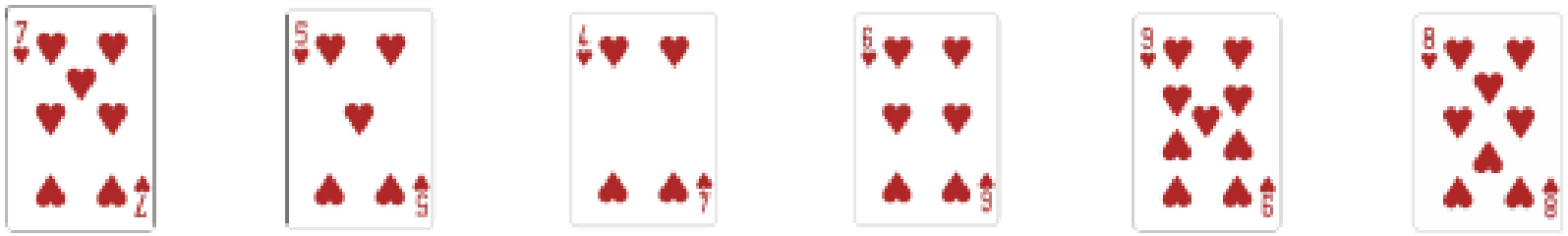
- Como os elementos são trocados (borbulhados) frequentemente, há um alto custo com troca de elementos;

Ordenação pelo método da bolha

- Técnica básica;
- Comparam-se dois elementos e trocam-se suas posições se o segundo elemento é menor do que o primeiro;
- São feitas várias passagens pelo vetor;
- Em cada passagem, comparam-se dois elementos adjacentes;
- Se estes elementos estiverem fora de ordem, eles são trocados;

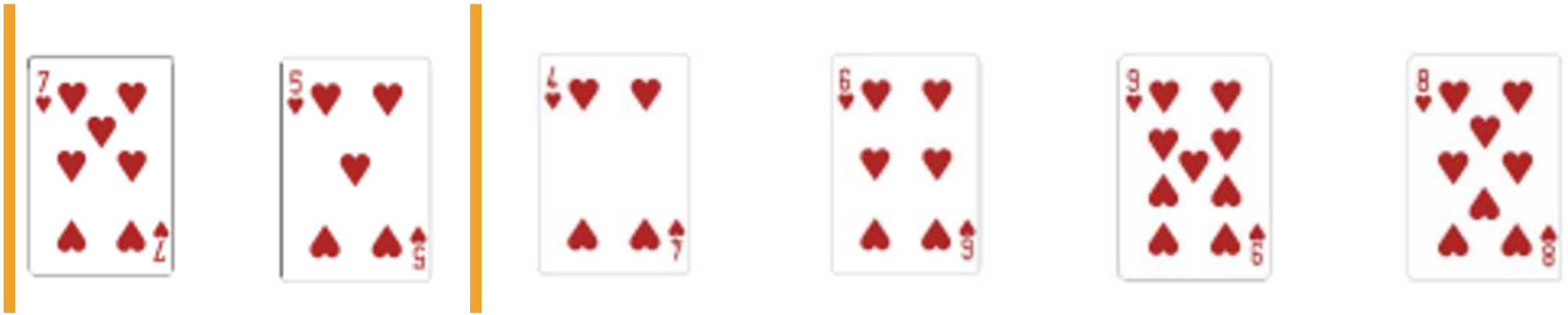
Ordenação pelo método da bolha

- Considere um arranjo com os seguintes elementos:



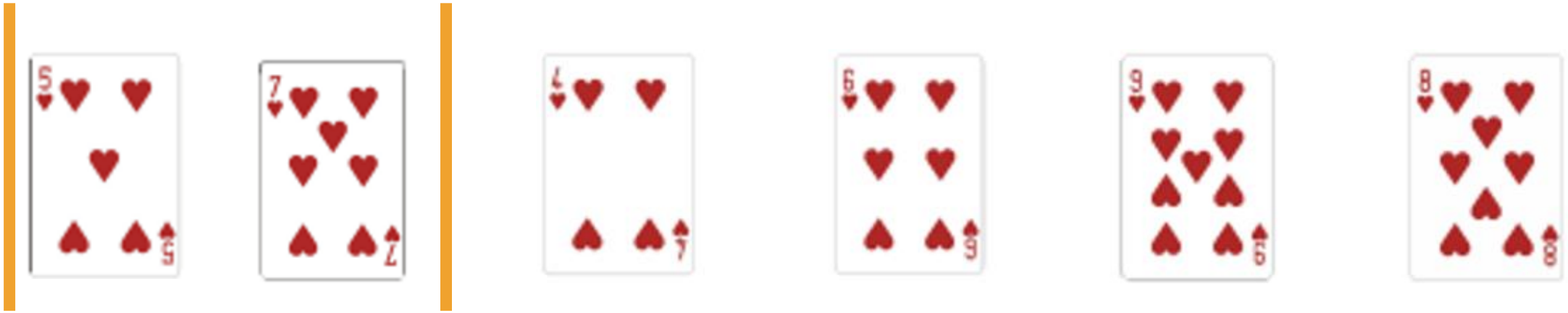
Ordenação pelo método da bolha

- Comparamos o primeiro elemento do vetor com o segundo elemento;



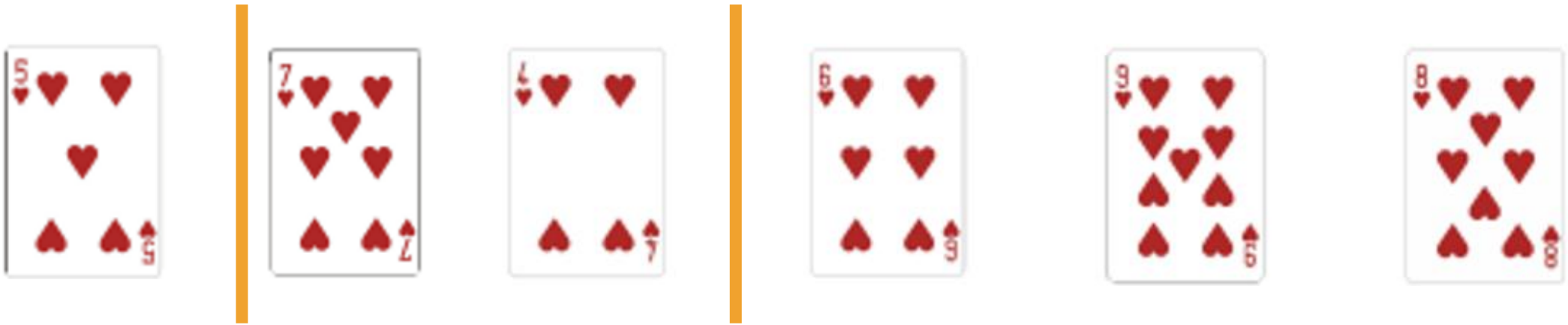
Ordenação pelo método da bolha

- Como o segundo elemento é menor, efetua-se a troca;



Ordenação pelo método da bolha

- Comparamos o segundo elemento do vetor com o terceiro elemento;

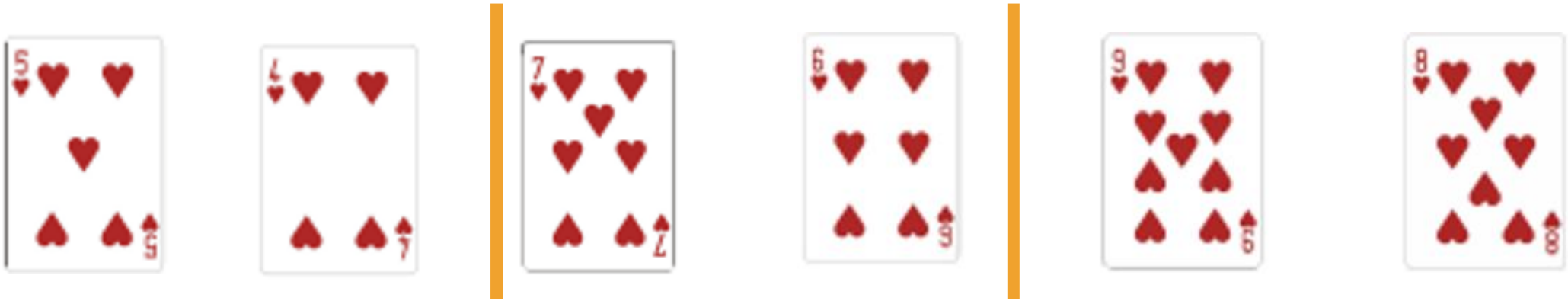


Ordenação pelo método da bolha

- Como o terceiro elemento é menor, efetua-se a troca;

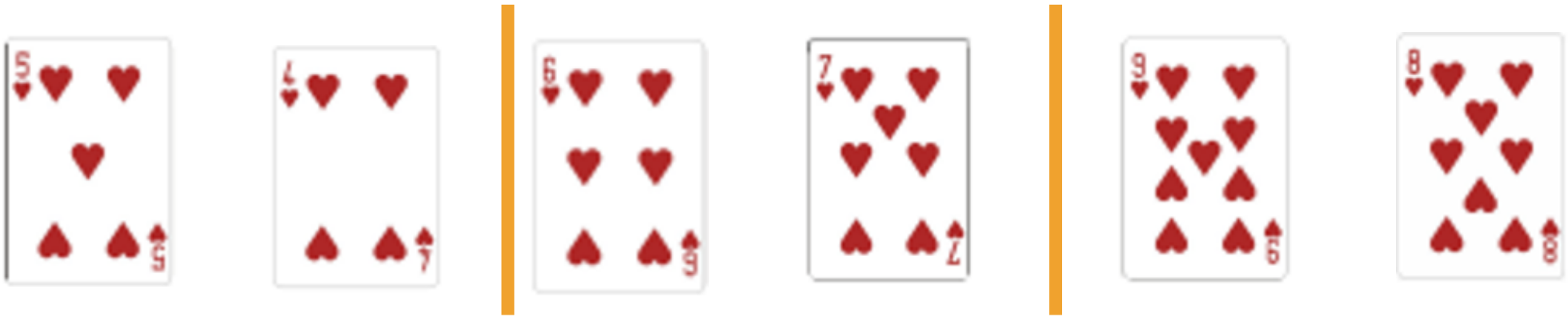
Ordenação pelo método da bolha

- Comparamos o terceiro elemento do vetor com o quarto elemento;



Ordenação pelo método da bolha

- Como o quarto elemento é menor, efetua-se a troca;



Ordenação pelo método da bolha

- Comparamos o quarto elemento do vetor com o quinto elemento;

Ordenação pelo método da bolha

- Como o quinto elemento não é menor, então não efetua-se a troca;

Ordenação pelo método da bolha

- Repetimos esse processo até chegarmos na comparação entre o penúltimo elemento do vetor com o último elemento;

Ordenação pelo método da bolha

- Como o último elemento é menor, então efetua-se a troca;

- Após essa iteração, podemos concluir que o último elemento é o maior elemento do vetor;

- O processo se repete até que todos os elementos estejam ordenados;

Ordenação pelo método da bolha

```
void bolha(int a[], int n){
    int i, j; // indices
    int aux; //elemento
    for(i = 0; i < n-1; i++){
        //percorremos o vetor n vezes
        for(j = 1; j < n-1; j++){
            //verificamos a cada iteracao qual o menor
            //valor entre um elemento e seu antecessor
            if(a[j] < a[j-1]){
                //se o sucessor for menor do que o
                //elemento anterior, efetua-se a troca
                aux = a[j];
                a[j] = a[j-1];
                a[j-1] = aux;
            }
        }
    }
}
```

Ordenação pelo método da bolha

- Vantagens:
  - Simplicidade do algoritmo;
  - Estável;
  - *In situ*;

- Desvantagens:
  - Lentidão;
  - Não aproveita uma ordenação prévia do vetor (não adaptável);

- Indicado em:
  - Vetores muito pequenos;
  - Demonstrações didáticas;

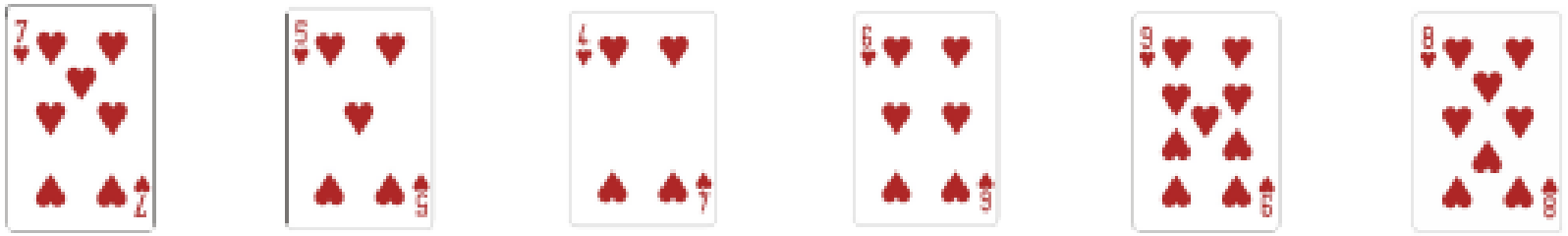
Ordenação por Seleção - SelectionSort

Ordenação por Seleção

- Este é um dos algoritmos mais simples de ordenação
- É um algoritmo onde a cada passo de repetição:
  - Se escolhe o menor elemento do arranjo
  - Troca-o com o elemento da primeira posição
  - Repete este procedimento para os outros n-1 elementos

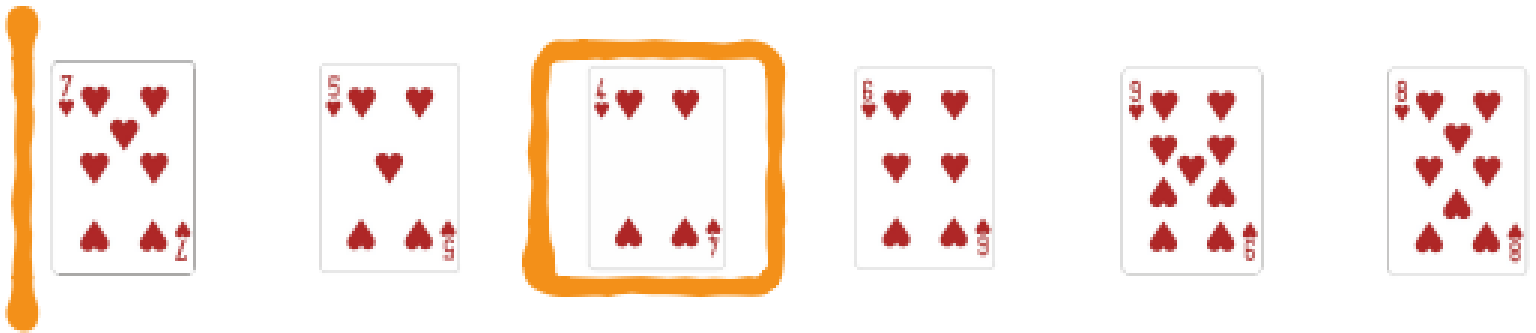
Ordenação por Seleção

- Considere um arranjo com os seguintes elementos:



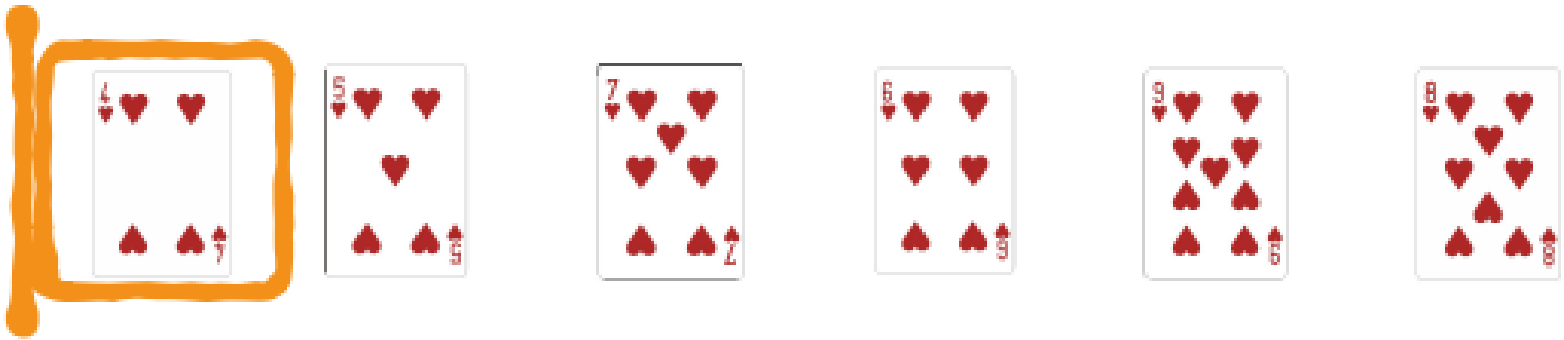
Ordenação por Seleção

- A partir do primeiro elemento, procuramos o menor elemento do arranjo



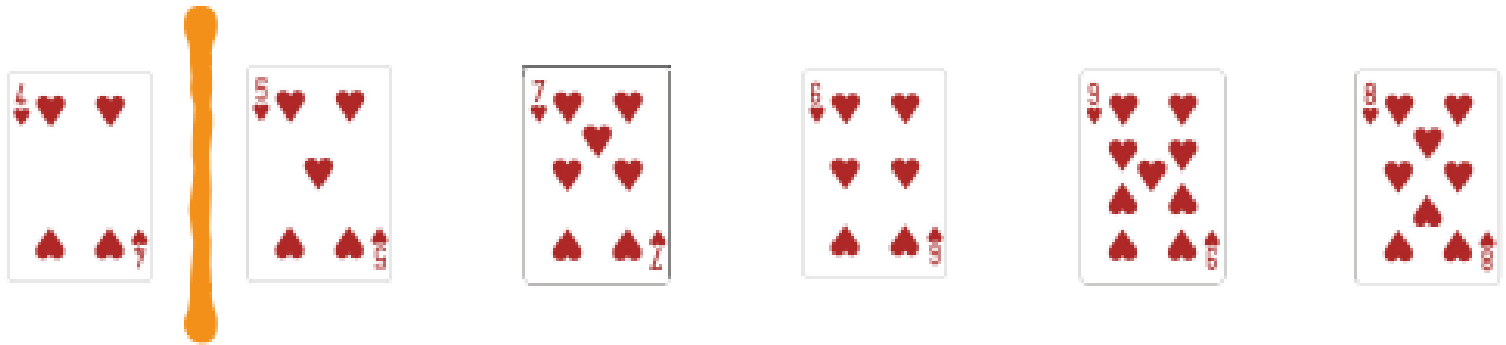
Ordenação por Seleção

- Este elemento é trocado com o elemento da primeira posição



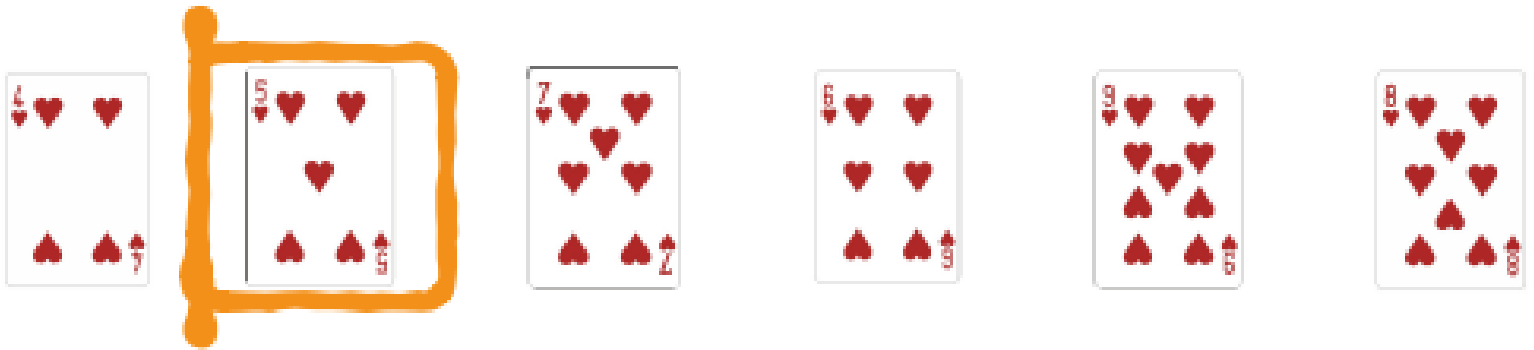
Ordenação por Seleção

- Sabemos agora que o arranjo está ordenado até o primeiro elemento



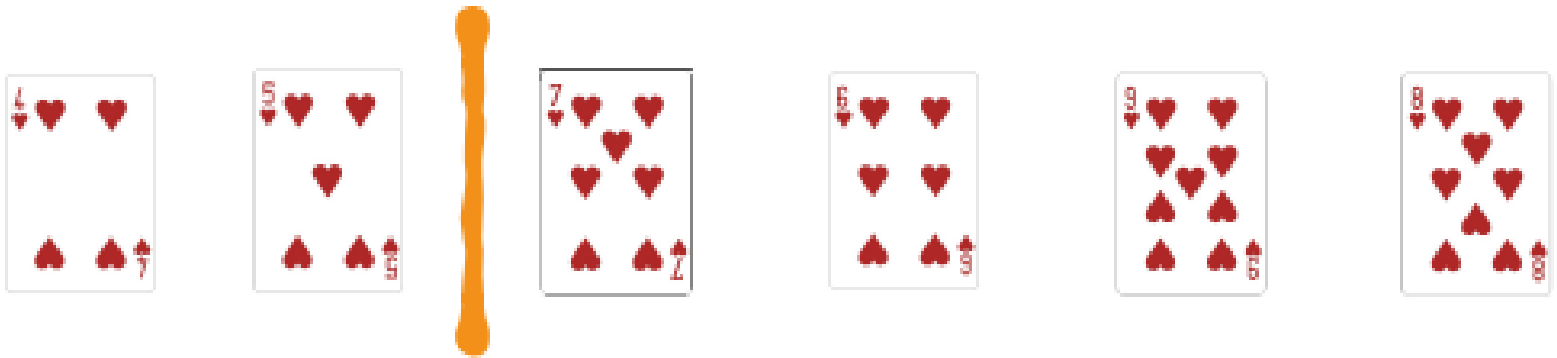
Ordenação por Seleção

- A partir do segundo elemento, procuramos o menor elemento do arranjo



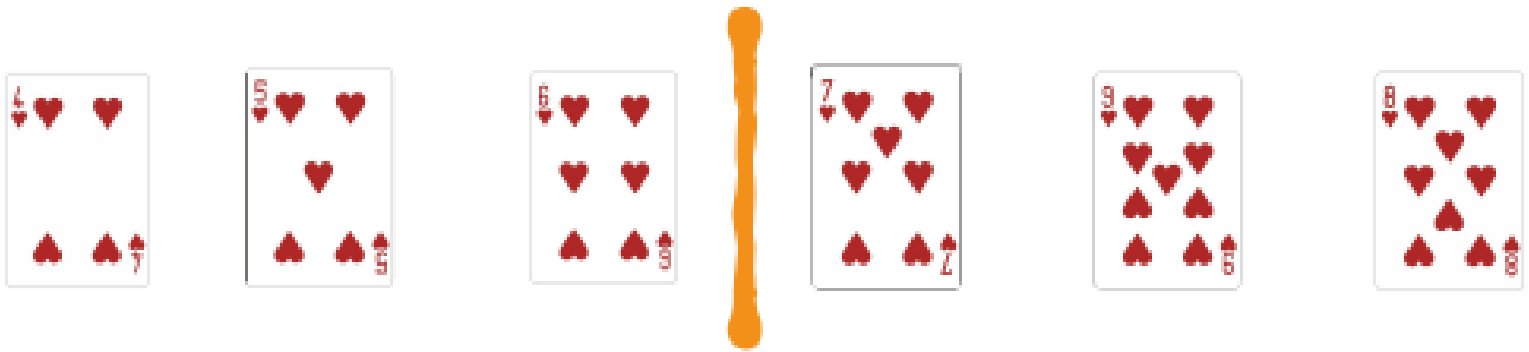
Ordenação por Seleção

- Este é colocado na segunda posição e sabemos que os dois primeiros elementos do arranjo já estão ordenados



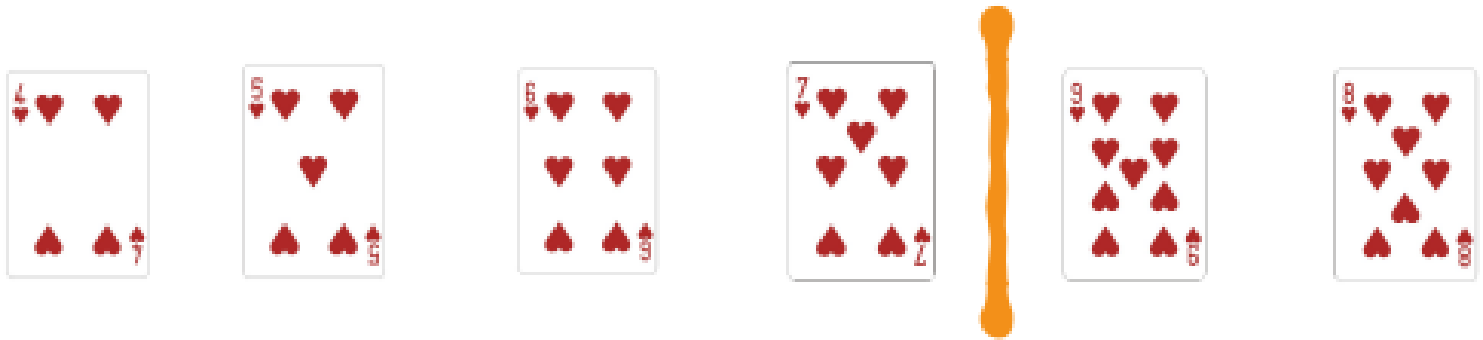
Ordenação por Seleção

- A partir do terceiro elemento, procuramos o menor elemento e o colocamos na terceira posição



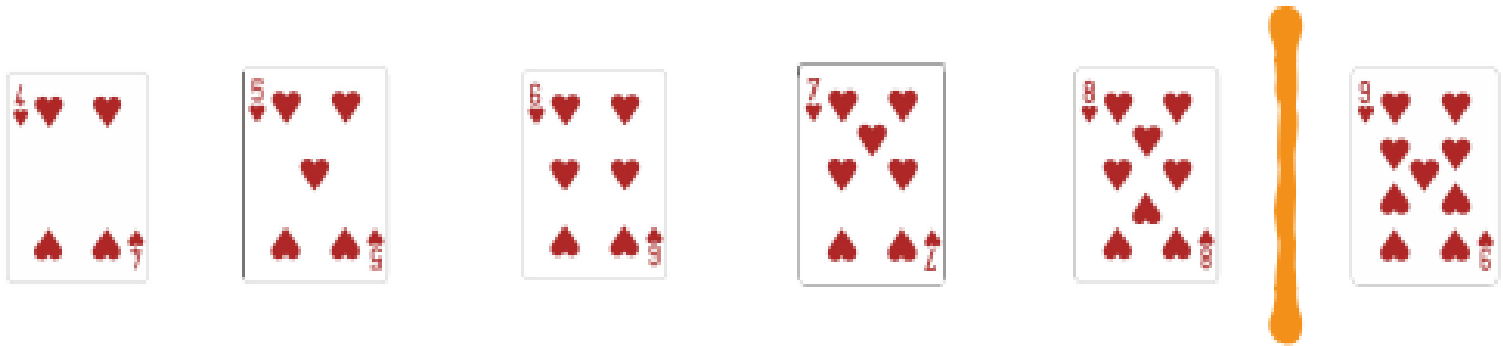
Ordenação por Seleção

- A partir do quarto elemento, procuramos o menor elemento e o colocamos na quarta posição



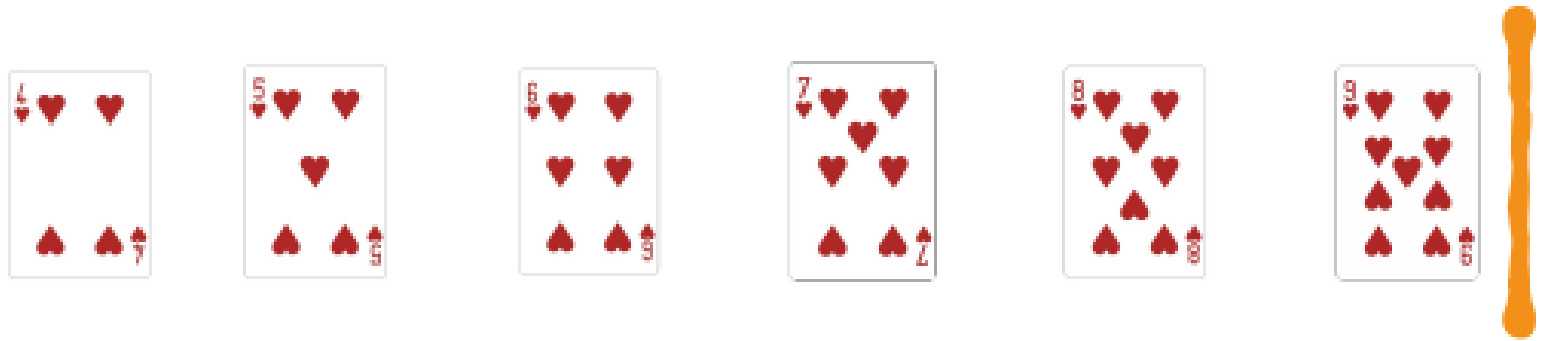
Ordenação por Seleção

- A partir do quinto elemento, procuramos o menor elemento e o colocamos na quinta posição



Ordenação por Seleção

- Como há apenas um elemento no arranjo não ordenado, já sabemos que ele está em sua posição correta então podemos encerrar o método



Ordenação por Seleção

Arranjo	7	5	4	6	9	8
Passo 1	4	5	7	6	9	8
Passo 2	4	5	7	6	9	8
Passo 3	4	5	6	7	9	8
Passo 4	4	5	6	7	9	8
Passo 5	4	5	6	7	8	9
Arranjo	4	5	6	7	8	9

Troca | Desordenado | Ordenado

Ordenação por Seleção

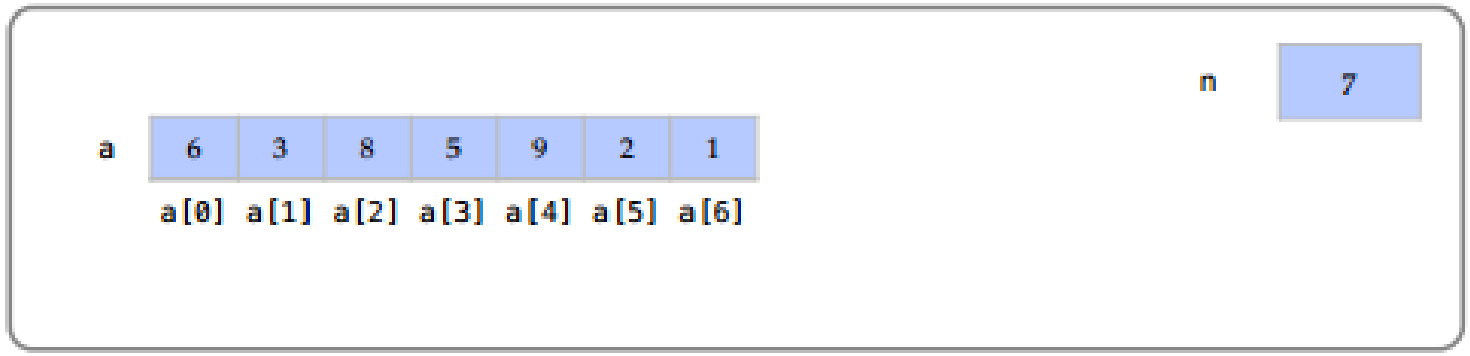
- Ordenação por seleção para um arranjo de inteiros

```
void selecao(int a[], int n){
    int i, j, min; // indices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```

Ordenação por Seleção

- A função ordena o arranjo a, que contém n elementos

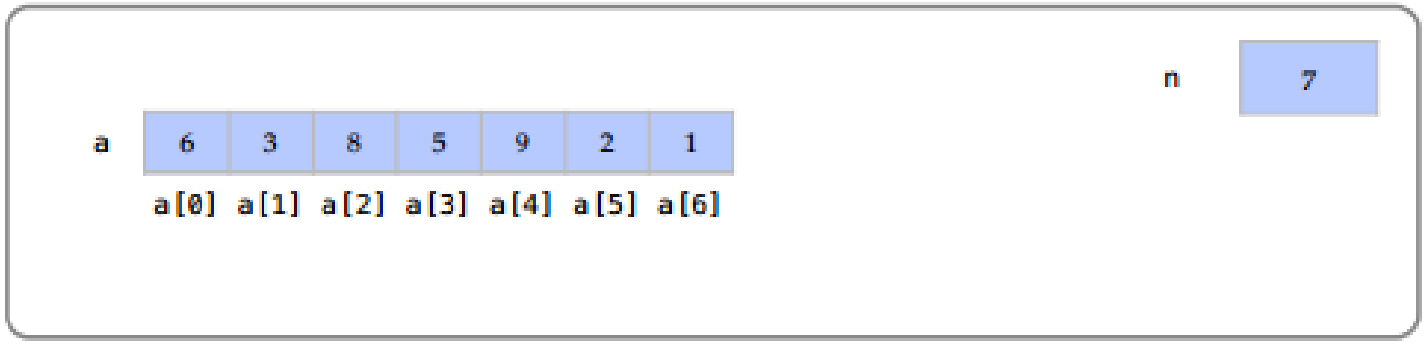
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Como sabemos que arranjos são passados por referência em C++, não precisamos retornar nada desta função

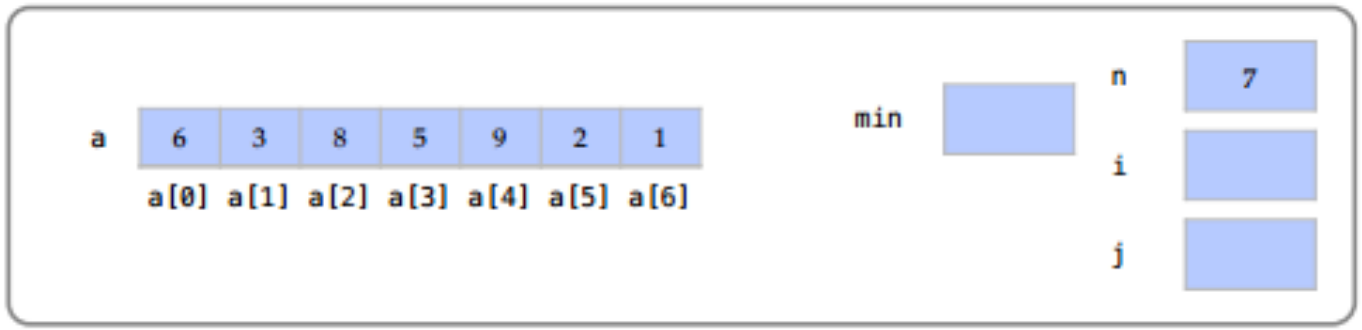
```
void selecao(int a[], int n){
    int i, j, min; // indices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Os índices i, j e min são utilizados para percorrermos o arranjo em um for aninhado e procurar seu menor elemento a cada passo

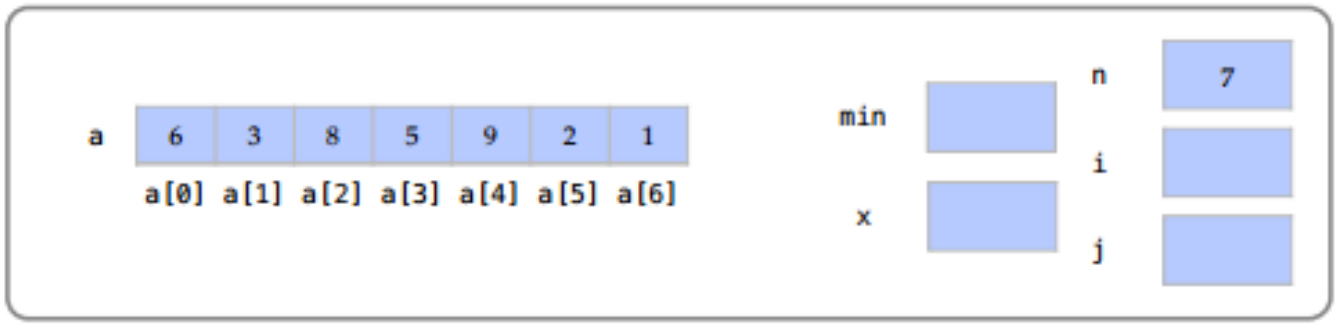
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- A variável x deve ser do mesmo tipo dos elementos do arranjo, pois será uma variável temporária para fazer as trocas entre elementos

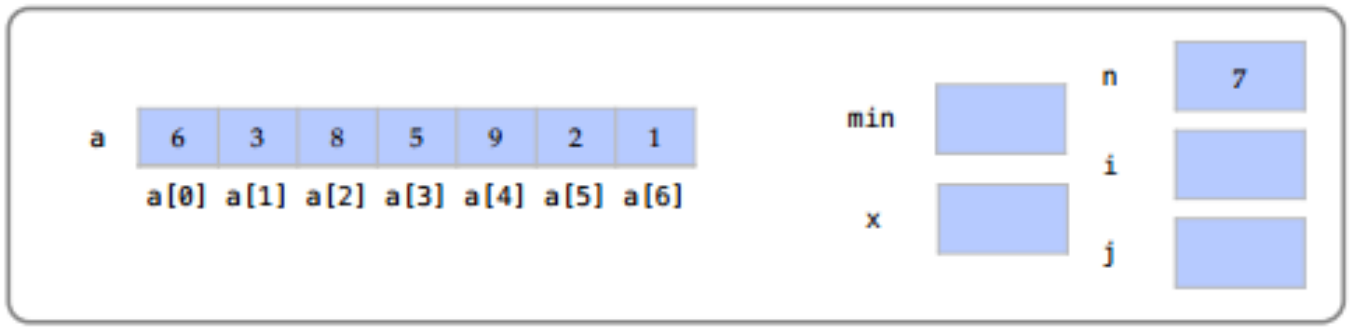
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Entramos em um for que, para cada posição do arranjo, colocará em seu lugar o elemento mínimo entre os elementos posteriores

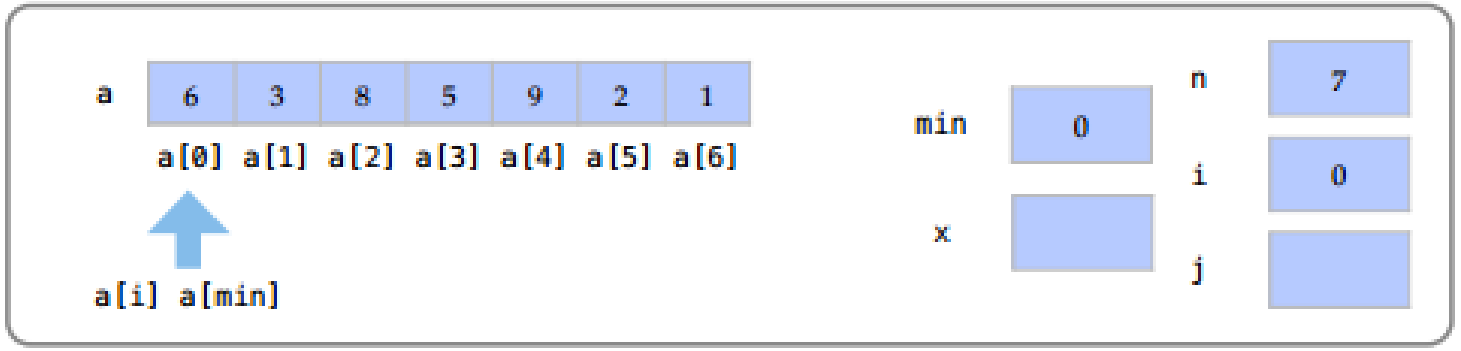
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Por exemplo, quando i é igual a 0 (zero), supomos que o menor elemento entre as posições 0 e n está na posição 0

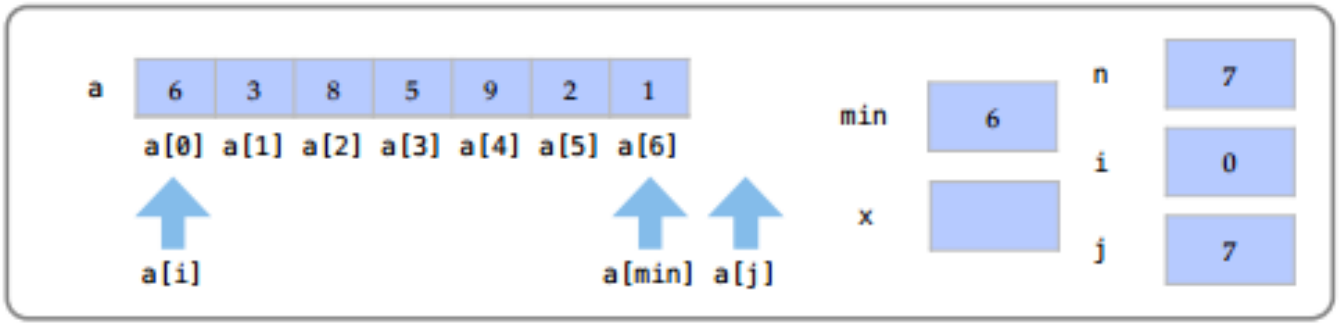
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Percorremos então os elementos entre i+1 e n-1 atualizando a posição do menor elemento. Descobrimos que o menor elemento está em a[6]

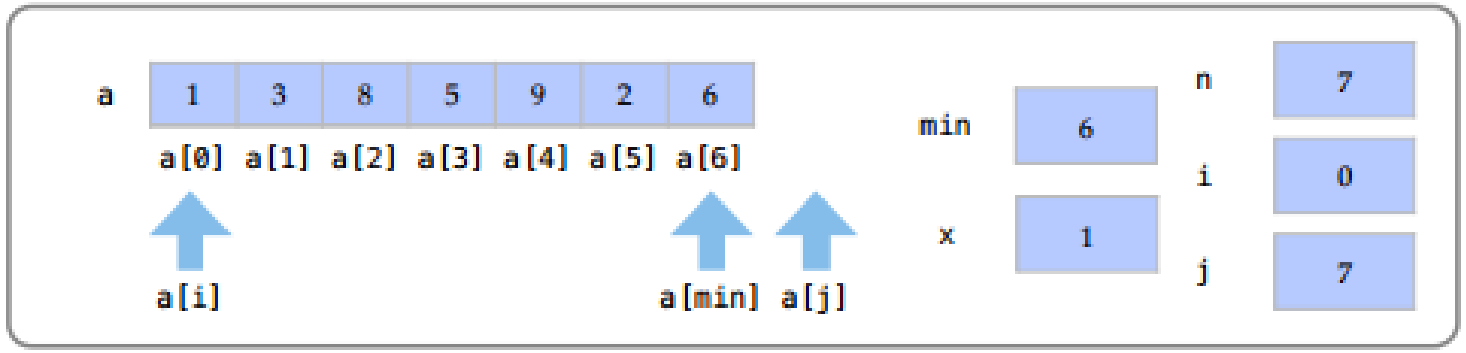
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Trocamos então os elementos da posição a[0] e a[6]. Precisamos da variável auxiliar x para fazer a troca

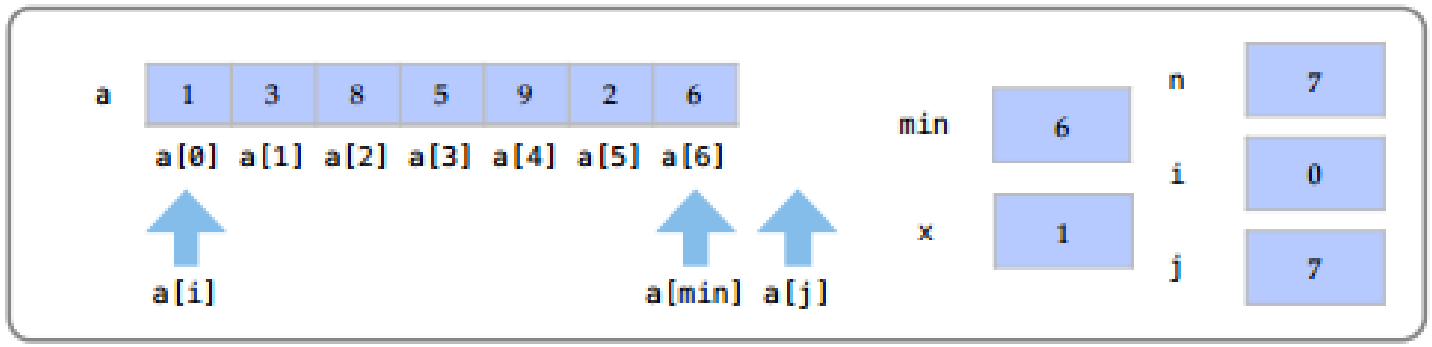
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Ao fim do primeiro passo, procuramos então o menor elemento entre a[0] e a[n-1] e colocamos no lugar de a[0]

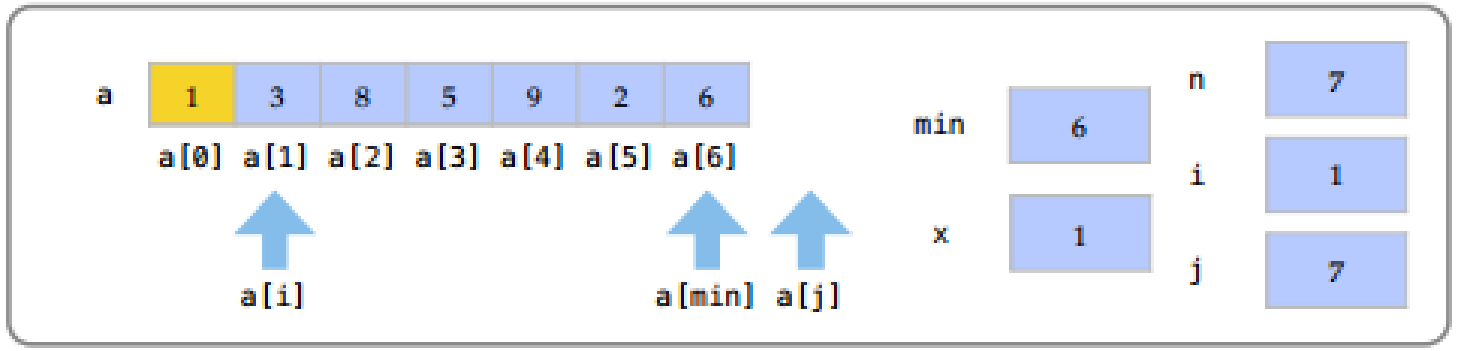
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Atualizamos i para 1 e fazemos mais um passo. Sabemos que o vetor até a[i] já está ordenado

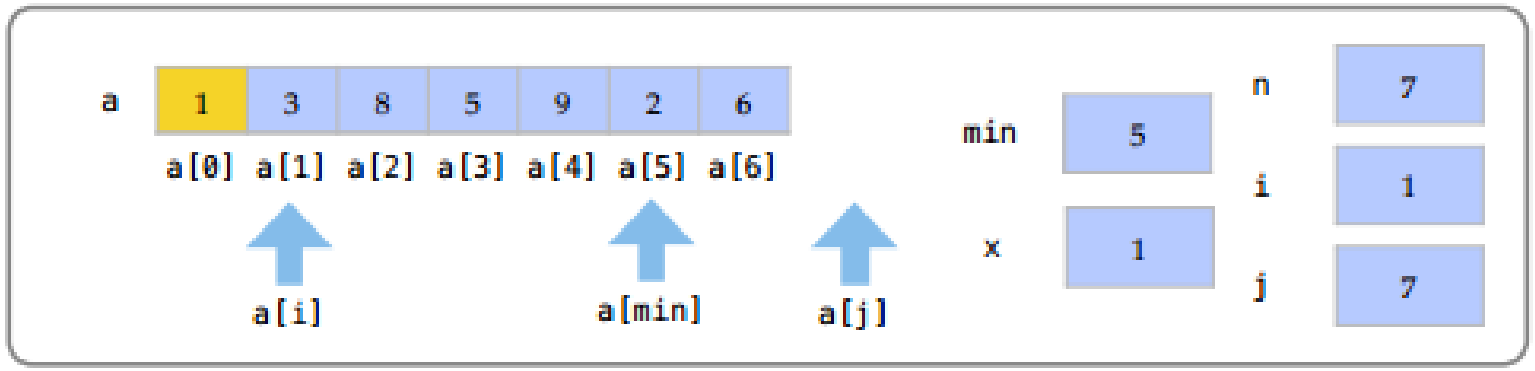
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Esta iteração procura o menor elemento entre a[i] e a[n-1]

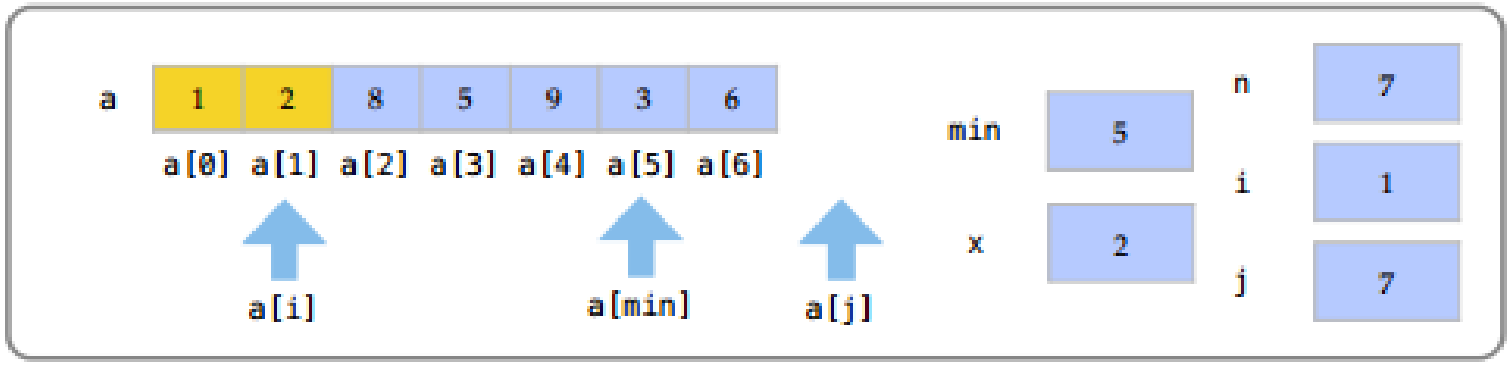
```
void selecao(int a[], int n){
    int i, j, min; // indices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Este elemento a[min] é trocado com a[i]

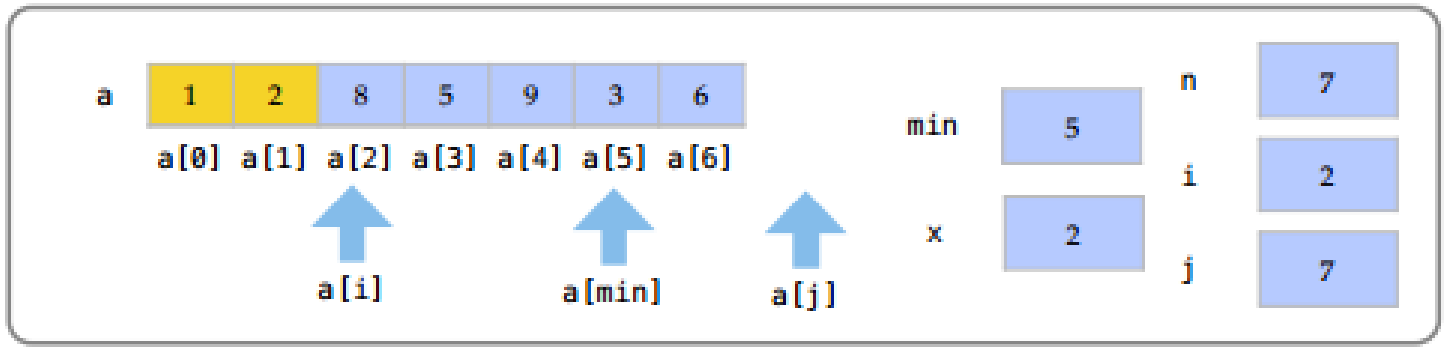
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Atualizamos novamente i para 2 e sabemos que agora que o arranjo até a[i] já está ordenado

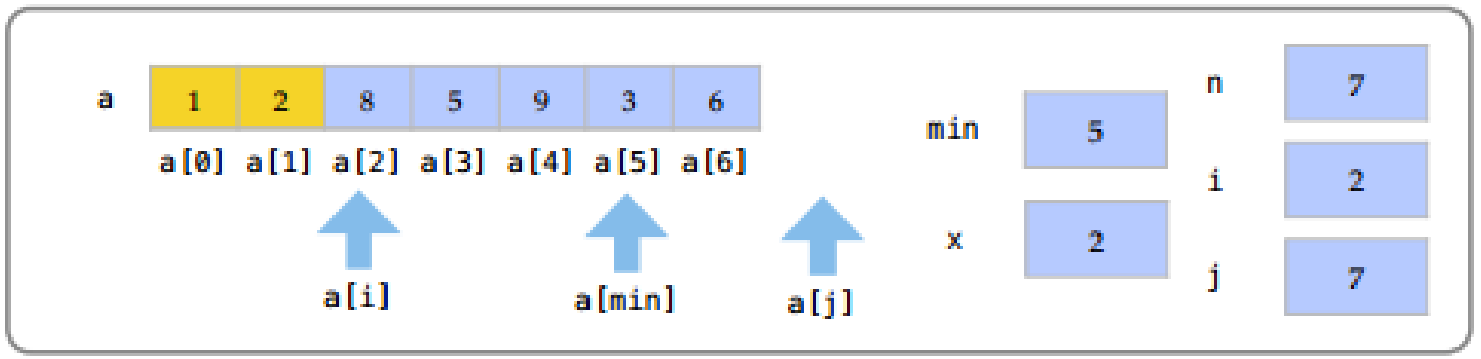
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Achamos o menor entre a[i] e a[n-1]

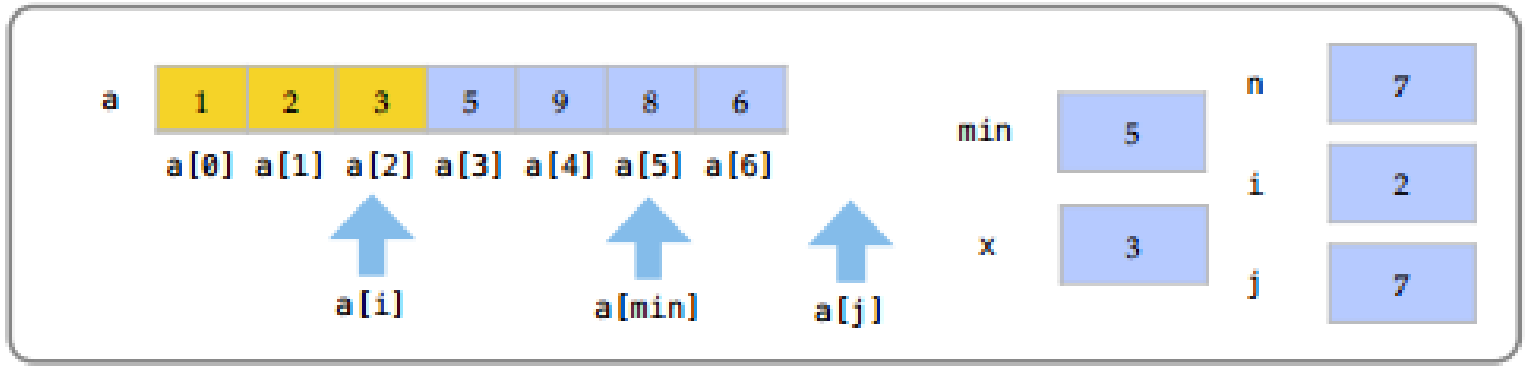
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Trocamos a[i] com a[min]

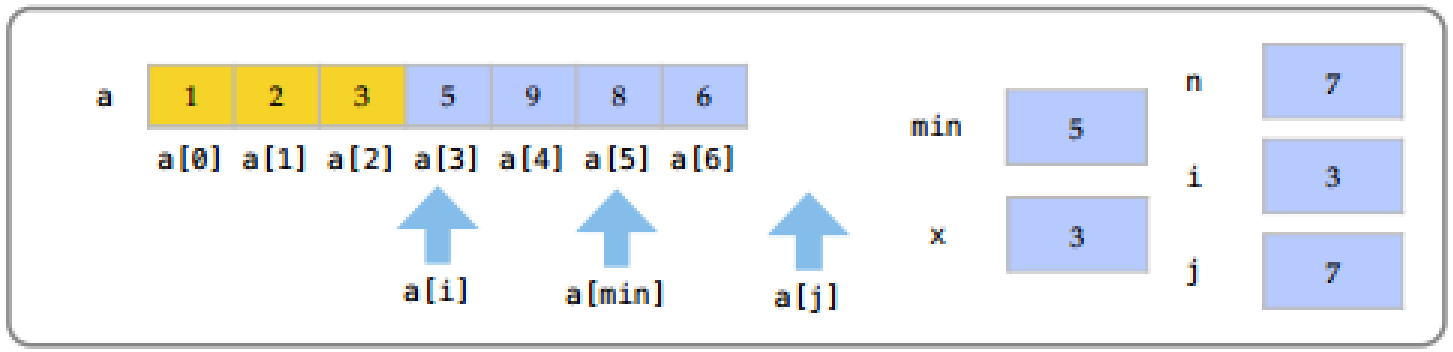
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Incrementamos i. Sabemos que o arranjo está ordenado até a posição a[i]

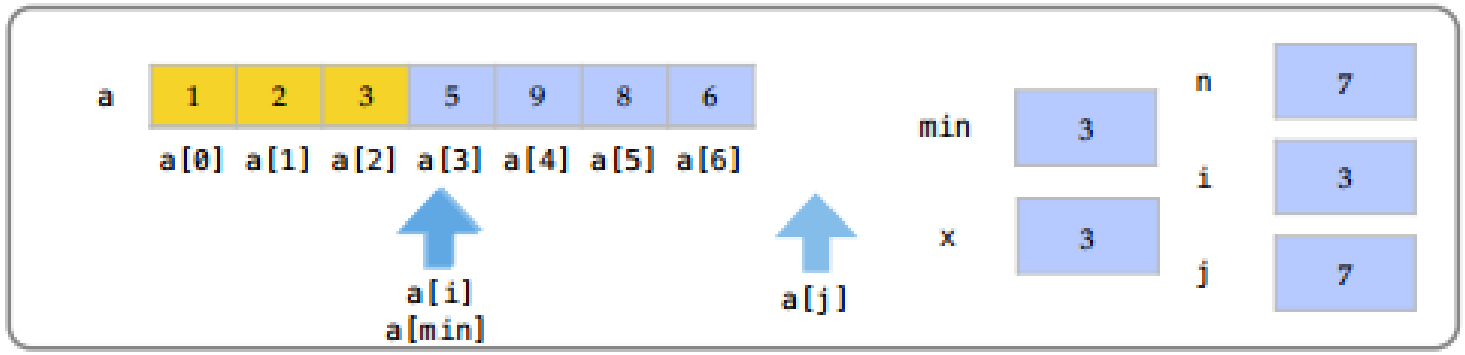
```
void selecao(int a[], int n){
    int i, j, min; // indices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Encontramos o menor elemento entre a[i] e a[n-1]

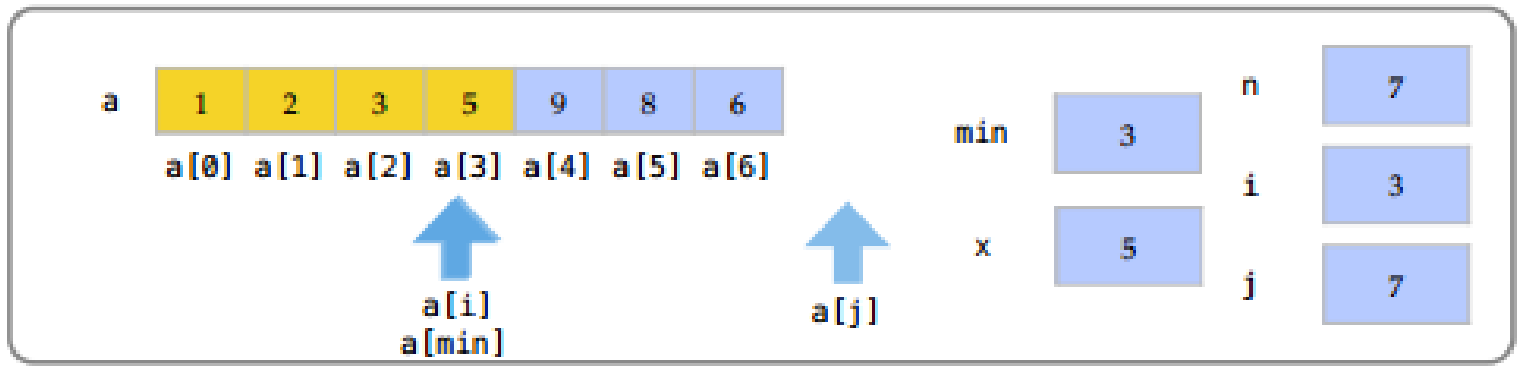
```
void selecao(int a[], int n){
    int i, j, min; // indices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Trocamos o elemento a[i] com o elemento a[min]

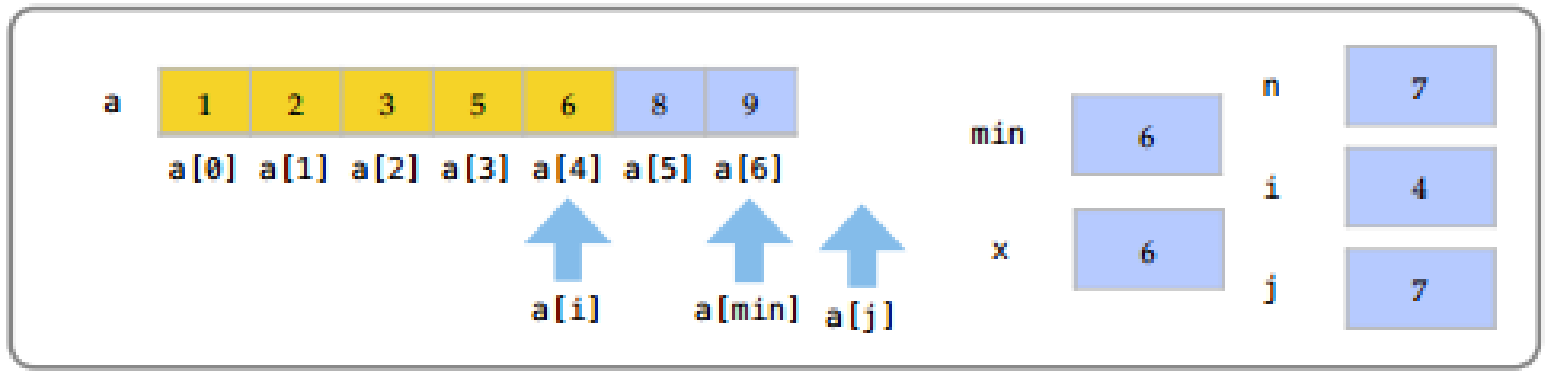
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Na próxima iteração, colocamos o menor em a[4]

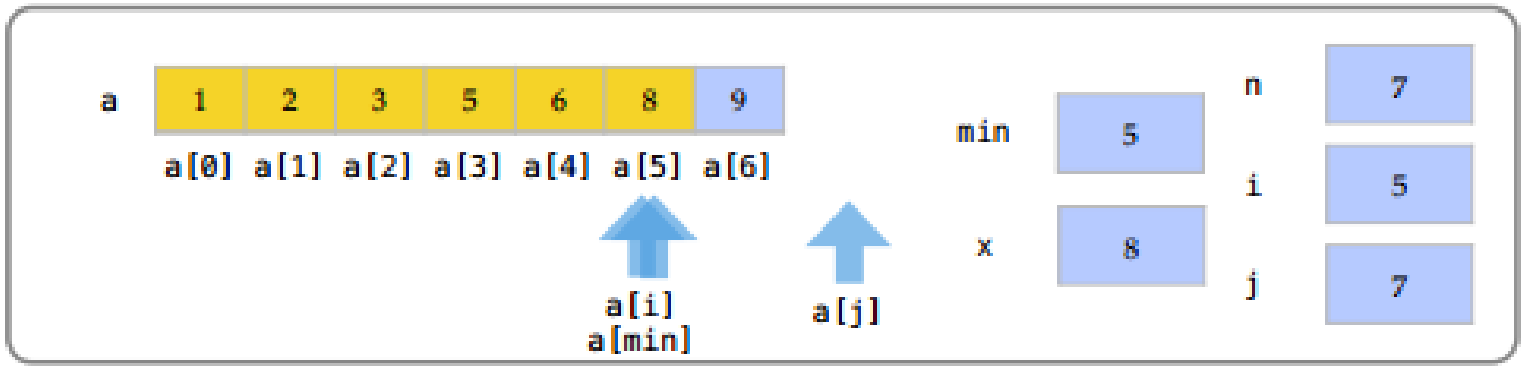
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Na última iteração, colocamos o menor em a[5]

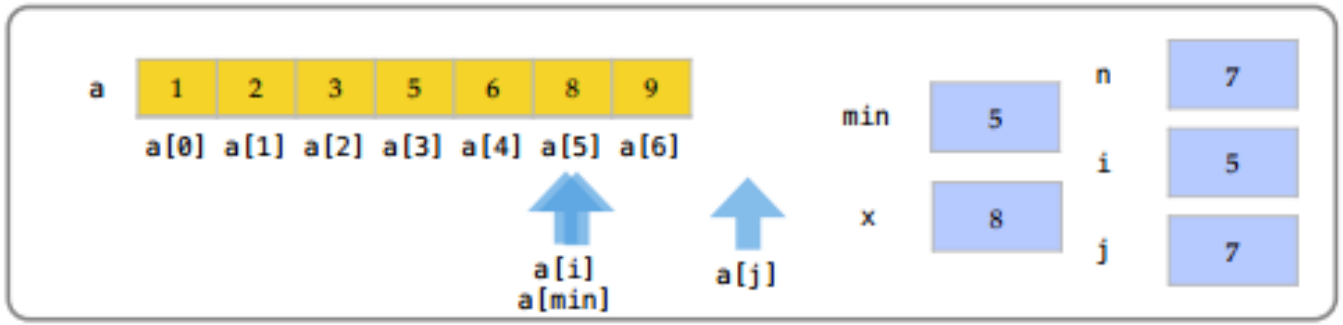
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Veja que não precisamos de uma iteração para o último elemento. Se há apenas um elemento, sabemos que ele seria o menor

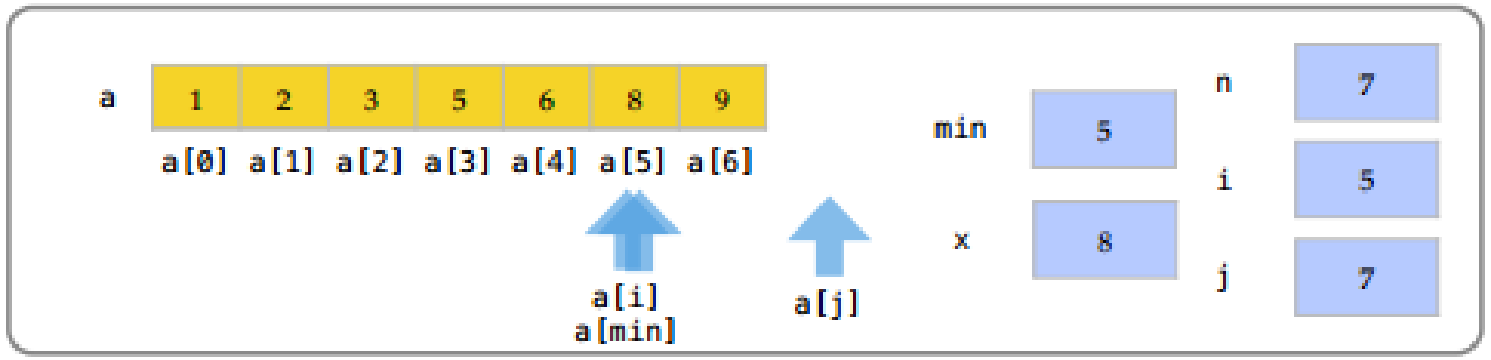
```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção

- Assim, terminamos a função com todos os elementos do arranjo **a** ordenados

```
void selecao(int a[], int n){
    int i, j, min; // índices
    int x; // elemento
    // para cada posição
    for (i = 0; i < n - 1; i++){
        // procuramos o menor entre i+1 e n e colocamos em i
        min = i; //mínimo é o i
        for (j = i + 1; j < n; j++){
            if (a[j] < a[min]){
                min = j; //mínimo é o j
            }
        }
        // troca a[i] com a[min]
        x = a[min];
        a[min] = a[i];
        a[i] = x;
    }
}
```



Ordenação por Seleção - Análise

- No laço interno, sempre que procuramos o menor elemento entre **i** e **n**, fazemos **n-1** comparações

- No laço mais externo, estas n-1 comparações são feitas n-1 vezes

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

- Assim, no total, o algoritmo faz $O(n^2)$ comparações

Ordenação por Seleção - Análise

- Em relação ao número de atribuições, cada troca envolve 3 atribuições. Como são feitas *n-1* trocas, temos *3(n-1) = O(n)* atribuições de movimentação

- Além destas, temos a atualização da posição do menor

  - Esta ocorre em média *n log n* vezes
  - Com estas, o custo total de atribuições é *O(n log n)*

Ordenação por Seleção – Análise - **Vantagens**

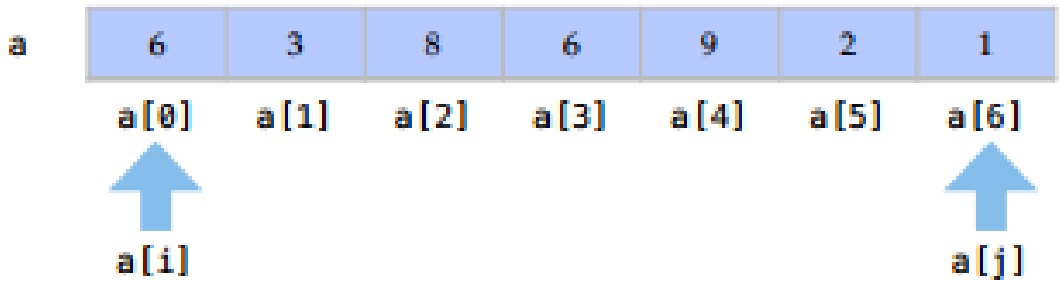
- Em relação ao custo de movimentação, temos um custo linear *O(n)*
    - É então um bom algoritmo onde estas movimentações por algum motivo custem muito
- O algoritmo é interessante para arranjos pequenos

Ordenação por Seleção – Análise - **Desvantagens**

- Se o arranjo já estiver ordenado, isto não ajuda o algoritmo, pois o custo de comparações continua $O(n^2)$
  - Quando isso ocorre, dizemos que o método não é adaptável
- O algoritmo não é estável pois a troca entre elementos pode destruir a ordem relativa destes

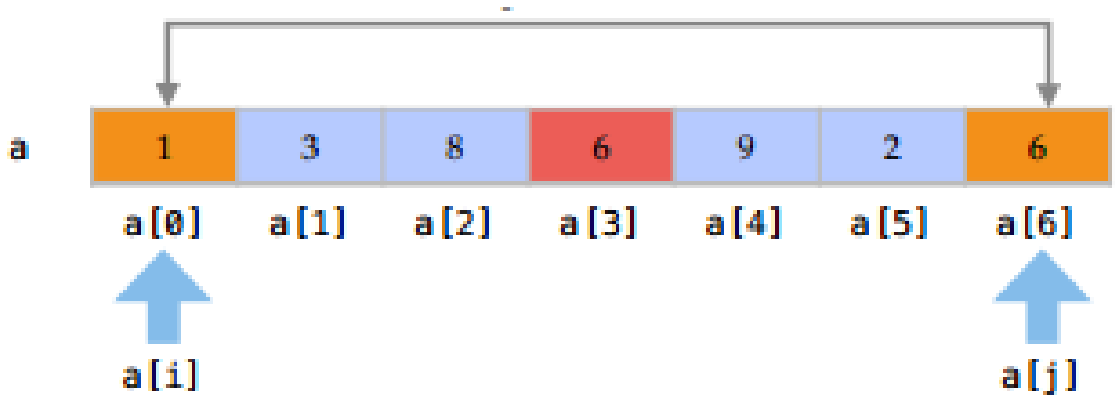
Ordenação por Seleção – Exemplo de perda de estabilidade

- Suponha um passo onde trocaremos o elemento a[i] (6) com o elemento a[j] (1)



Ordenação por Seleção – Exemplo de perda de estabilidade

- Ao fazer esta troca, o elemento a[i] (6) perde sua ordem relativa entre qualquer elemento entre a[i+1] e a[j-1]



- Neste caso, o elemento a[i] perdeu sua ordem relativa com o elemento a[3], que tem o mesmo valor

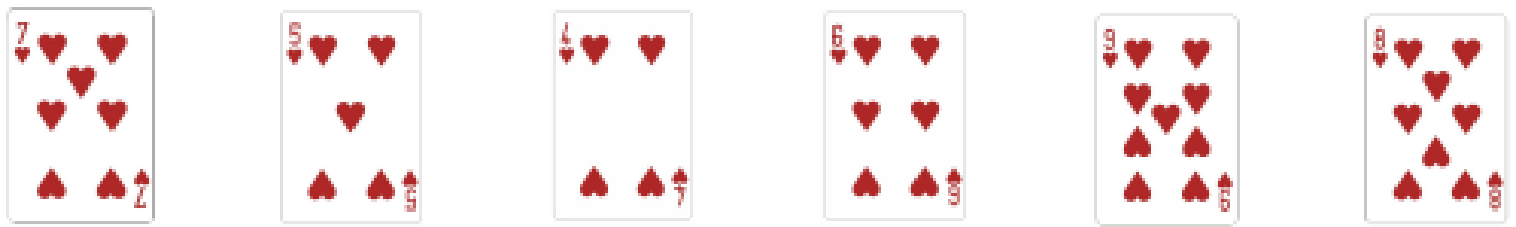
Ordenação por Inserção - InsertionSort

Ordenação por Inserção

- Este é o algoritmo preferido dos jogadores de cartas
- É um algoritmo onde a cada passo de repetição:
  - Temos um arranjo ordenado até um certo ponto
  - Pegamos o próximo elemento
  - Colocamos este elemento na posição correta entre o primeiro e ele mesmo

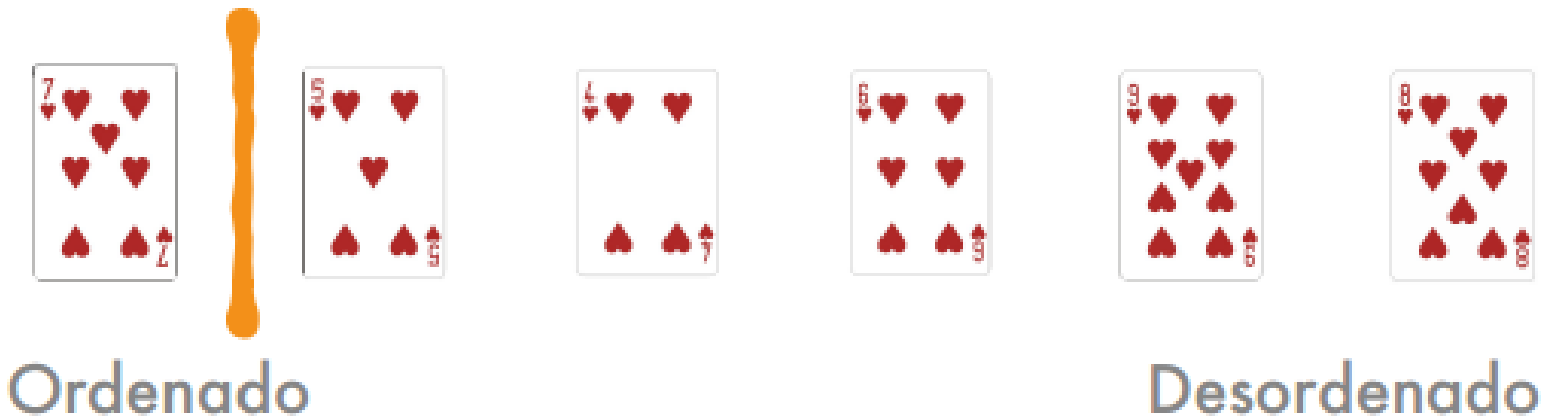
Ordenação por Inserção

- Considere um arranjo com os seguintes elementos



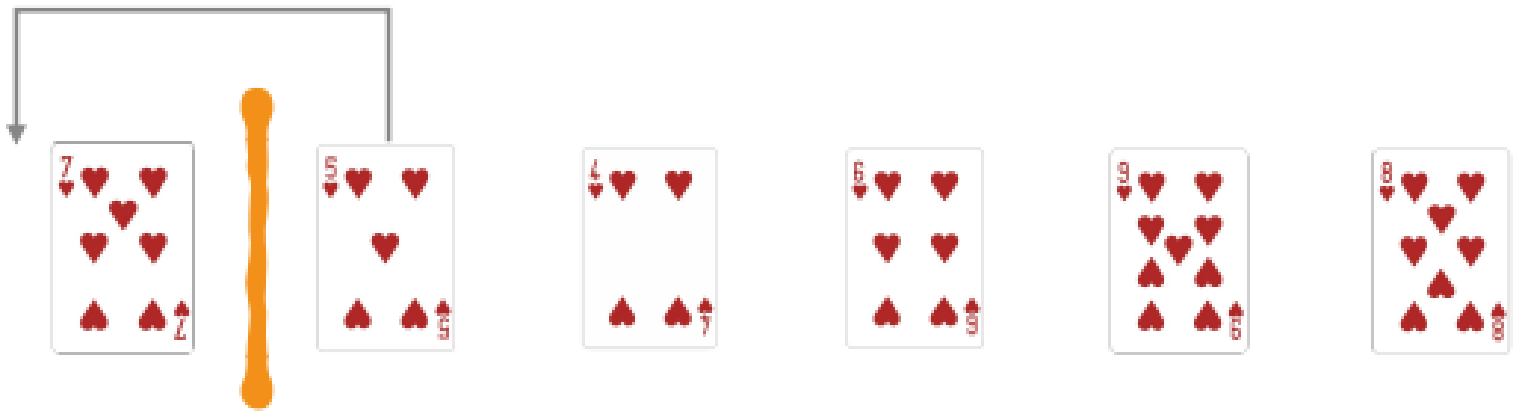
Ordenação por Inserção

- Sabemos que o arranjo até o segundo elemento já está ordenado pois um arranjo de apenas um elemento está sempre ordenado



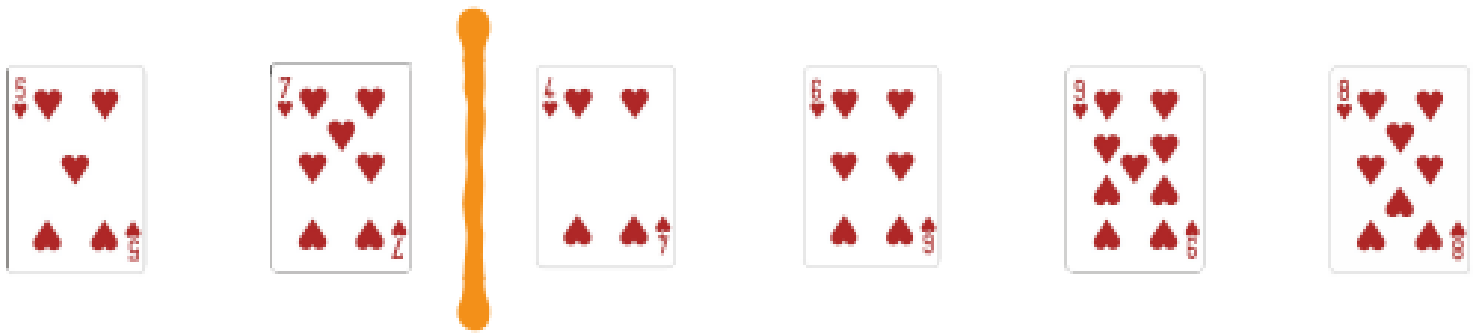
Ordenação por Inserção

- Colocamos então o próximo elemento na posição correta do arranjo ordenado



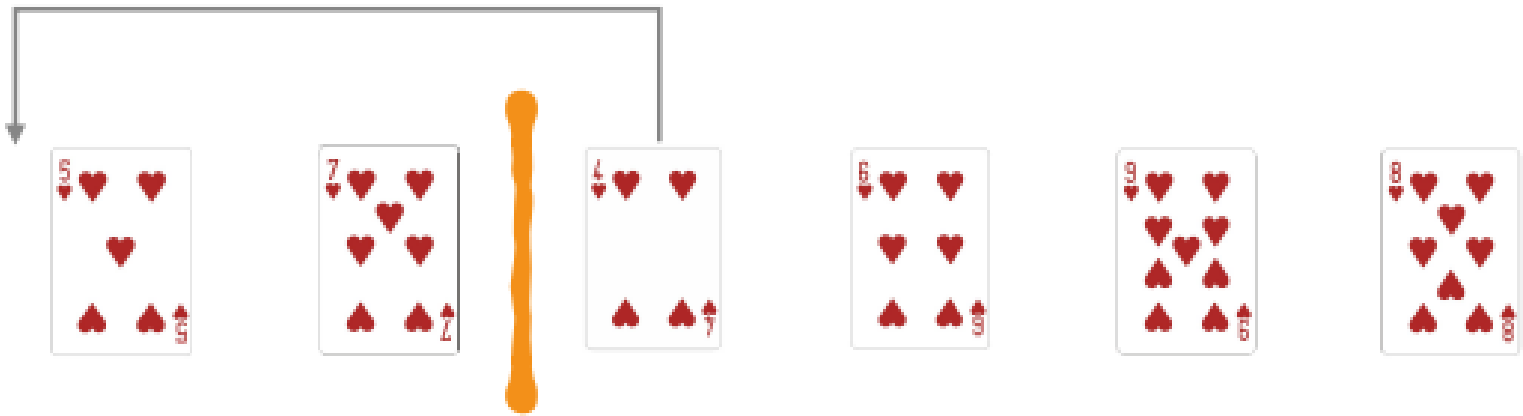
Ordenação por Inserção

- Sabemos agora então que os dois primeiros elementos estão ordenados



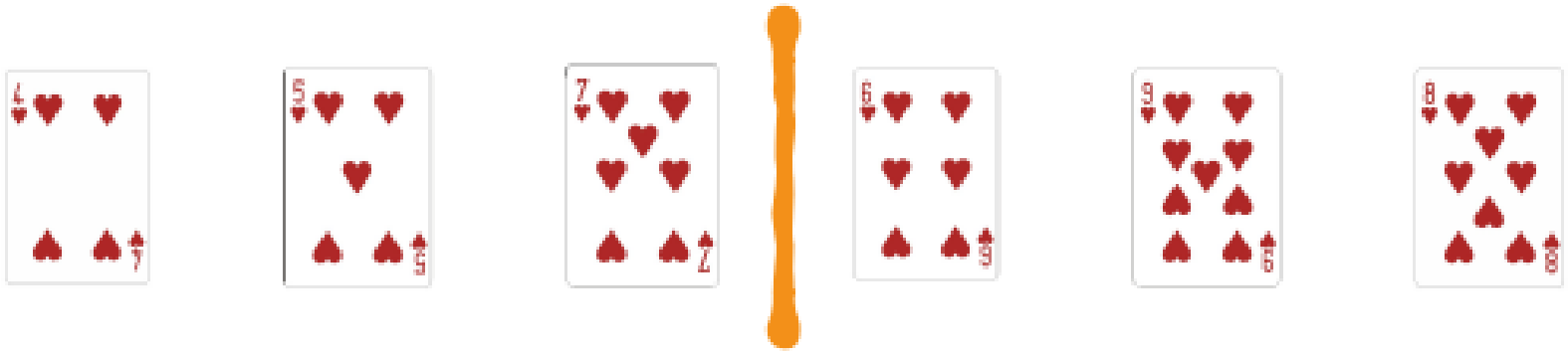
Ordenação por Inserção

- O próximo elemento deverá ser colocado em sua posição correta



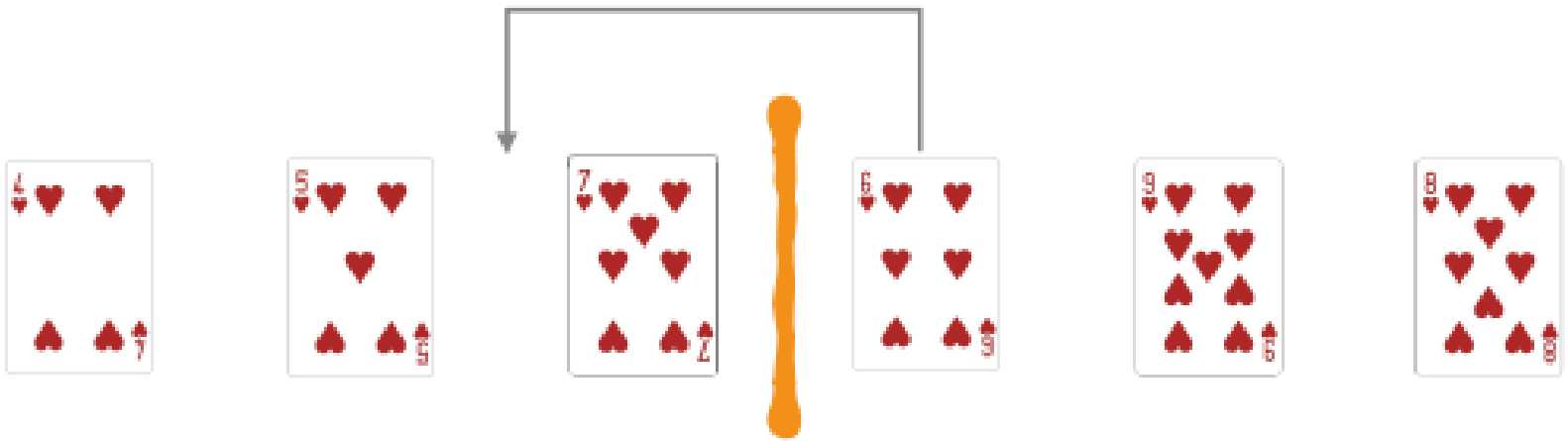
Ordenação por Inserção

- Sabemos agora que os três primeiros elementos estão ordenados



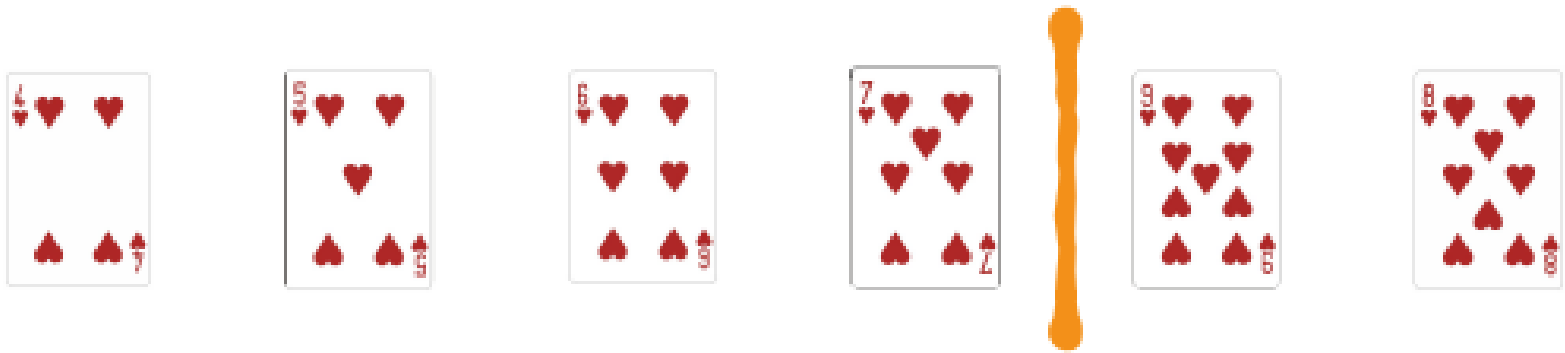
Ordenação por Inserção

- O próximo elemento deverá ser colocado em sua posição correta



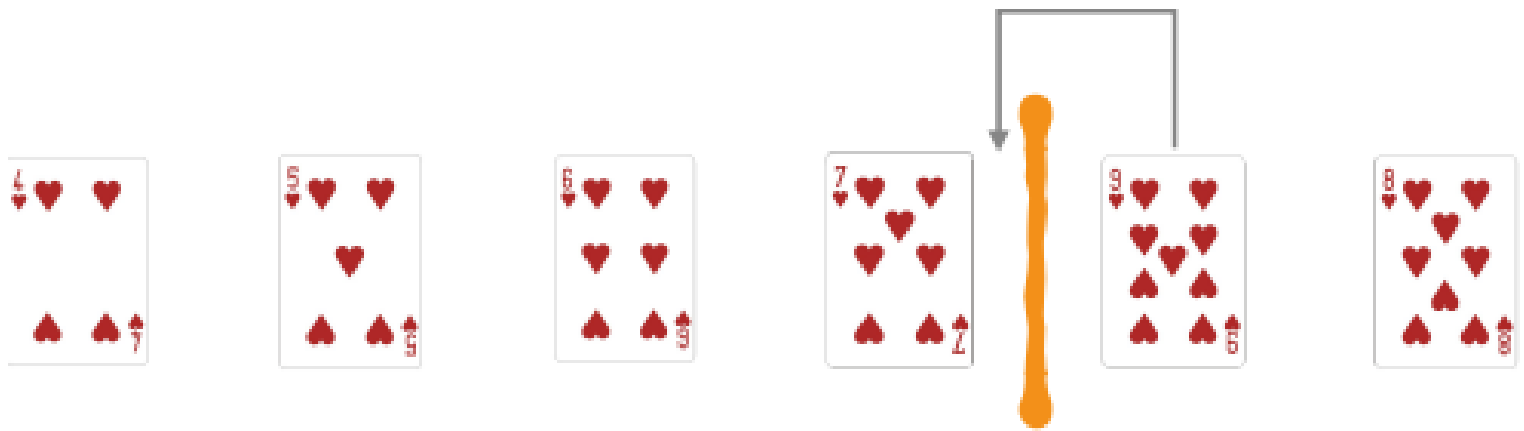
Ordenação por Inserção

- Sabemos agora que os quatro primeiros elementos estão ordenados



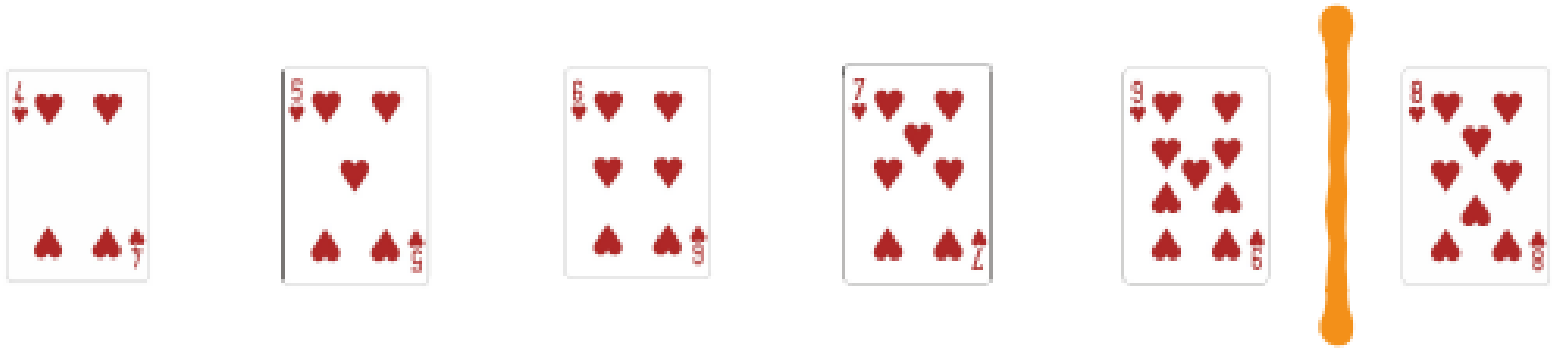
Ordenação por Inserção

- O próximo elemento deverá ser colocado em sua posição correta



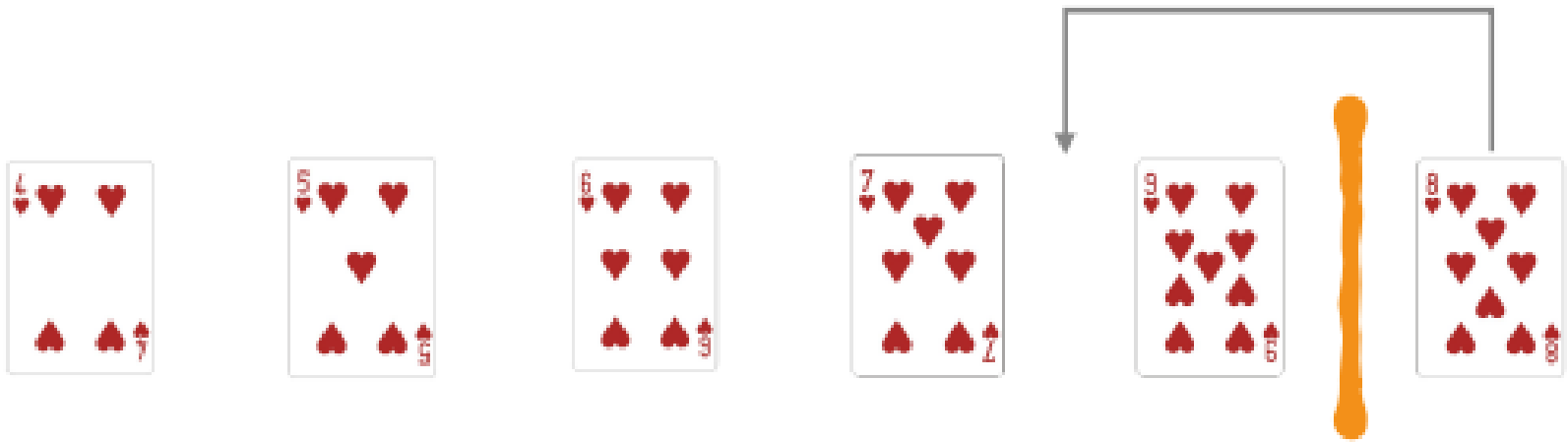
Ordenação por Inserção

- Sabemos agora que os cinco primeiros elementos estão ordenados



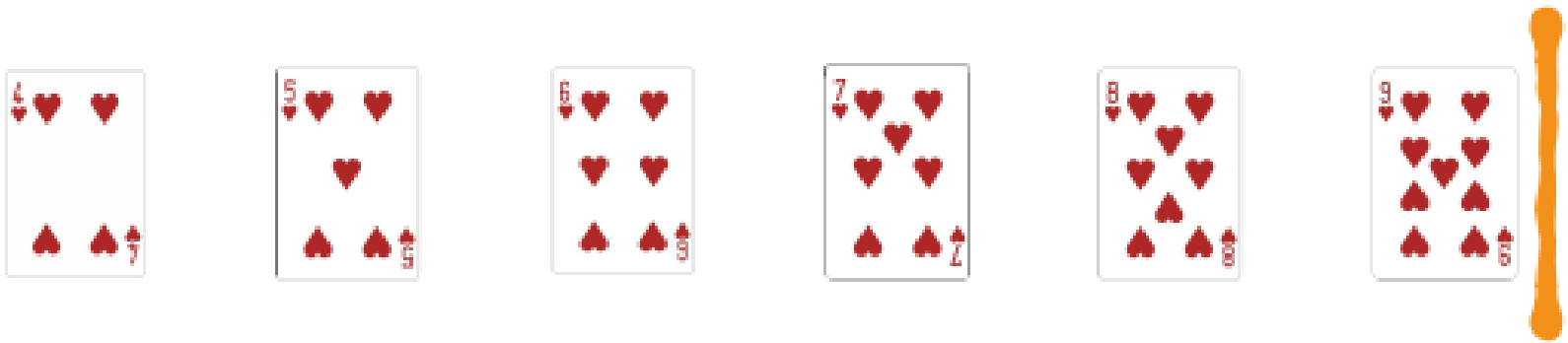
Ordenação por Inserção

- O próximo elemento deverá ser colocado em sua posição correta



Ordenação por Inserção

- Sabemos agora que todos os elementos estão ordenados



Ordenação por Inserção

Arranjo	7	5	4	6	9	8
Passo 1	7	5	4	6	9	8
Passo 2	5	7	4	6	9	8
Passo 3	4	5	7	6	9	8
Passo 4	4	5	6	7	9	8
Passo 5	4	5	6	7	9	8
Arranjo	4	5	6	7	8	9

Movimentação | Desordenado | Ordenado

Ordenação por Inserção

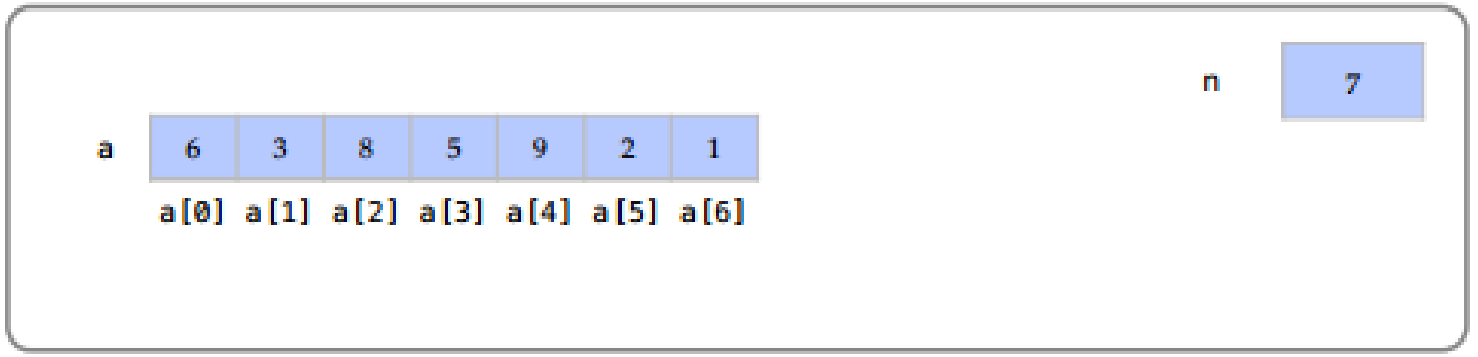
- Ordenação por inserção para um arranjo de inteiros

```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```

Ordenação por Inserção

- A função ordena o arranjo a, que contém n elementos

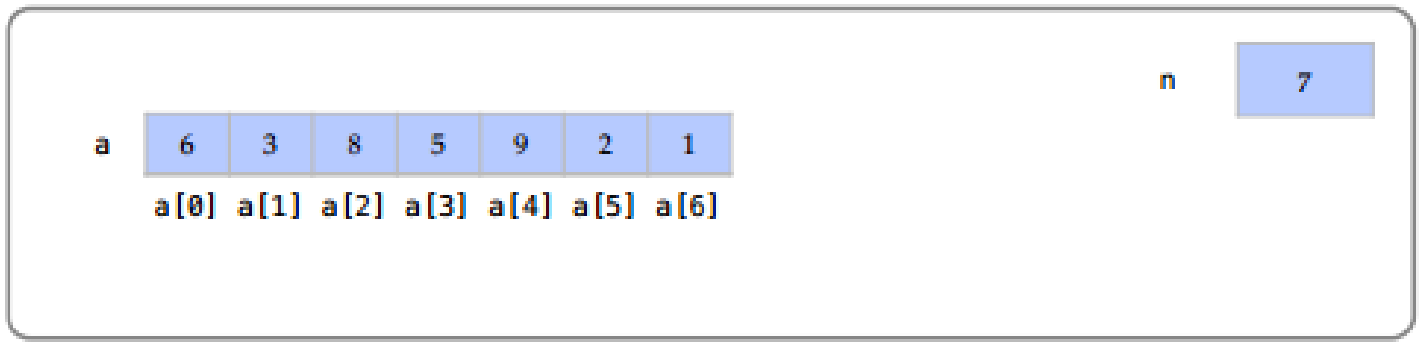
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Como sabemos que arranjos são passados por referência em C++, não precisamos retornar nada desta função

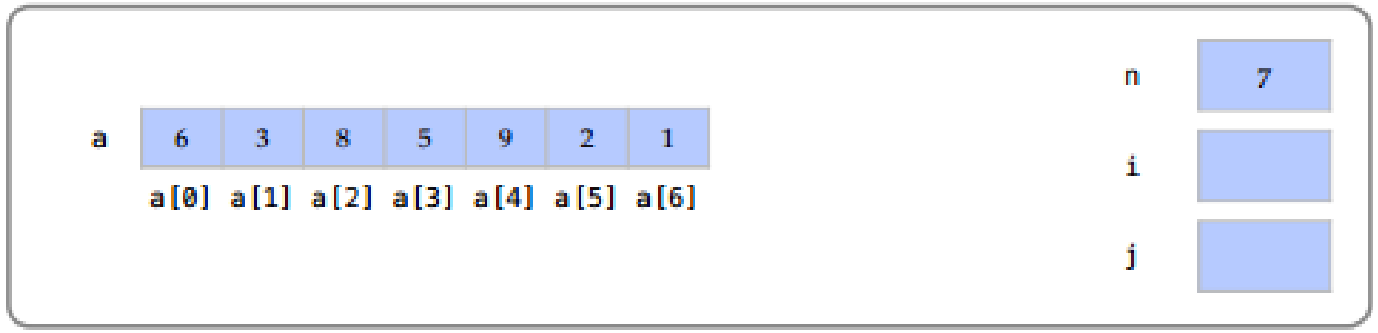
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Os índices i e j são utilizados para percorrermos o arranjo em um for aninhado e colocar cada elemento a[i] em sua posição no arranjo ordenado

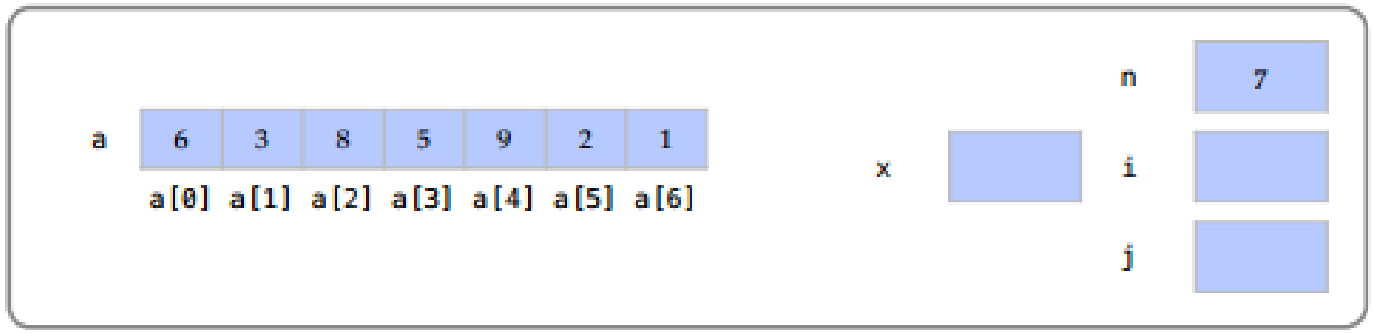
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- A variável x deve ser do mesmo tipo dos elementos do arranjo pois será uma variável temporária para fazer as trocas entre elementos

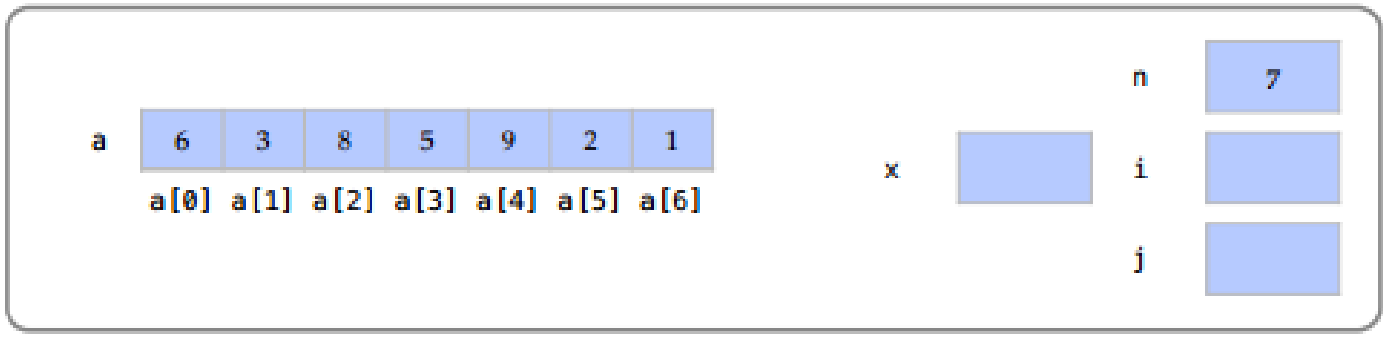
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Entramos em um for que, para cada posição i do arranjo, colocará o elemento a[i] em seu lugar correto do arranio ordenado de a[0] até a[i-1]

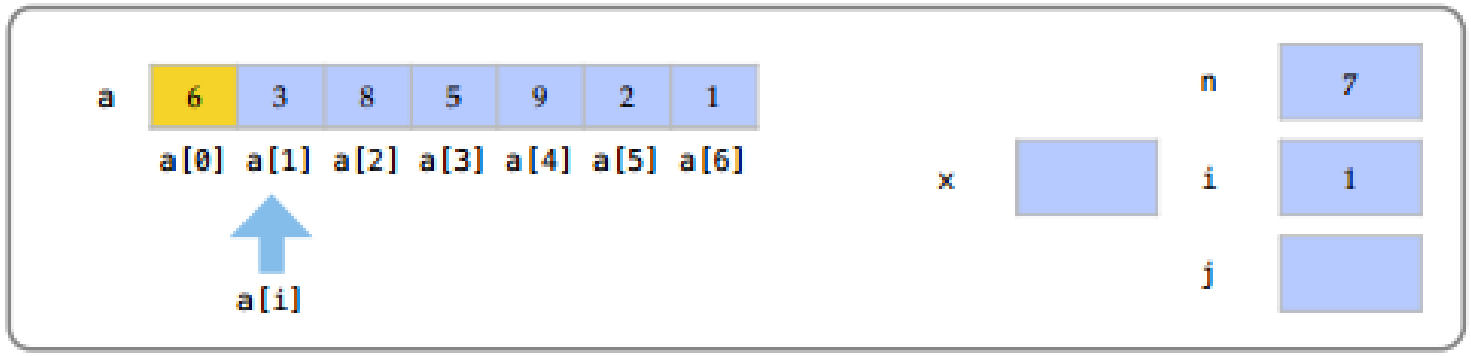
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Inicialmente, quando i é 1, sabemos que o arranjo até a[0] já está ordenado

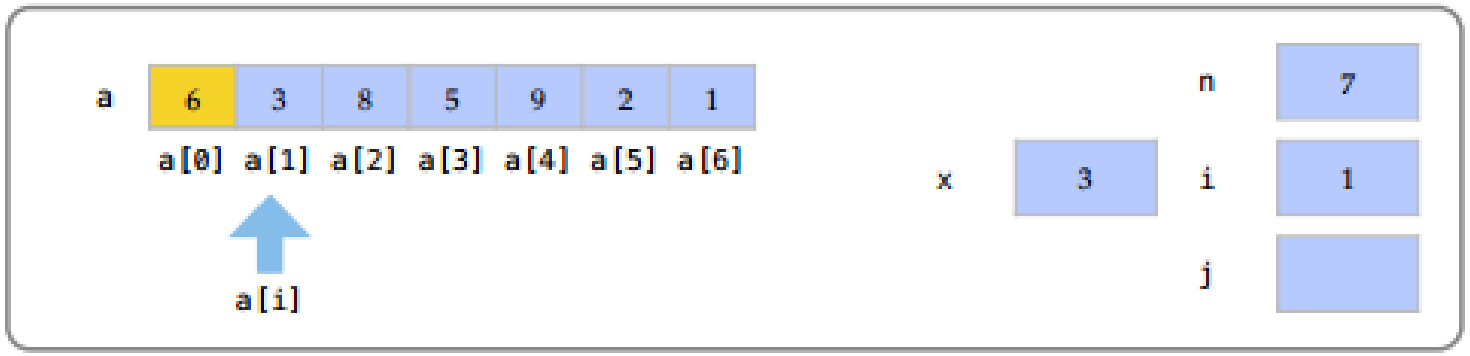
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- A variável x recebe uma cópia do elemento a[i]. Está cópia será colocada em sua posição correta no arranjo ordenado

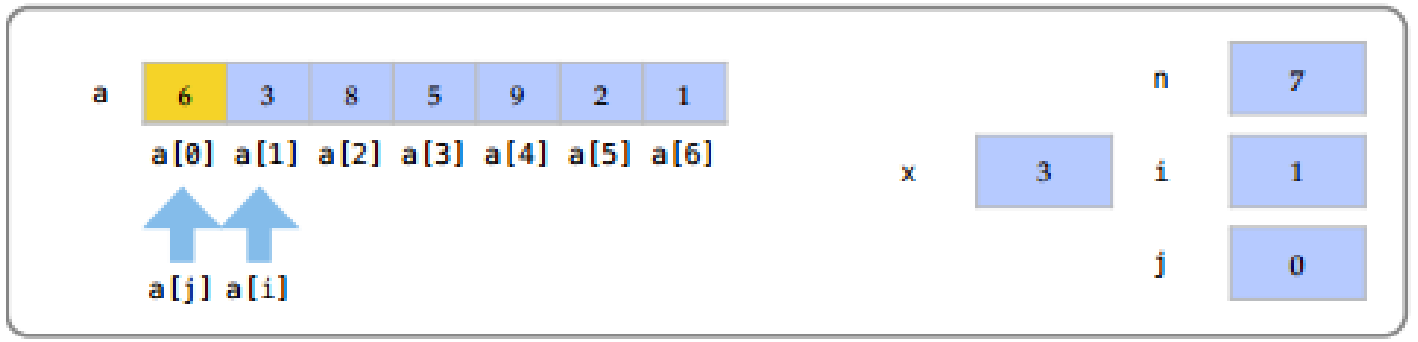
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- O índice j, recebe i-1. É a partir de a[i-1] que procuraremos a posição correta de a[i]

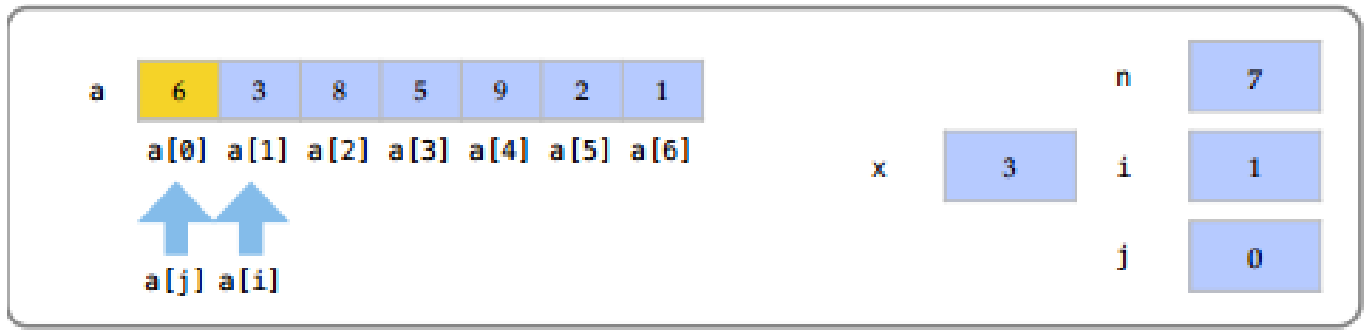
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- No while aninhado, deslocamos os elementos anteriores a a[i]. Isso até encontrarmos a posição de x ou até que todos os elementos tenham sido deslocados

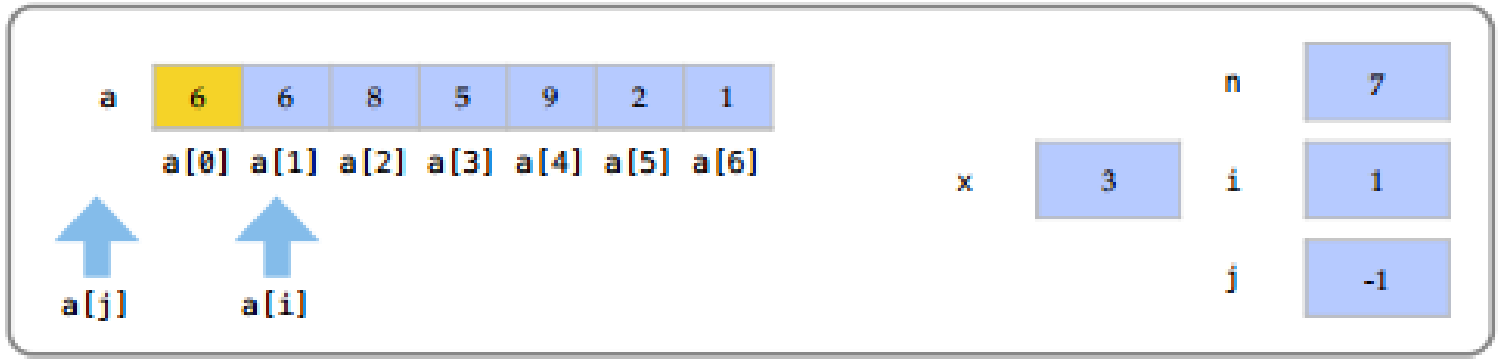
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Na primeira iteração, temos que a[0] é deslocado. O valor de a[i] ainda está salvo em x

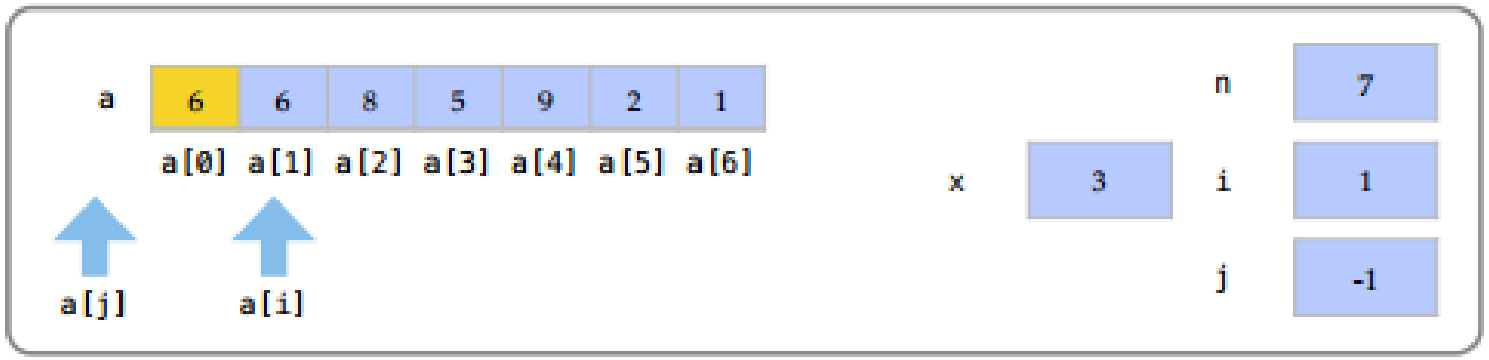
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Saímos deste loop pois todos os elementos foram deslocados

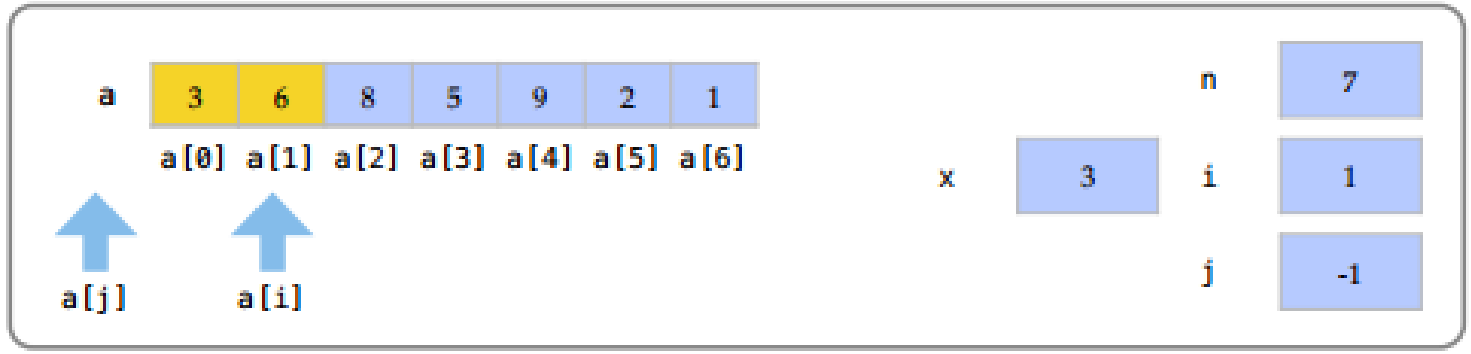
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- O elemento x, que saiu de a[i], é então inserido em sua posição correta a[j+1]

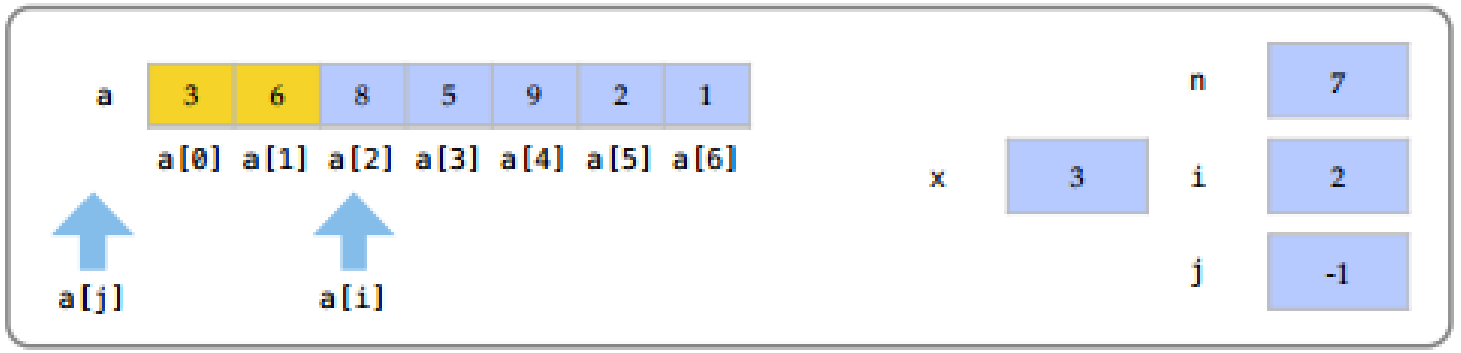
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Incrementamos i para 2, e sabemos agora que o arranjo até a[1] já está ordenado

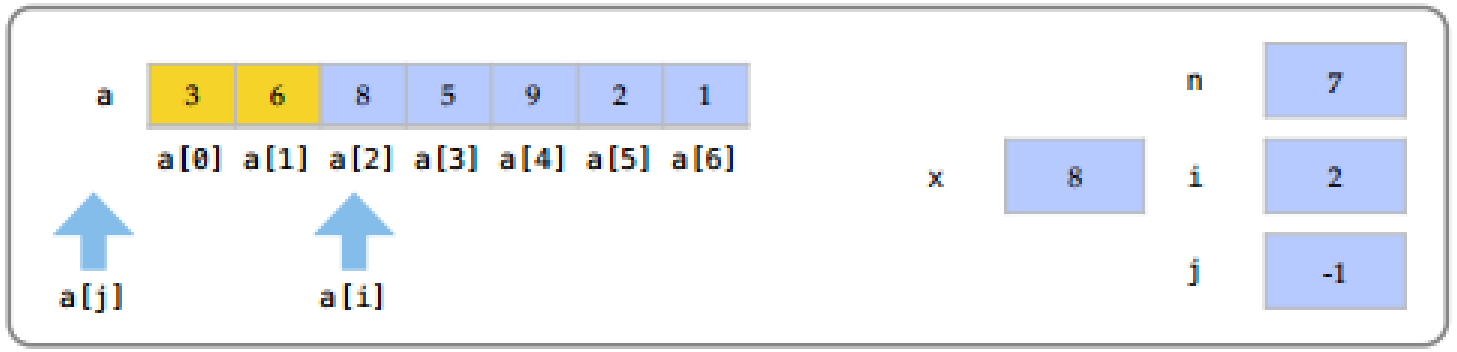
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Repetindo o processo, x guarda uma cópia do elemento a[i] em questão

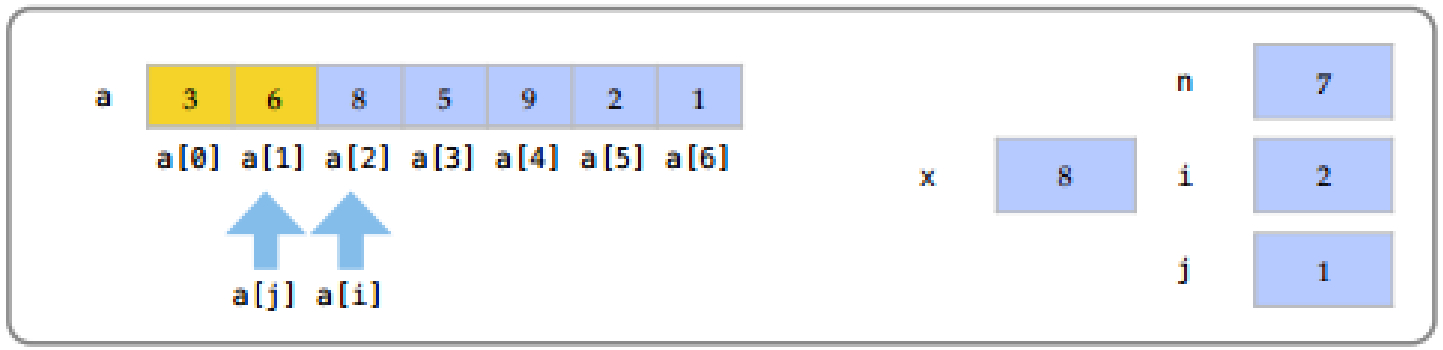
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- A partir de a[i-1] (agora a[j]) procuraremos a posição correta para o elemento a[i]

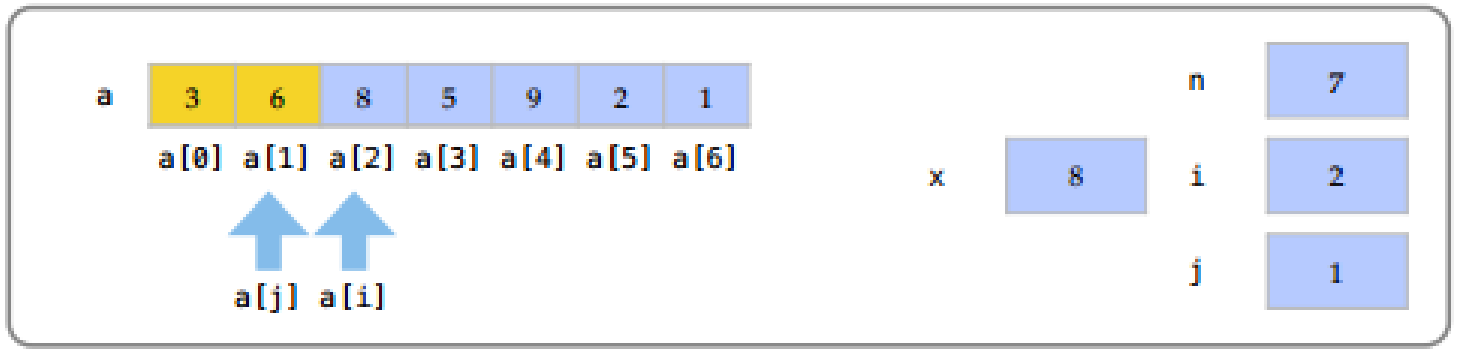
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Como o elemento guardado x não é menor que o elemento a[j], já achamos a posição correta para ele

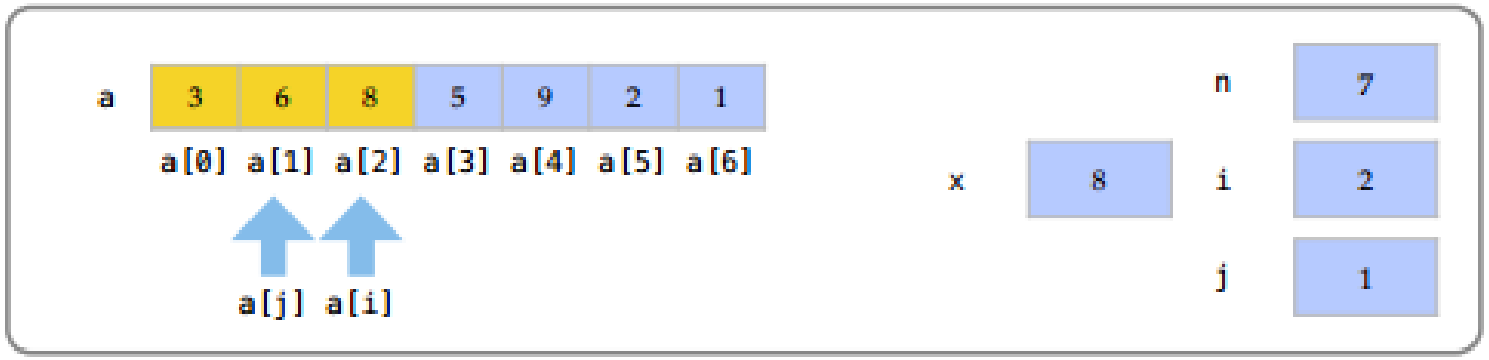
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- O elemento x é colocado na posição a[j+1], que já era a sua posição original

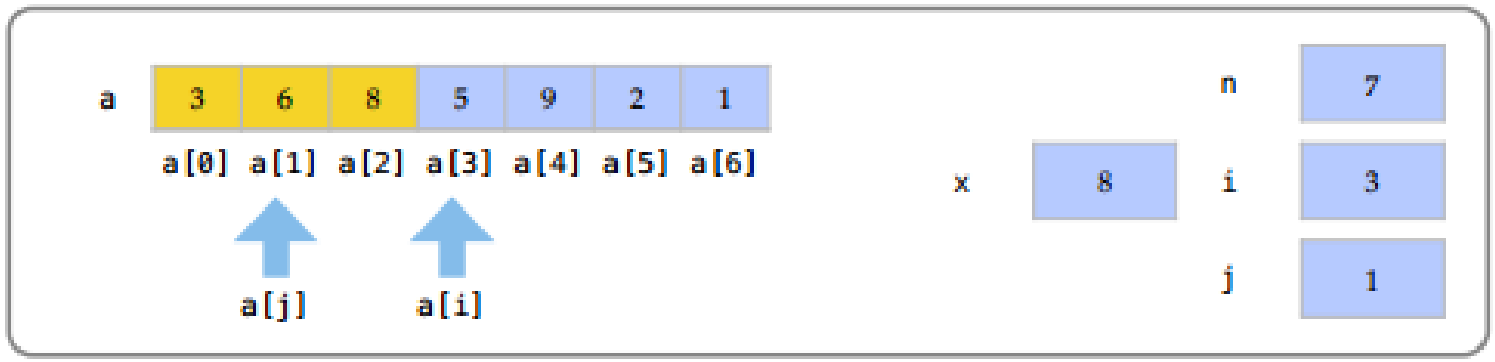
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Sabemos que o vetor até a[2] está ordenado e incrementamos i para 3

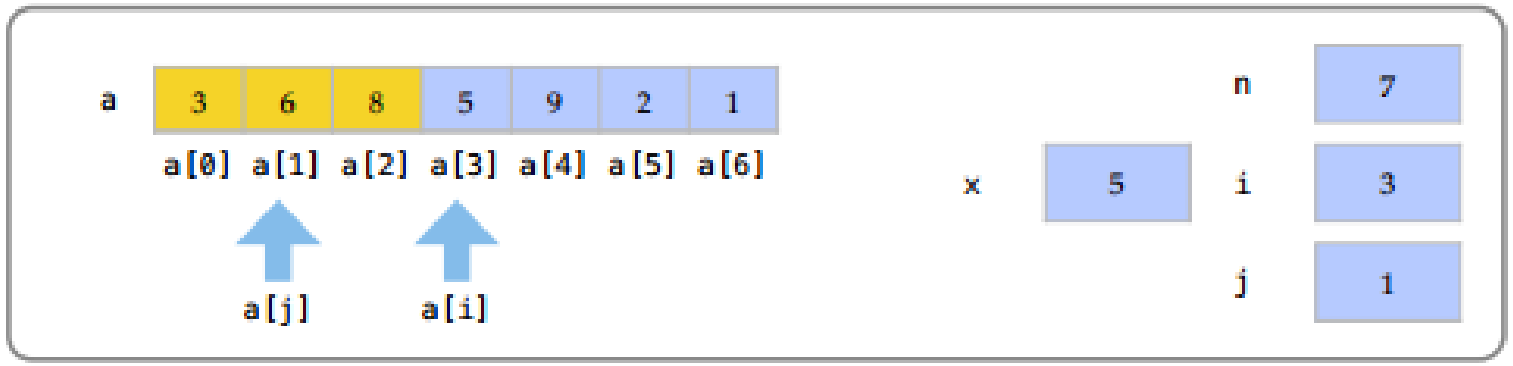
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- A variável x recebe uma cópia de a[i]

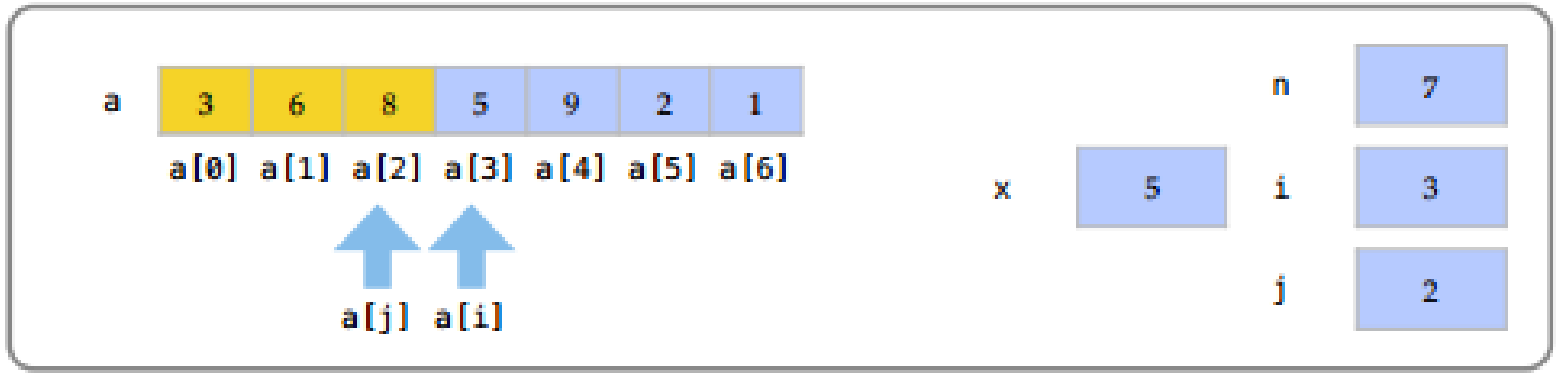
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- O índice j marca que procuraremos a posição a partir de a[i-1]

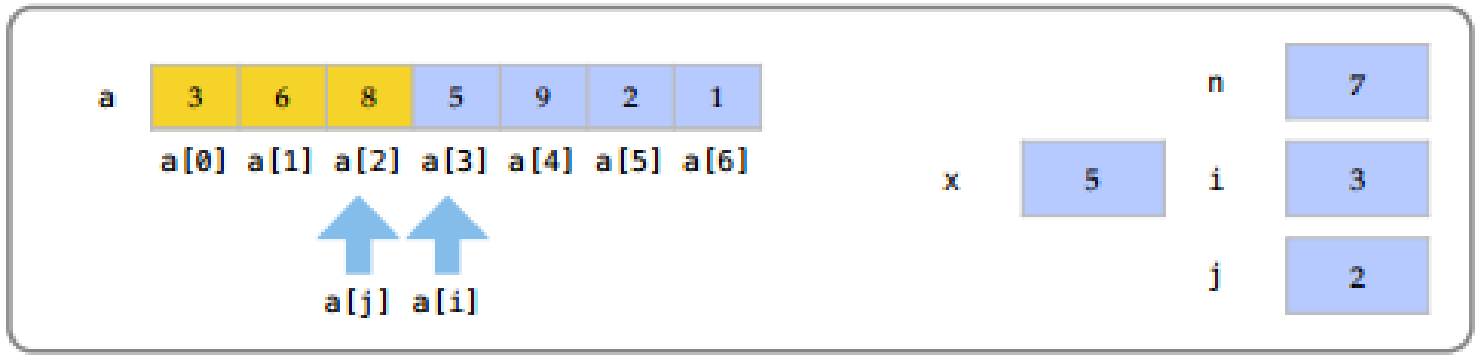
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Enquanto não achamos a posição do elemento ou não deslocamos todos os elementos

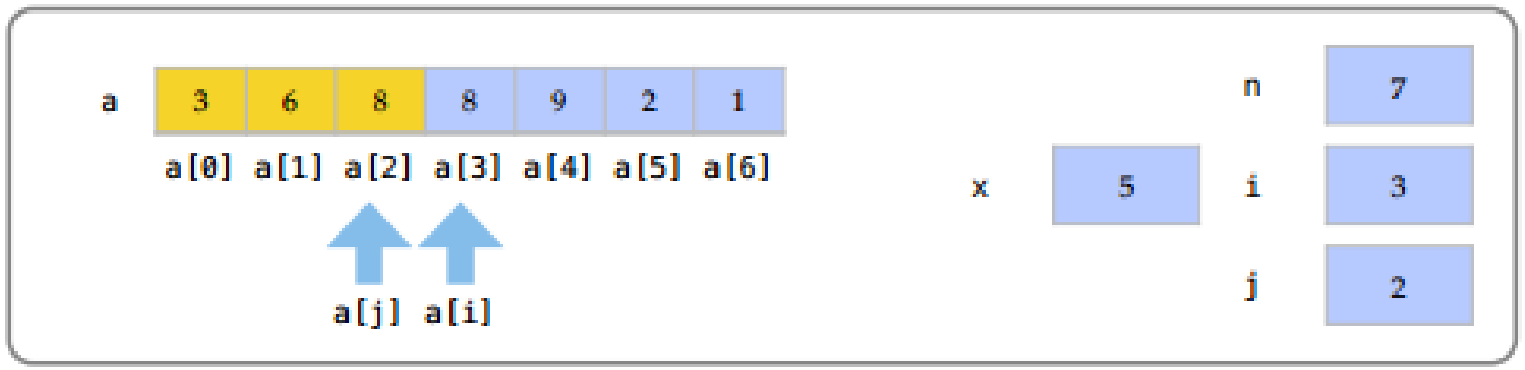
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Deslocamos o elemento a[j] em uma posição

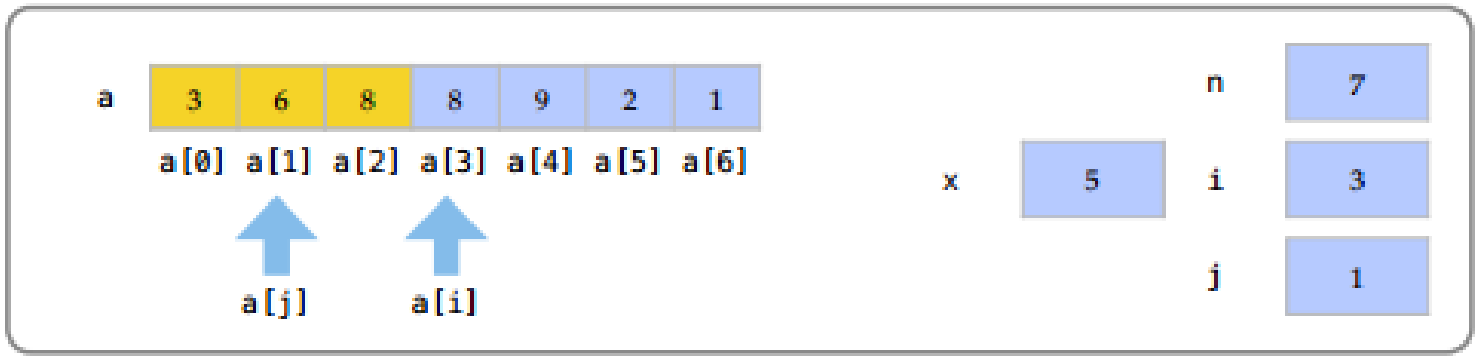
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Decrementamos j para testar o próximo elemento

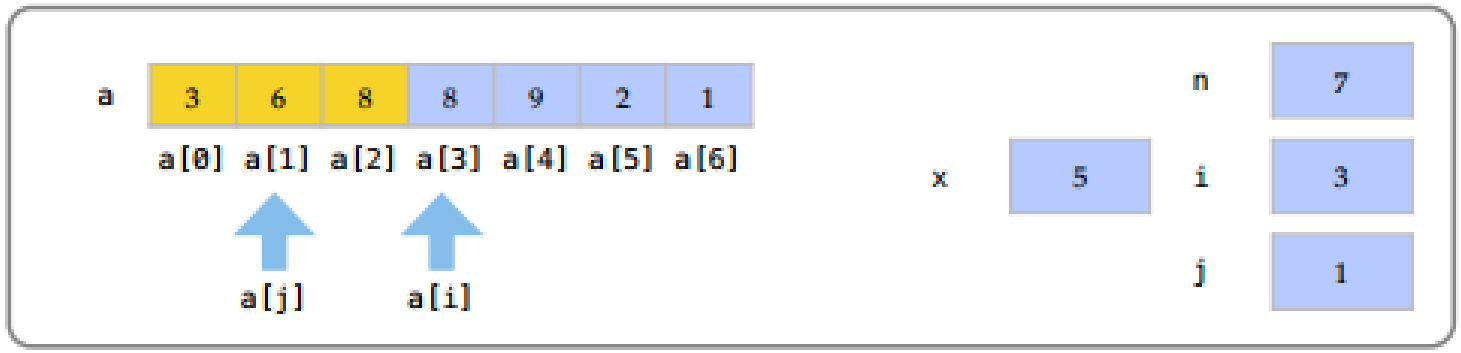
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Ainda não achamos a posição correta de x nem deslocamos todos os elementos

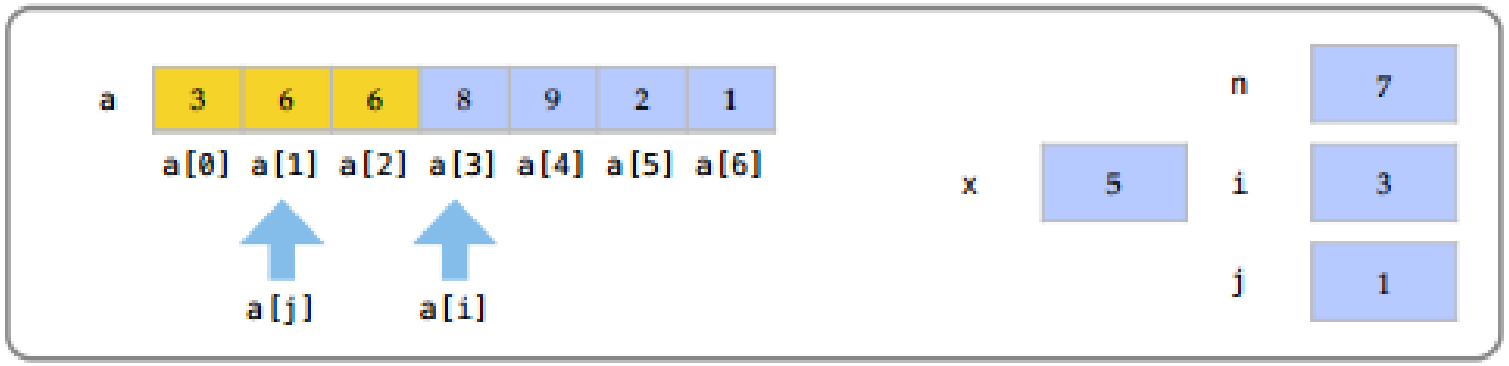
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Deslocamos o elemento a[j] em uma posição

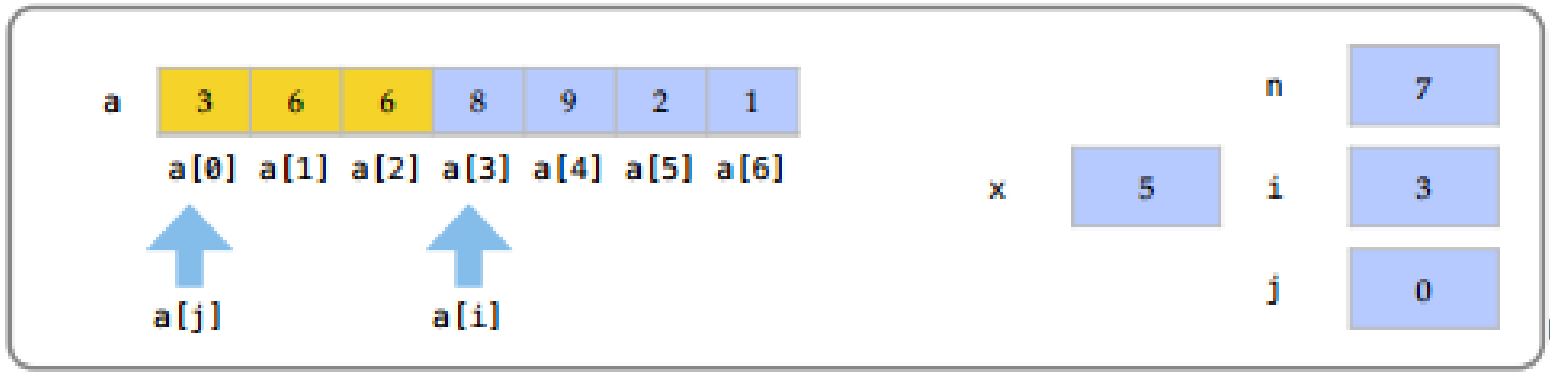
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Decrementamos j para testar a próxima posição

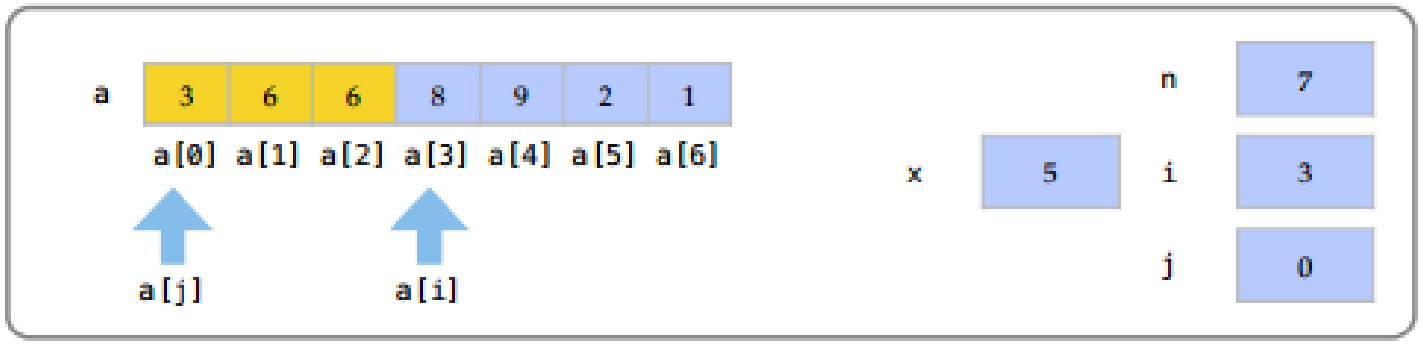
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Como o elemento x não é menor que a[j], encontramos sua posição correta

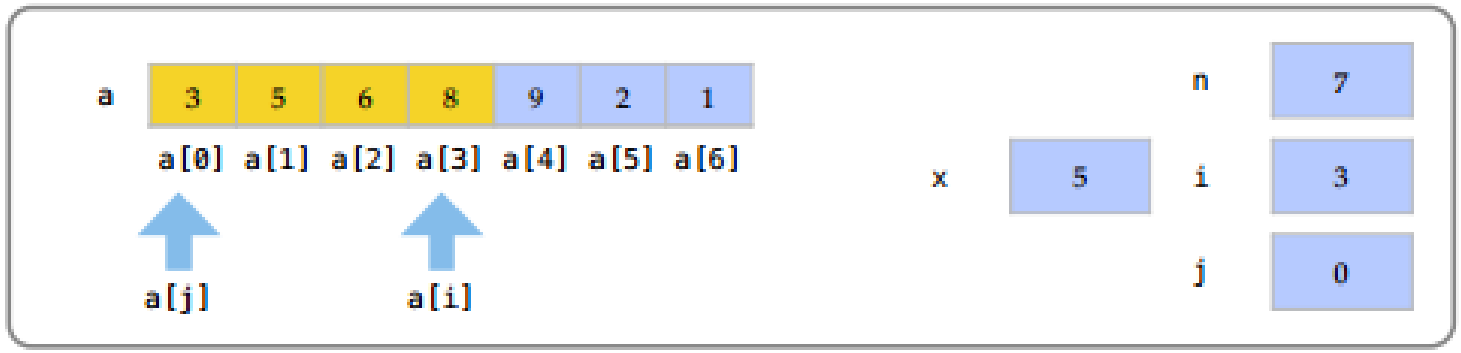
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- O elemento guardado em x é então colocado em sua posição correta

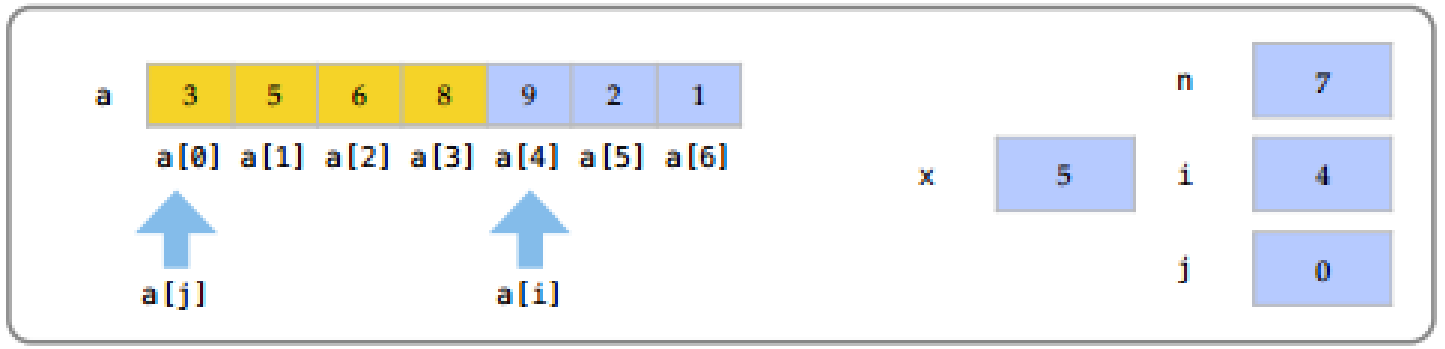
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Sabemos que o arranjo está ordenado até a[3] e incrementamos i para 4

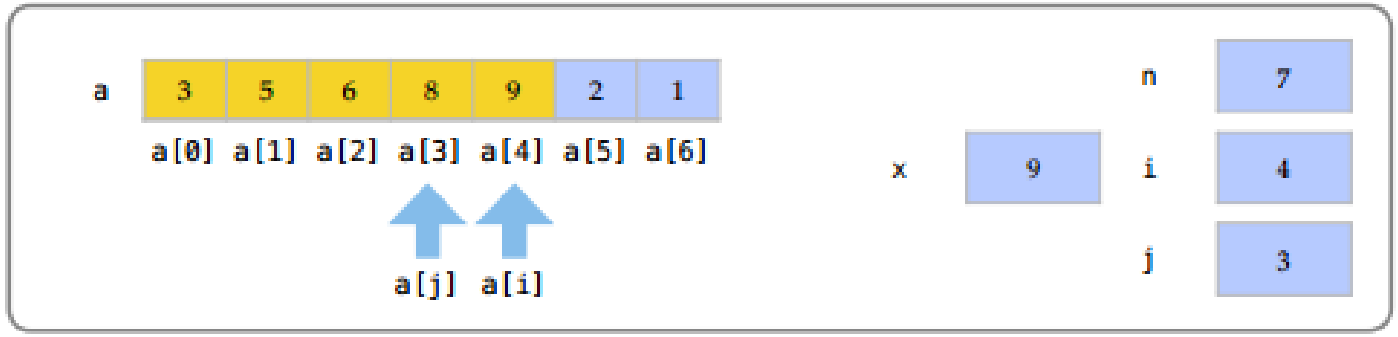
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Como vimos neste trecho de código, elementos maiores que a[i] serão deslocados no arranjo ordenado e a[i] será inserido em sua posição correta

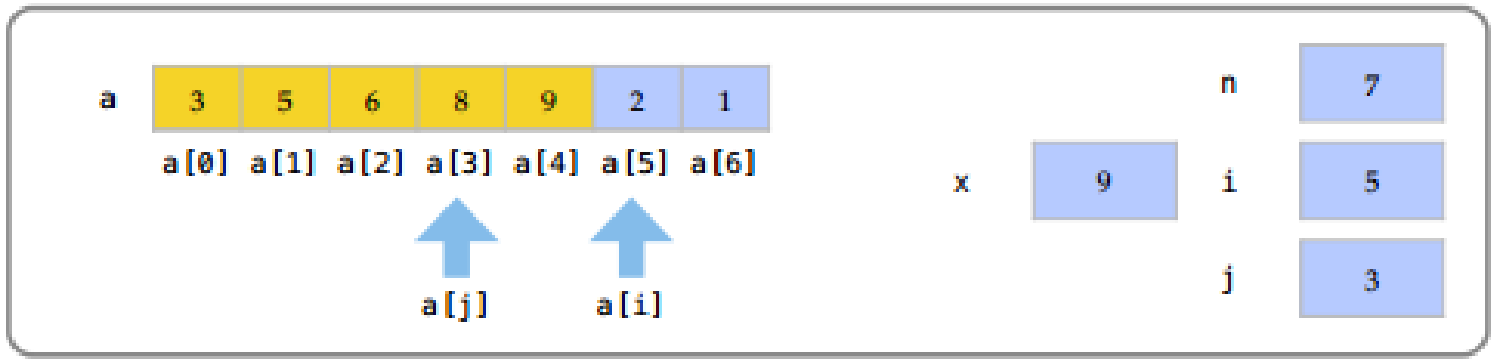
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Sabemos que o arranjo está ordenado até a[4] e incrementamos i para 5

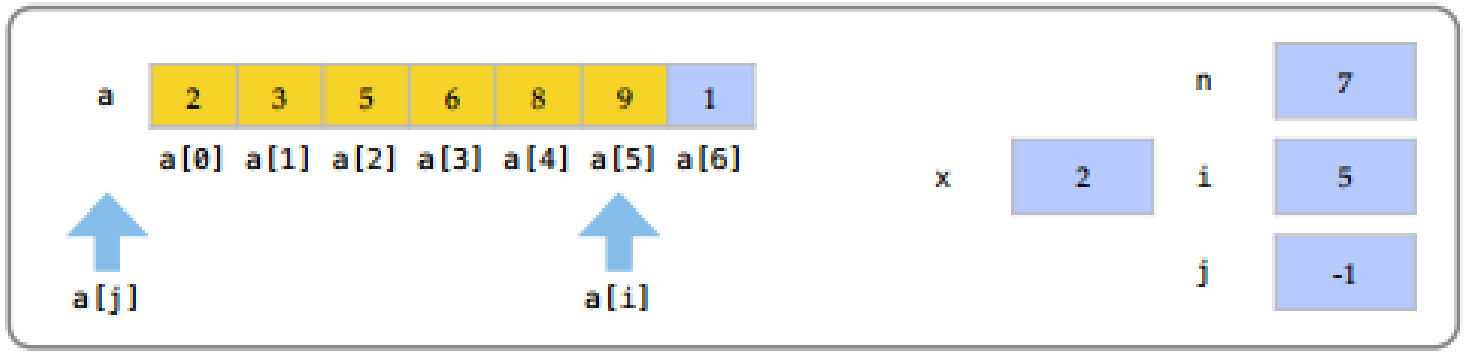
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Elementos maiores que a[i] serão deslocados no arranjo ordenado e a[i] será inserido em sua posição correta

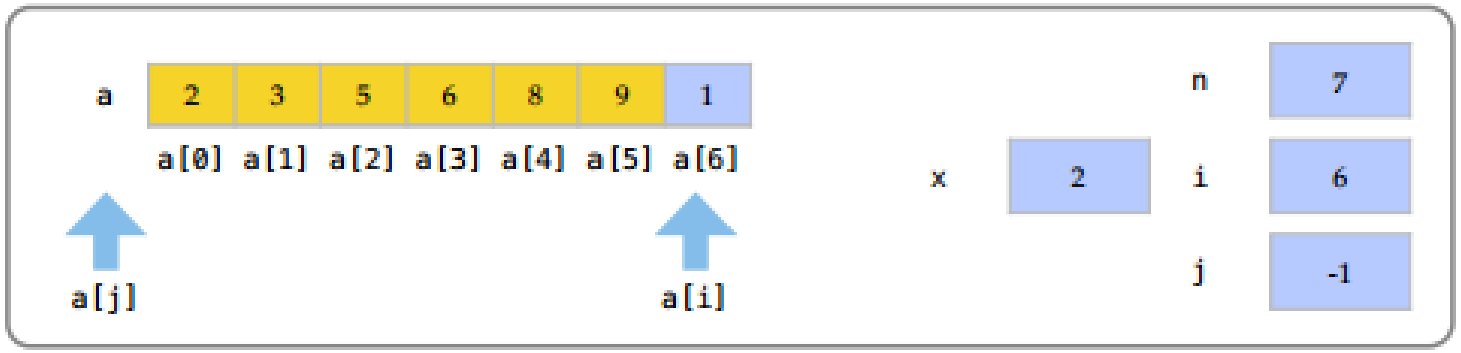
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Sabemos que o arranjo está ordenado até a[5] e incrementamos i para 6

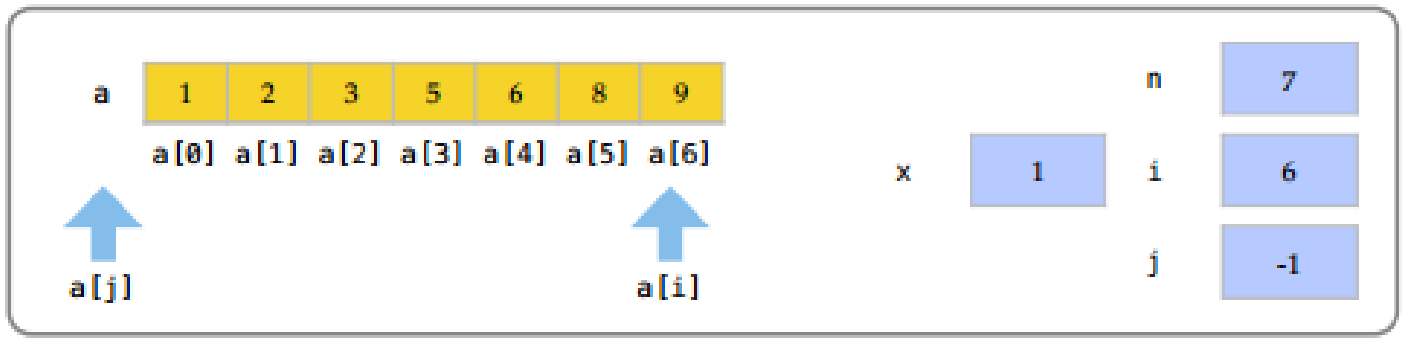
```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Elementos maiores que a[i] serão deslocados no arranjo ordenado e a[i] será inserido em sua posição correta

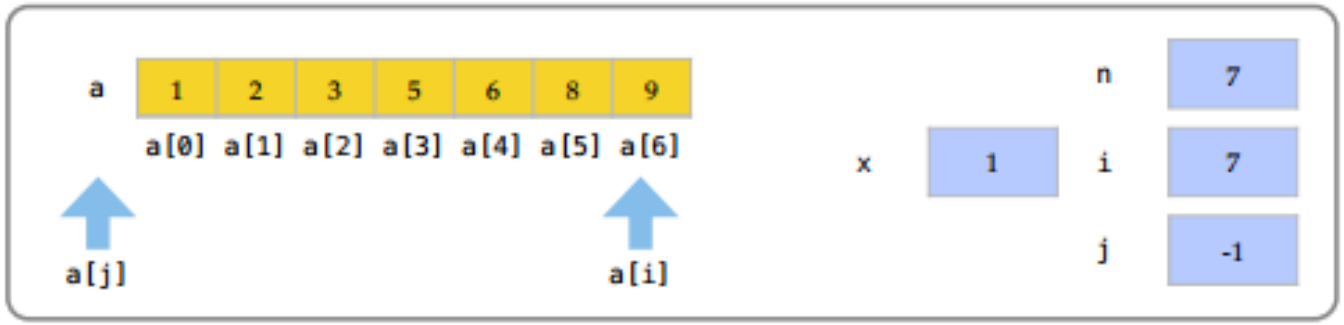
```
void insercao(int a[], int n)
{
    int i, j; // índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção

- Sabemos agora que todo o arranjo está ordenado e o critério de parada é então atingido quando i chega a 7, encerrando a função

```
void insercao(int a[], int n)
{
    int i, j; // Índices
    int x; // elemento
    // para cada posição a partir de i = 1
    for (i = 1; i < n; i++){
        // coloca o elemento na posição correta entre 0 e i - 1
        x = a[i];
        j = i - 1;
        while (x < a[j] && j >= 0){
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```



Ordenação por Inserção - Análise

- Há duas condições que podem terminar o laço mais interno do algoritmo:
  - A posição do elemento já ter sido encontrada
  - Todas as posições já terem sido testadas

Ordenação por Inserção - Análise

- Em relação aos casos possíveis para o laço interno, temos:
  - Melhor caso: quando é feita apenas uma comparação pois o elemento já está sempre na posição correta *O(1)*
  - Pior caso: i comparações são feitas até testarmos todas as posições *i = O(i)*
  - Caso médio: probabilidade 1/i em cada caso:

$$\frac{1}{i}(1+2+3\cdots+i) = \frac{1}{i}\sum_{j=1}^{i} j = \frac{i+1}{2} = O(i)$$

Ordenação por Inserção - Análise

- Como temos n-1 iterações do laço mais externo, temos:
  - Melhor caso:

$$1 + 1 + \cdots + 1 = \sum_{i=2}^{n} 1 = n - 1 = O(n)$$

  - Pior caso:

$$2 + 3 + \cdots + n = \sum_{i=2}^{n} i = \frac{n + n^2 - 2}{2} = O(n^2)$$

  - Caso médio:

$$\frac{3 + 4 + 5 \ldots n + 1}{2} = \sum_{i=2}^{n} \frac{i + 1}{2} = \frac{3n + n^2 - 4}{4} = O(n^2)$$

Ordenação por Inserção - Análise

- No algoritmo de ordenação por inserção, o número de movimentações é proporcional ao número de comparações
  - Isso porque os elementos são movimentados até que uma comparação indique que a posição correta foi encontrada

Ordenação por Inserção - Análise

- Assim, em relação ao número de movimentações, temos também:
  - Melhor caso: $O(n)$
  - Pior caso: $O(n^2)$
  - Caso médio: $O(n^2)$

Ordenação por Inserção – Análise - **Vantagens**

- O algoritmo tem um melhor caso que ocorre quando os elementos já estão ordenados
    - Uma ordenação prévia é aproveitada pelo algoritmo. Dizemos que o método é adaptável
- É um bom método para se adicionar alguns poucos elementos a um arranjo ordenado, pois terá um custo baixo em um arranjo “quase ordenado”
- Este algoritmo de ordenação é estável, pois a ordem relativa dos elementos é mantida no deslocamento dos elementos

Ordenação por Inserção – Análise - **Vantagens**

- Apesar de pouco provável na maior parte das aplicações práticas, o custo máximo do algoritmo ocorre quando os elementos estão em ordem reversa

- Se o número de movimentações é o fator mais relevante, ele se torna ineficiente em relação à ordenação por seleção

Bolha X Seleção X Inserção

Algoritmo	Bolha	Seleção	Inserção
Pior caso	$O(n^2)$	$O(n^2)$	$O(n^2)$
Melhor caso	$O(n^2)$	$O(n^2)$	$O(n)$
Caso médio	**$O(n^2)$**	**$O(n^2)$**	**$O(n^2)$**
Estabilidade	Sim	Não	Sim
Adaptabilidade	Não	Não	Sim
Movimentações	$O(n^2)$	$O(n)$	Mesmo que comp.

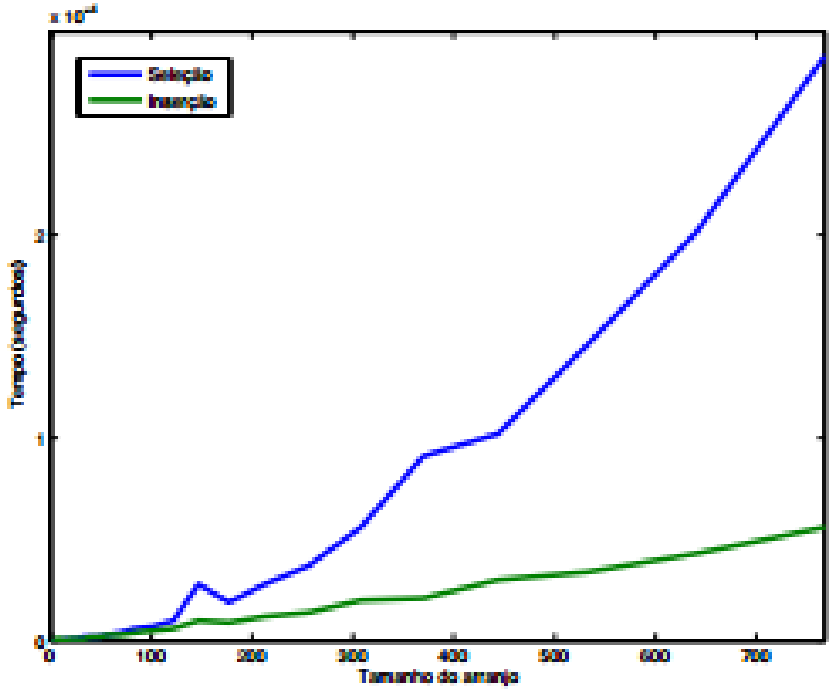
Testes Reais: Seleção x Inserção

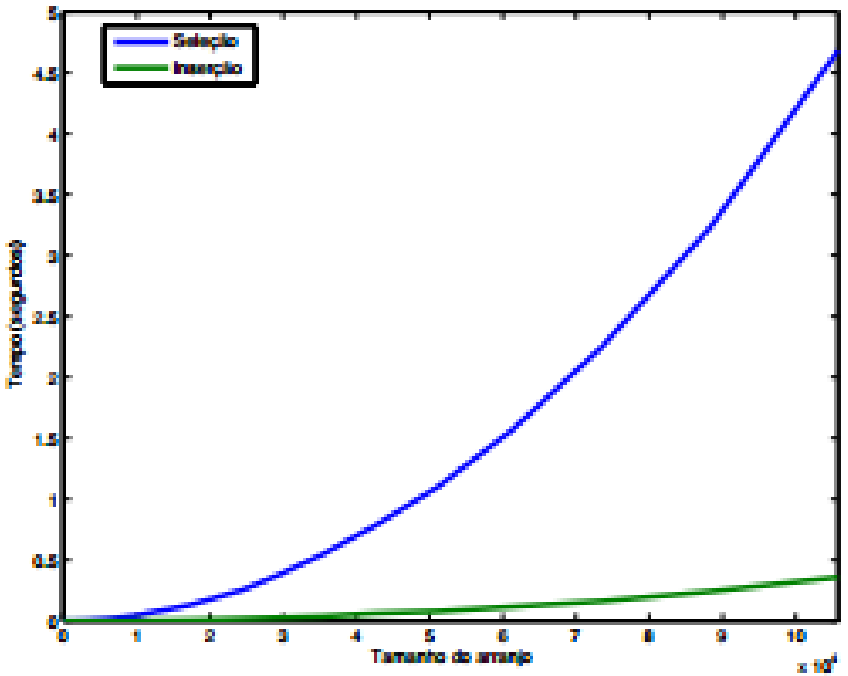
- Como os dois algoritmos são $O(n^2)$ no caso médio, podemos fazer comparações reais de tempo para ver a performance em segundos dos algoritmos

- Para as comparações apresentadas a seguir, foram utilizadas versões mais eficientes dos algoritmos

Testes Reais

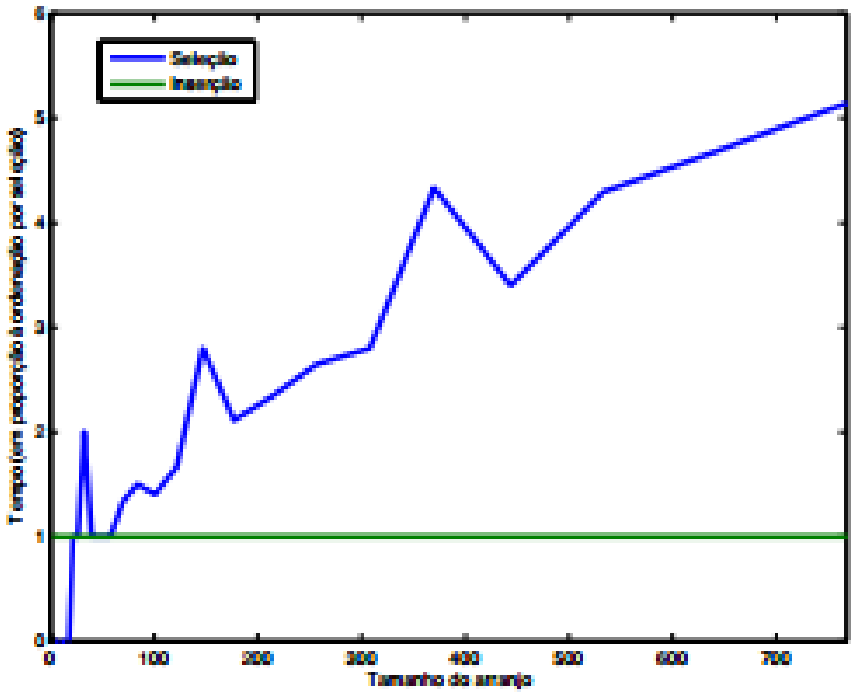
- Arranjos em ordem inicial aleatória

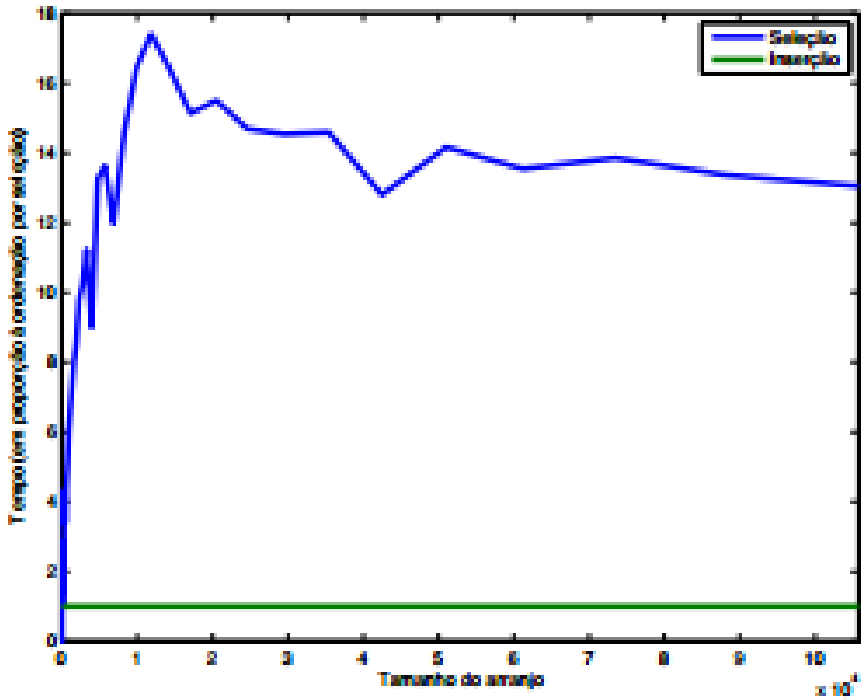




Testes Reais

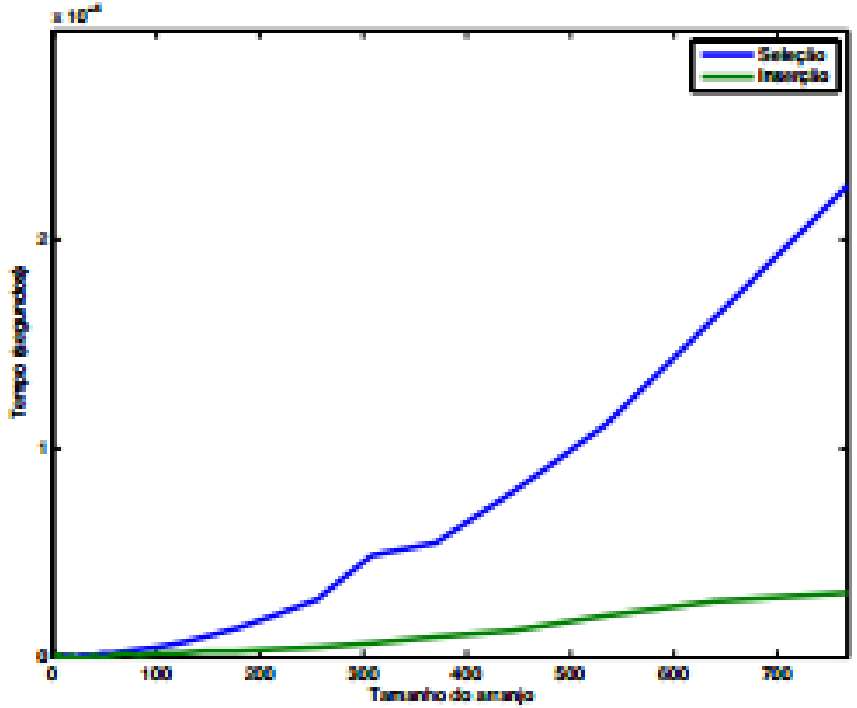
- Em proporção, a seleção chega a ser quase 18 vezes mais lenta que a inserção

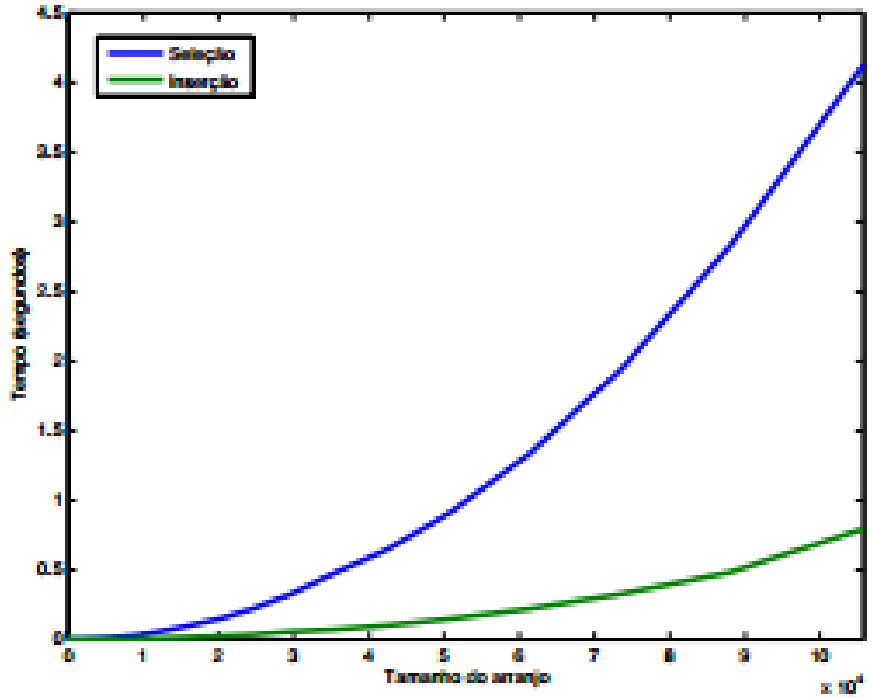




Testes Reais

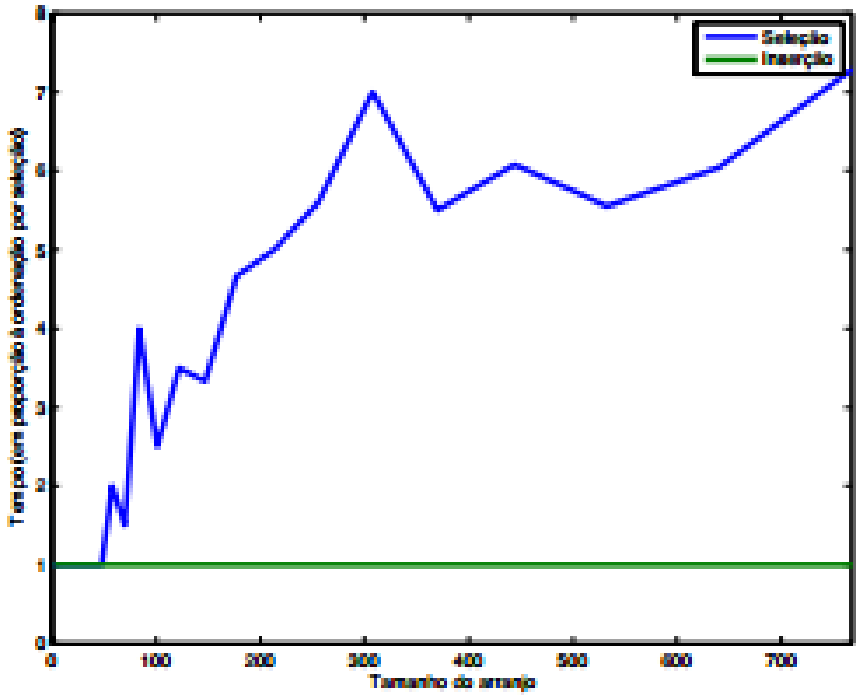
- Arranjos em ordem inicial decrescente (pior caso da inserção)

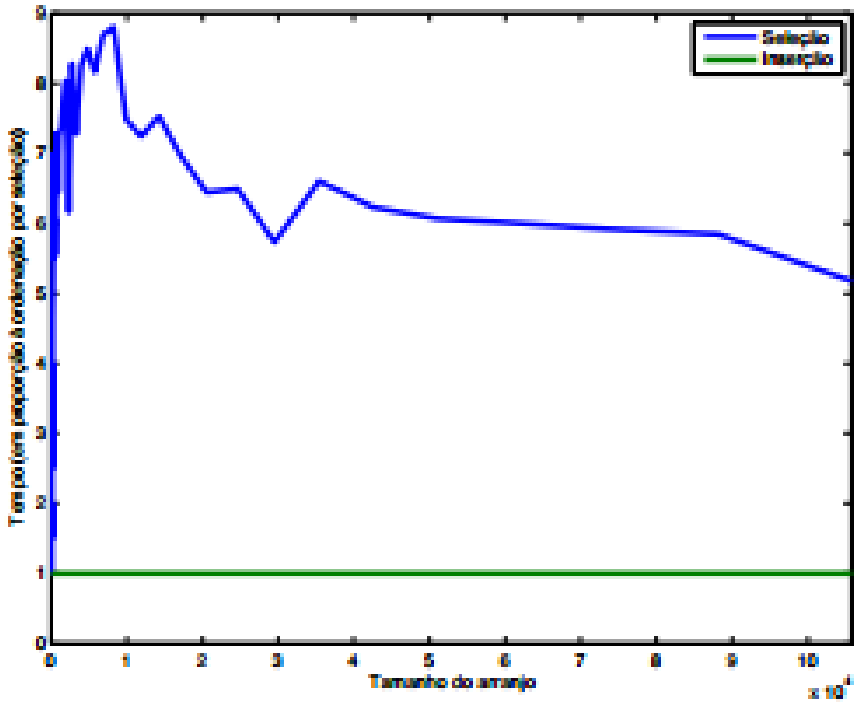




Testes Reais

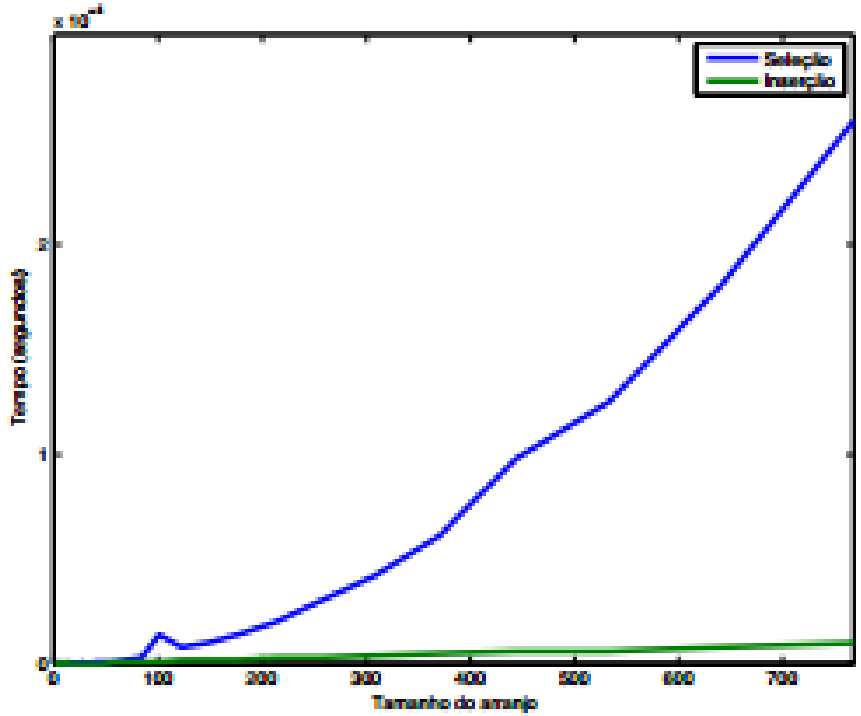
- Em proporção, a seleção chega a ser quase 9 vezes mais lenta que a inserção, mesmo em seu pior caso

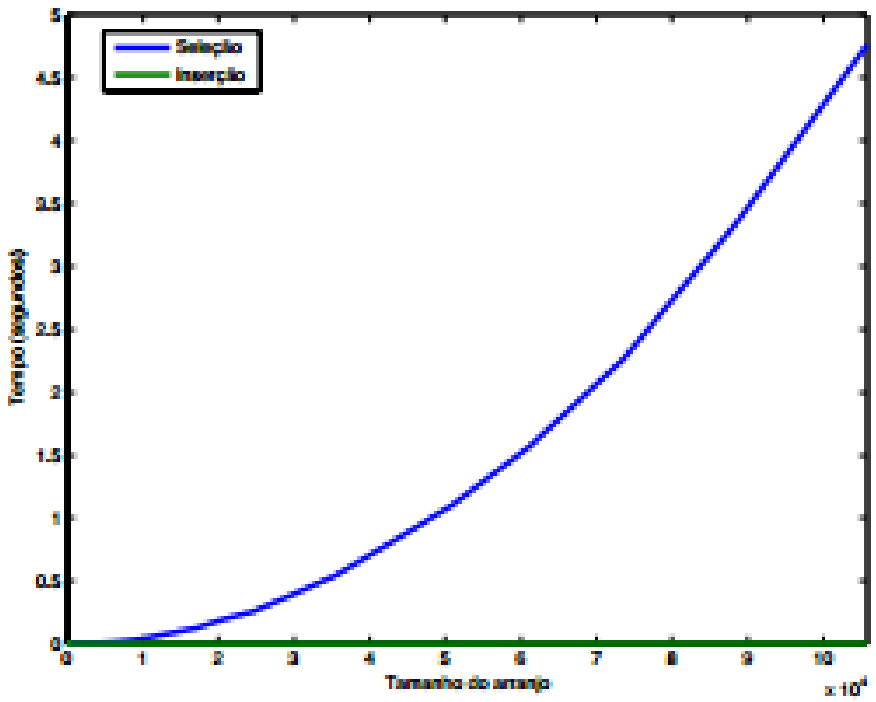




Testes Reais

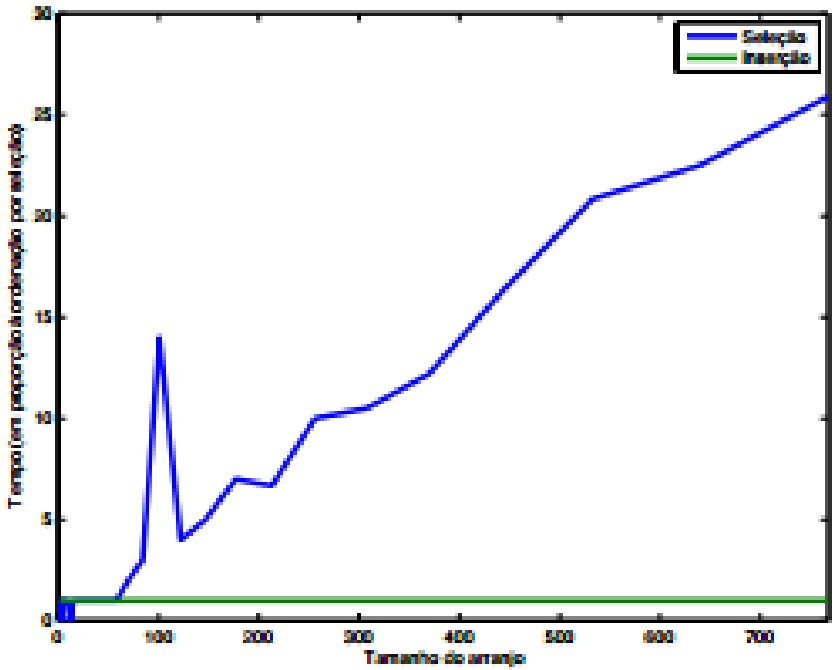
- Arranjos em ordem inicial crescente (melhor caso da inserção)

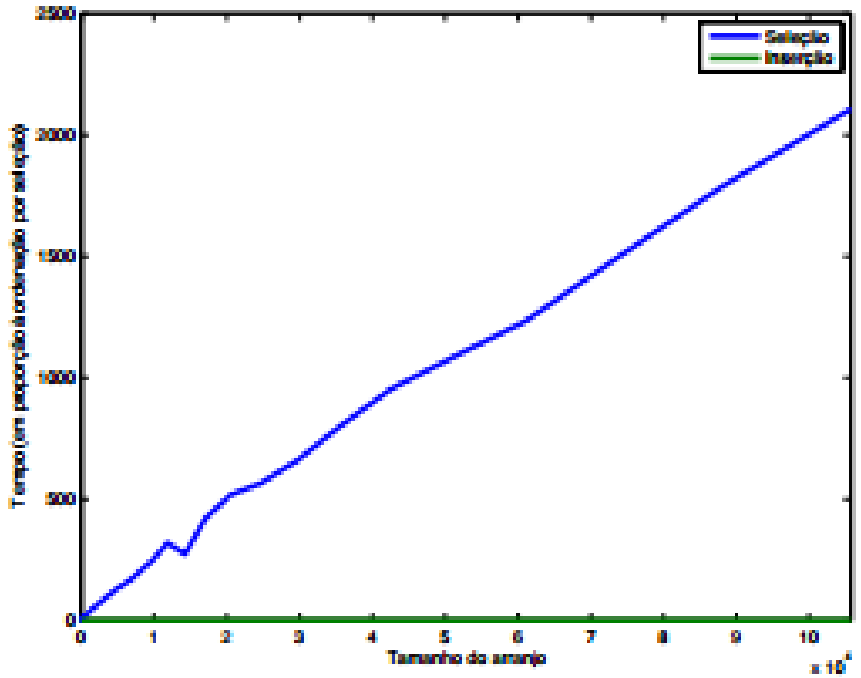




Testes Reais

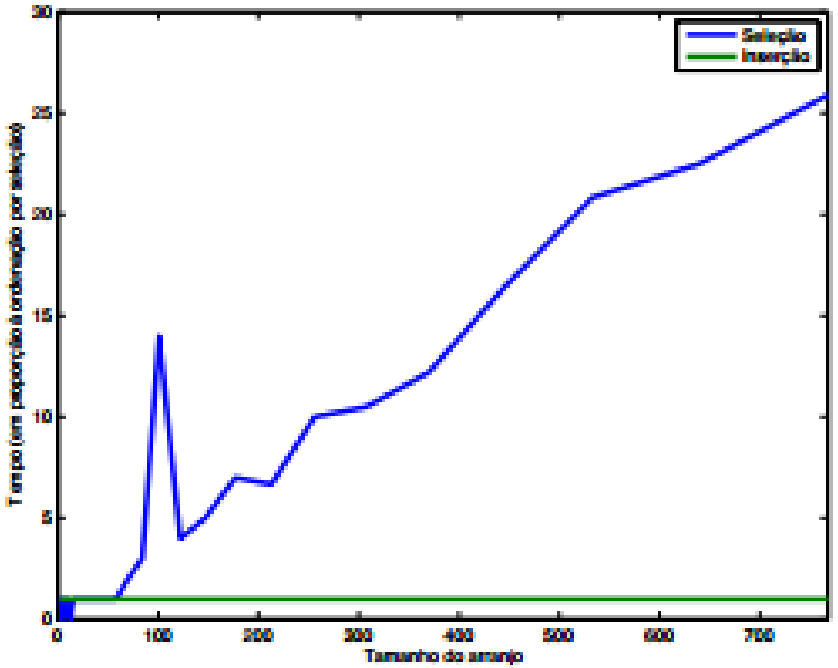
- Em proporção, a seleção chega a ser quase 2500 vezes mais lenta que a inserção, em seu melhor caso

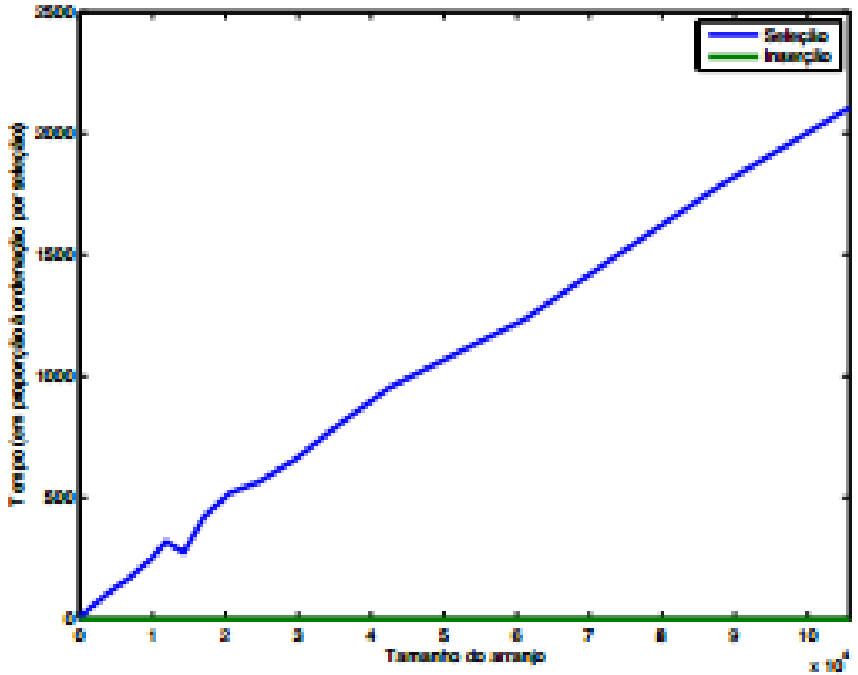




Testes Reais

- Como o melhor caso da inserção é O(n), em comparação a O(n^2) seleção, isto era esperado





Vídeos de apresentação dos algoritmos de ordenação simples

- Bolha
  - https://www.youtube.com/watch?v=lyZQPjUT5B4
- Seleção
  - https://www.youtube.com/watch?v=Ns4TPTC8whw
- Inserção
  - https://www.youtube.com/watch?v=ROalU379l3U

Exercício

- Criar um arranjo com 10 elementos aleatórios entre 1 e 50. Observação, crie um arranjo desordenado.

- Desenhar o arranjo várias vezes demonstrando os passos de uma ordenação por:

  - Bolha
  - Seleção
  - Inserção

- Colocar um círculo nos elementos movimentados. Colocar um traço entre os elementos ordenados e desordenados

Algoritmos e Estruturas de Dados II

- Bibliografia:
  - Básica:
    - CORMEN, Thomas, RIVEST, Ronald, STEIN, Clifford, LEISERSON, Charles. Algoritmos. Rio de Janeiro: Elsevier, 2002.
    - EDELWEISS, Nina, GALANTE, Renata. Estruturas de dados. Porto Alegre: Bookman. 2009. (Série livros didáticos informática UFRGS,18).
    - ZIVIANI, Nívio. Projeto de algoritmos com implementação em Pascal e C. São Paulo: Cengage Learning, 2010.
  - Complementar:
    - ASCENCIO, Ana C. G. Estrutura de dados. São Paulo: Pearson, 2011. ISBN: 9788576058816.
    - PINTO, W.S. Introdução ao desenvolvimento de algoritmos e estrutura de dados. São Paulo: Érica, 1990.
    - PREISS, Bruno. Estruturas de dados e algoritmos. Rio de Janeiro: Campus, 2000.
    - TENEMBAUM. Aaron M. Estruturas de dados usando C. São Paulo: Makron Books. 1995. 884 p. ISBN: 8534603480.
    - VELOSO, Paulo A. S. Complexidade de algoritmos: análise, projeto e métodos. Porto Alegre, RS: Sagra Luzzatto, 2001

Algoritmos e Estruturas de Dados II