Ordenação: algoritmos elementares

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

Campus Quixadá

2º semestre/2025

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.

- Um vetor pode ser ordenado de muitas maneiras diferentes: algumas elementares, outras mais sofisticadas e eficientes.

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.

- Um vetor pode ser ordenado de muitas maneiras diferentes: algumas elementares, outras mais sofisticadas e eficientes.

- Pode-se usar basicamente duas estratégias para ordenar os dados:

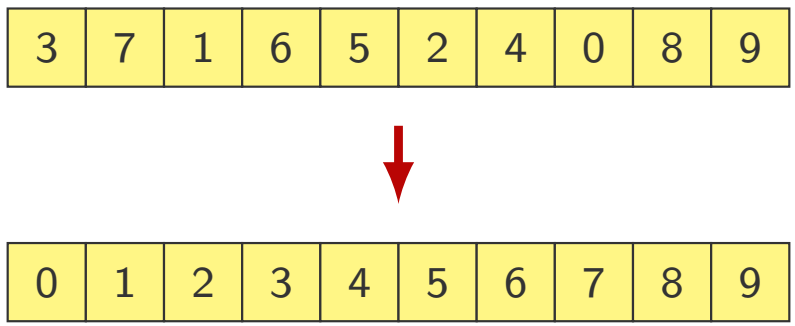
(1) inserir os dados na estrutura respeitando sua ordem.

(2) a partir de um conjunto de dados já criado, aplicar um algoritmo para ordenar seus elementos.

Ordenação

O problema da ordenação de um vetor consiste em:

- rearranjar os elementos de um vetor $A[0 \ldots n-1]$ de tal modo que ele se torne crescente, ou seja, de modo que $A[0] \leq \cdots \leq A[n-1]$.

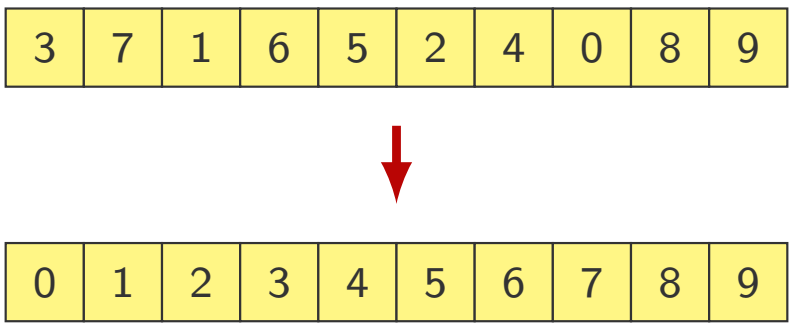


Ordenação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O problema da ordenação de um vetor consiste em:

- rearranjar os elementos de um vetor $A[0 \ldots n-1]$ de tal modo que ele se torne crescente, ou seja, de modo que $A[0] \leq \cdots \leq A[n-1]$.



Nos códigos vamos ordenar vetores de `int`

- Mas é fácil alterar para comparar `double` ou `string`
- ou comparar `struct` por algum de seus campos
  - O valor usado para a ordenação é a *chave* de ordenação
  - Podemos até desempatar por outros campos

BubbleSort

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele

Estado inicial: [7,6,5,4,1]

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele

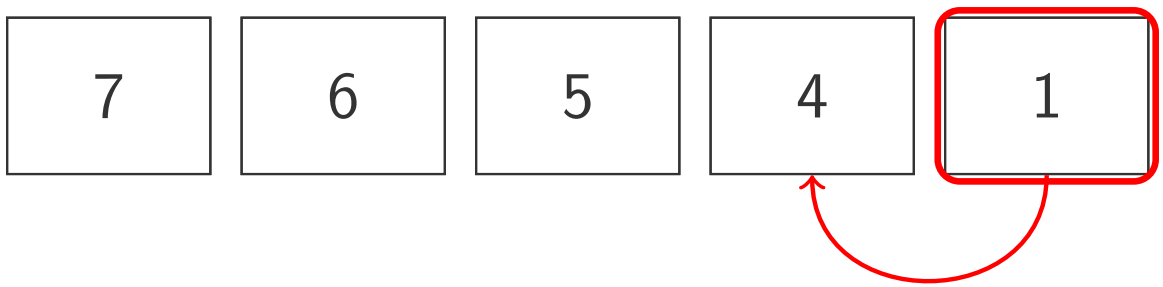
Destaque da bolha (1)

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele



Comparando $1 \rightarrow 4$

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele

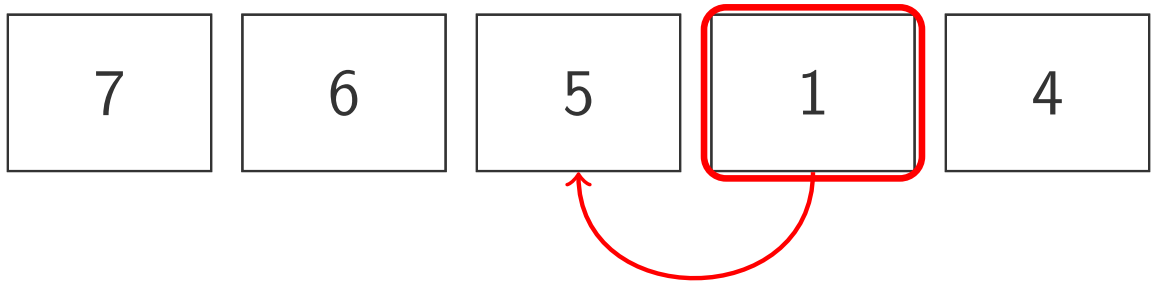
Troca realizada: [7,6,5,1,4]

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele



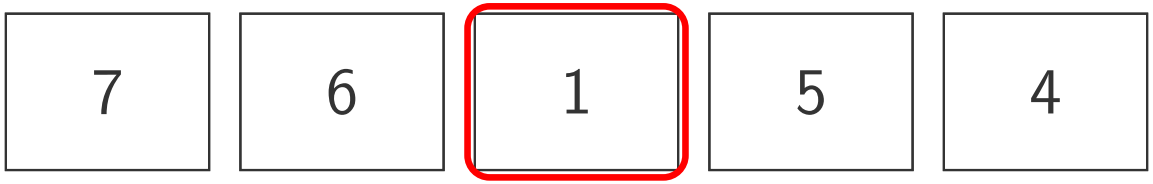
Comparando $1 \rightarrow 5$

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele



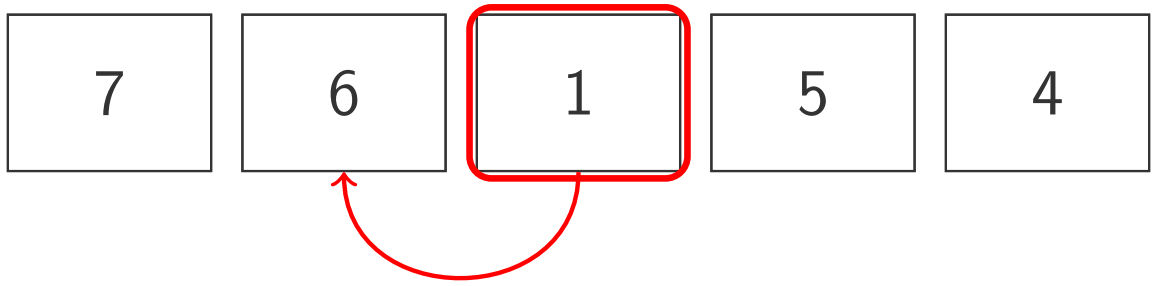
Troca realizada: [7,6,1,5,4]

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele



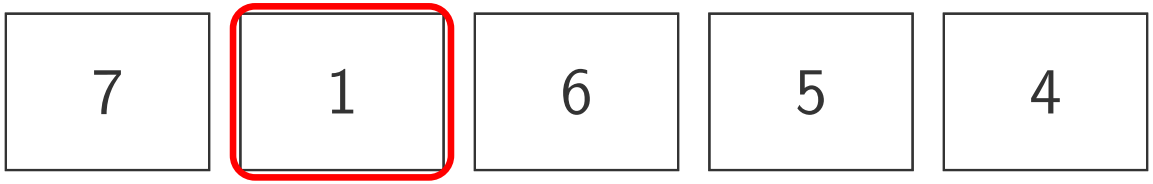
Comparando $1 \rightarrow 6$

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele



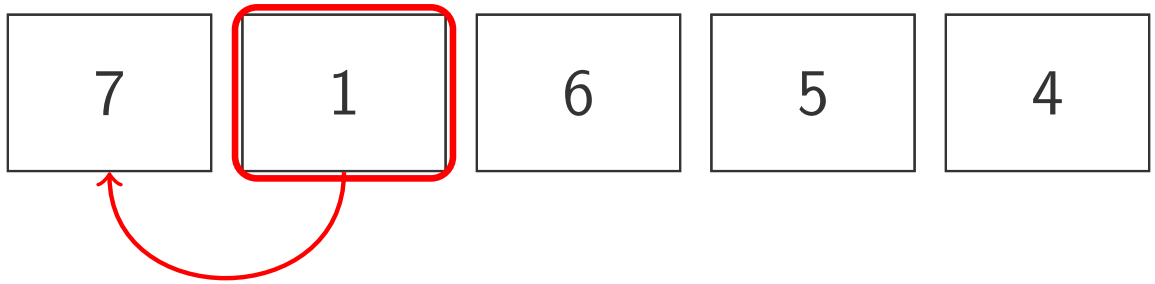
Troca realizada: [7,1,6,5,4]

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele



Comparando $1 \rightarrow 7$

BubbleSort — Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

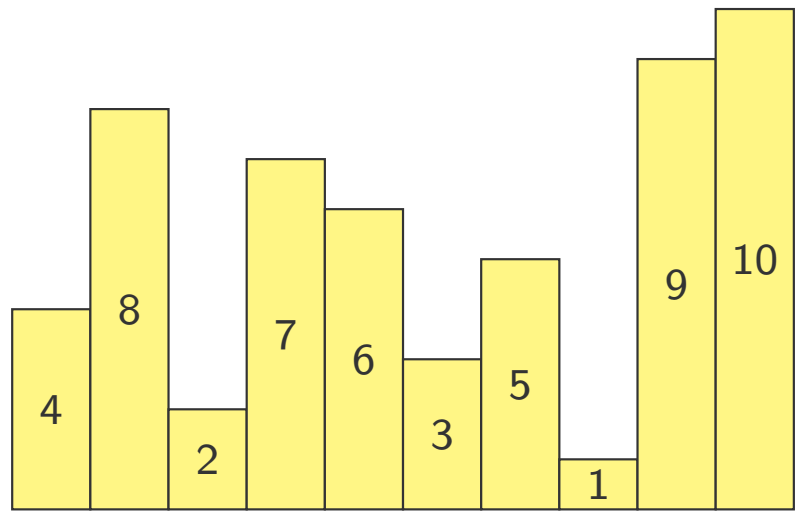
Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento. Ele será trocado com todos os elementos que estiverem antes dele

Troca final: [1,7,6,5,4]

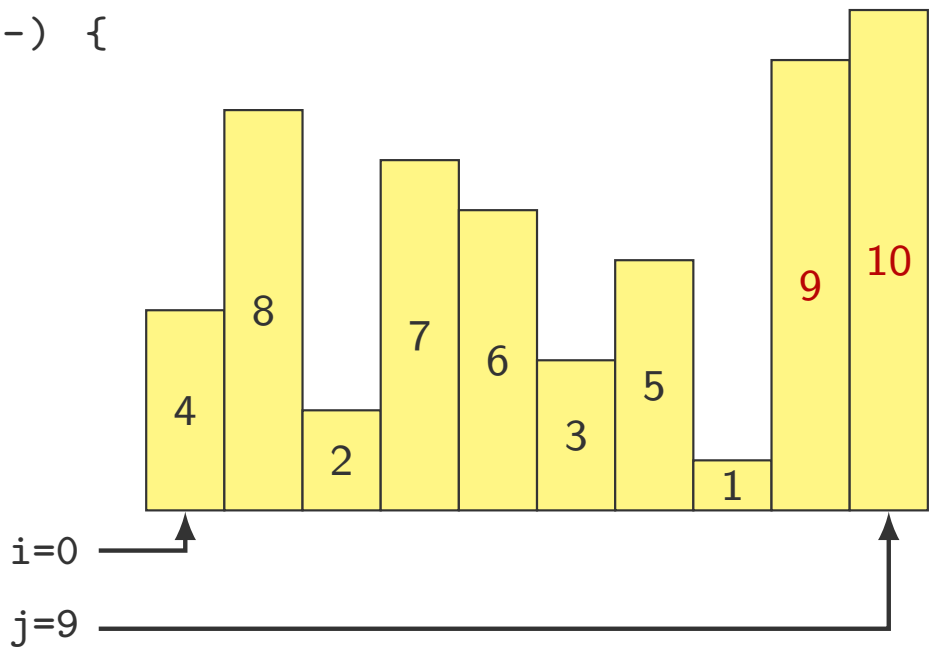
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



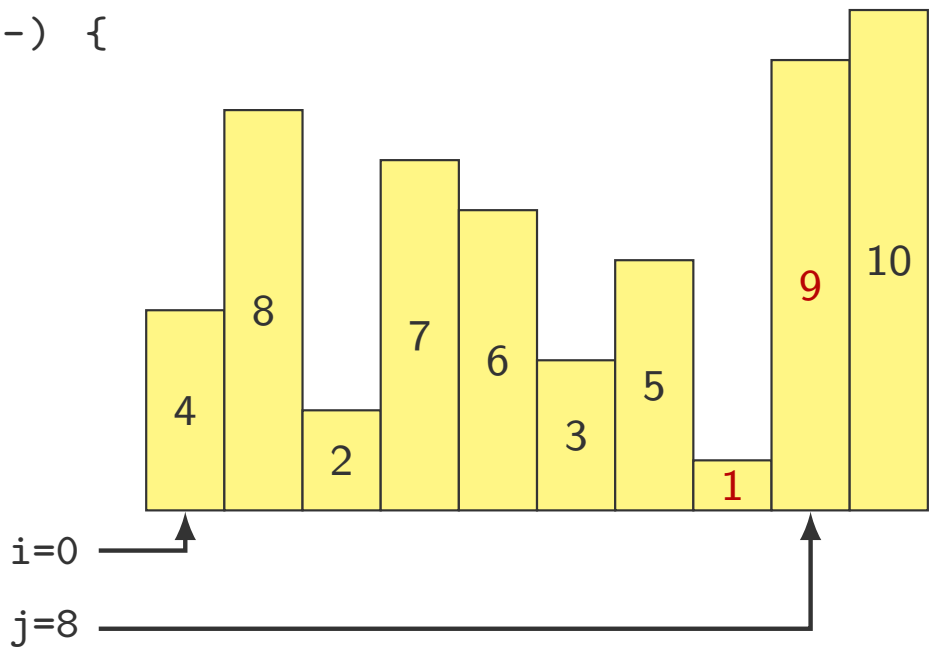
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



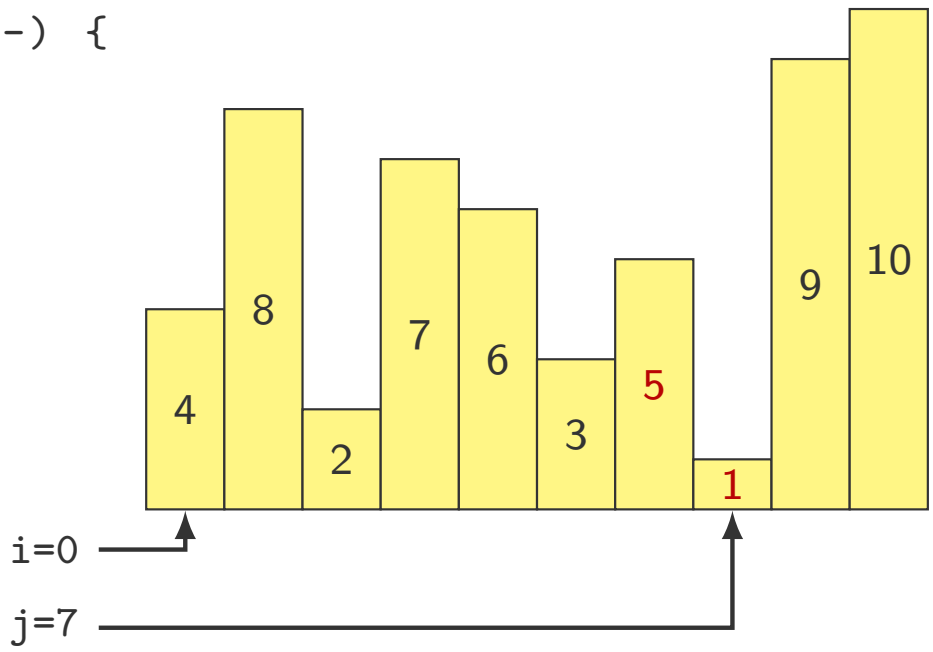
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



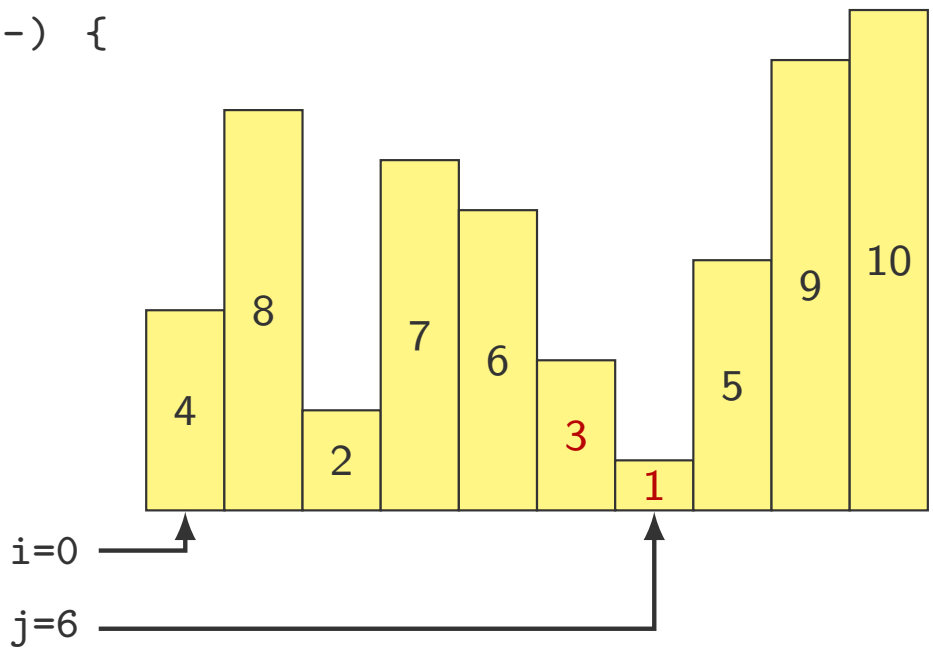
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



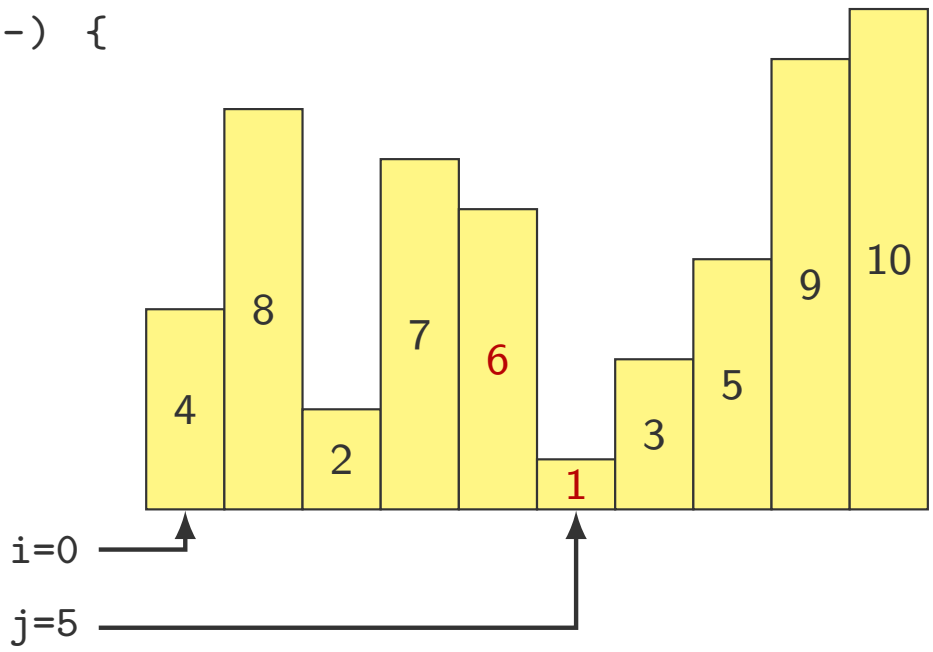
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



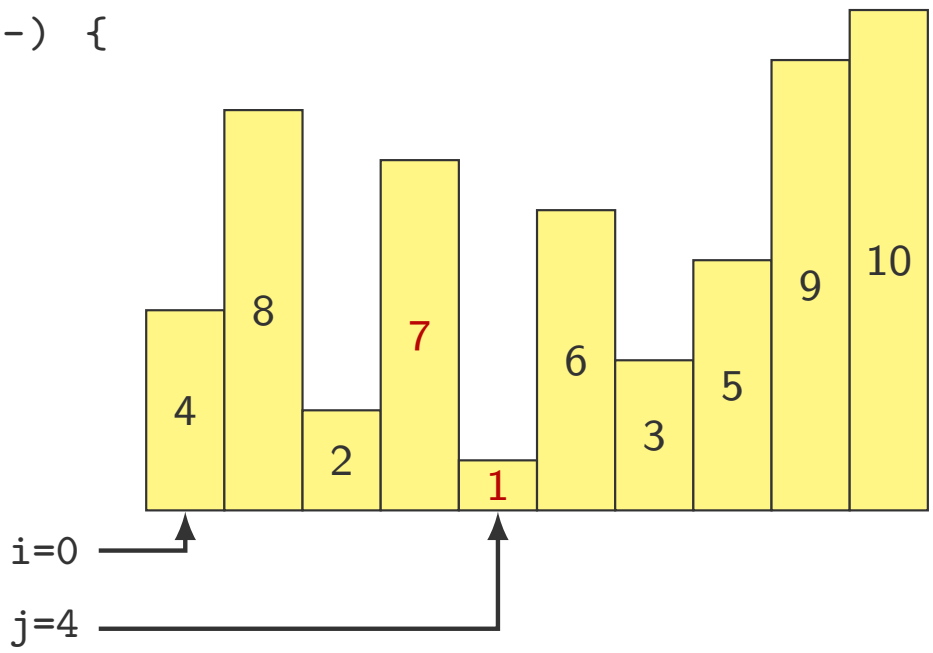
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



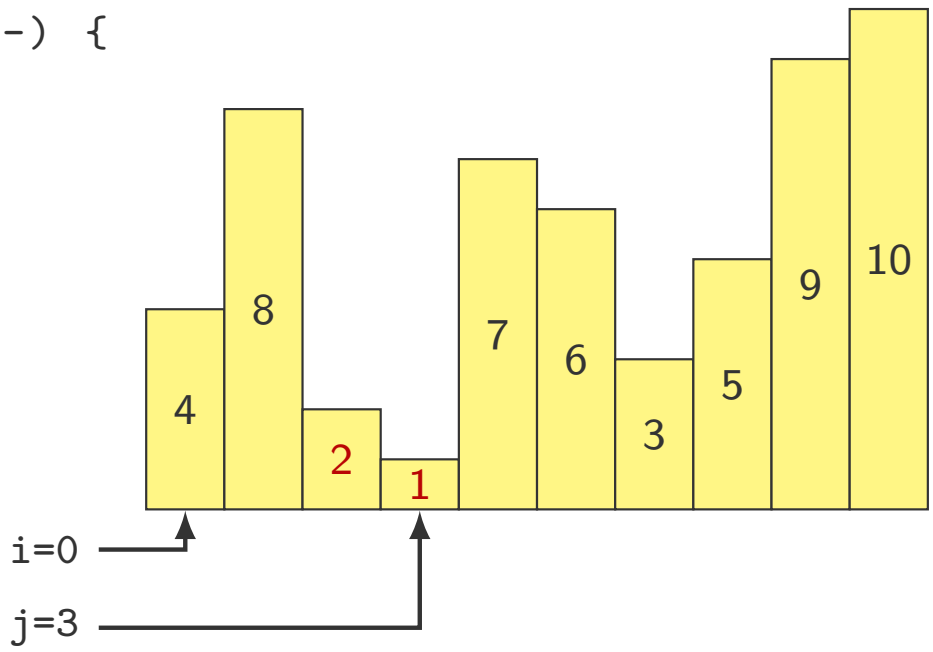
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



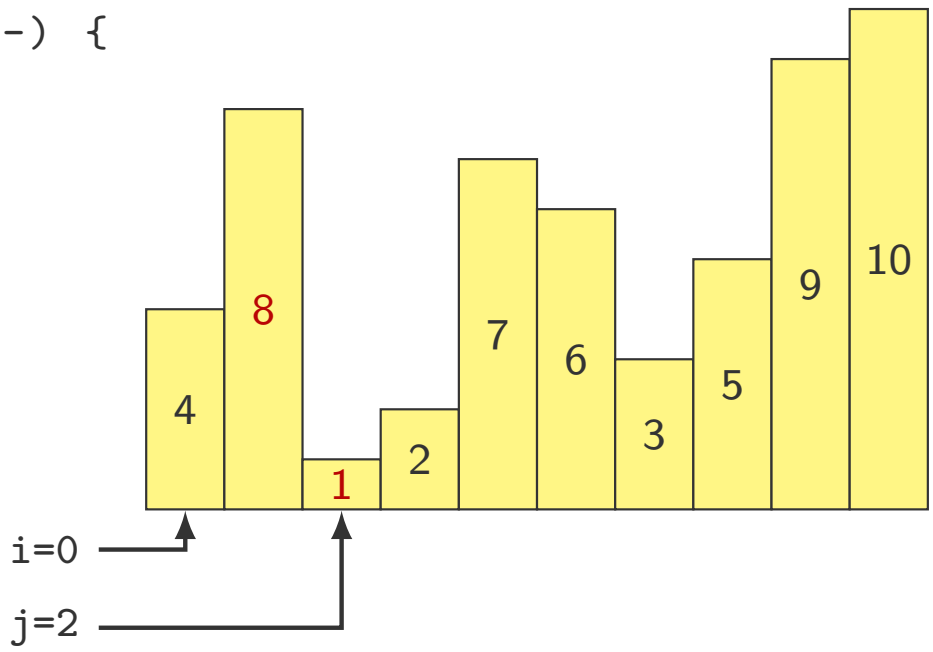
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



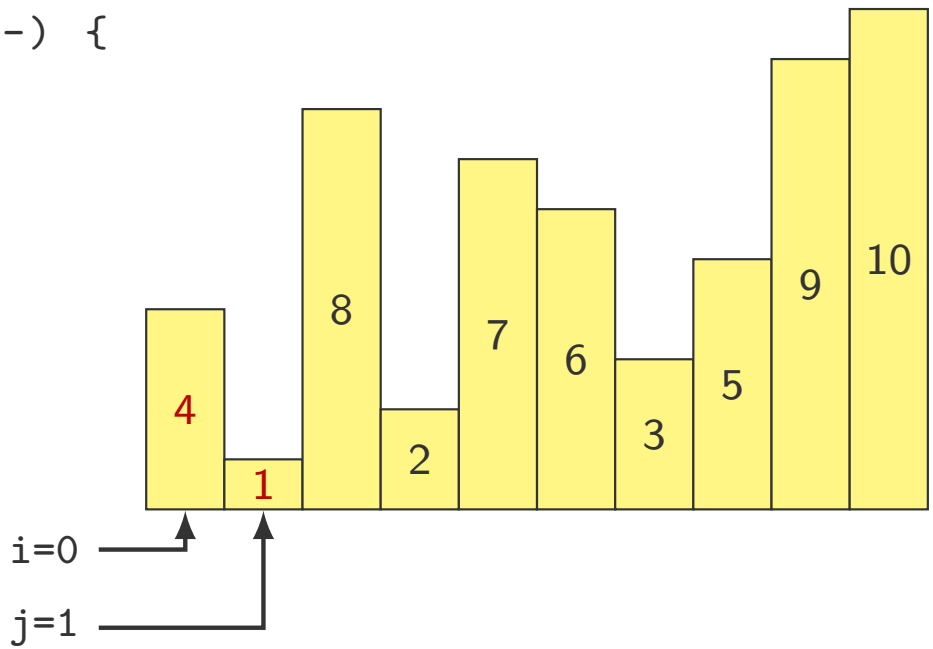
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



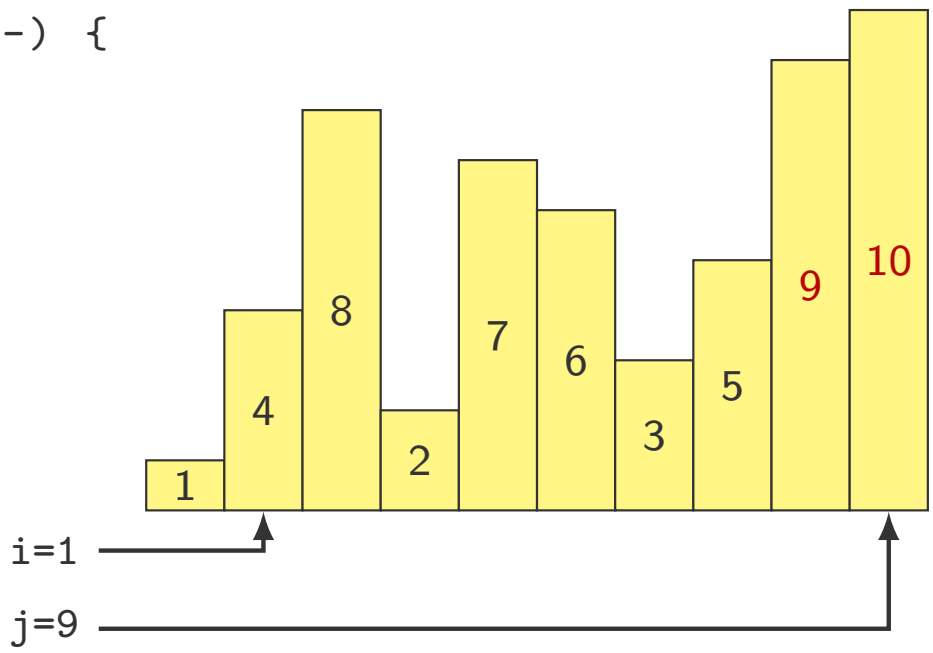
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



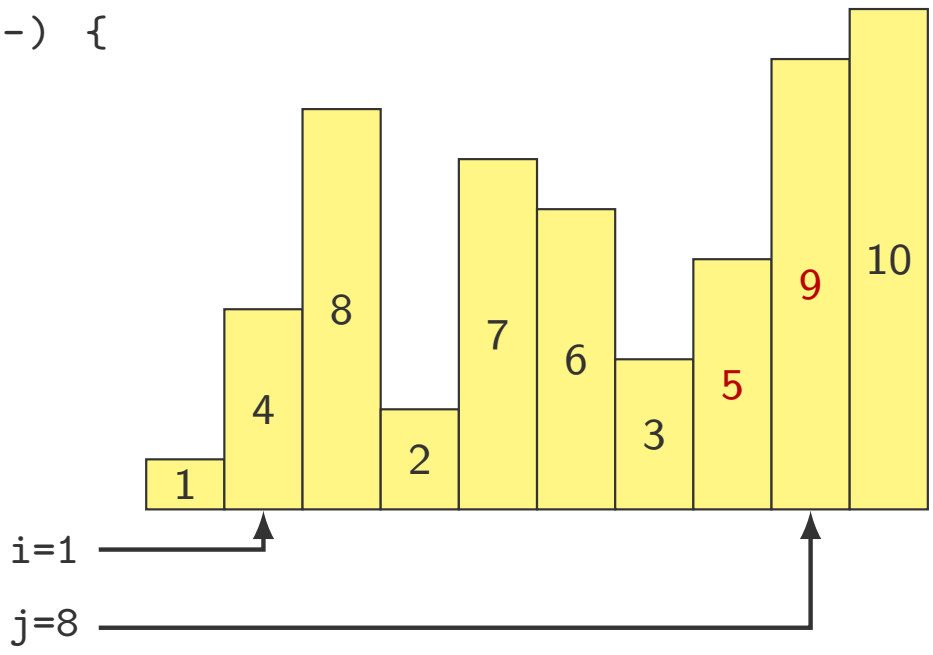
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



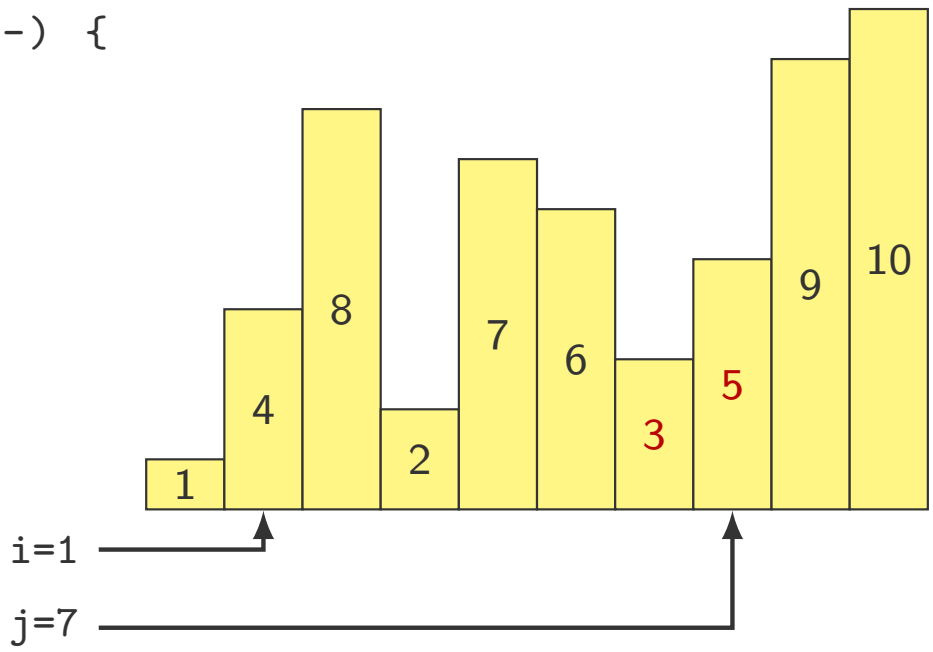
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



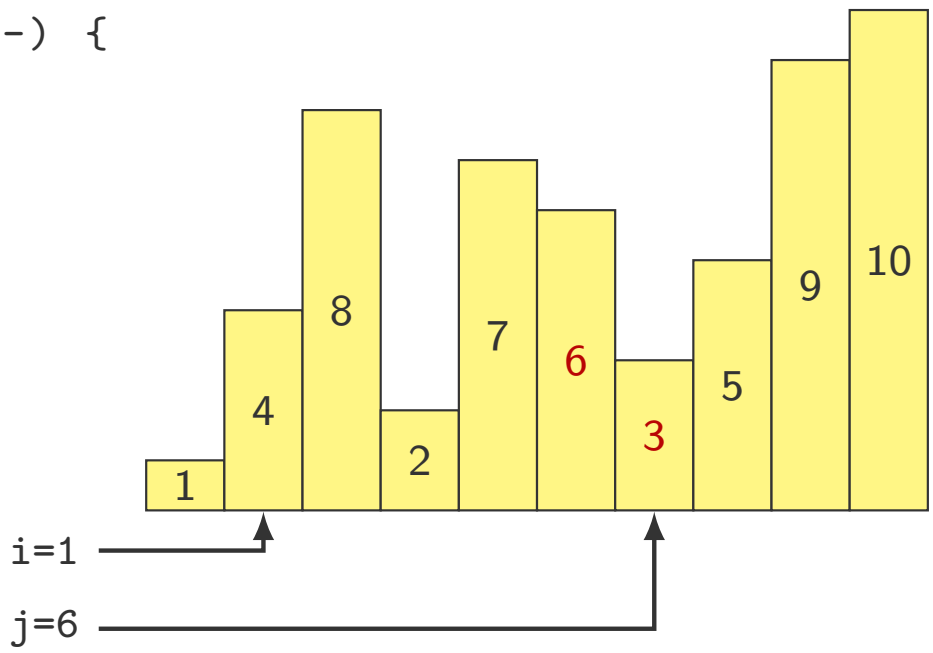
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



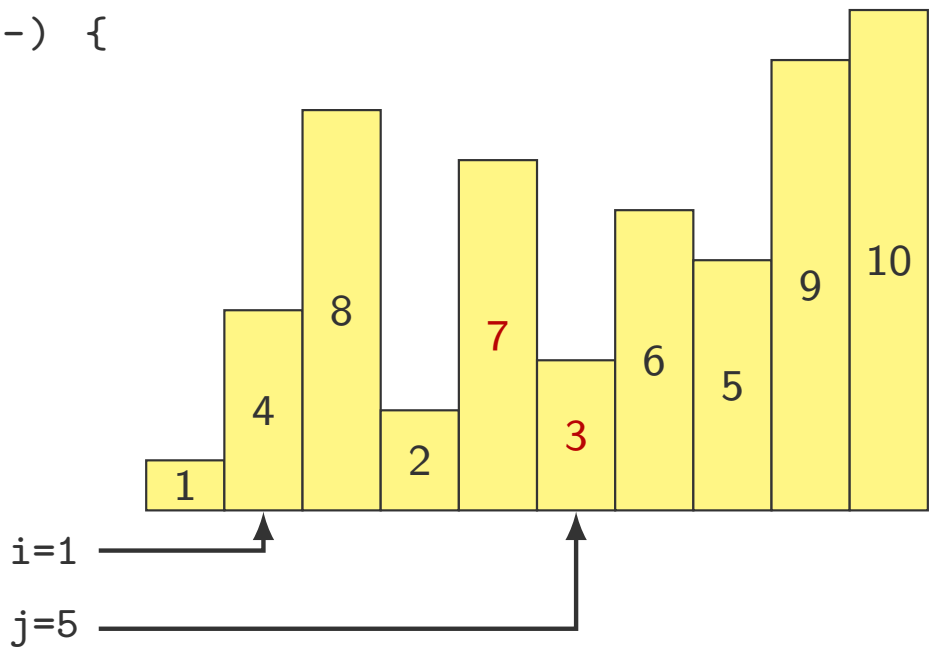
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



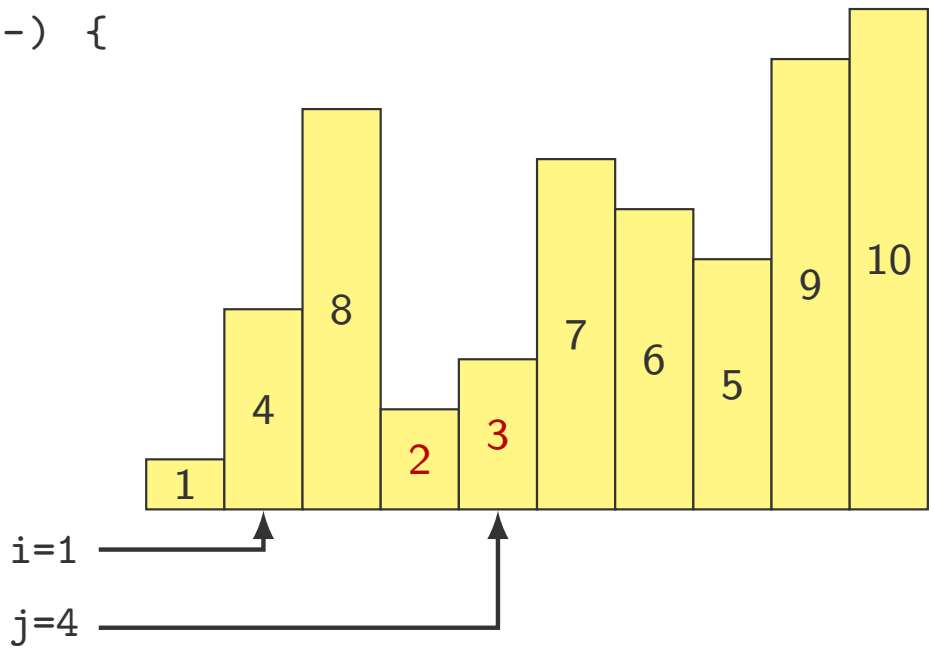
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



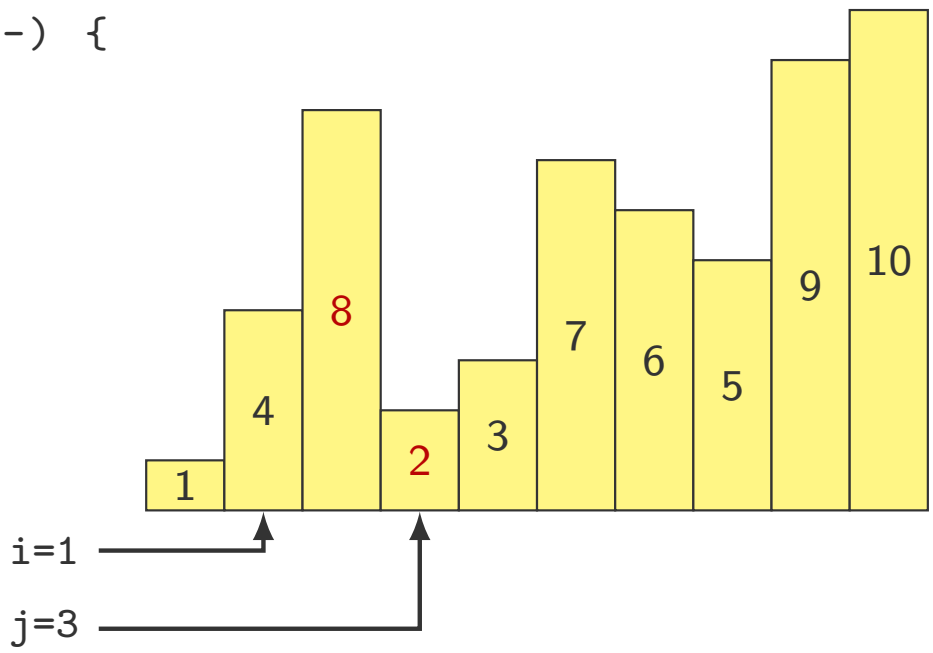
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



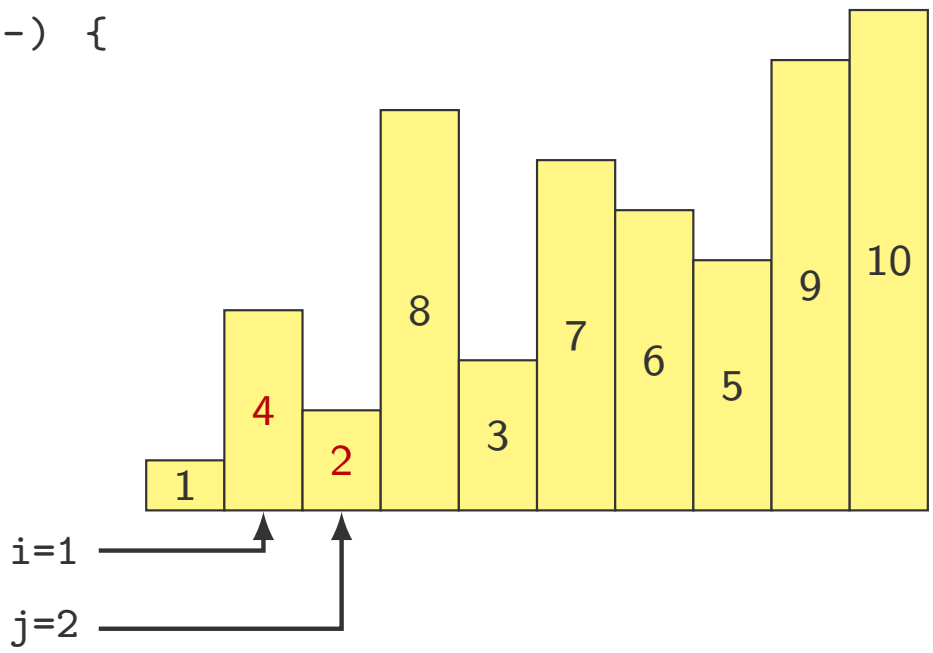
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



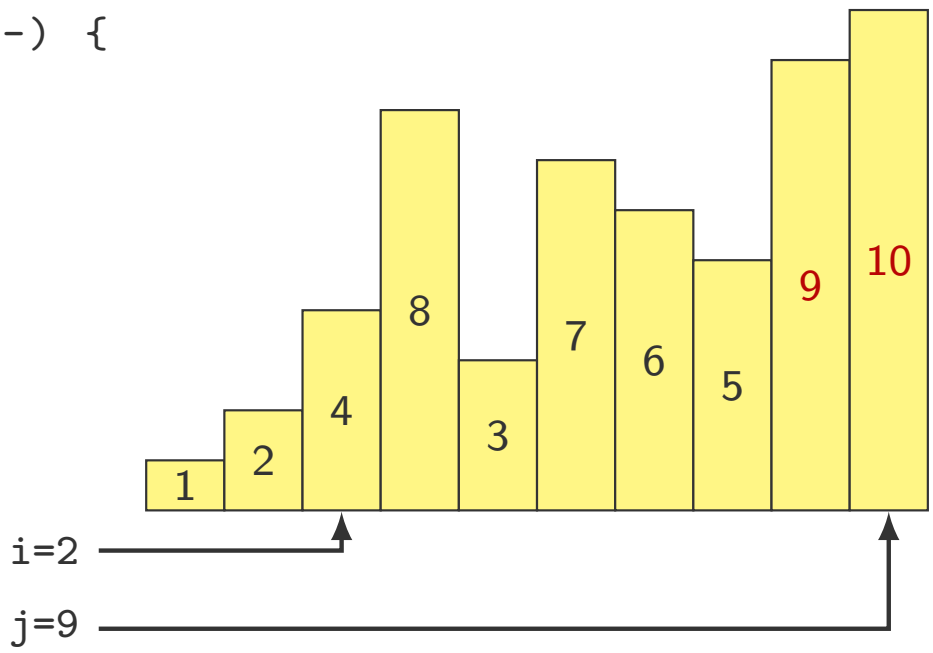
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



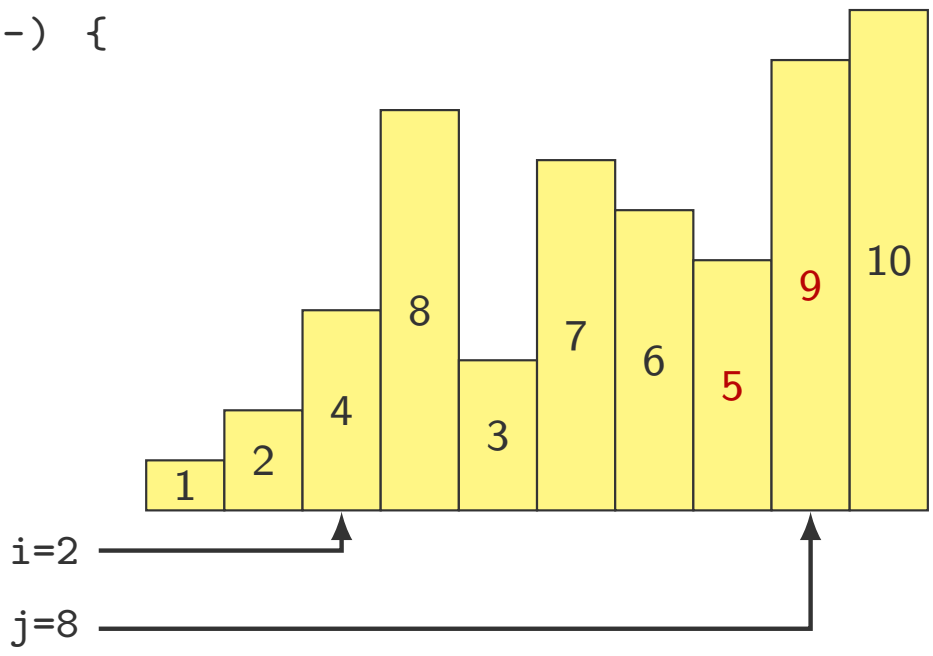
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



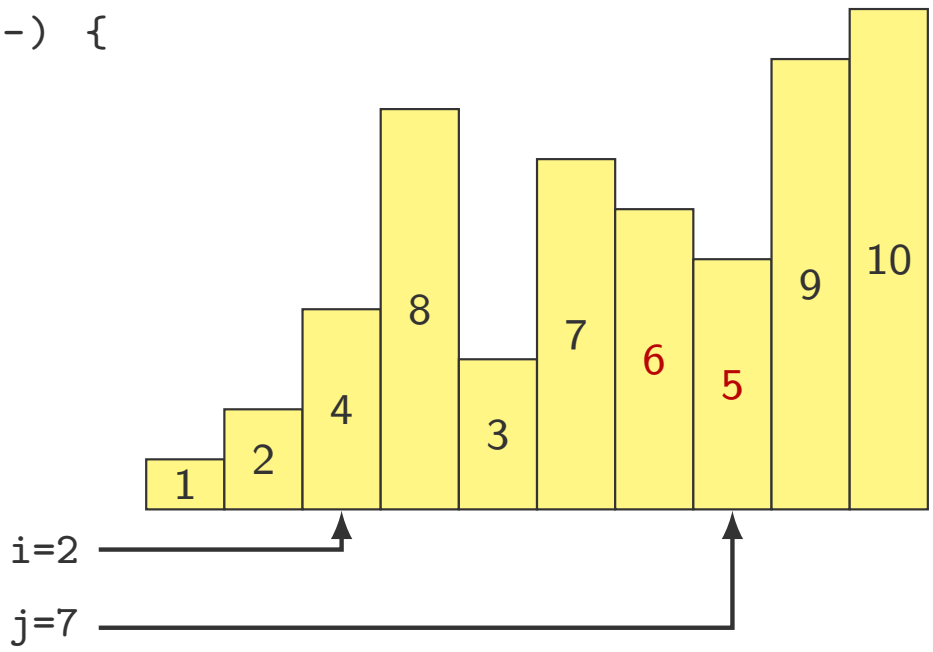
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



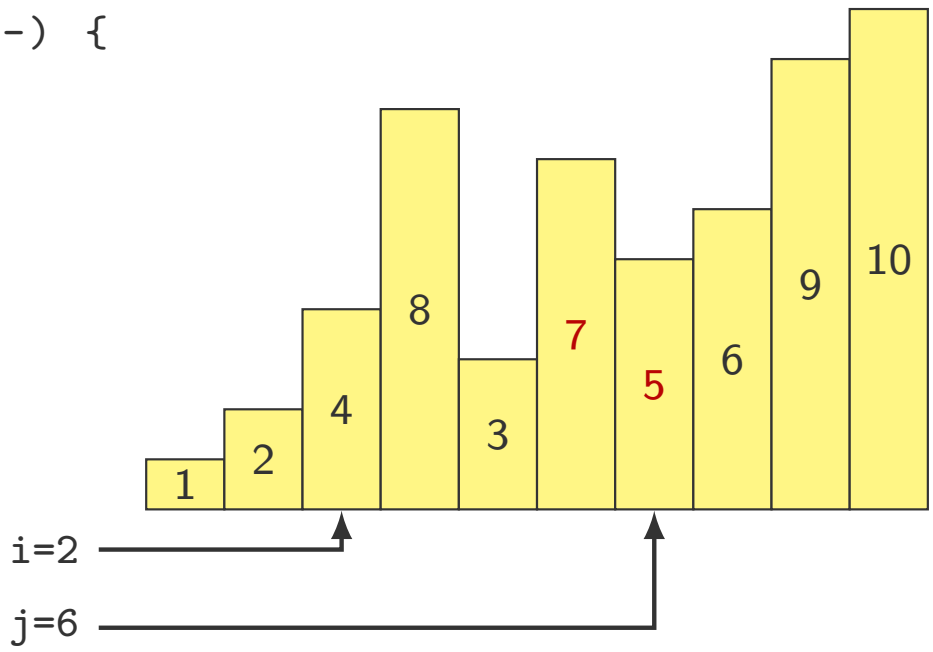
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



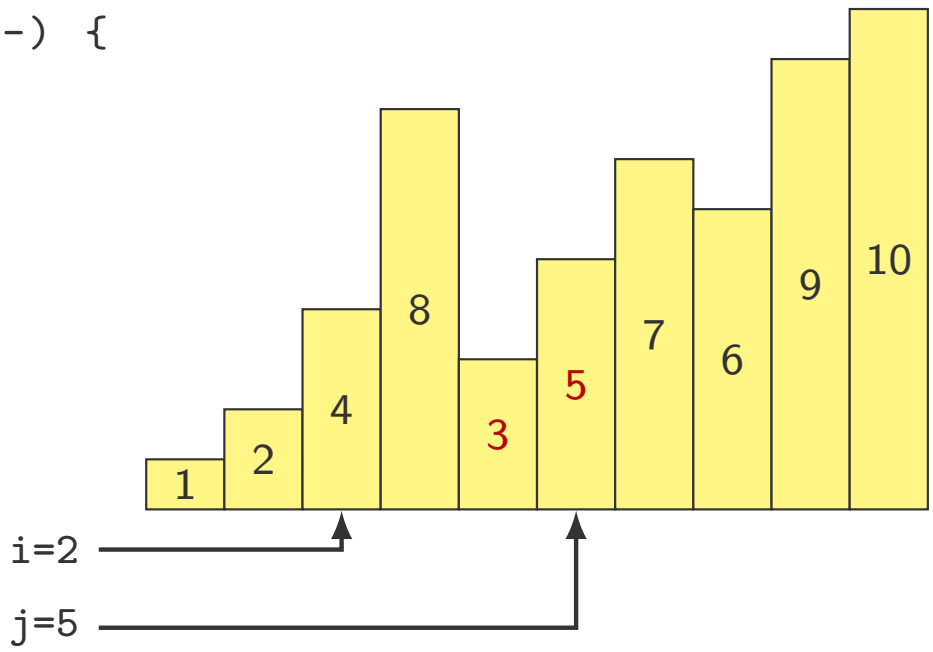
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



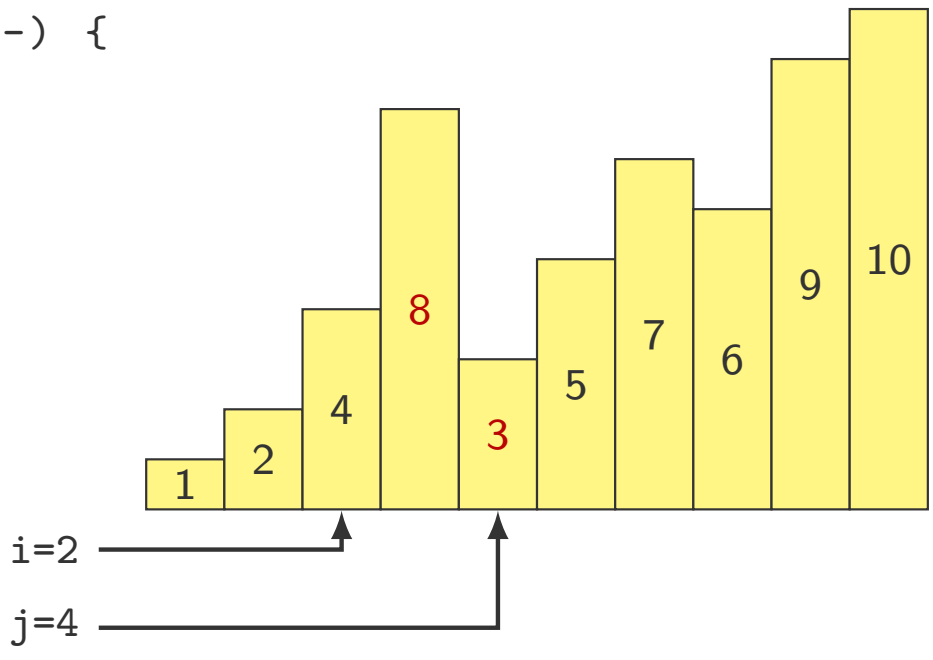
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



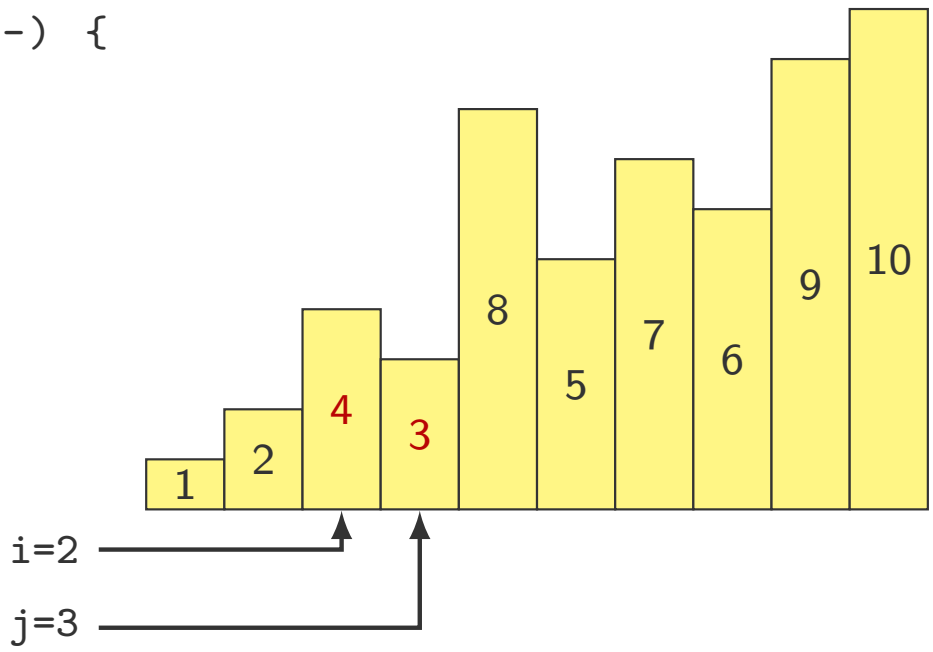
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



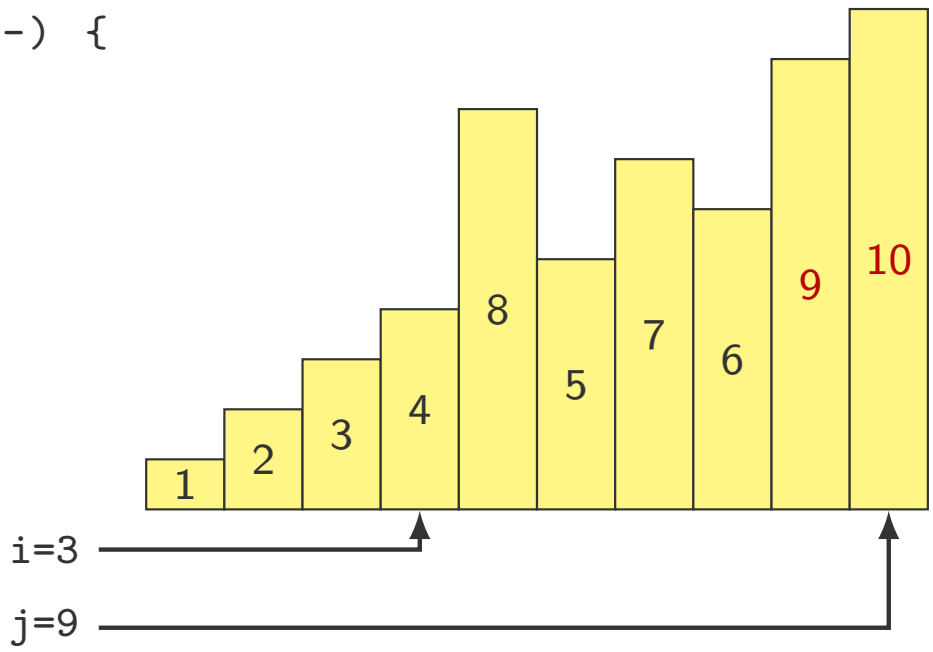
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



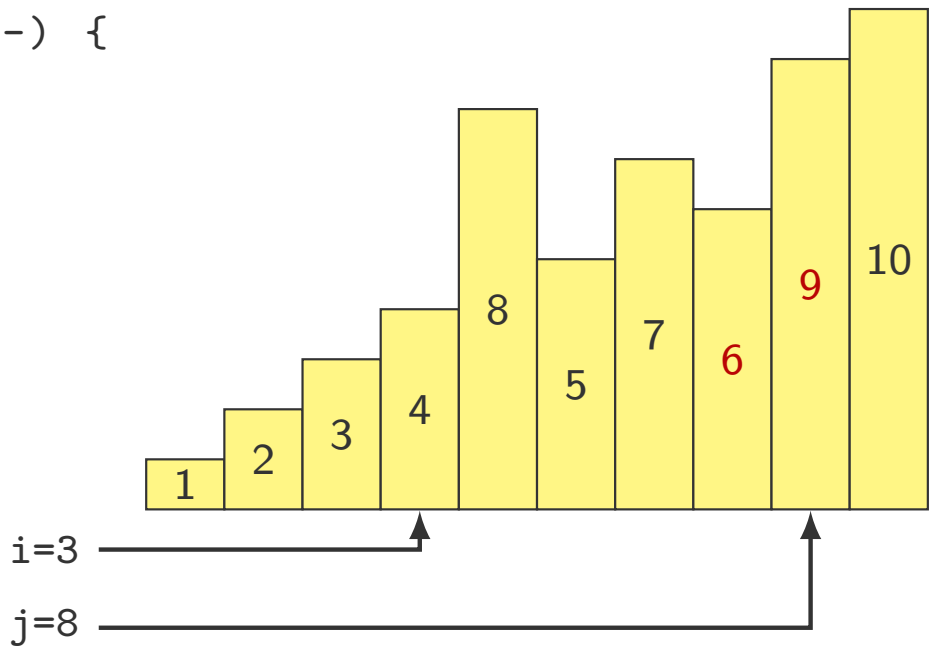
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



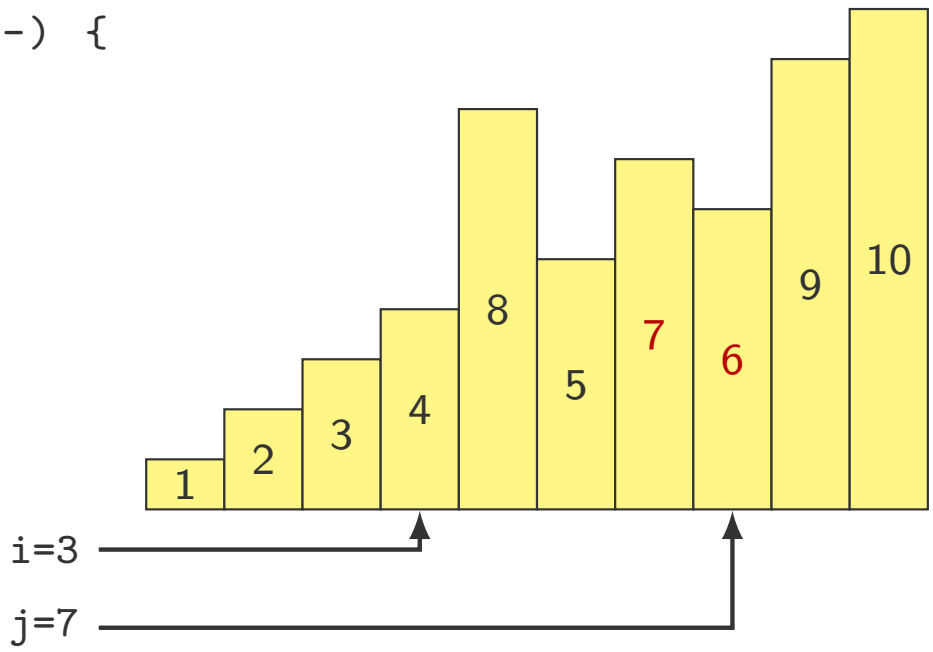
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



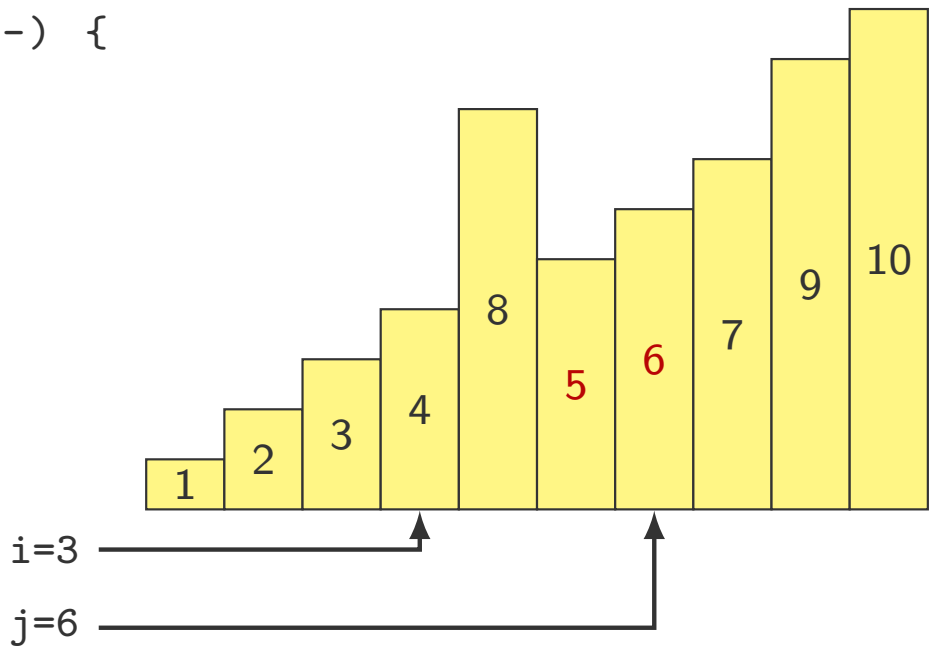
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



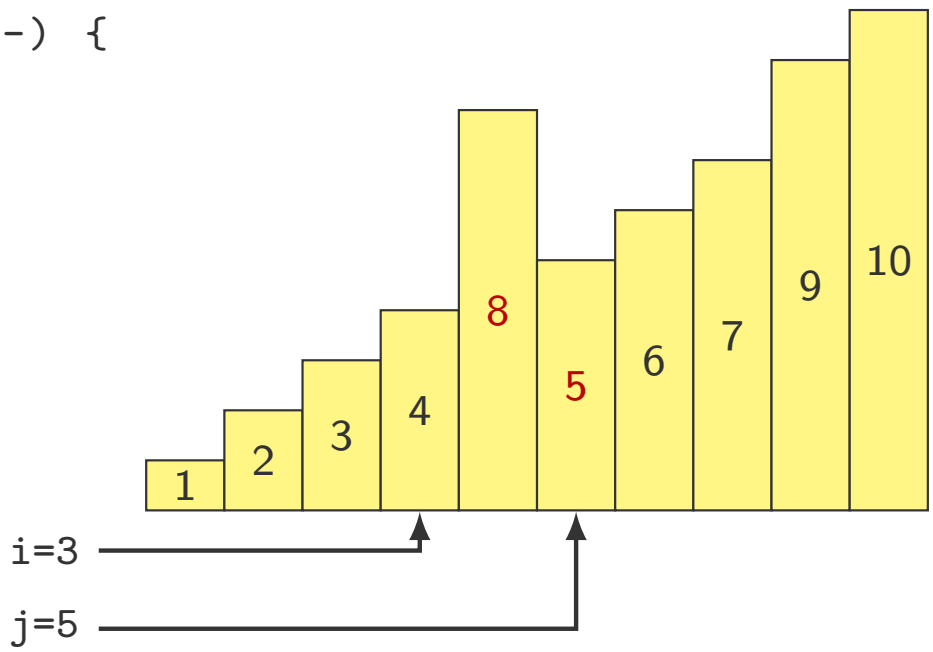
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



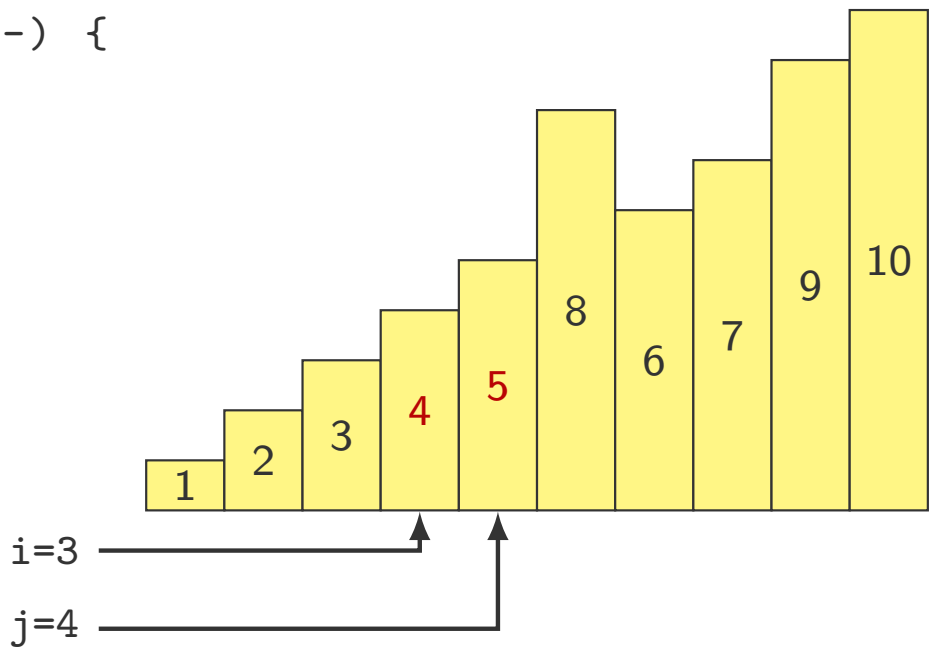
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



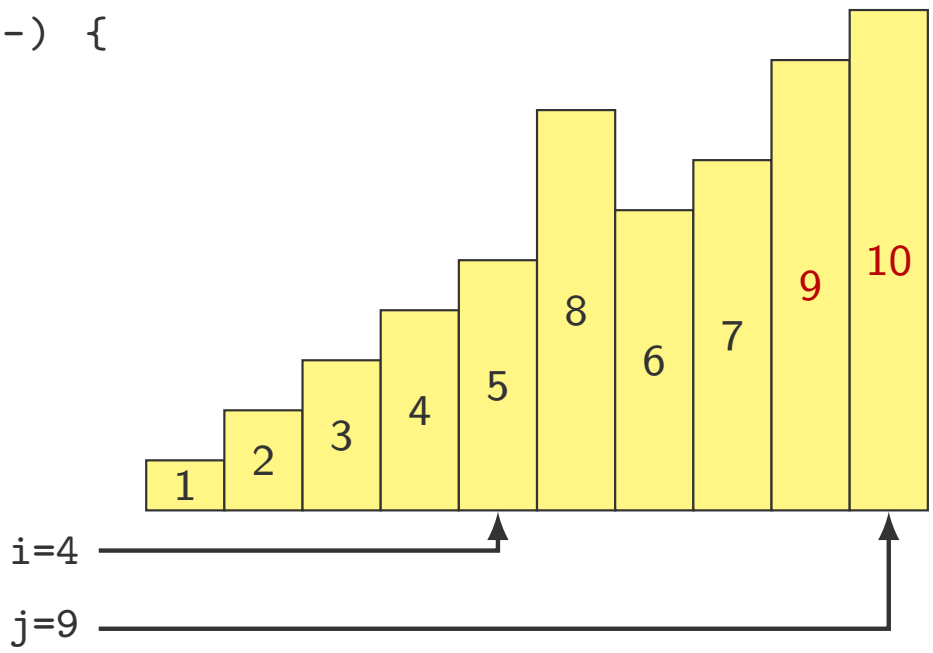
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



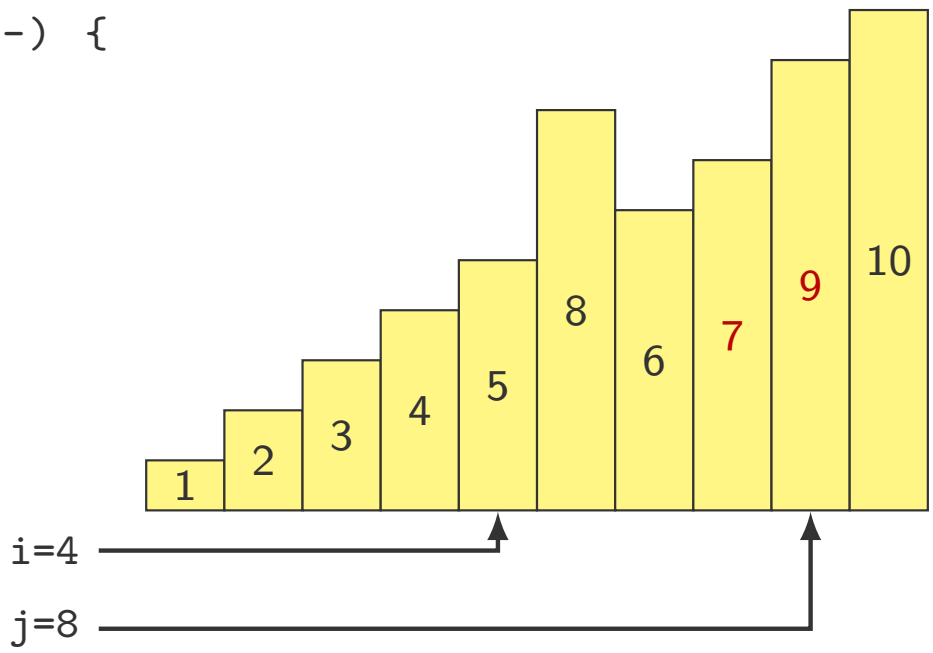
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



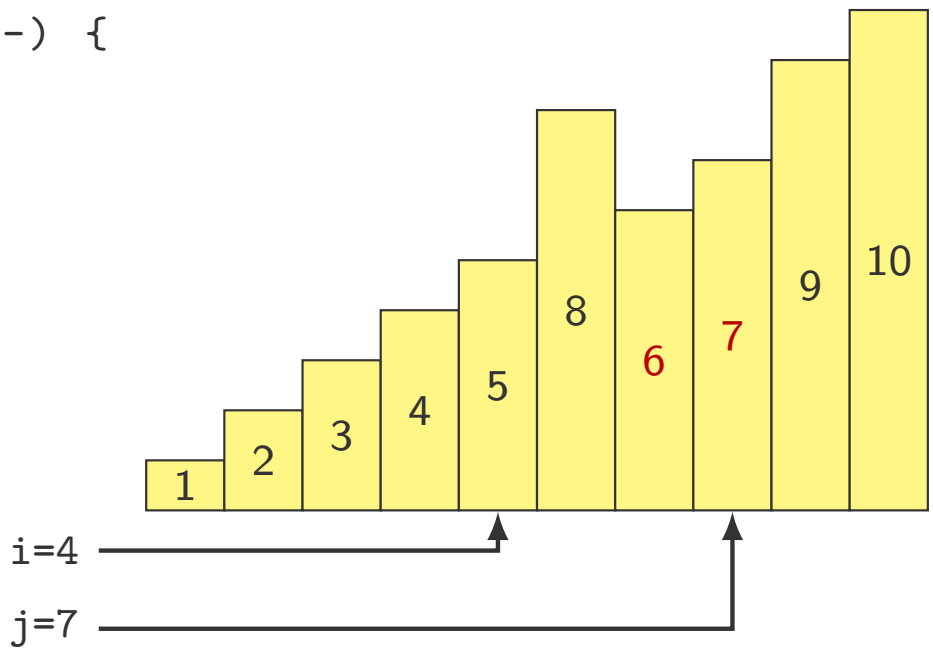
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



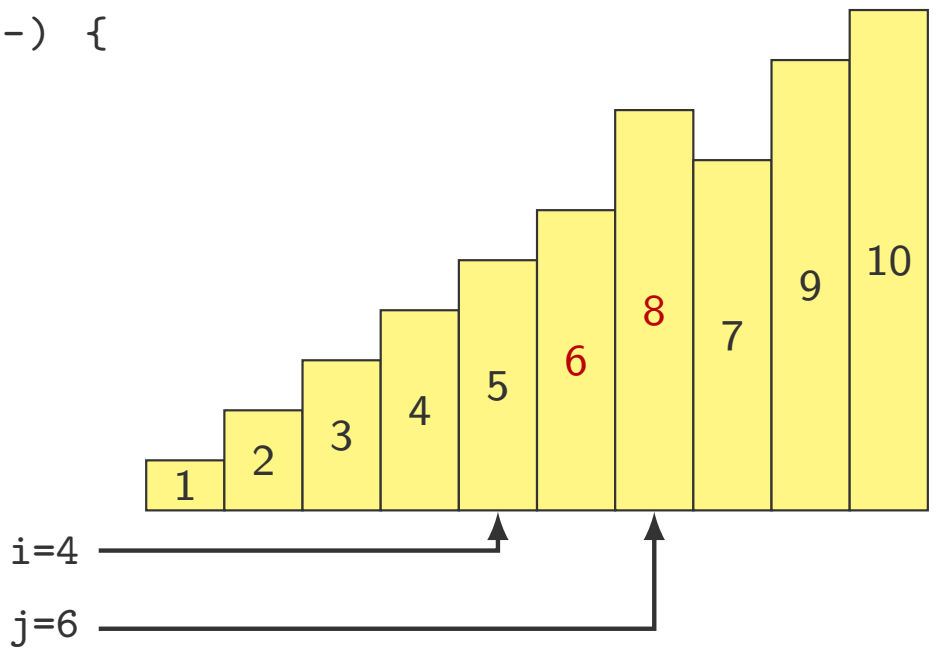
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



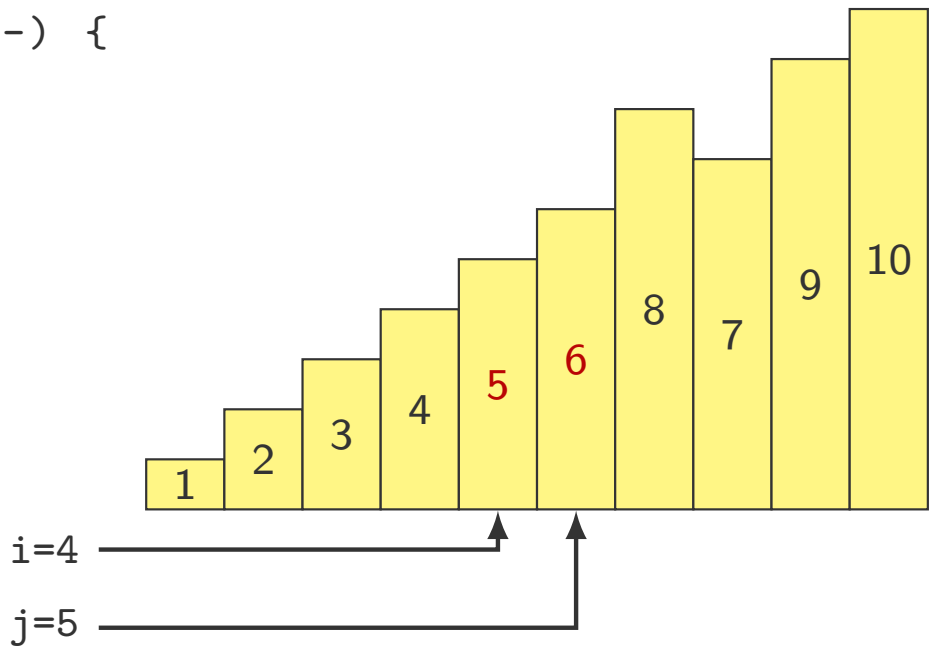
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



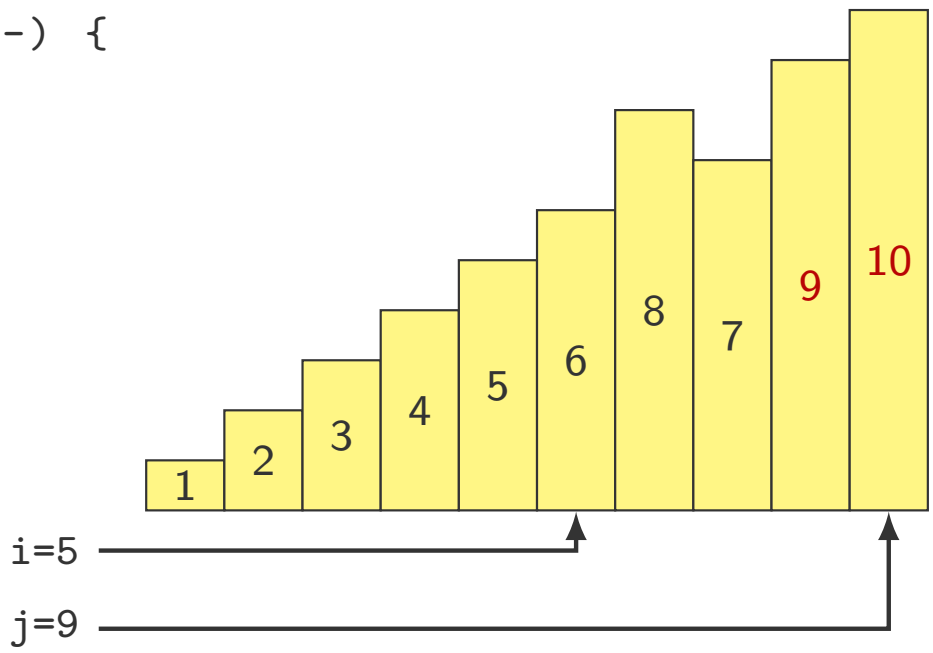
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



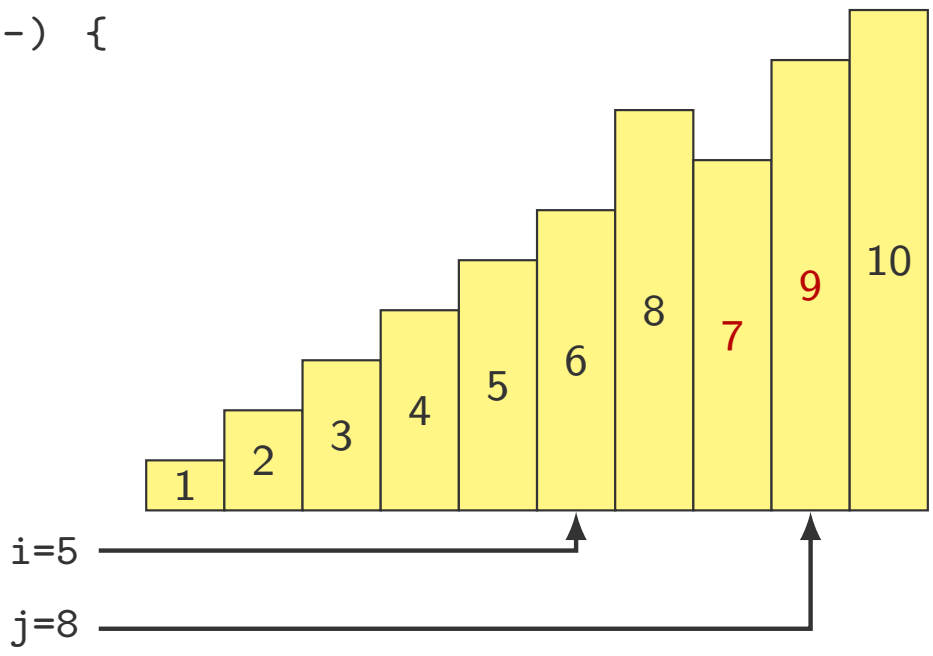
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



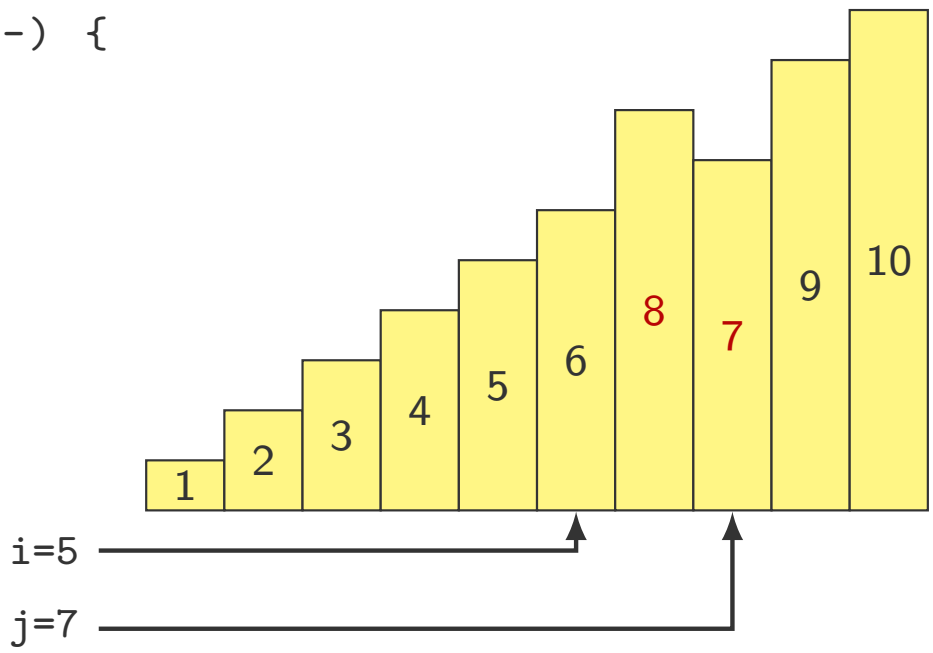
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



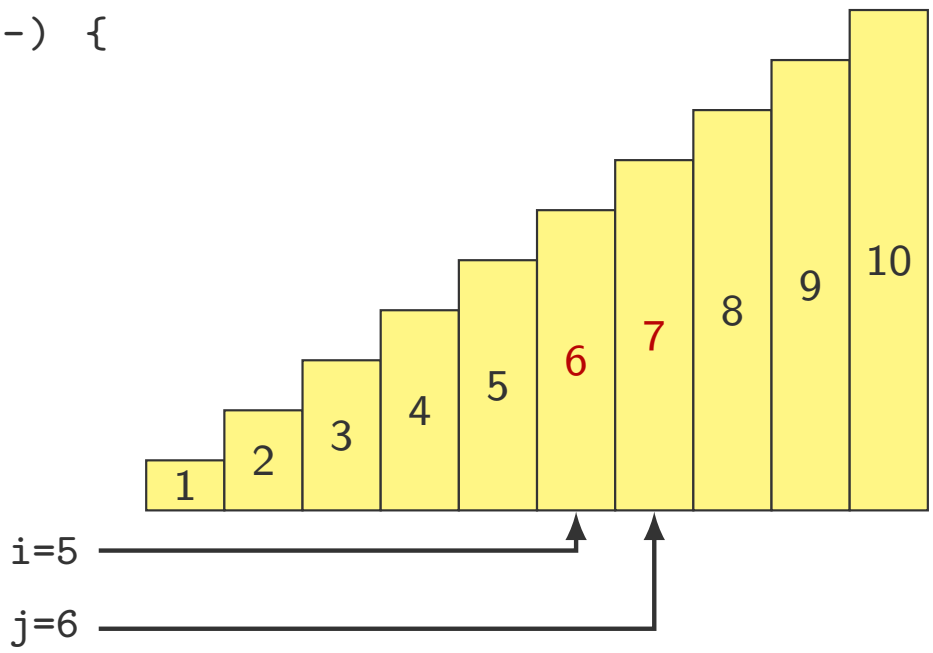
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



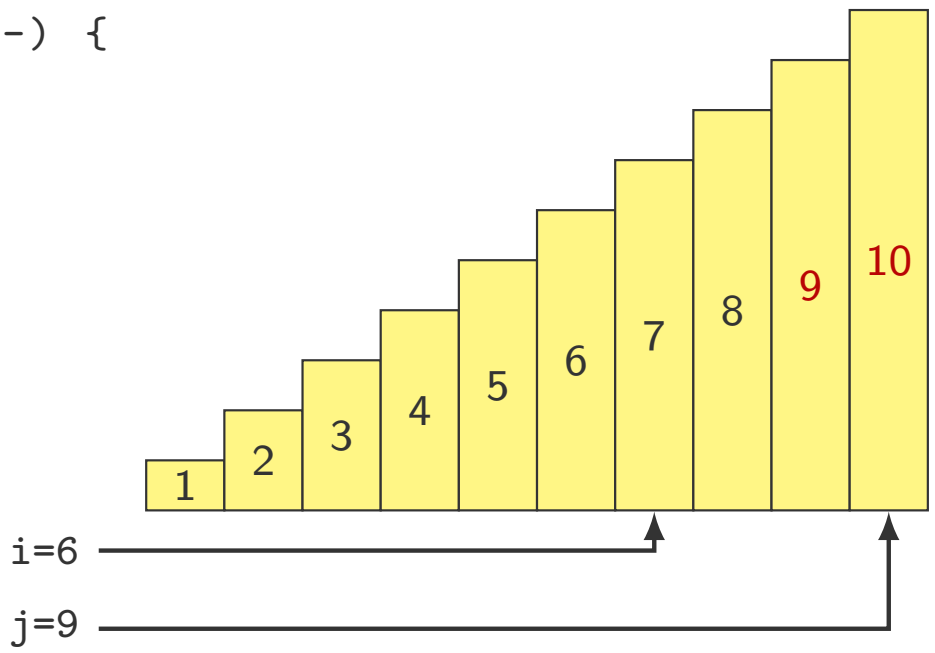
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



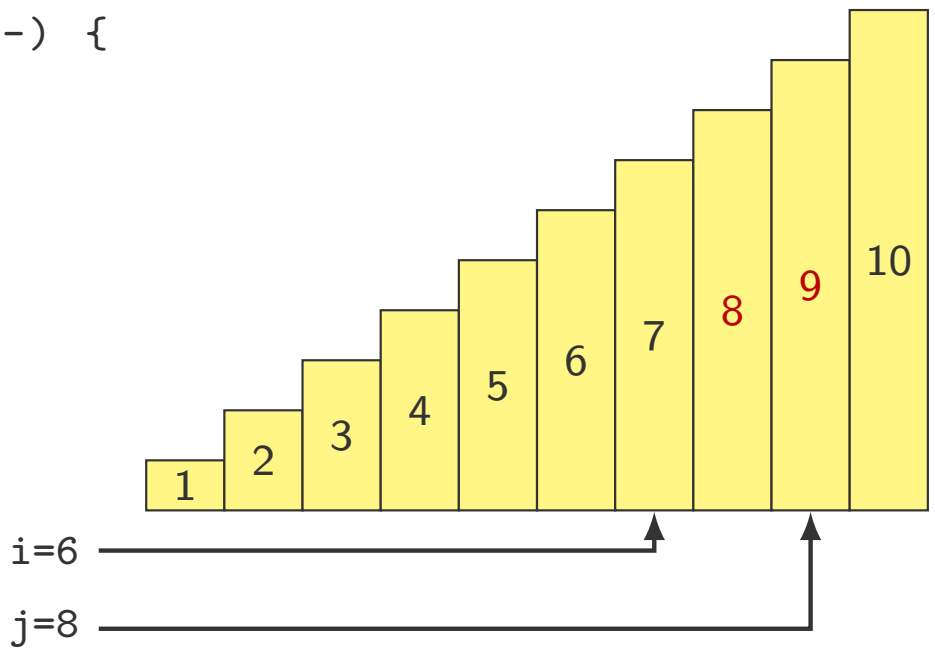
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



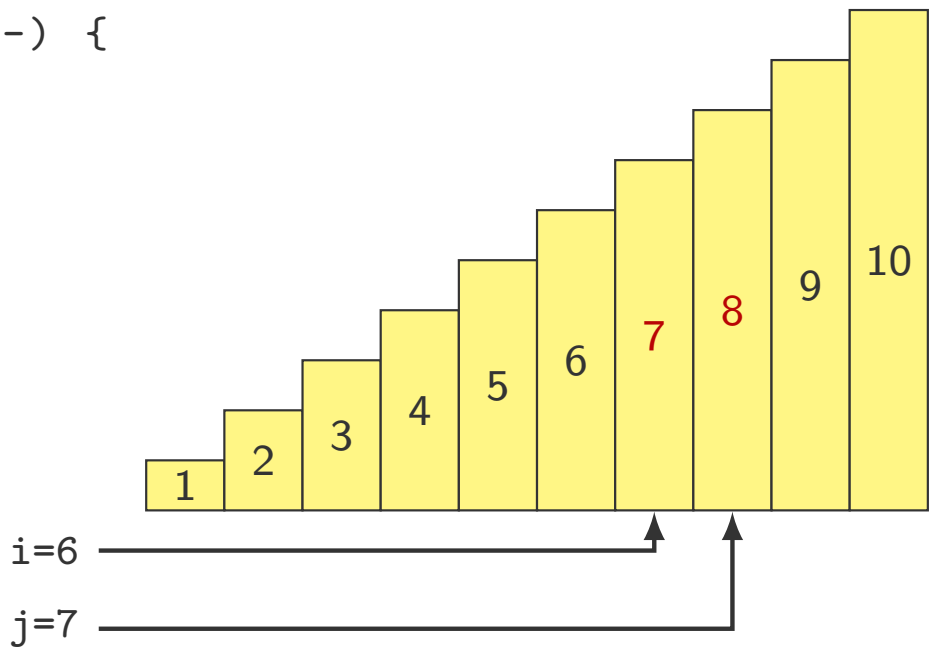
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



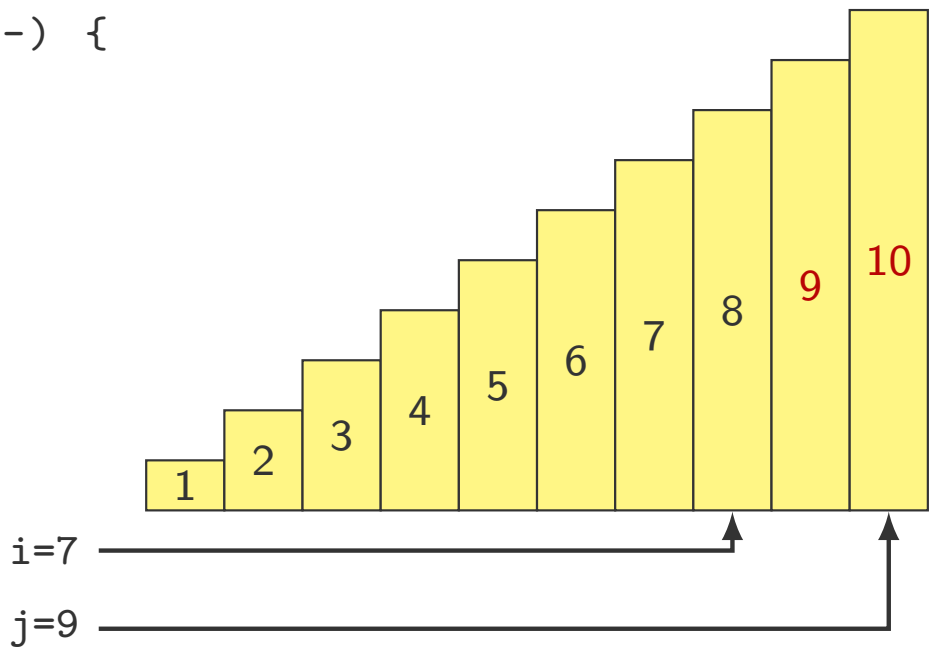
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



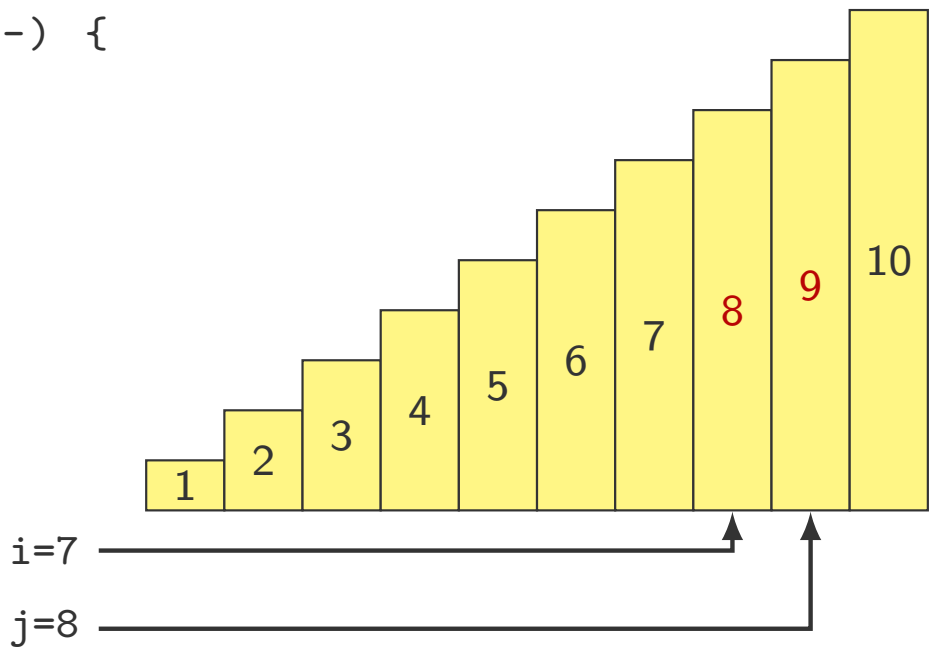
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



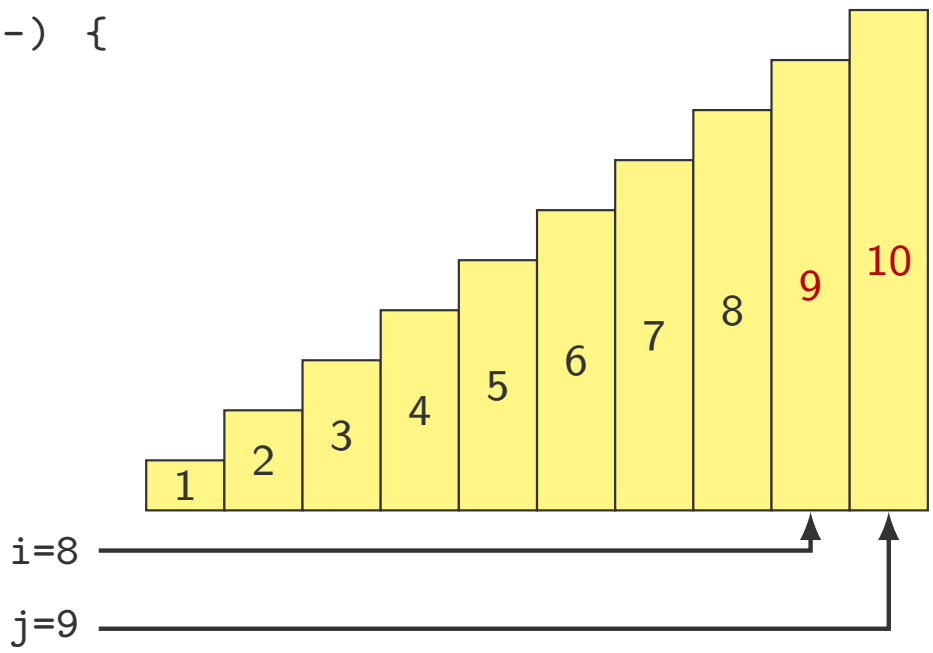
BubbleSort – Ordenação por flutuação

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



BubbleSort – Ordenação por flutuação

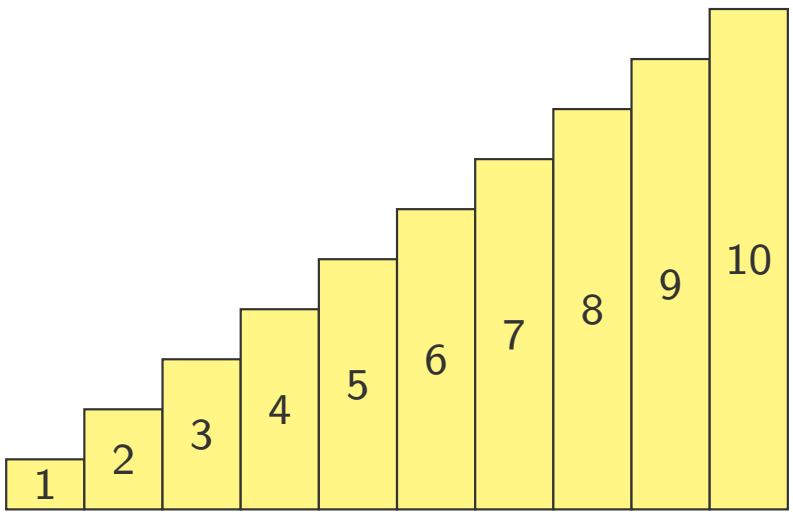
```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



BubbleSort – Ordenação por flutuação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```



i

j

BubbleSort — Complexidade do algoritmo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

No caso médio:

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

No caso médio:

- comparações: $\approx n^2/2 = O(n^2)$

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

No caso médio:

- comparações: $\approx n^2/2 = O(n^2)$
- trocas: $\approx n^2/2 = O(n^2)$

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

No caso médio:

- comparações: $\approx n^2/2 = O(n^2)$
- trocas: $\approx n^2/2 = O(n^2)$

No melhor caso:

BubbleSort — Complexidade do algoritmo


```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

No caso médio:

- comparações: $\approx n^2/2 = O(n^2)$
- trocas: $\approx n^2/2 = O(n^2)$

No melhor caso:

- comparações: $\approx n^2 = O(n^2)$

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int n) {
2      int trocou = 1;
3      for (int i = 0; i < n && trocou; i++) {
4          trocou = 0;
5          for (int j = n-1; j > i; j--) {
6              if (A[j] < A[j-1]) {
7                  int aux = A[j];
8                  A[j] = A[j-1];
9                  A[j-1] = aux;
10                 trocou = 1;
11             }
12         }
13     }
14 }
```

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int n) {
2      int trocou = 1;
3      for (int i = 0; i < n && trocou; i++) {
4          trocou = 0;
5          for (int j = n-1; j > i; j--) {
6              if (A[j] < A[j-1]) {
7                  int aux = A[j];
8                  A[j] = A[j-1];
9                  A[j-1] = aux;
10                 trocou = 1;
11             }
12         }
13     }
14 }
```

No pior caso toda comparação gera uma troca:

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int n) {
2      int trocou = 1;
3      for (int i = 0; i < n && trocou; i++) {
4          trocou = 0;
5          for (int j = n-1; j > i; j--) {
6              if (A[j] < A[j-1]) {
7                  int aux = A[j];
8                  A[j] = A[j-1];
9                  A[j-1] = aux;
10                 trocou = 1;
11             }
12         }
13     }
14 }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int n) {
2      int trocou = 1;
3      for (int i = 0; i < n && trocou; i++) {
4          trocou = 0;
5          for (int j = n-1; j > i; j--) {
6              if (A[j] < A[j-1]) {
7                  int aux = A[j];
8                  A[j] = A[j-1];
9                  A[j-1] = aux;
10                 trocou = 1;
11             }
12         }
13     }
14 }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int n) {
2      int trocou = 1;
3      for (int i = 0; i < n && trocou; i++) {
4          trocou = 0;
5          for (int j = n-1; j > i; j--) {
6              if (A[j] < A[j-1]) {
7                  int aux = A[j];
8                  A[j] = A[j-1];
9                  A[j-1] = aux;
10                 trocou = 1;
11             }
12         }
13     }
14 }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$
- trocas: $n(n-1)/2 = O(n^2)$

No melhor caso: comparações: $O(n)$, trocas: $O(1)$

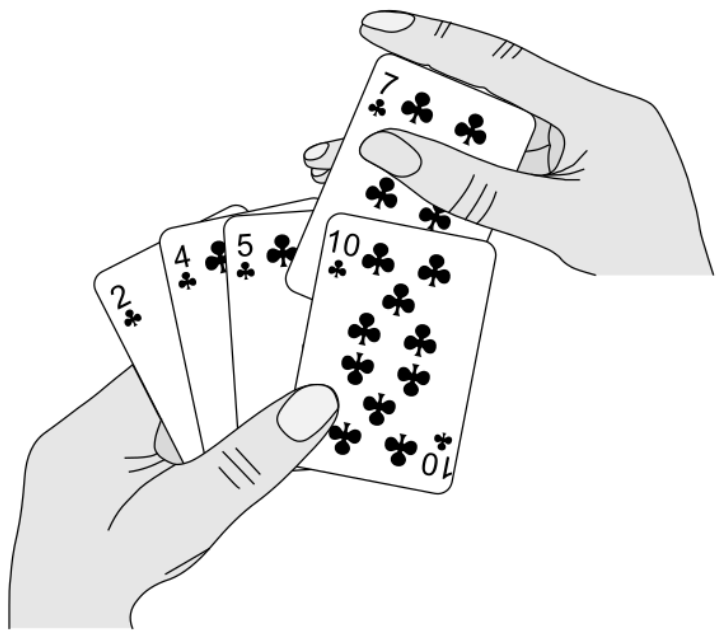
InsertionSort

Ordenação por Inserção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

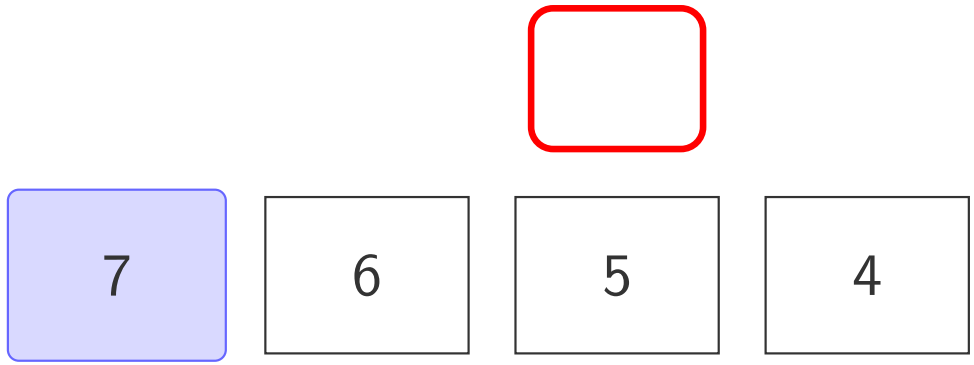
- Se já temos `v[0]`, `v[1]`, ..., `v[i-1]` ordenado, então inserimos `v[i]` na posição correta
  - Fazemos algo similar ao BubbleSort: ficamos com `v[0]`, `v[1]`, ..., `v[i]` ordenado



Retirado do livro do Cormen.

Insertion Sort — execução passo a passo

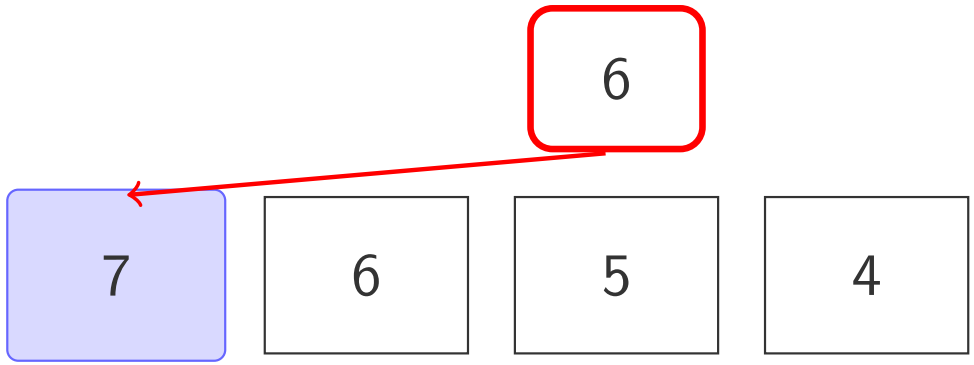
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Estado inicial: [7,6,5,4]

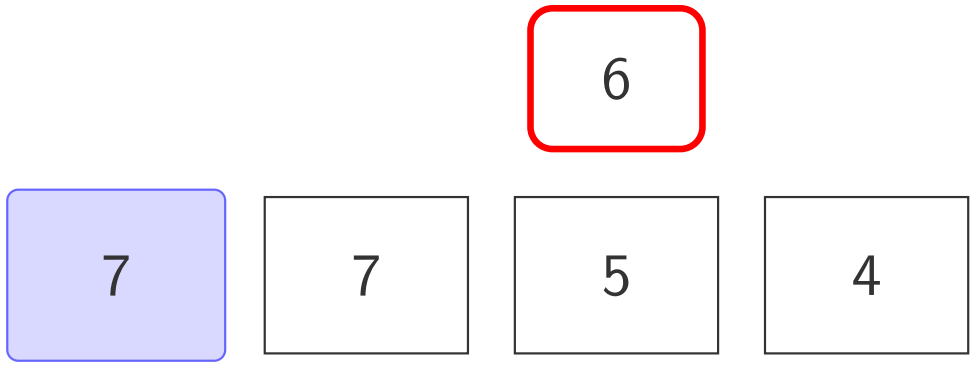
Insertion Sort — execução passo a passo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



j=1: copiando A[1]=6 para key e comparando com A[0]=7

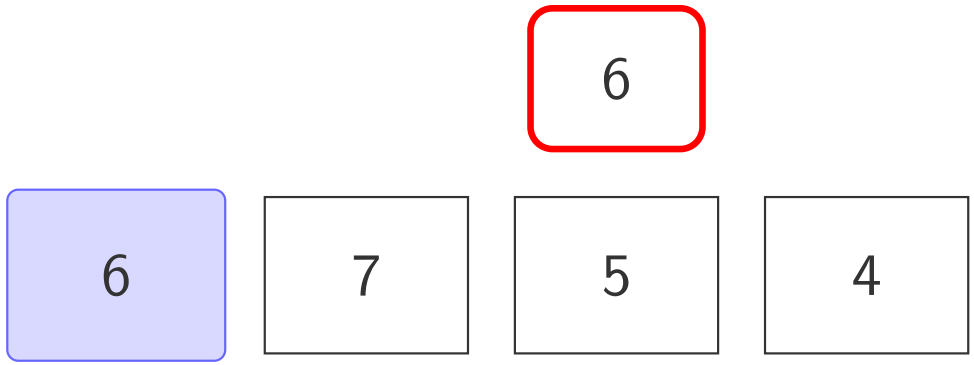
Insertion Sort — execução passo a passo



Shift: $A[1] \leftarrow A[0]$

Insertion Sort — execução passo a passo

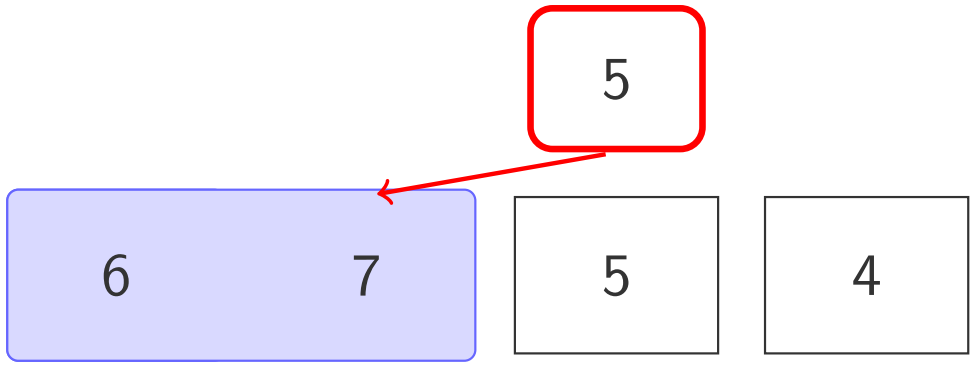
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



inserido: [6,7,5,4] $\Rightarrow$ prefixo A[0..1] ordenado (azul)

Insertion Sort — execução passo a passo

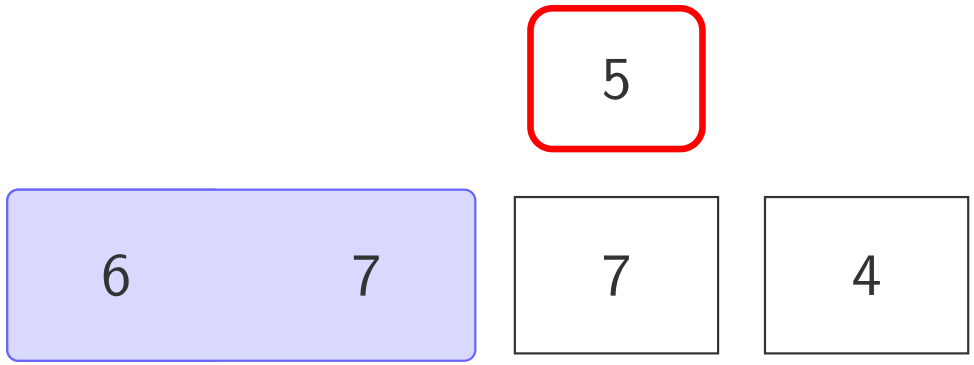
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



j=2: copiando A[2]=5 para key e comparando com A[1]=7

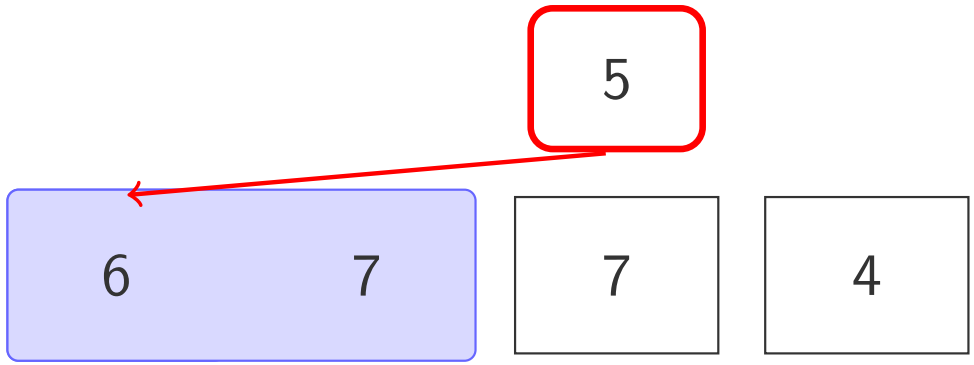
Insertion Sort — execução passo a passo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



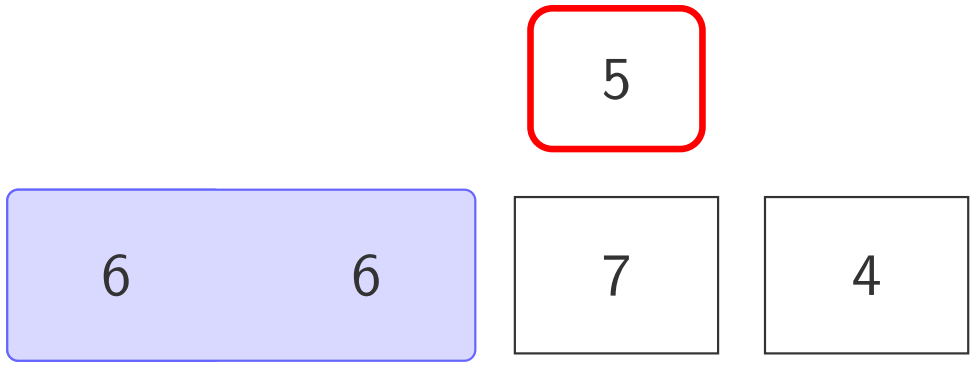
Shift: $A[2] \leftarrow A[1]$

Insertion Sort — execução passo a passo



Comparando key=5 com A[0]=6

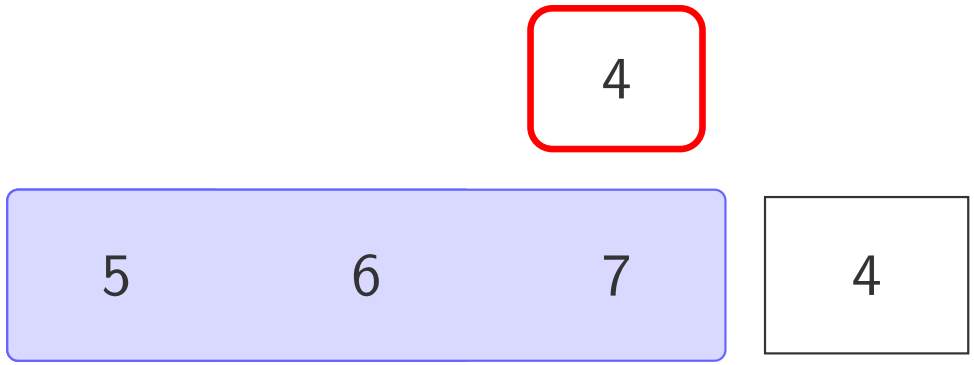
Insertion Sort — execução passo a passo



Shift: $A[1] \leftarrow A[0]$

Insertion Sort — execução passo a passo

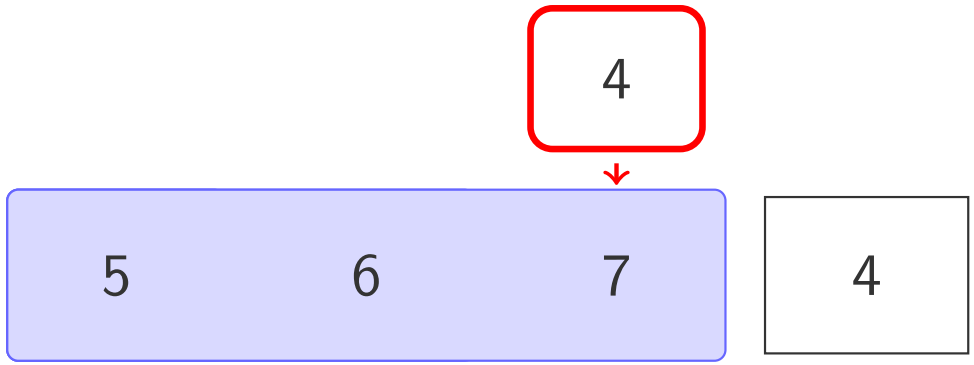
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



inserido: [5,6,7,4] $\Rightarrow$ prefixo A[0..2] ordenado (azul)

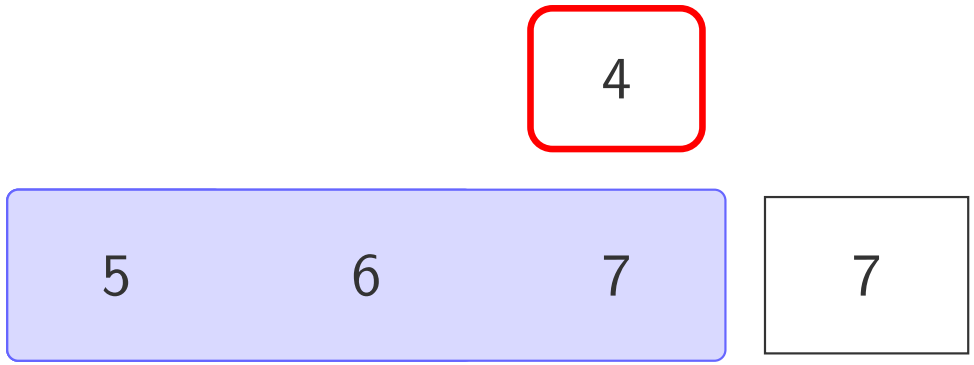
Insertion Sort — execução passo a passo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



j=3: copiando A[3]=4 para key e comparando com A[2]=7

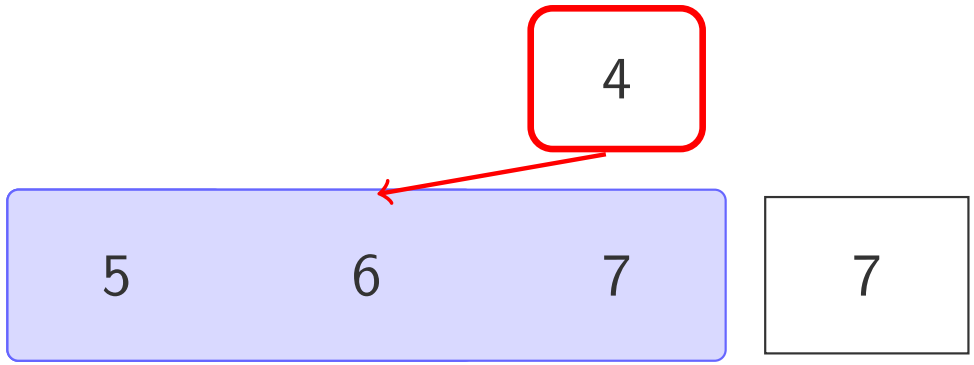
Insertion Sort — execução passo a passo



Shift: $A[3] \leftarrow A[2]$

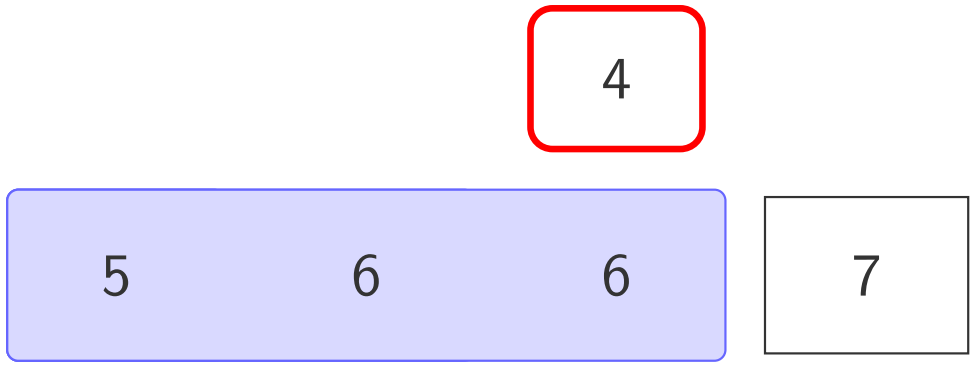
Insertion Sort — execução passo a passo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



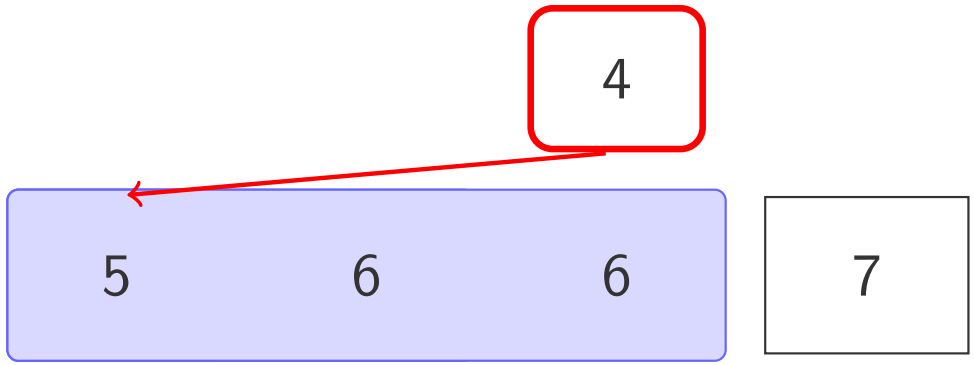
Comparando key=4 com A[1]=6

Insertion Sort — execução passo a passo



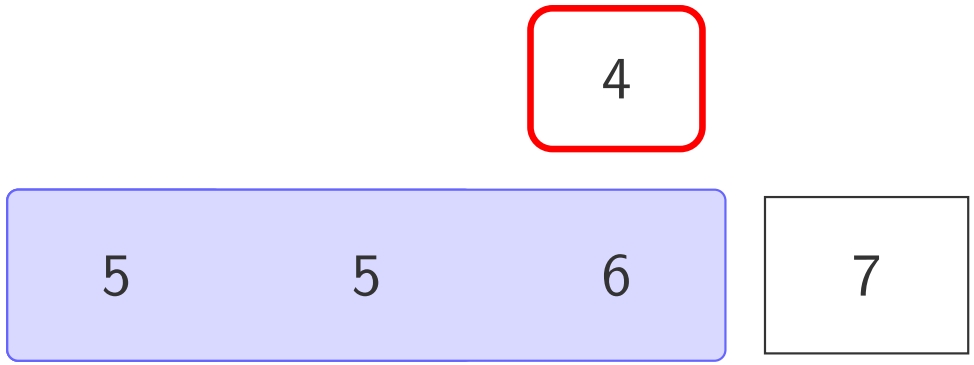
Shift: $A[2] \leftarrow A[1]$

Insertion Sort — execução passo a passo



Comparando key=4 com A[0]=5

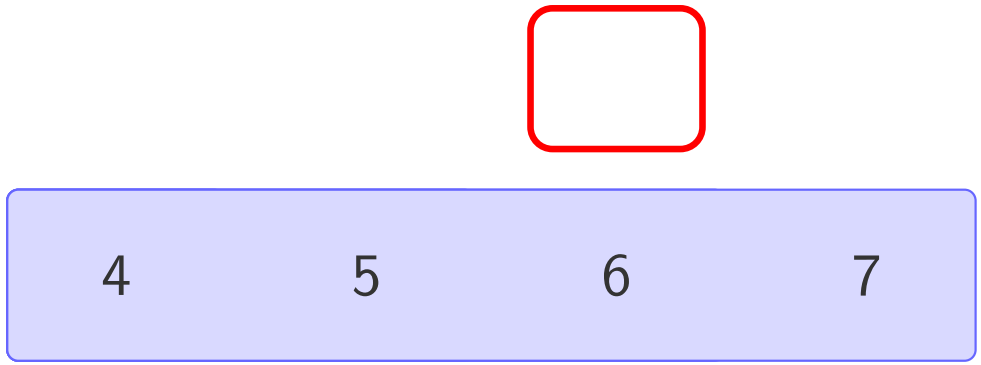
Insertion Sort — execução passo a passo



Shift: $A[1] \leftarrow A[0]$

Insertion Sort — execução passo a passo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



inserido: [4,5,6,7] $\Rightarrow$ array totalmente ordenado (azul)

Ordenação por Inserção

Algoritmo

```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

InsertionSort - Complexidade do algoritmo


```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

InsertionSort - Complexidade do algoritmo


```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

- O consumo de tempo do `insertionSort` é proporcional ao número de execuções da comparação $A[i] > key$.

InsertionSort - Complexidade do algoritmo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

- O consumo de tempo do `insertionSort` é proporcional ao número de execuções da comparação $A[i] > key$.
- Para cada j, a variável i assume no máximo j valores: $j-1, j-2, \ldots, 0$.

InsertionSort - Complexidade do algoritmo


```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

- O consumo de tempo do `insertionSort` é proporcional ao número de execuções da comparação $A[i] > key$.
- Para cada j, a variável i assume no máximo j valores: $j-1, j-2, \ldots, 0$.
- Como $1 \leq j \leq n-1$, o número de execuções da linha 5 é igual a $\sum_{j=1}^{n-1} j$ no pior caso.

InsertionSort - Complexidade do algoritmo

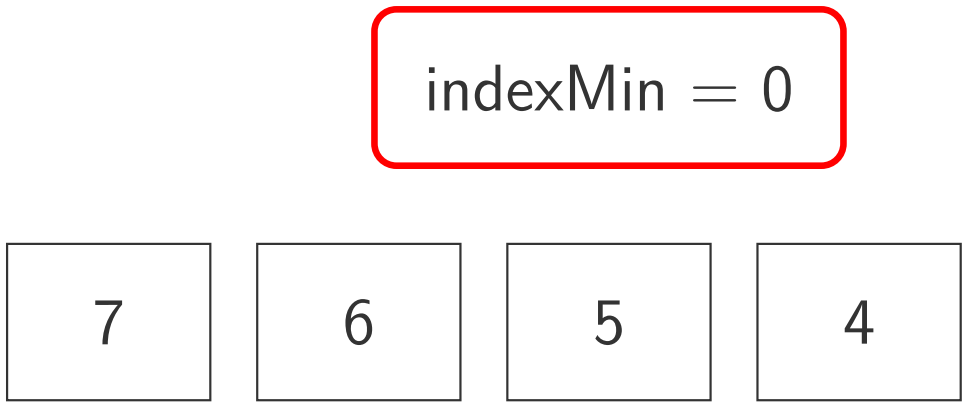

```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

- O consumo de tempo do `insertionSort` é proporcional ao número de execuções da comparação $A[i] > key$.
- Para cada j, a variável i assume no máximo j valores: $j-1, j-2, \ldots, 0$.
- Como $1 \leq j \leq n-1$, o número de execuções da linha 5 é igual a $\sum_{j=1}^{n-1} j$ no pior caso.
- Essa soma é igual a $n(n-1)/2 = O(n^2)$.

SelectionSort

Selection Sort

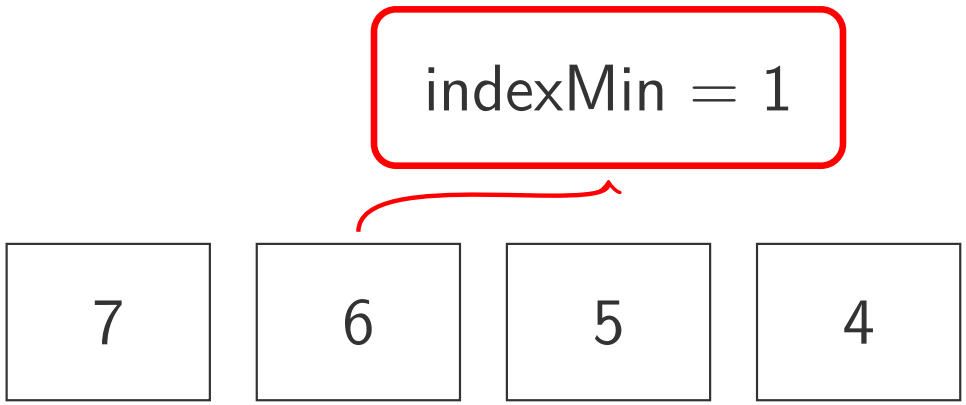
Selecionando o primeiro menor elemento



Estado inicial: [7,6,5,4], indexMin = 0

Selection Sort

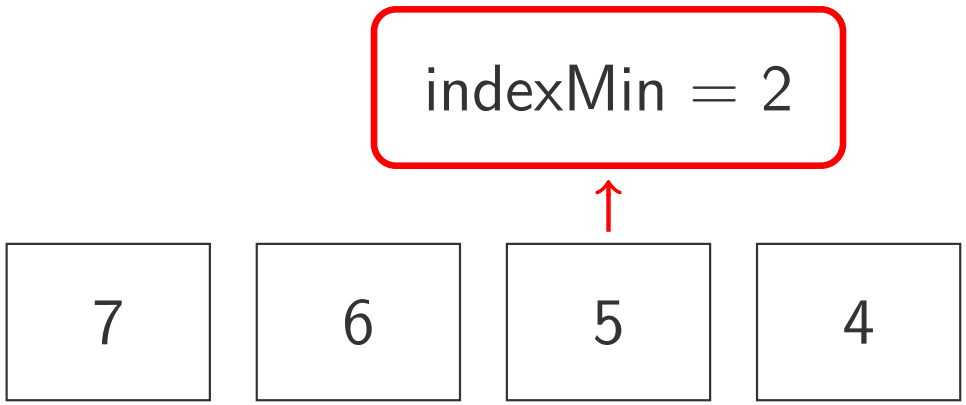
Selecionando o primeiro menor elemento



Comparando A[1]=6 com A[indexMin]=7, indexMin atualizado para 1

Selection Sort

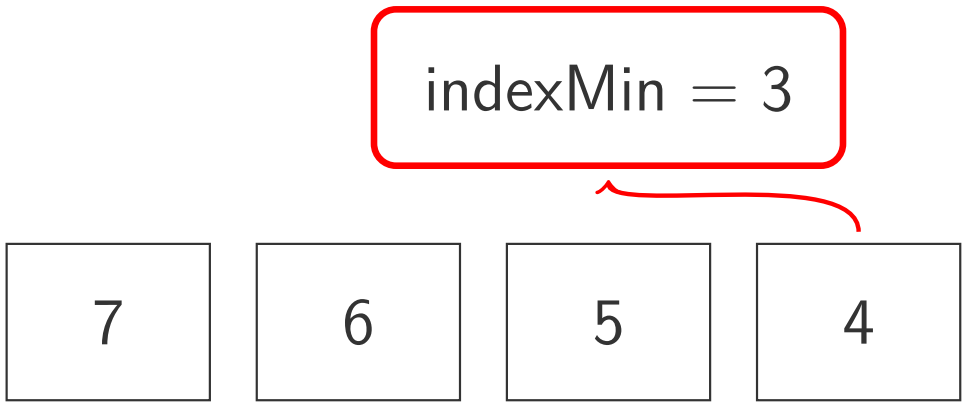
Selecionando o primeiro menor elemento



Comparando A[2]=5 com A[indexMin]=6, indexMin atualizado para 2

Selection Sort

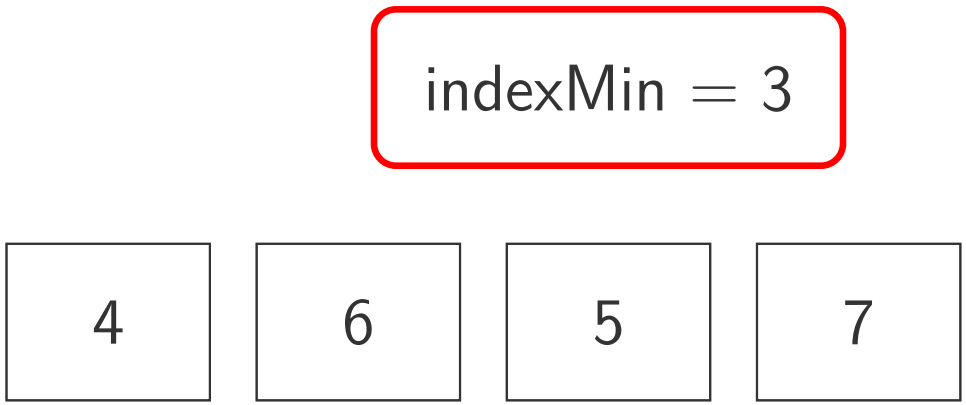
Selecionando o primeiro menor elemento



Comparando A[3]=4 com A[indexMin]=5, indexMin atualizado para 3

Selection Sort

Selecionando o primeiro menor elemento



Swap A[0] com A[indexMin]=4, array após a iteração: [4,6,5,7]

Ordenação por Seleção

Ideia:

Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0]`, `v[1]`, ..., `v[n-1]`

Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0]`, `v[1]`, ..., `v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1]`, `v[2]`, ..., `v[n-1]`

Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0]`, `v[1]`, ..., `v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1]`, `v[2]`, ..., `v[n-1]`
- ...

Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0]`, `v[1]`, ..., `v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1]`, `v[2]`, ..., `v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0]`, `v[1]`, ..., `v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1]`, `v[2]`, ..., `v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

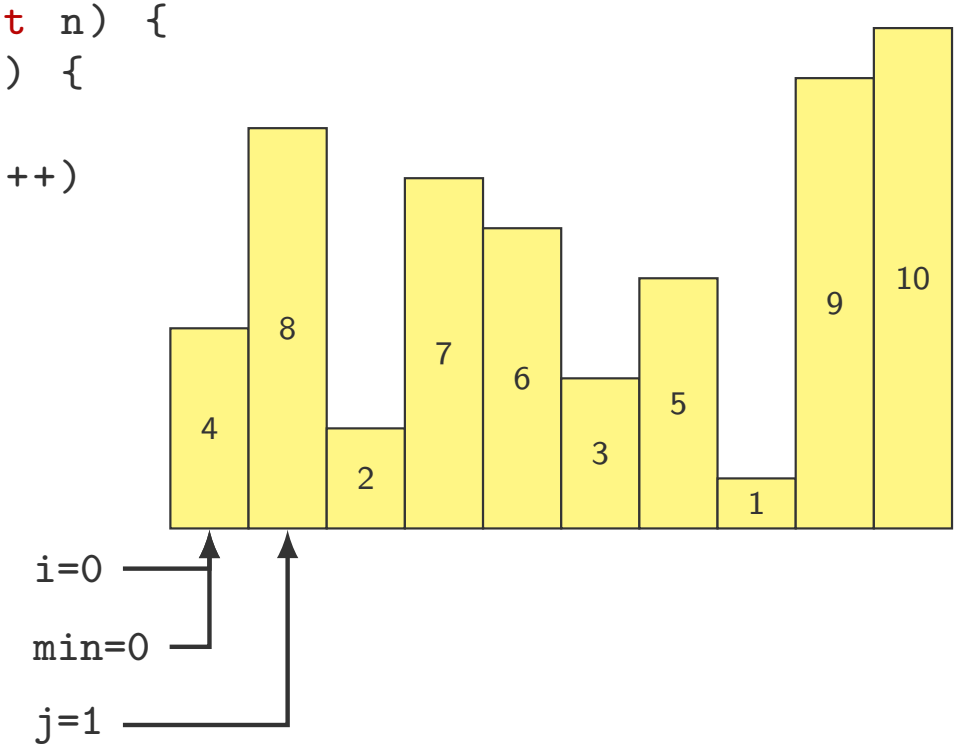
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



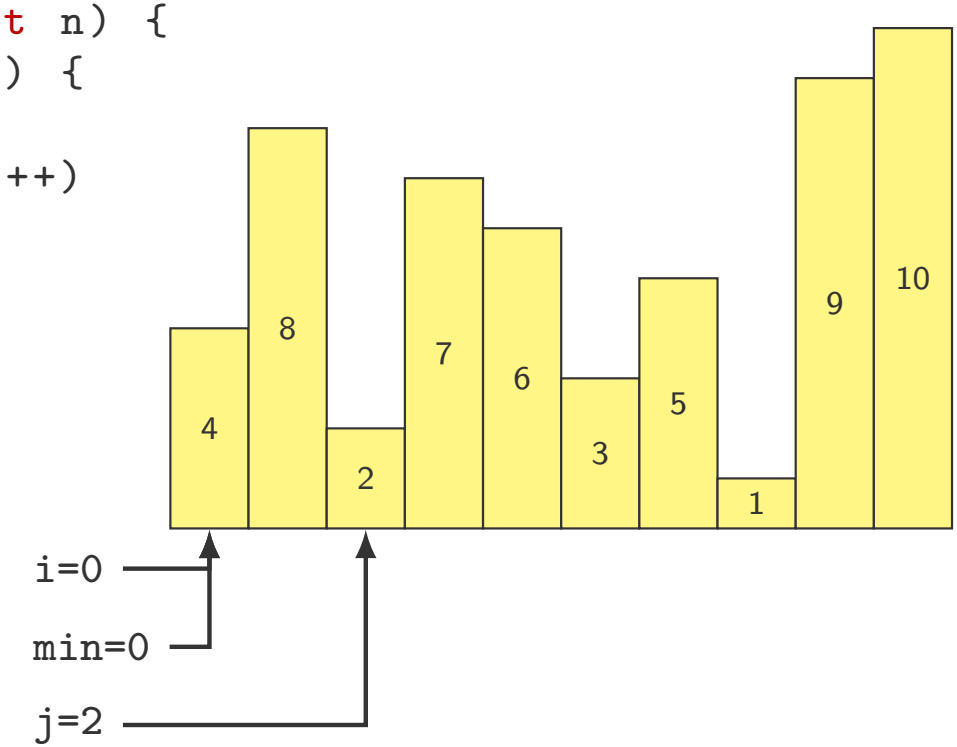
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0]`, `v[1]`, ..., `v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1]`, `v[2]`, ..., `v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



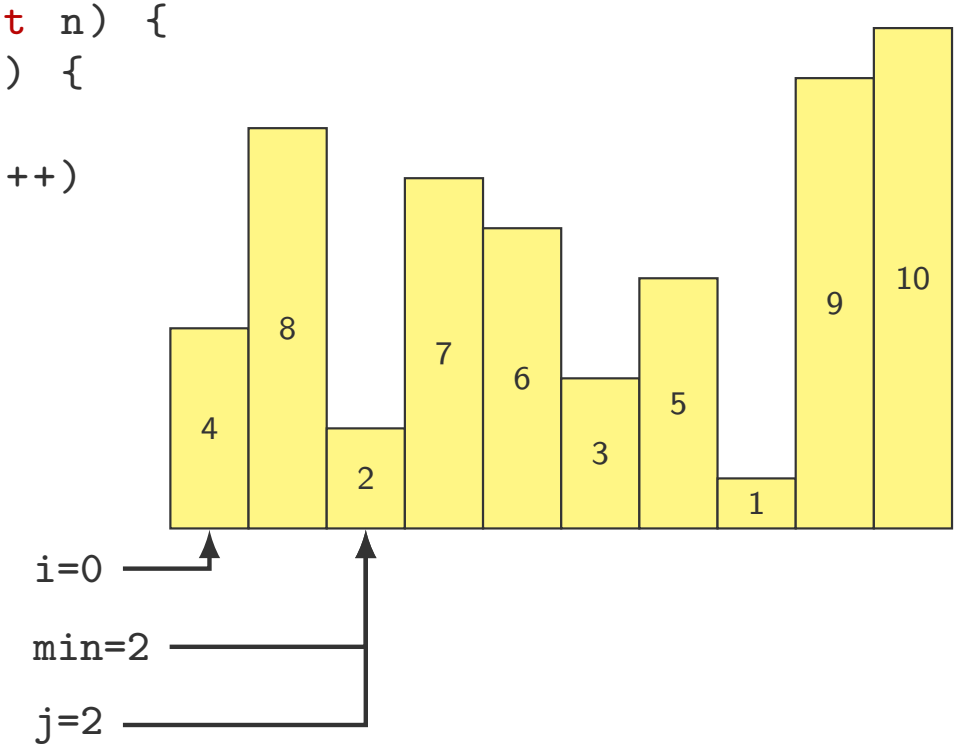
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



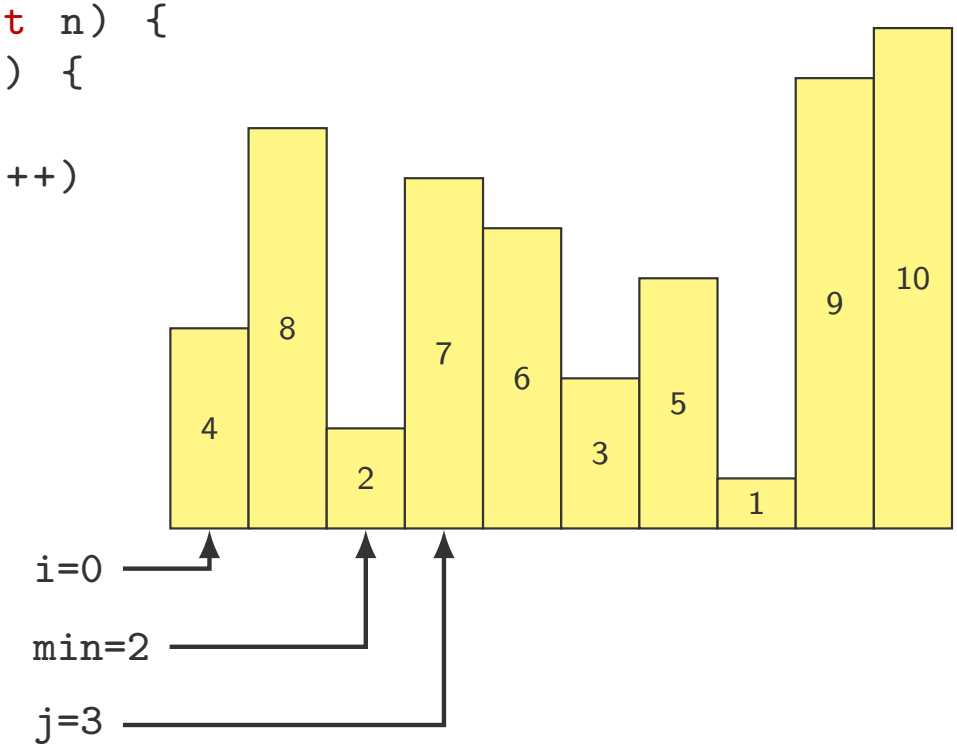
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

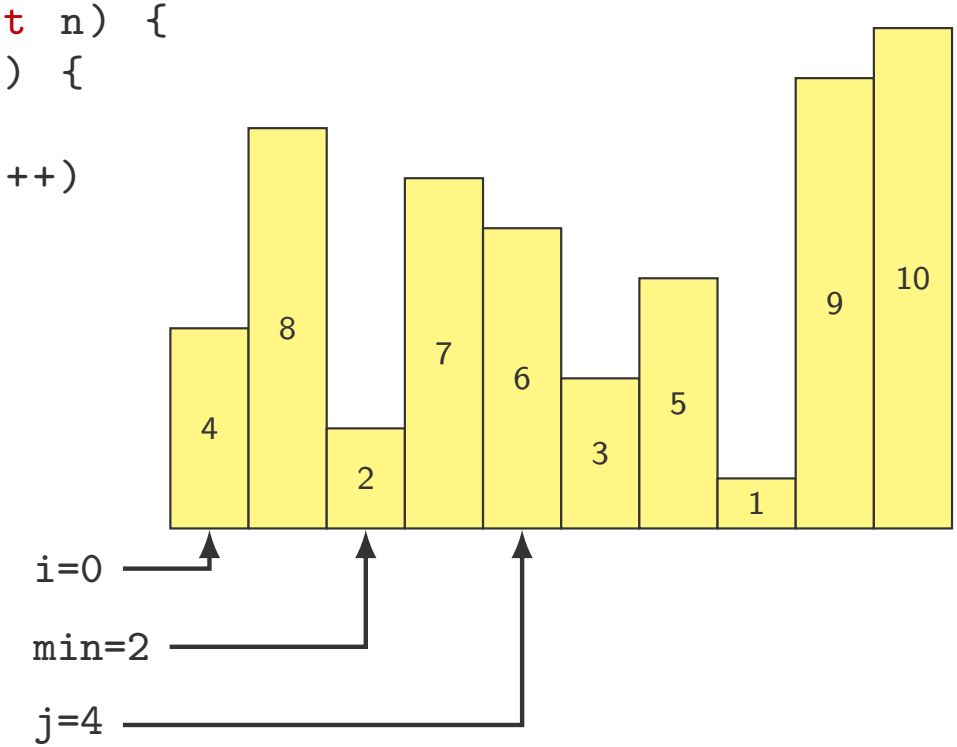


Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



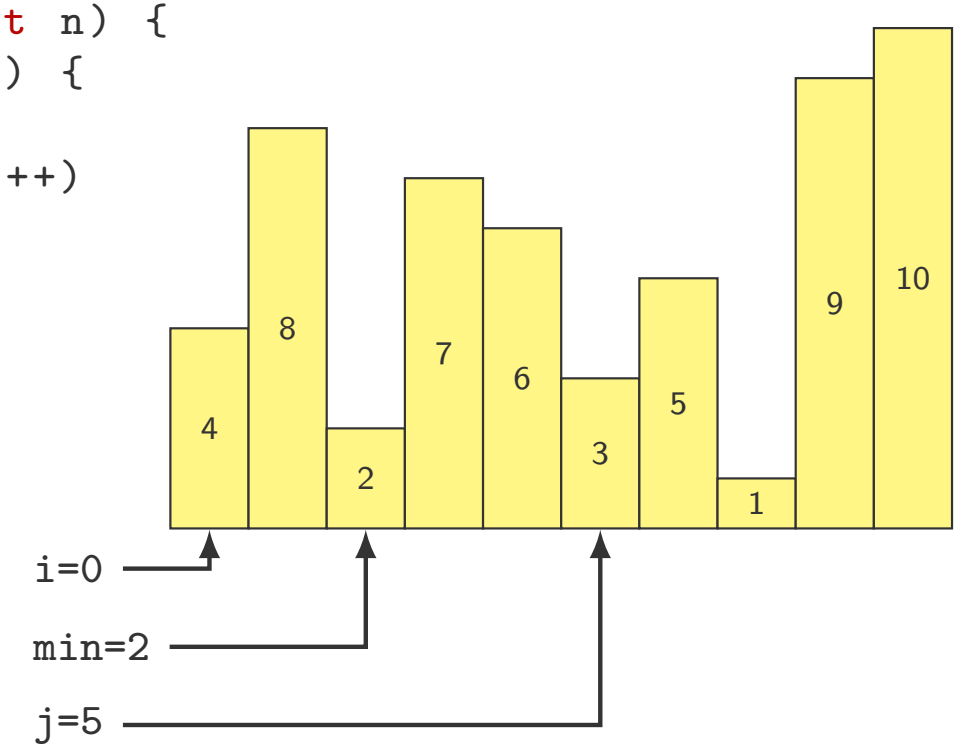
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



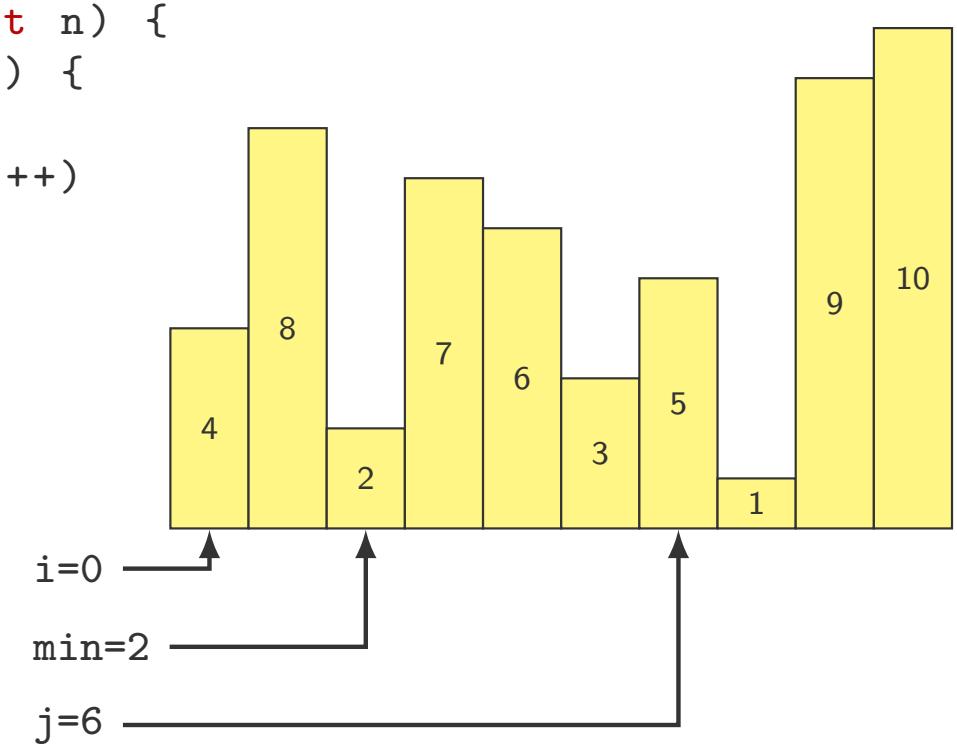
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



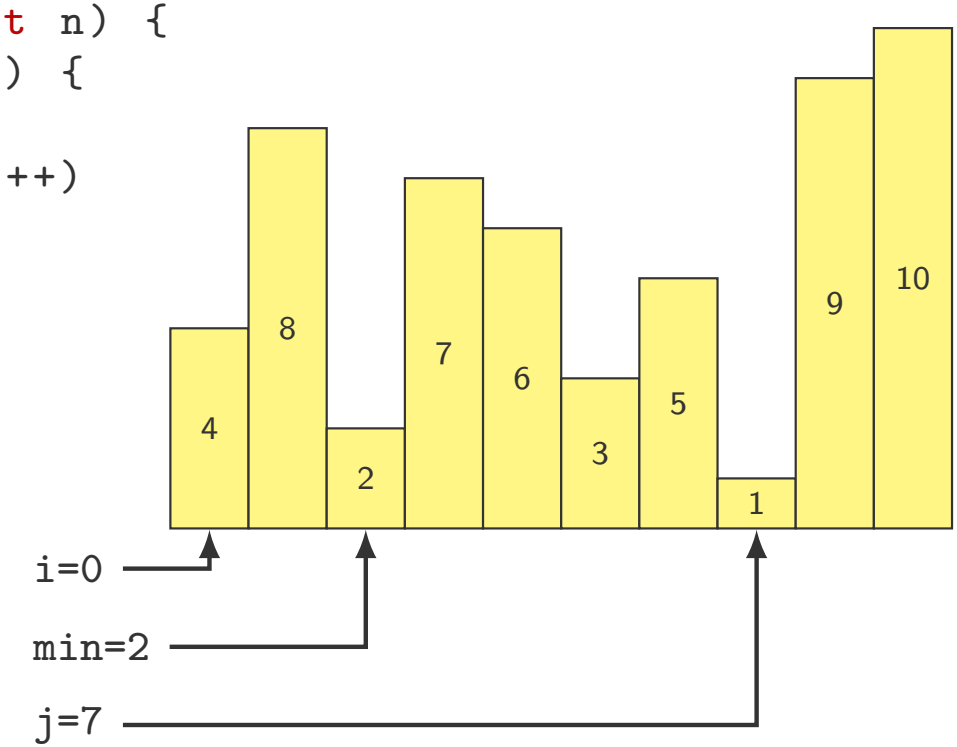
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



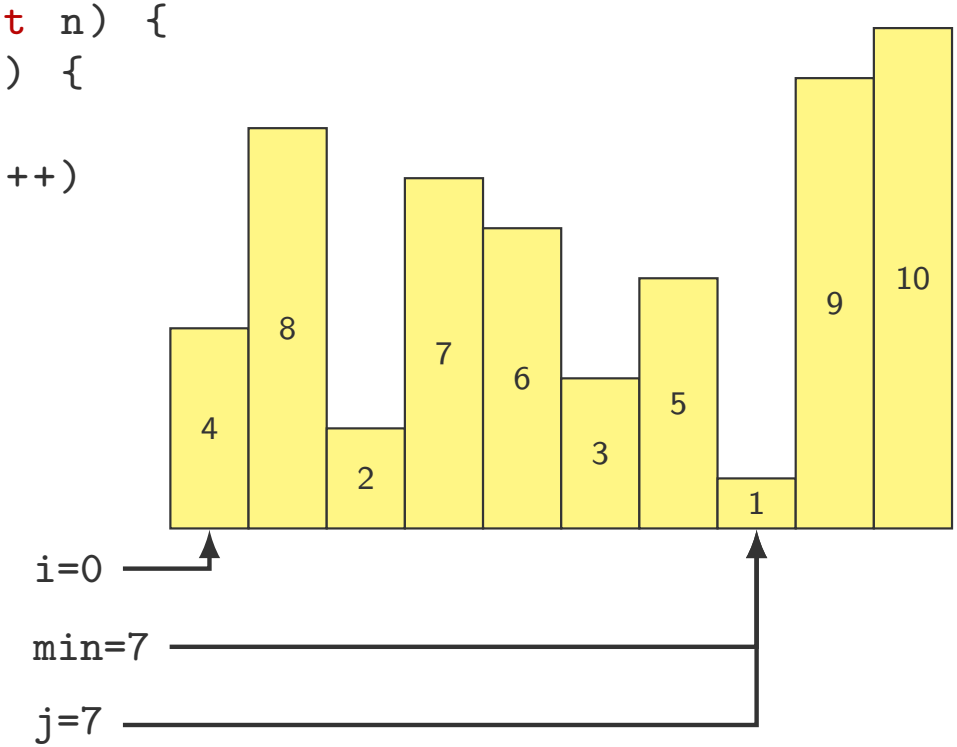
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



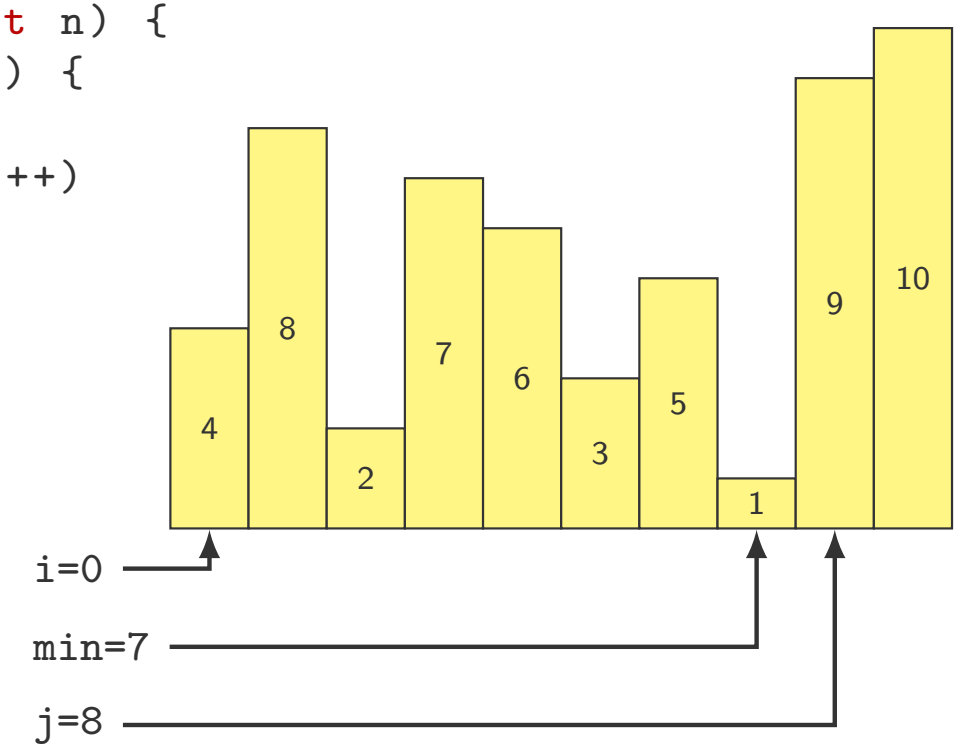
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



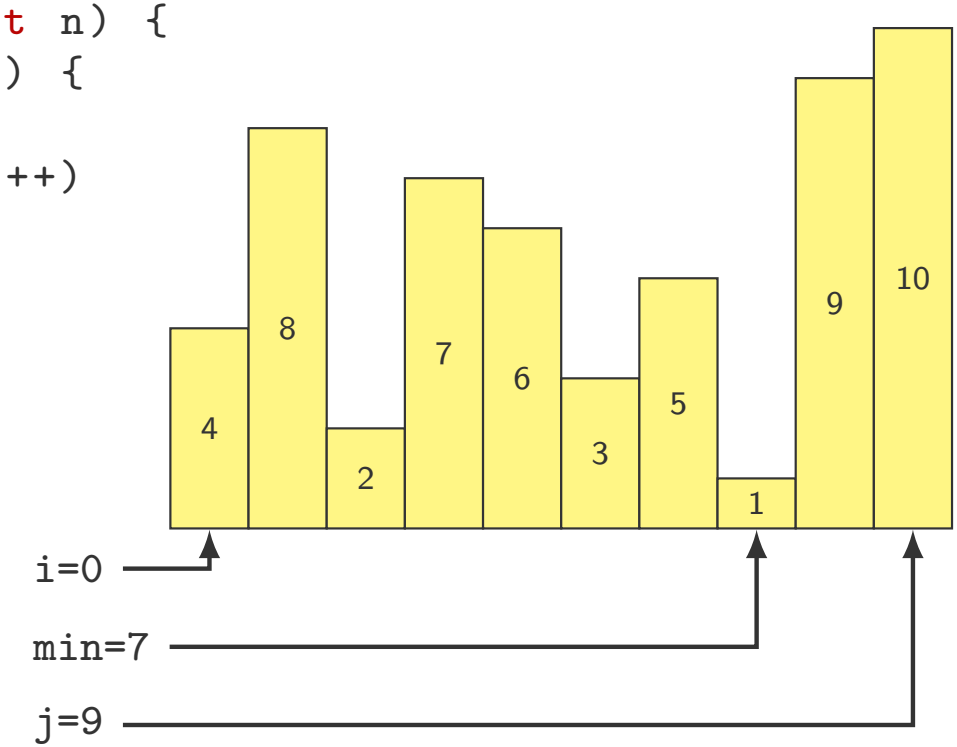
Ordenação por Seleção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



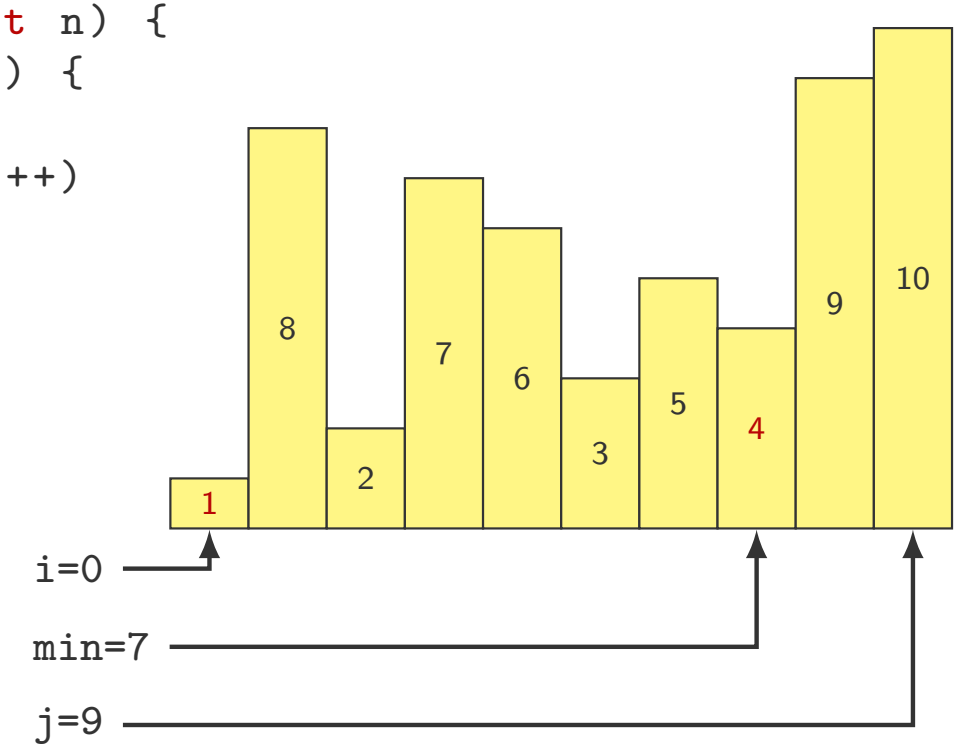
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



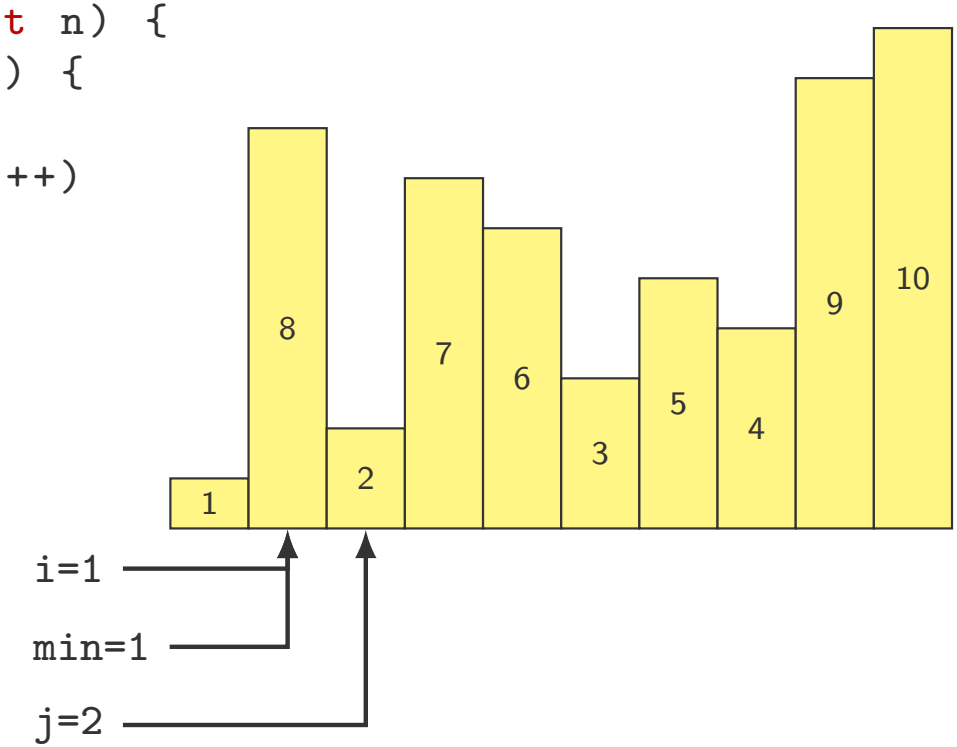
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



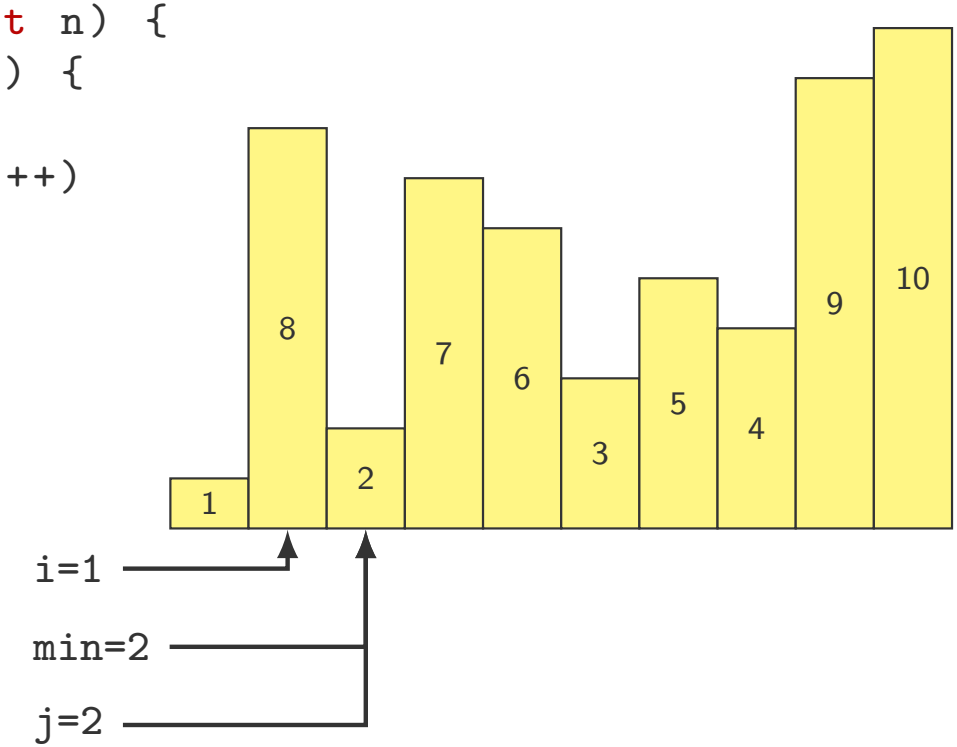
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



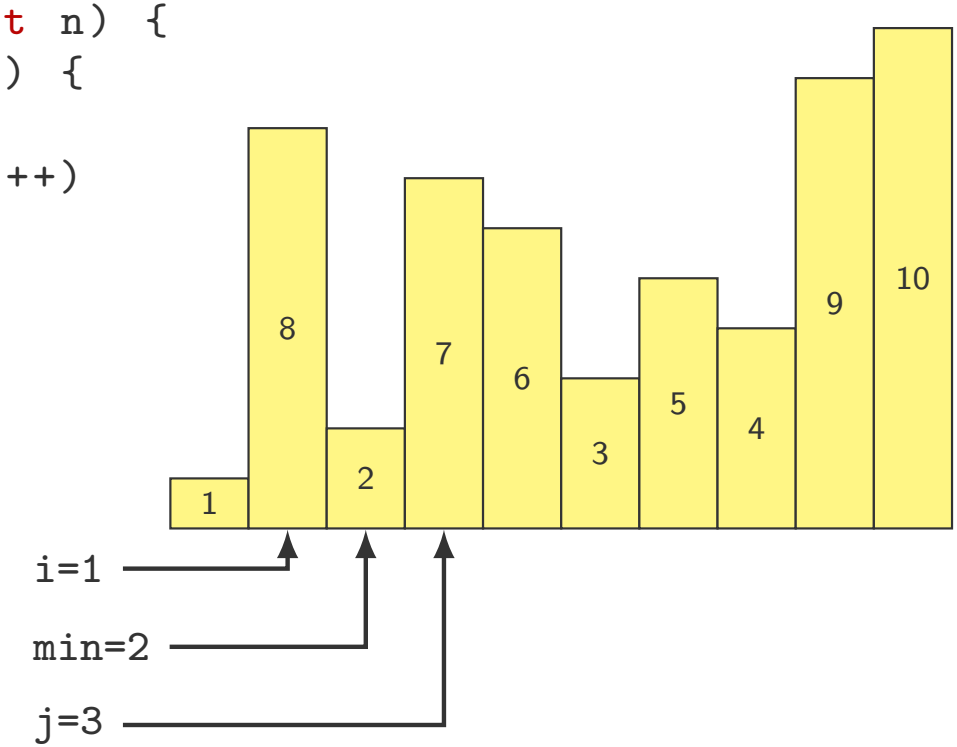
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

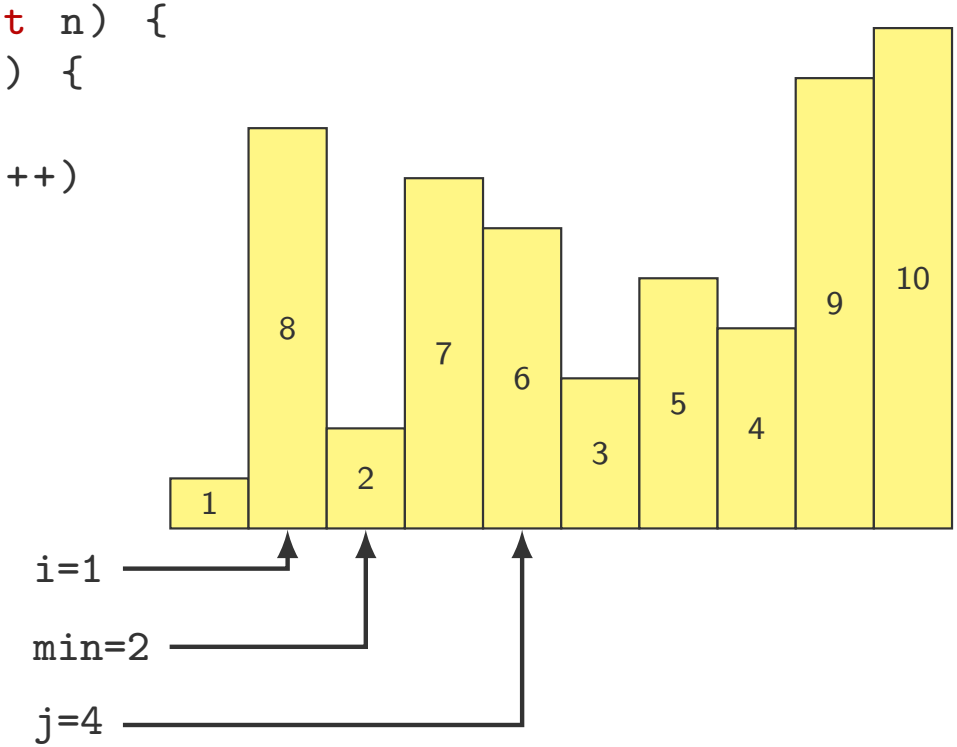


Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



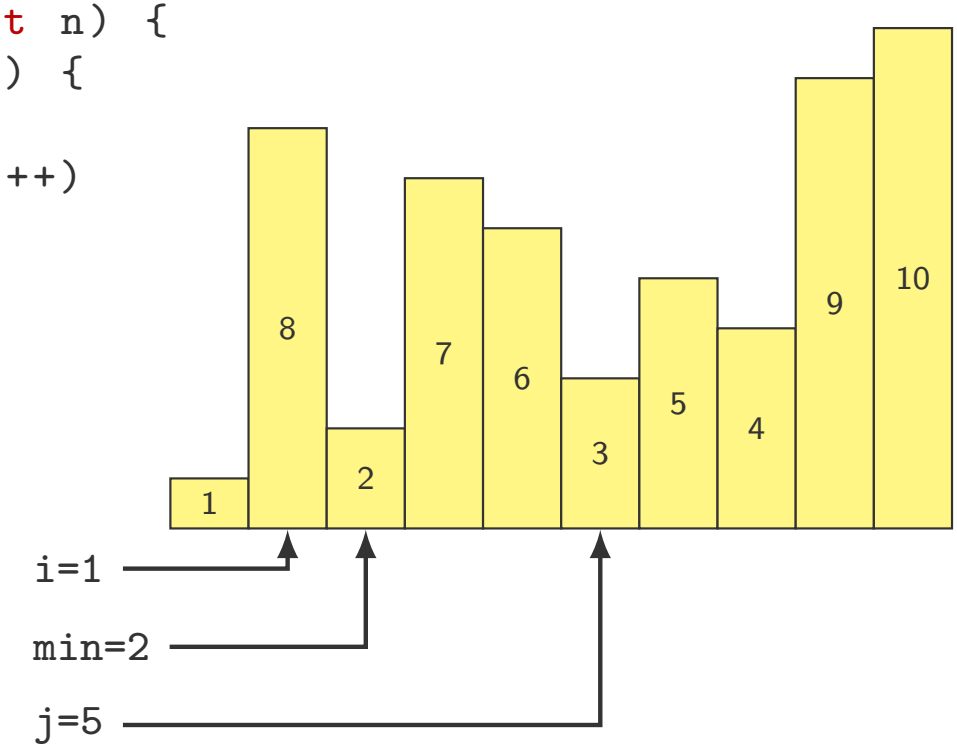
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



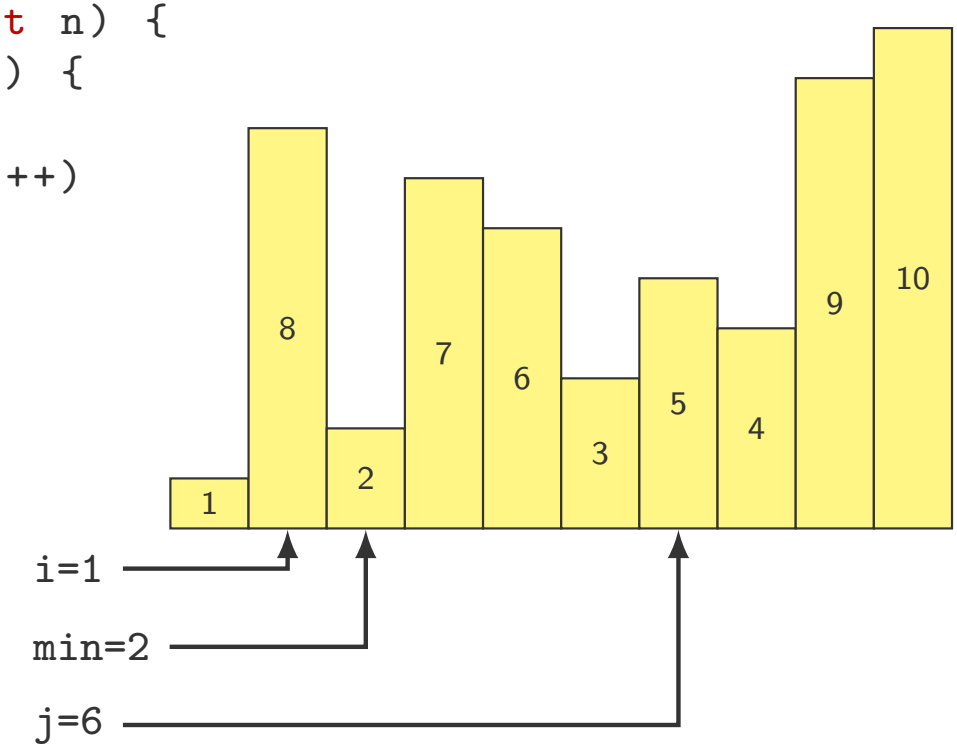
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



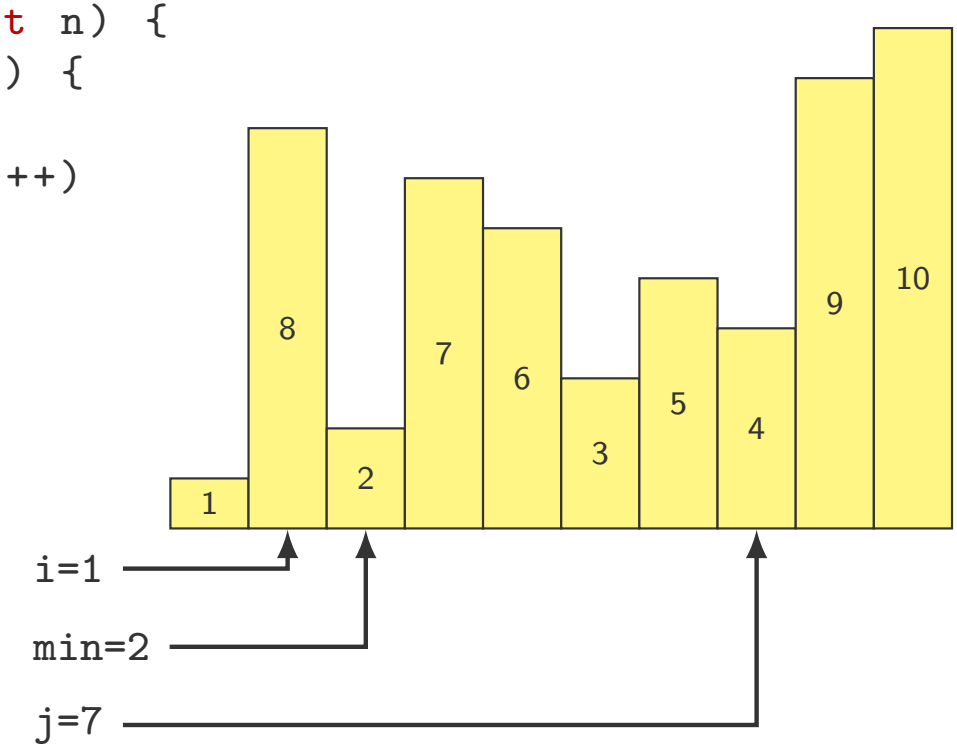
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



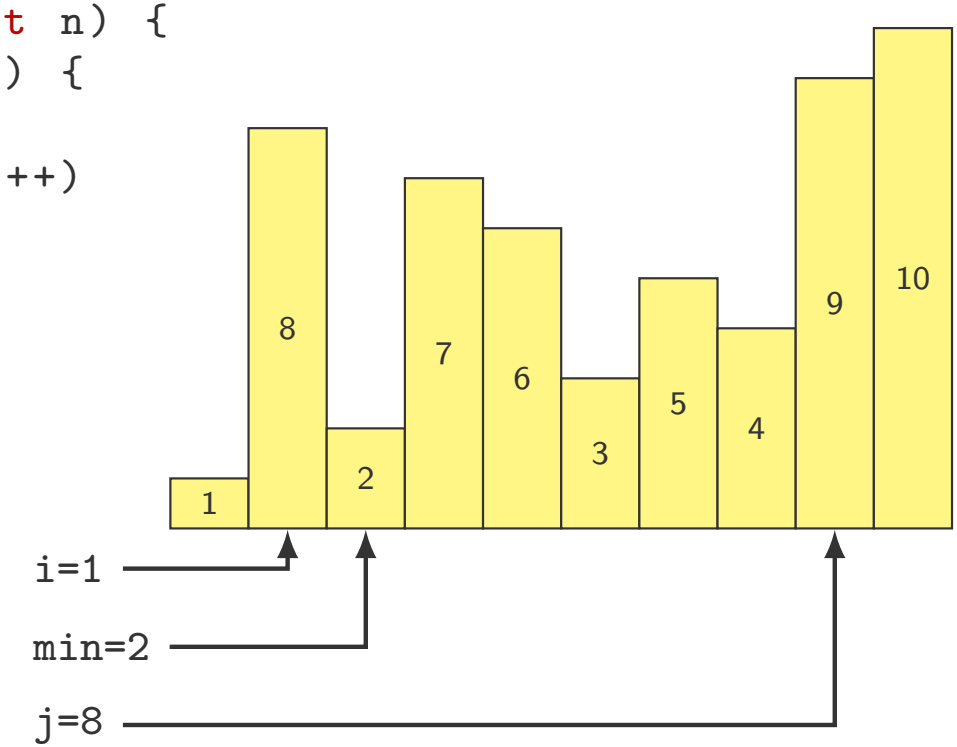
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



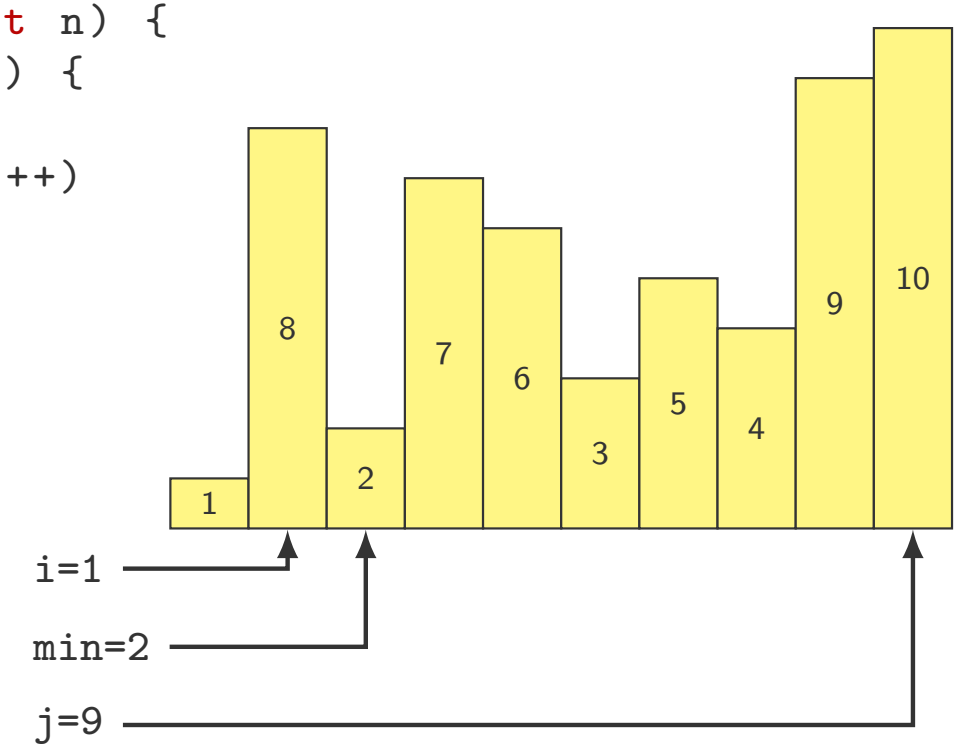
Ordenação por Seleção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



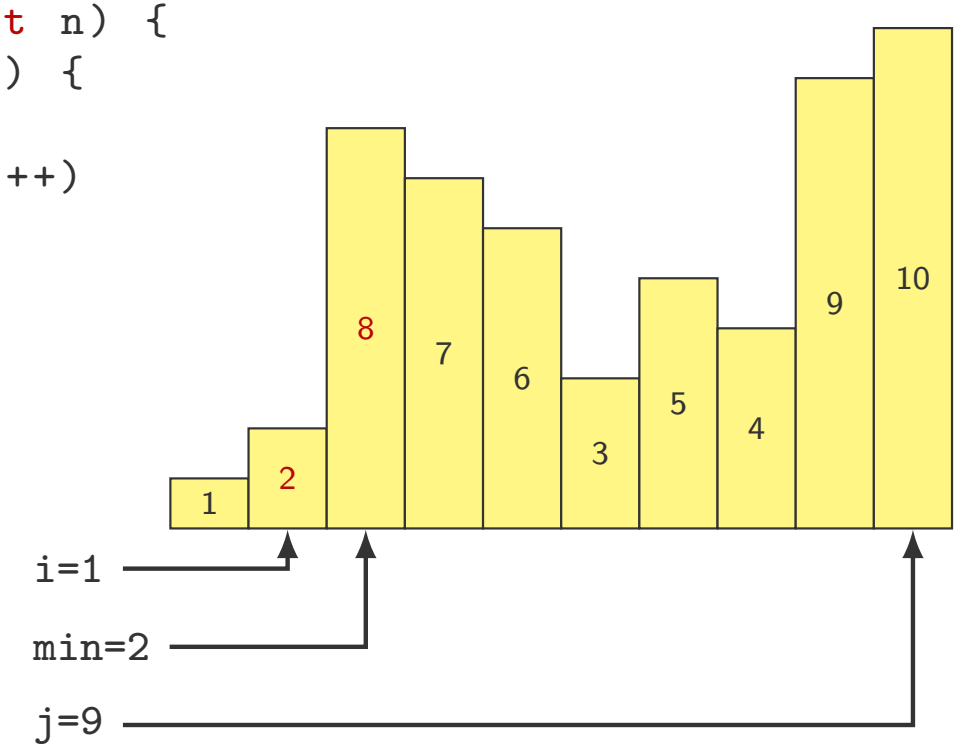
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

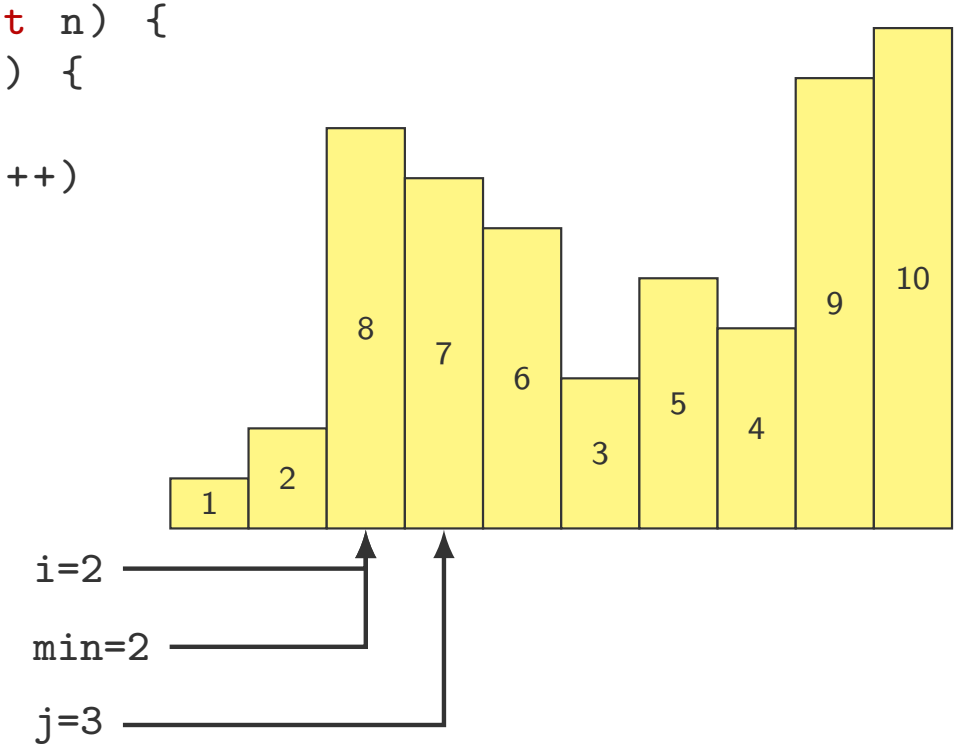


Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



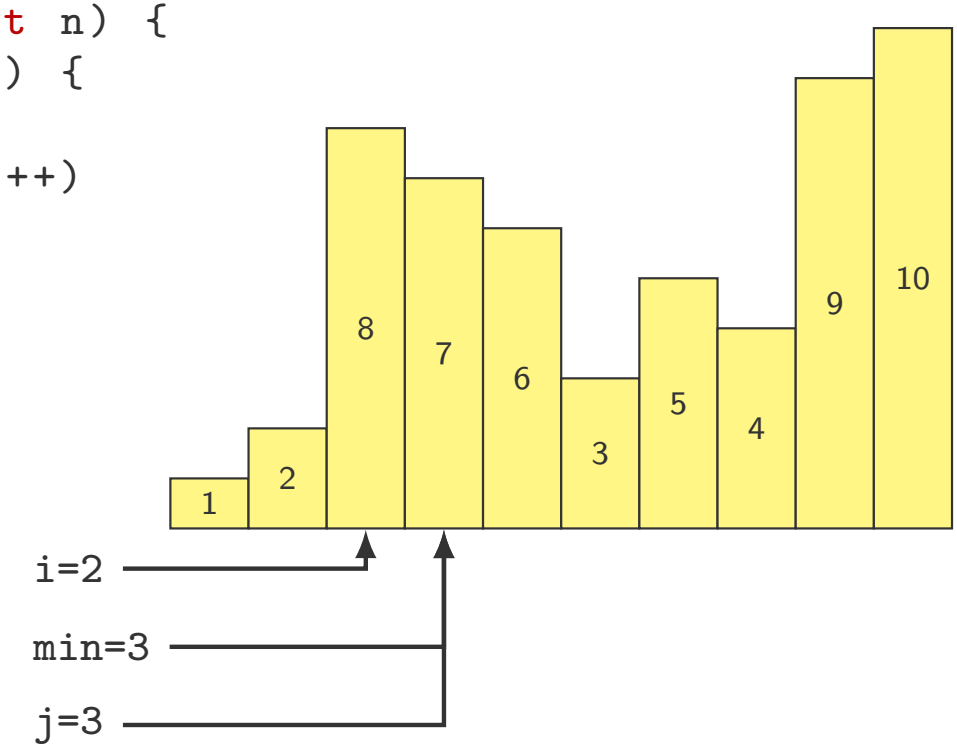
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



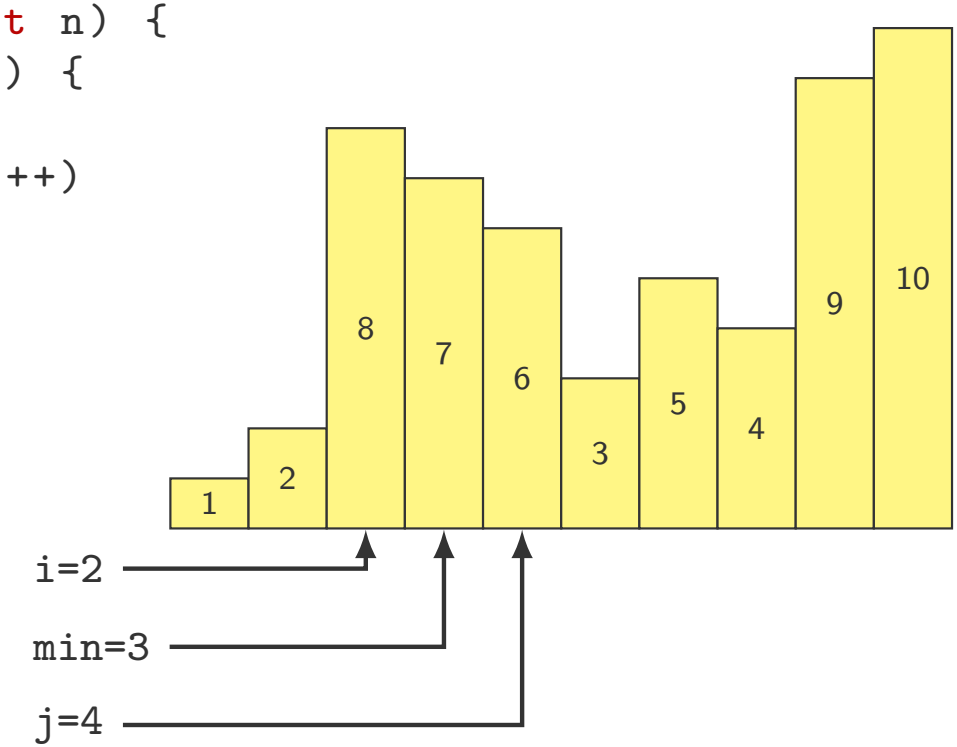
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



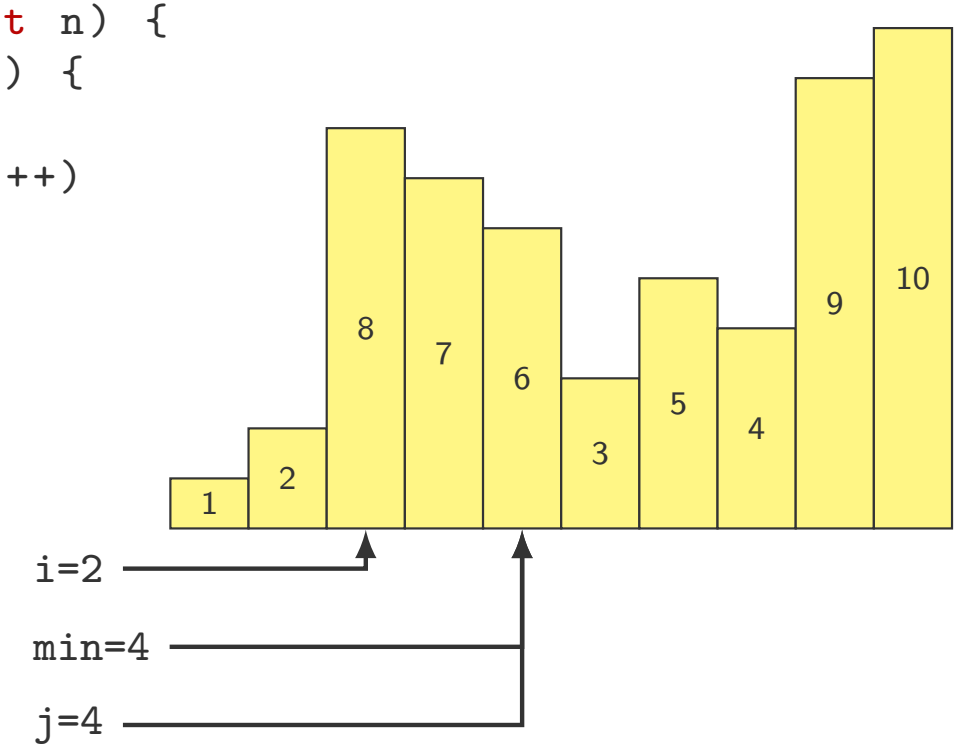
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



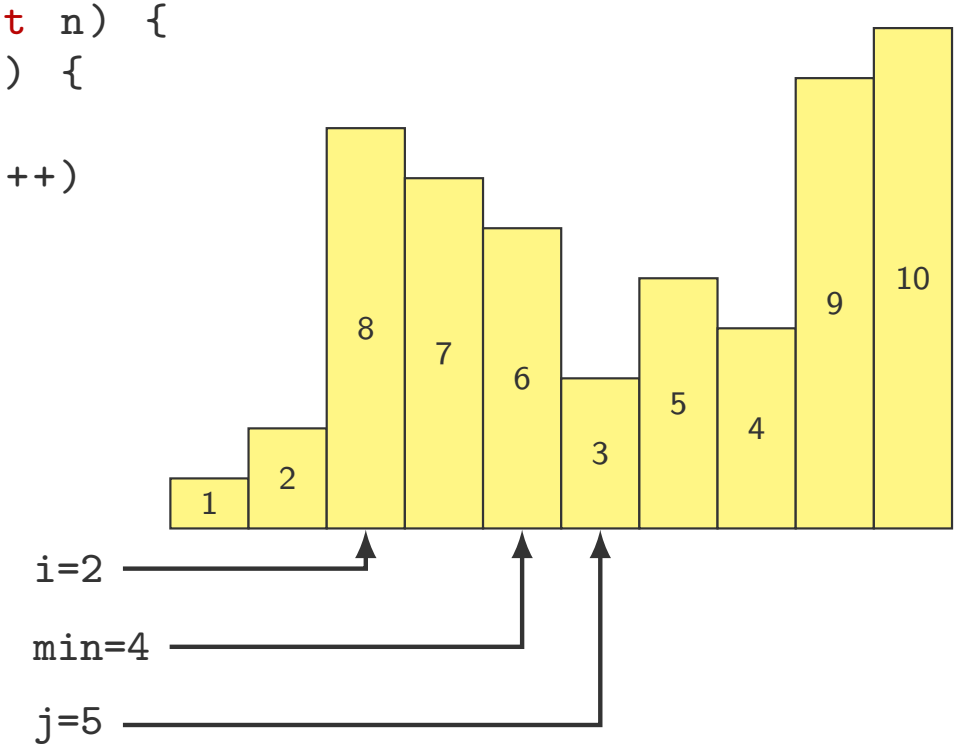
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



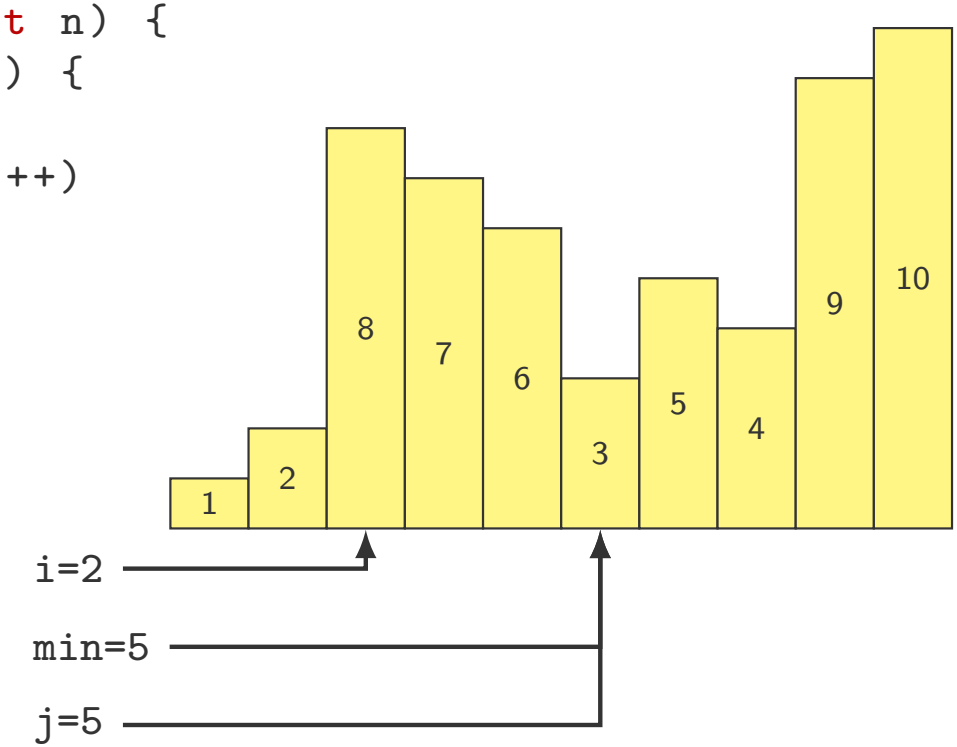
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



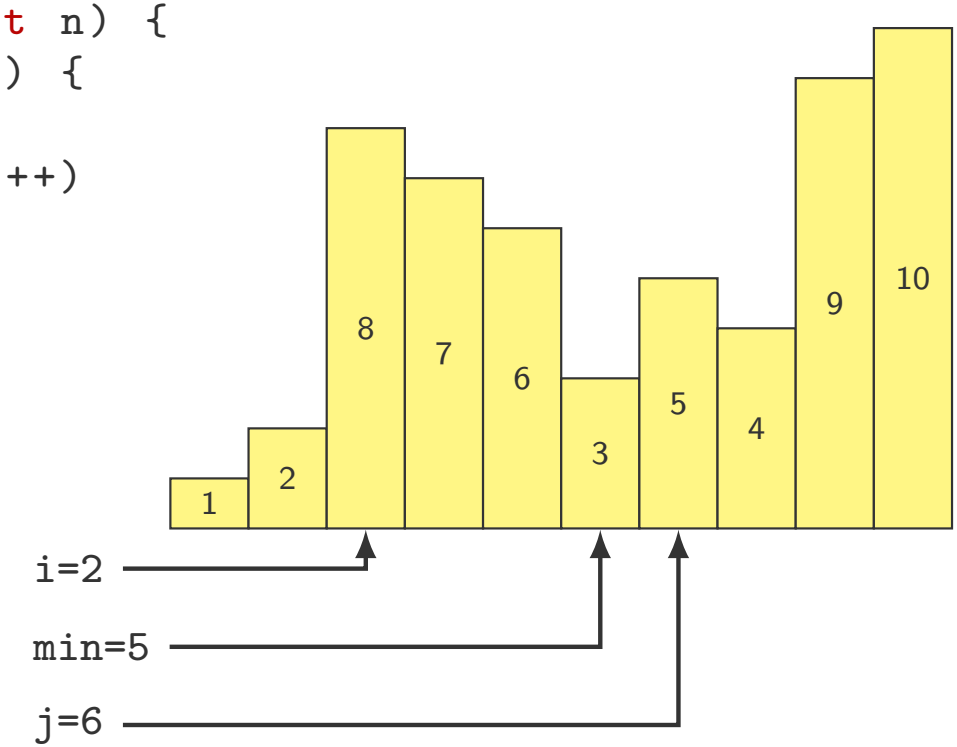
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



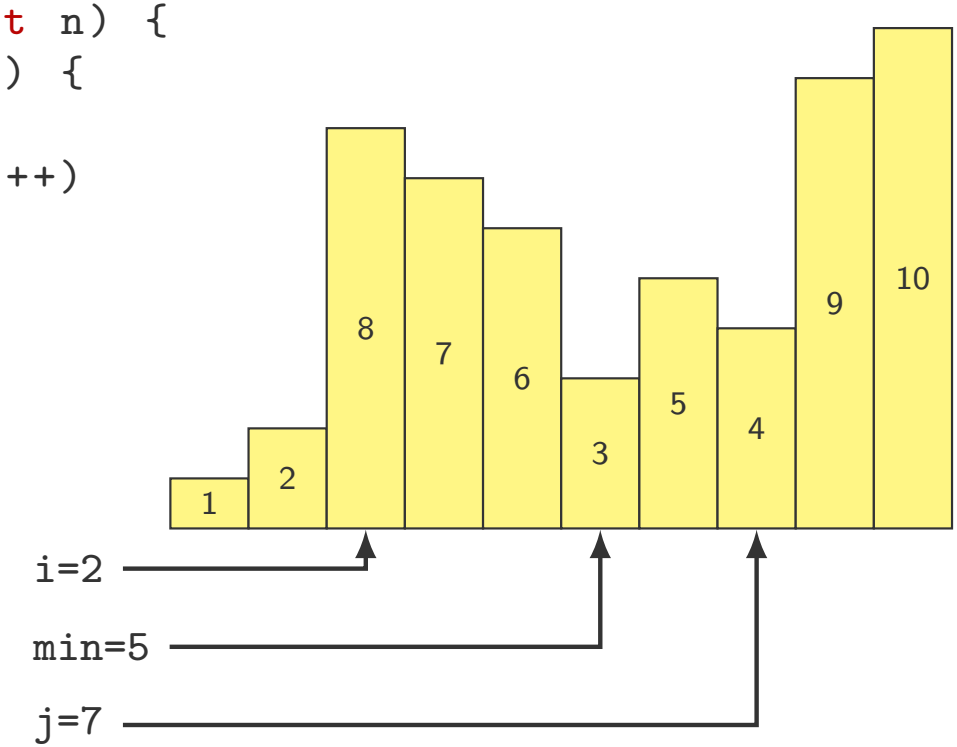
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



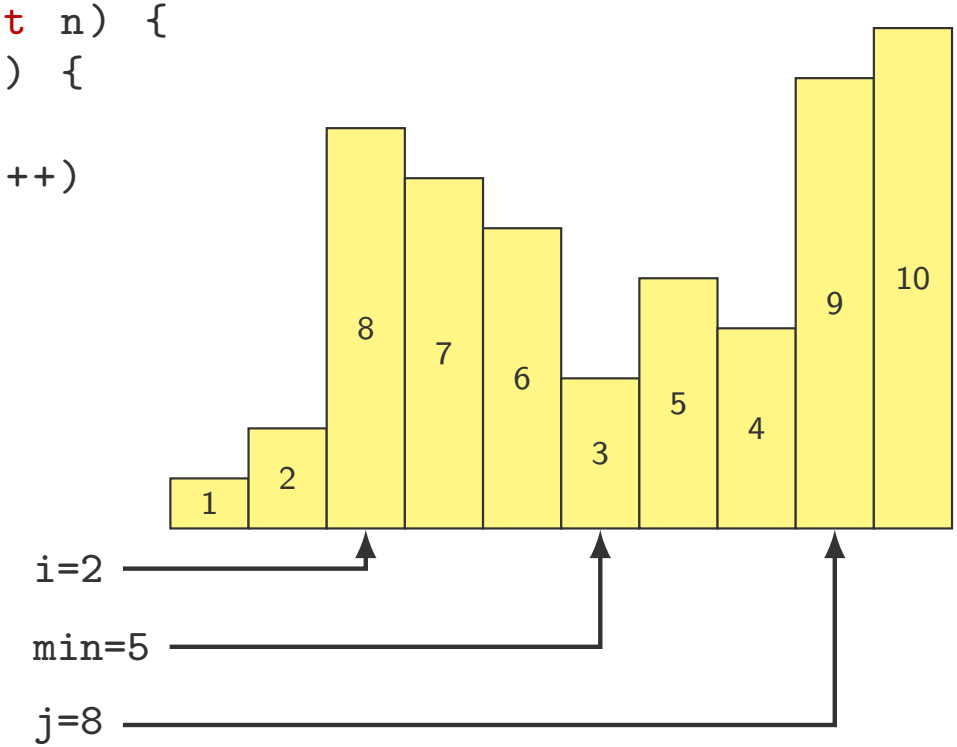
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



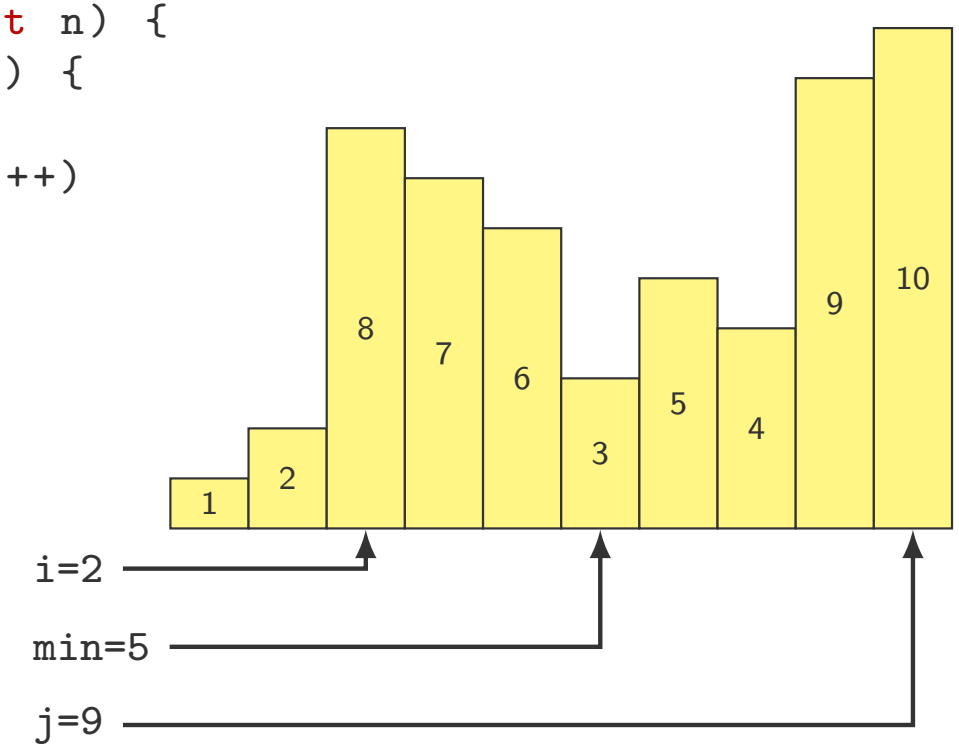
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



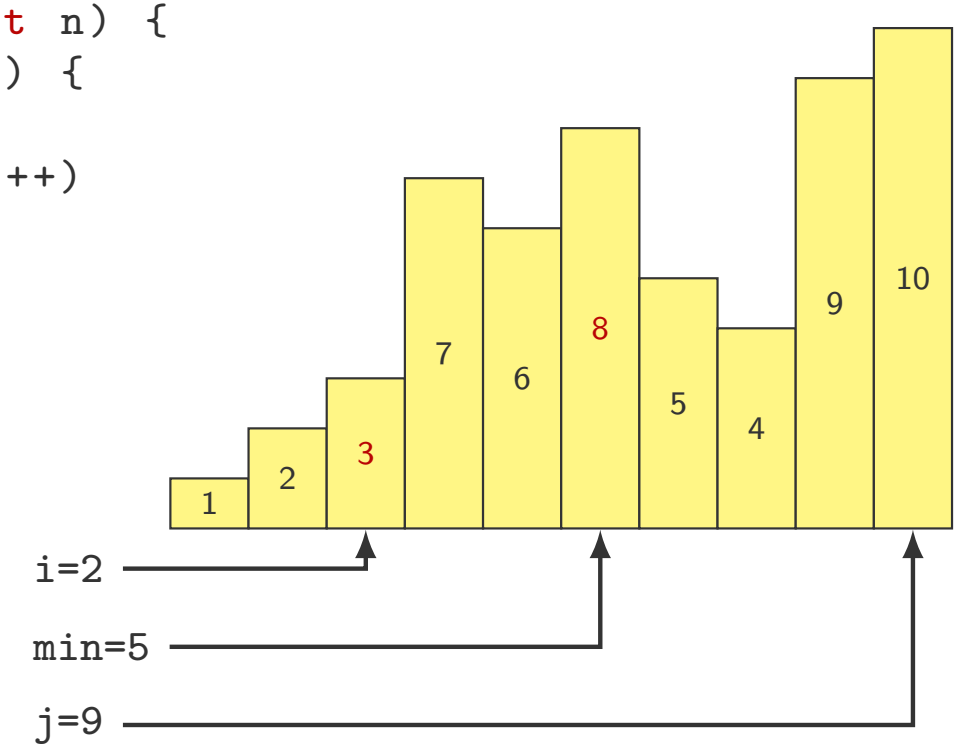
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



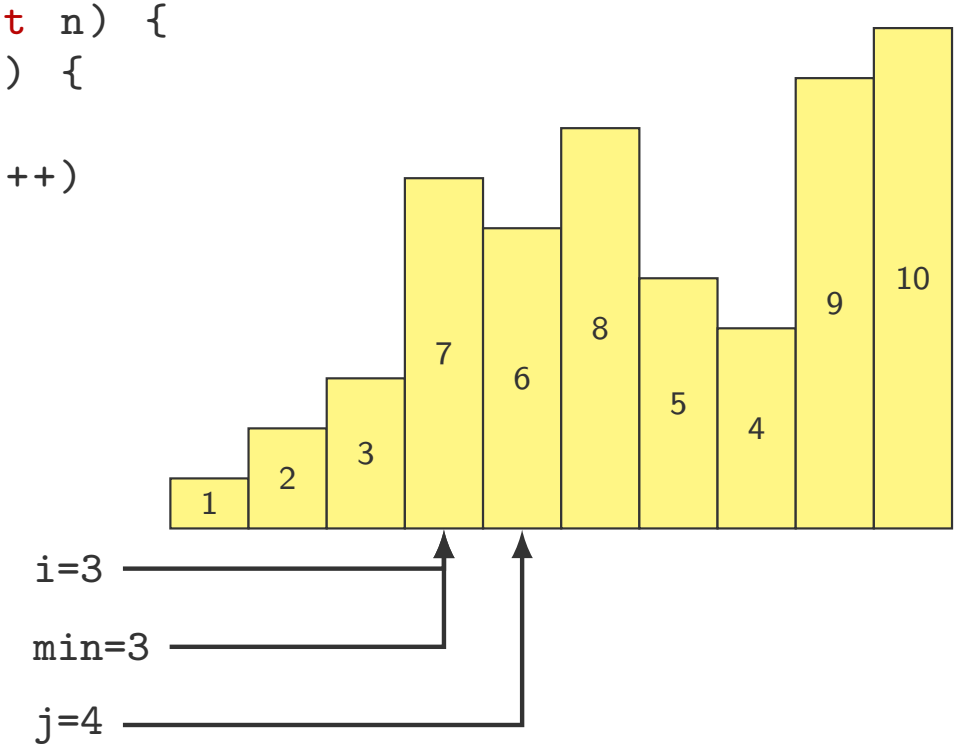
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



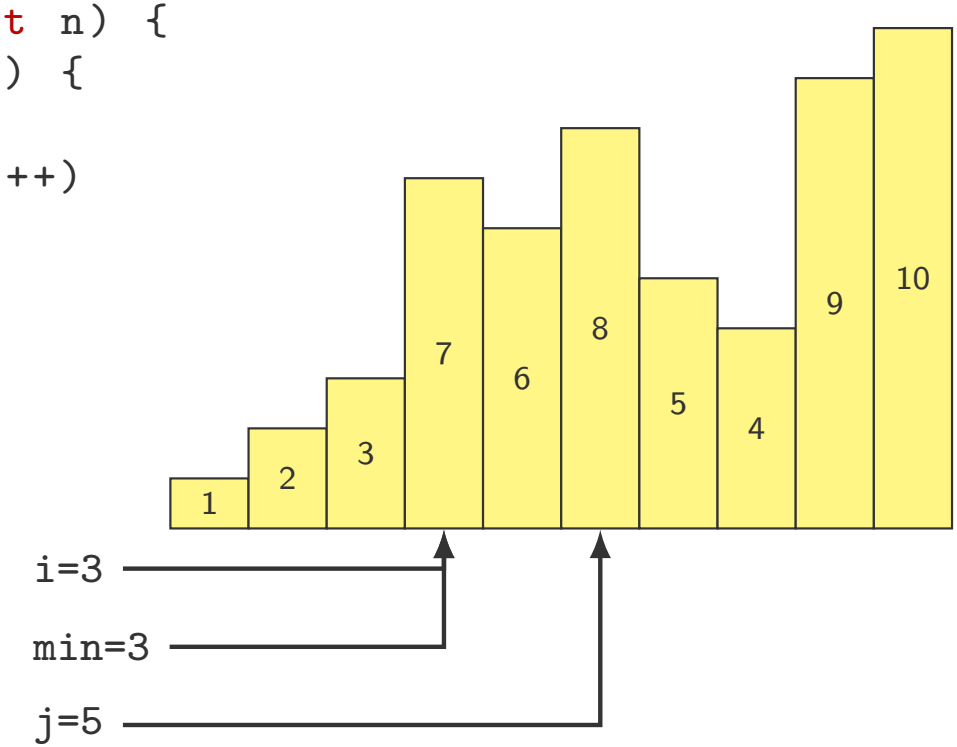
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



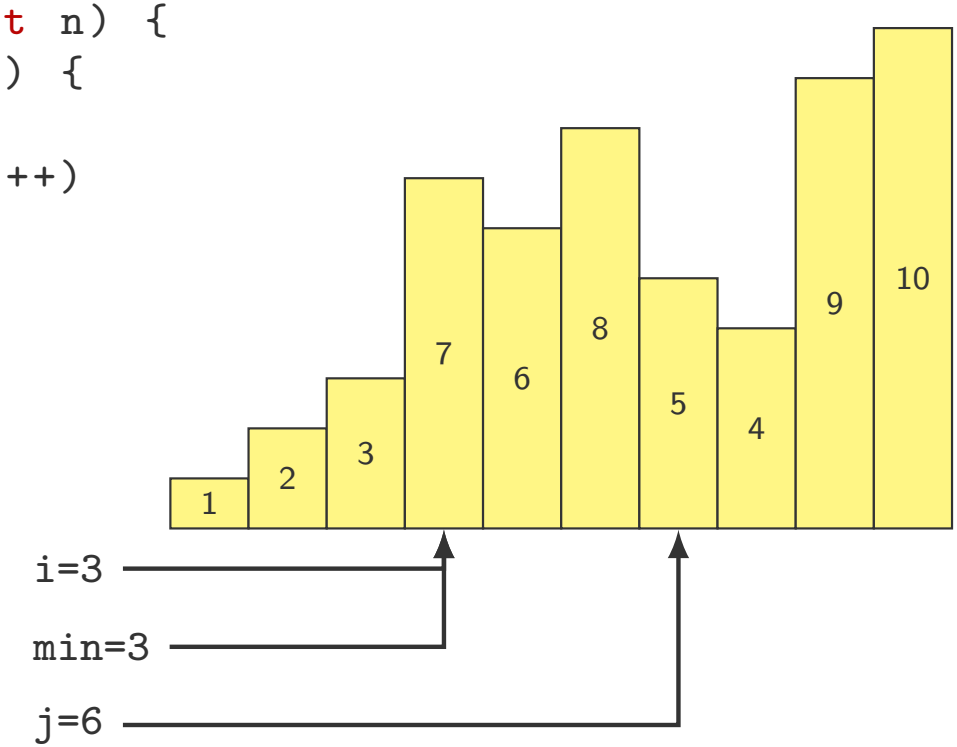
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

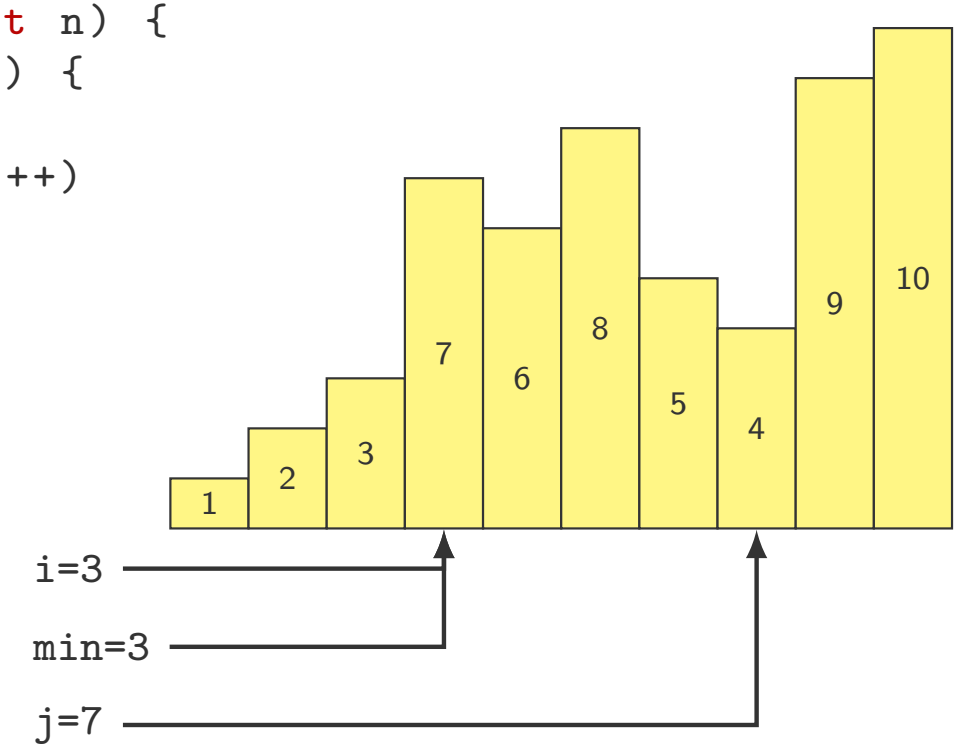


Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

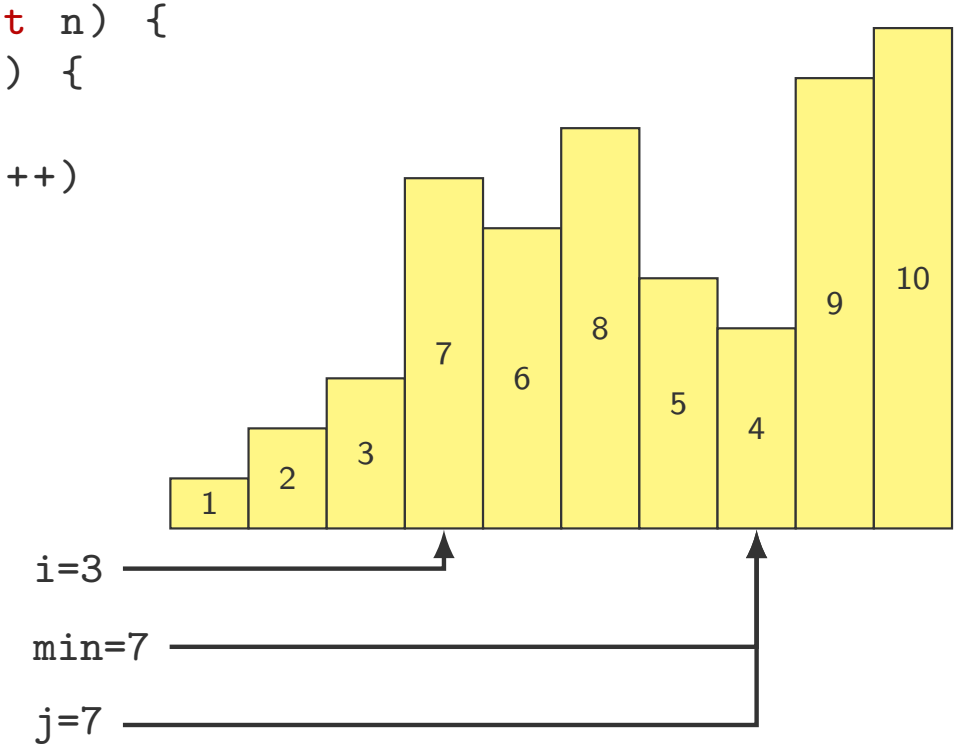


Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

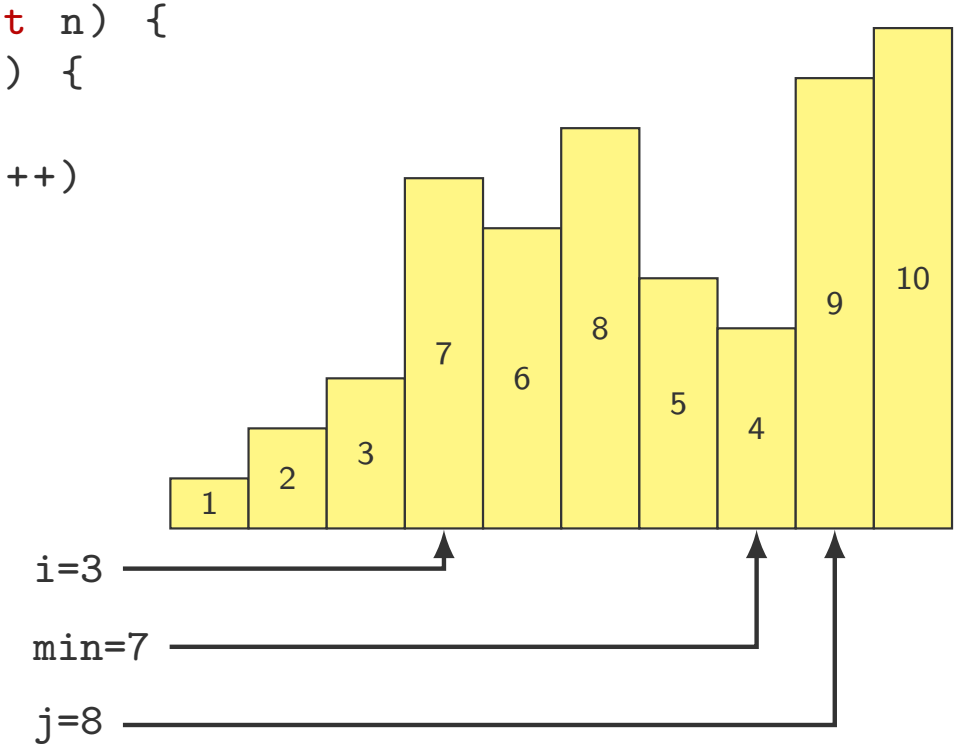


Ordenação por Seleção

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



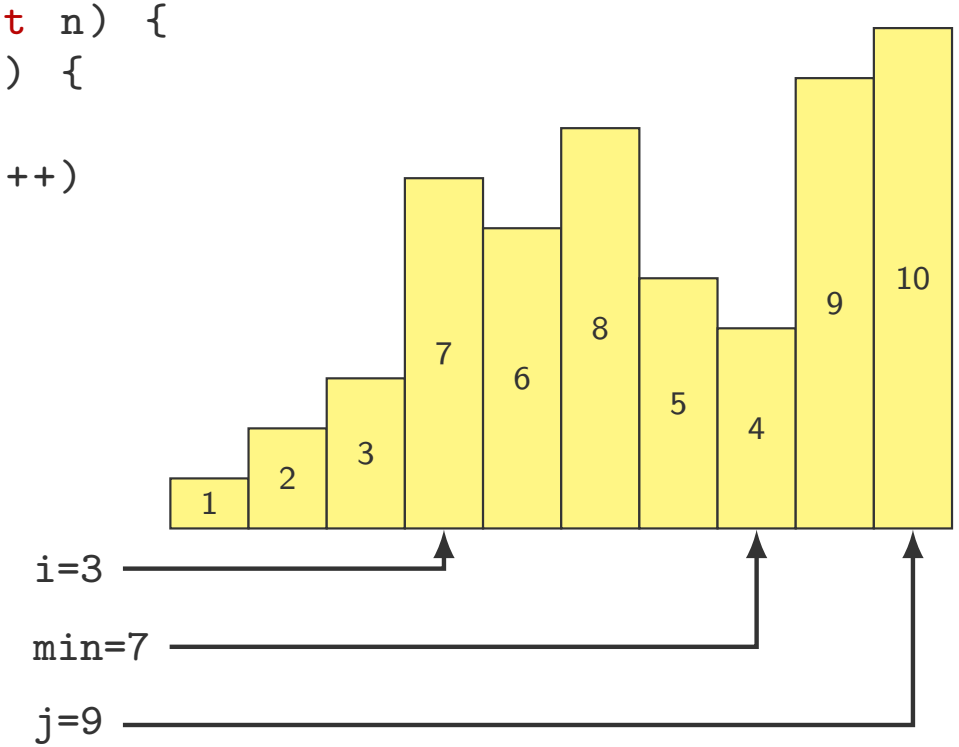
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



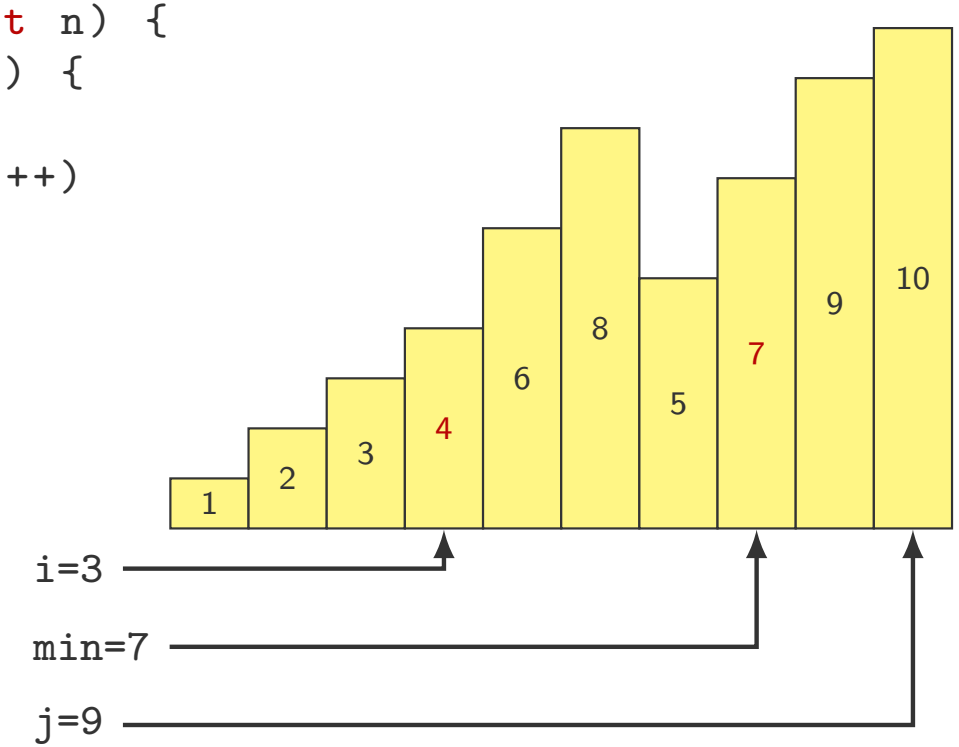
Ordenação por Seleção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



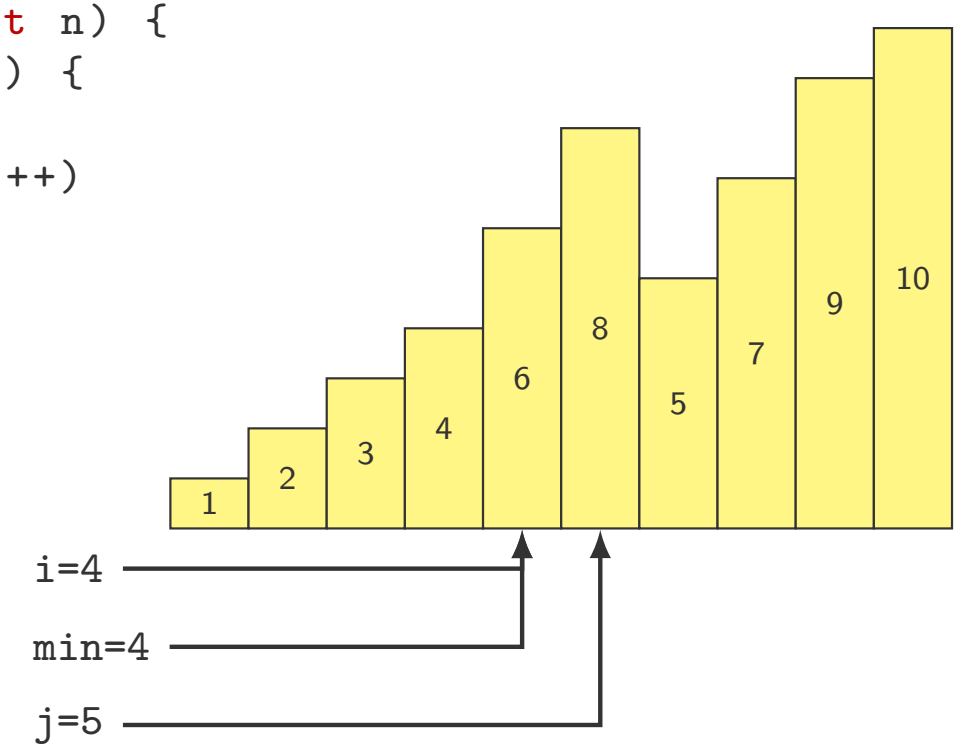
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



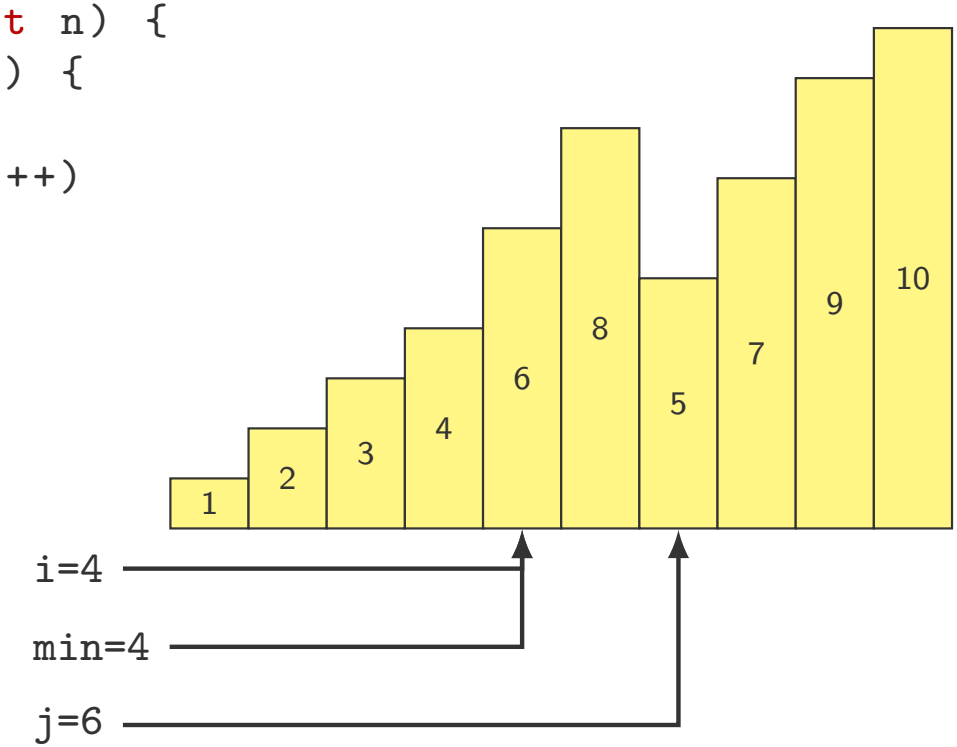
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



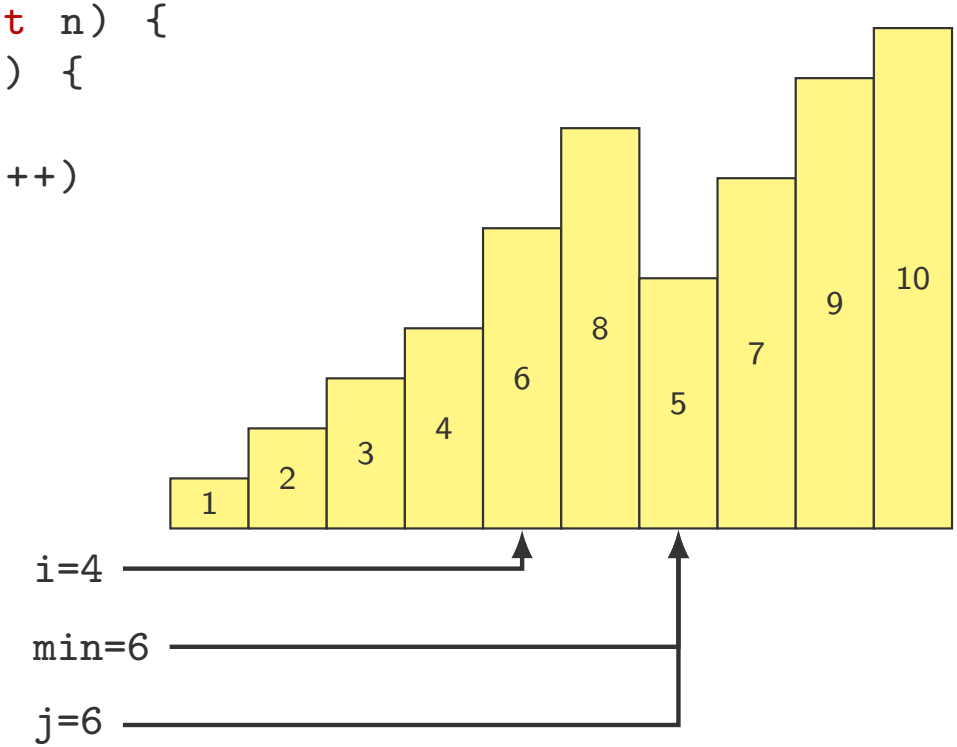
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



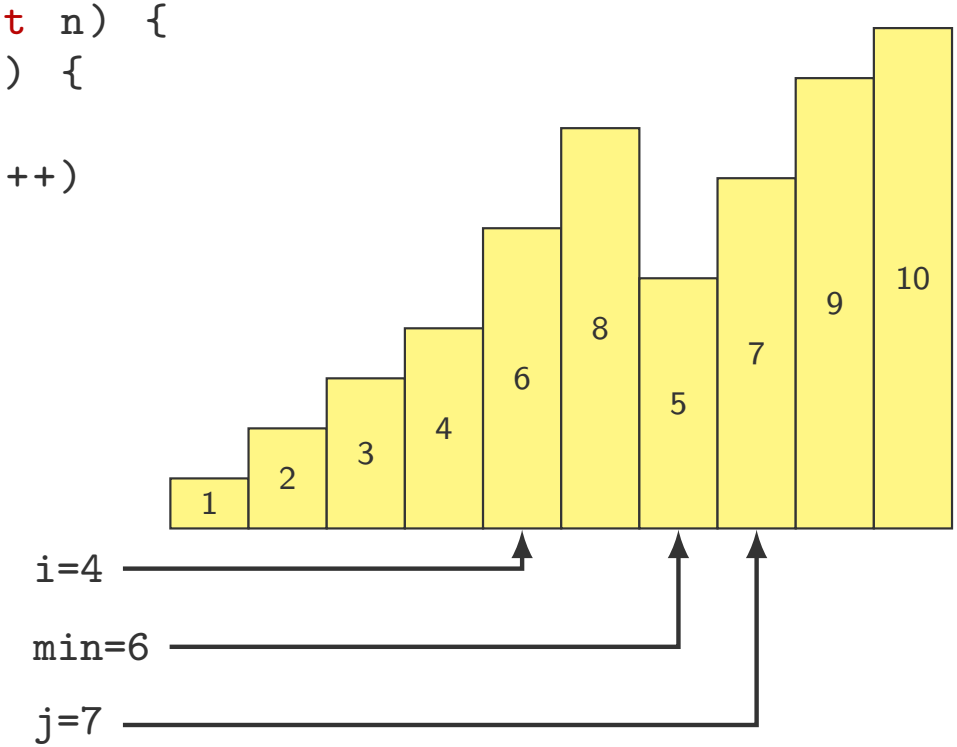
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



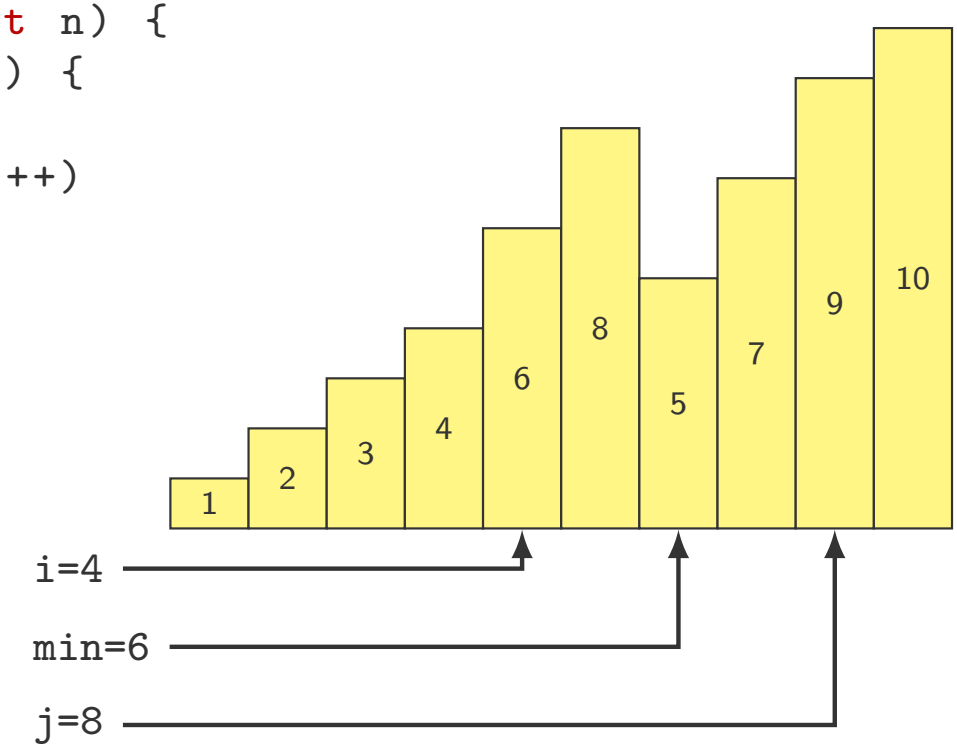
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



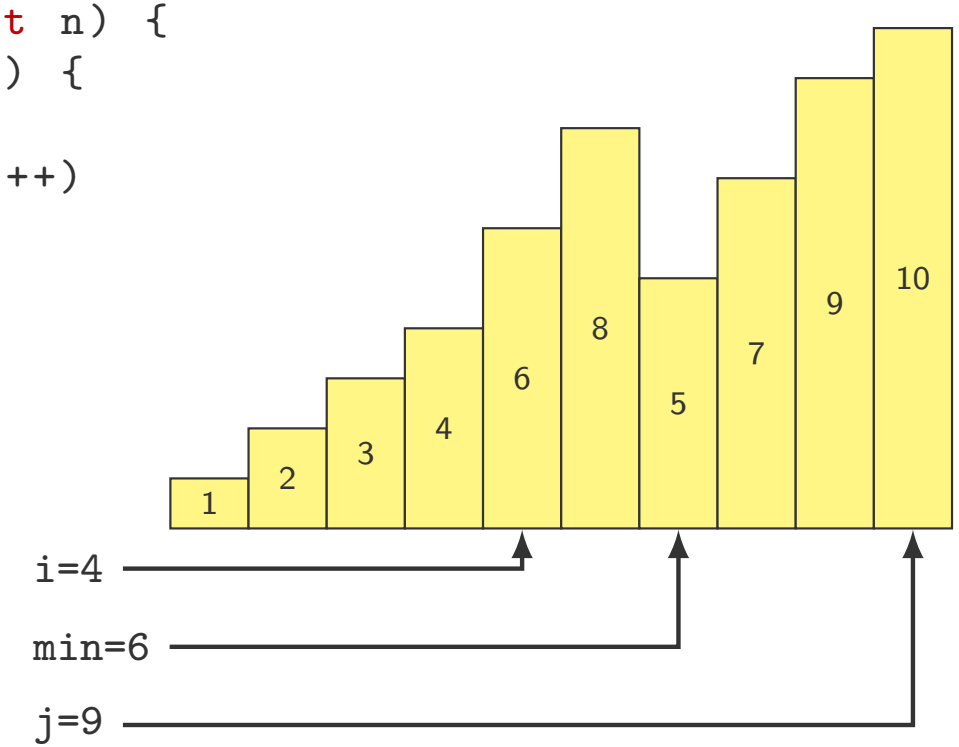
Ordenação por Seleção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



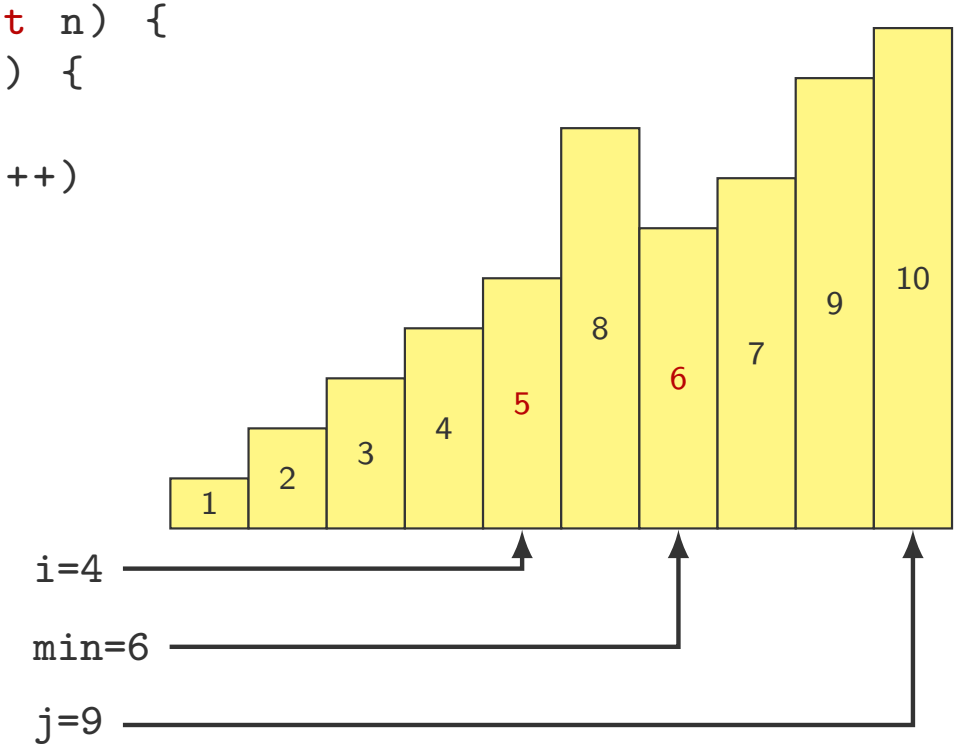
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



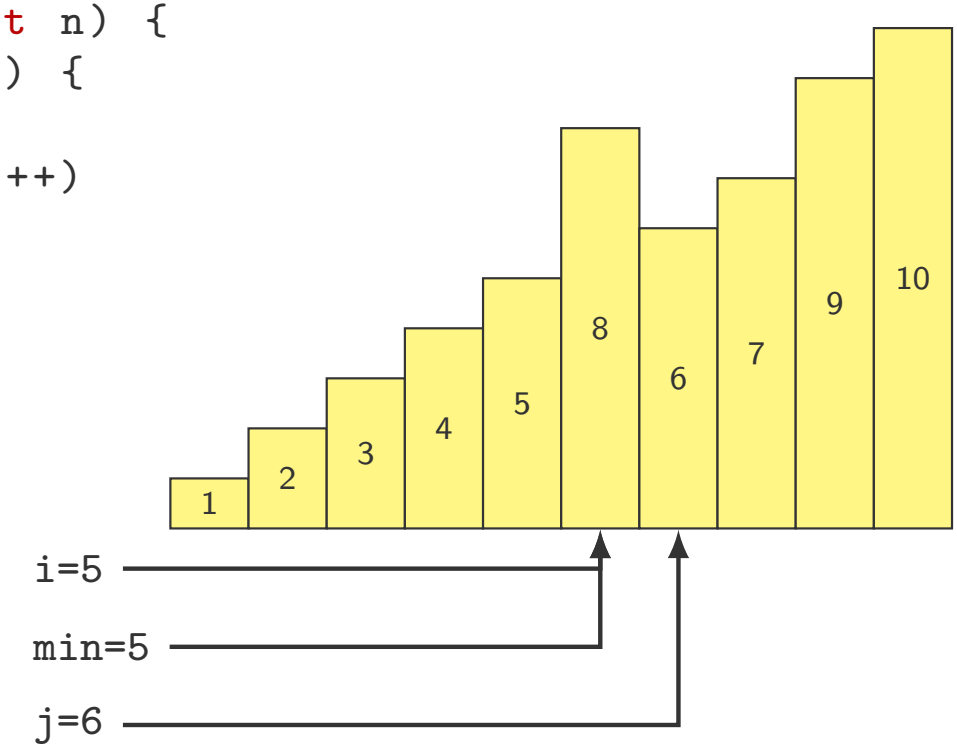
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



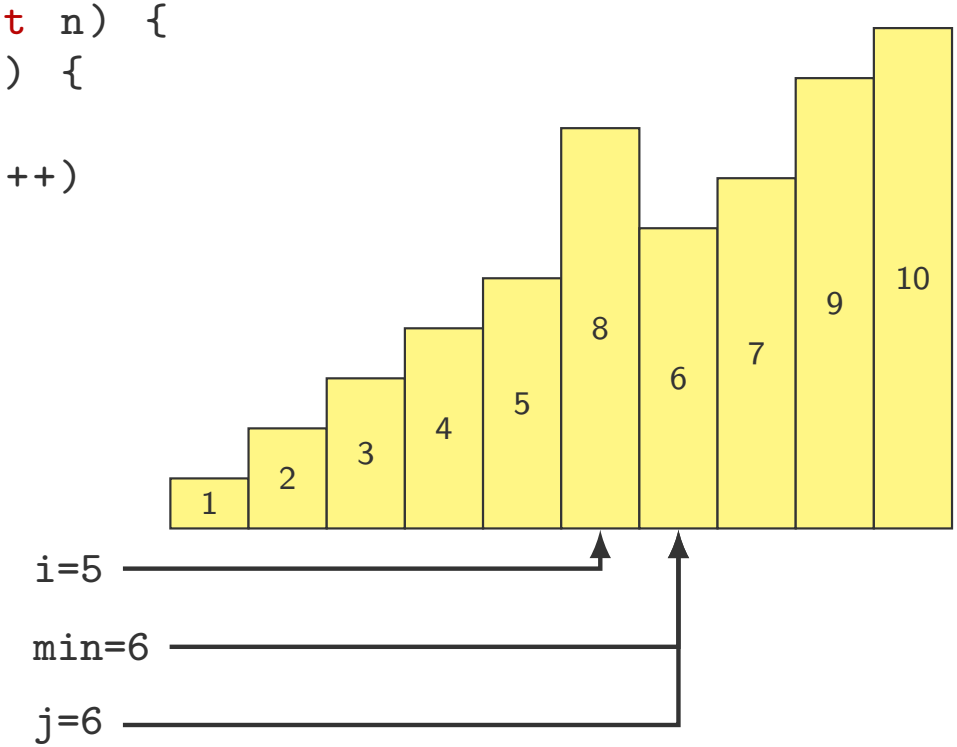
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



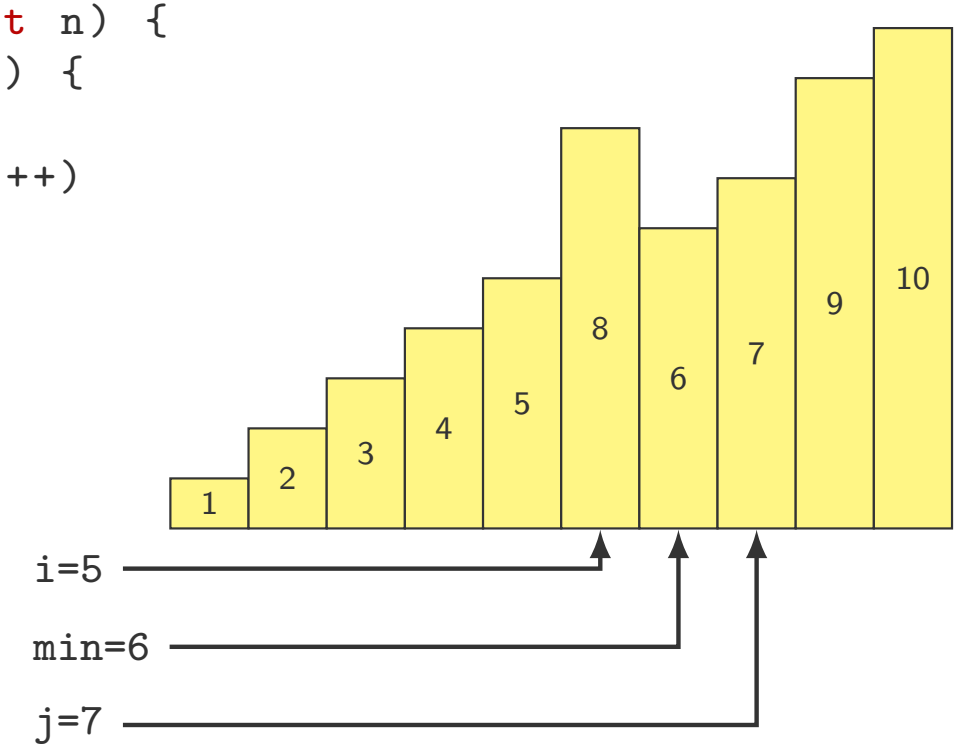
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



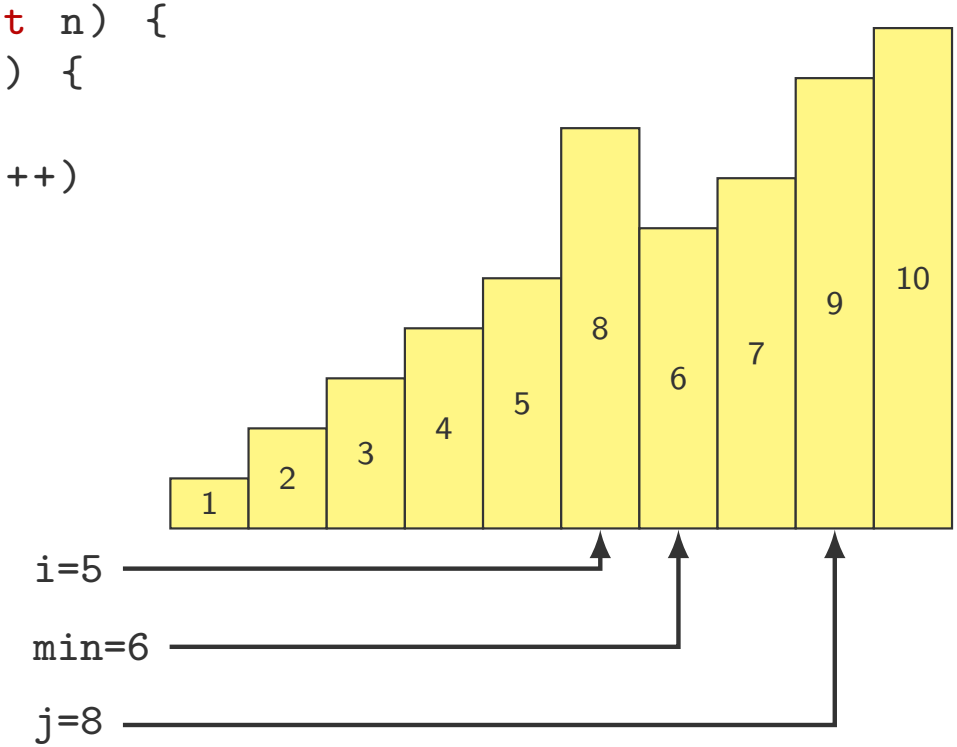
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



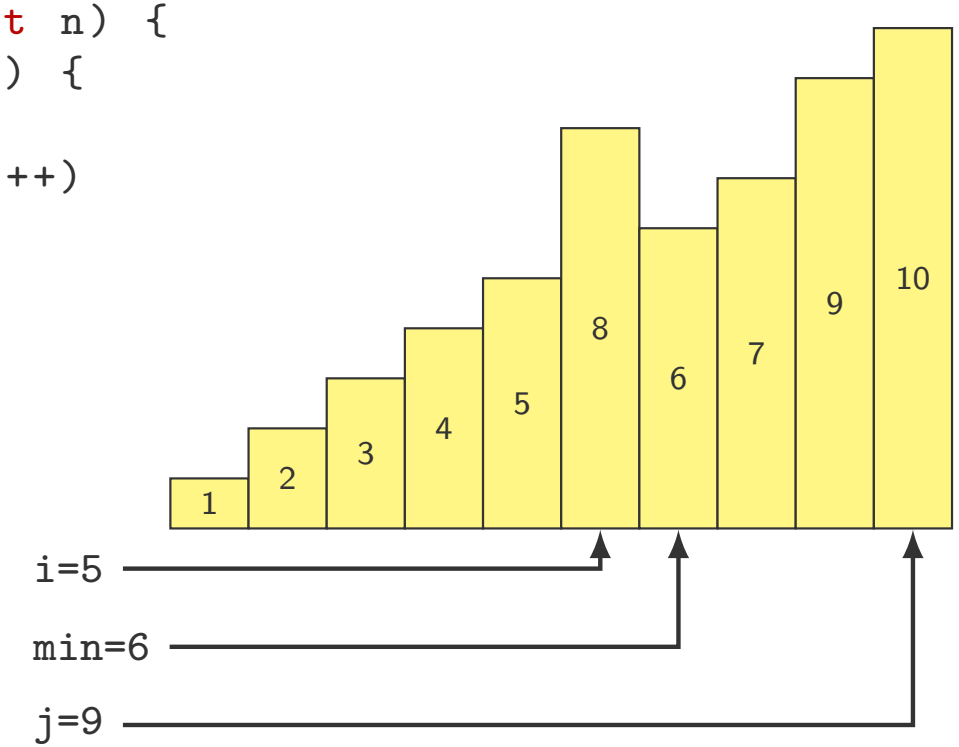
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



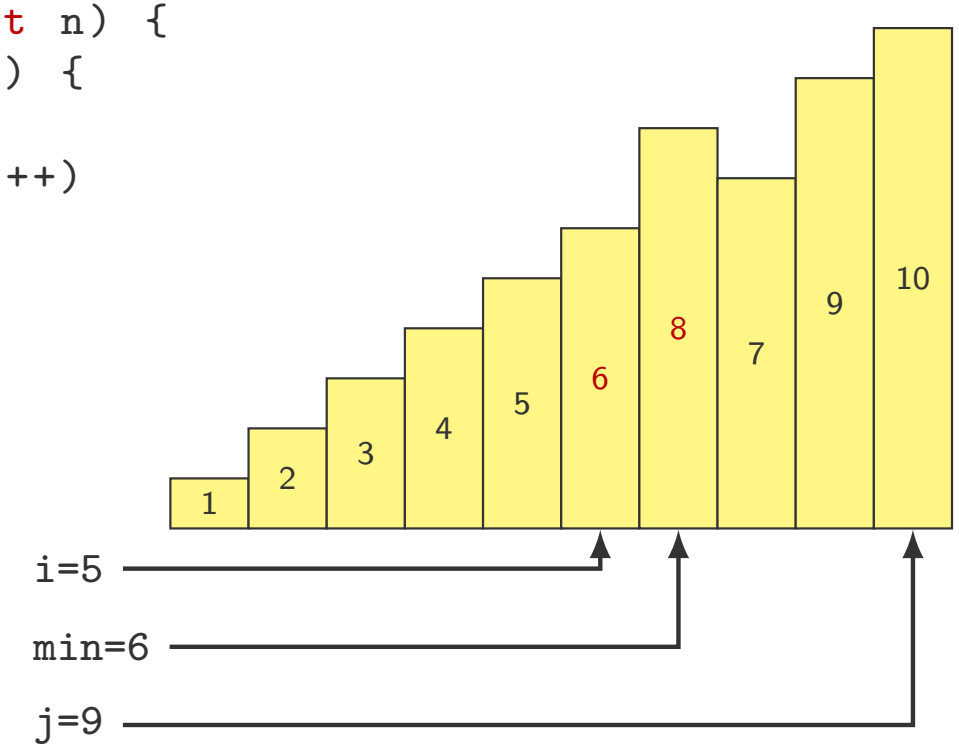
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



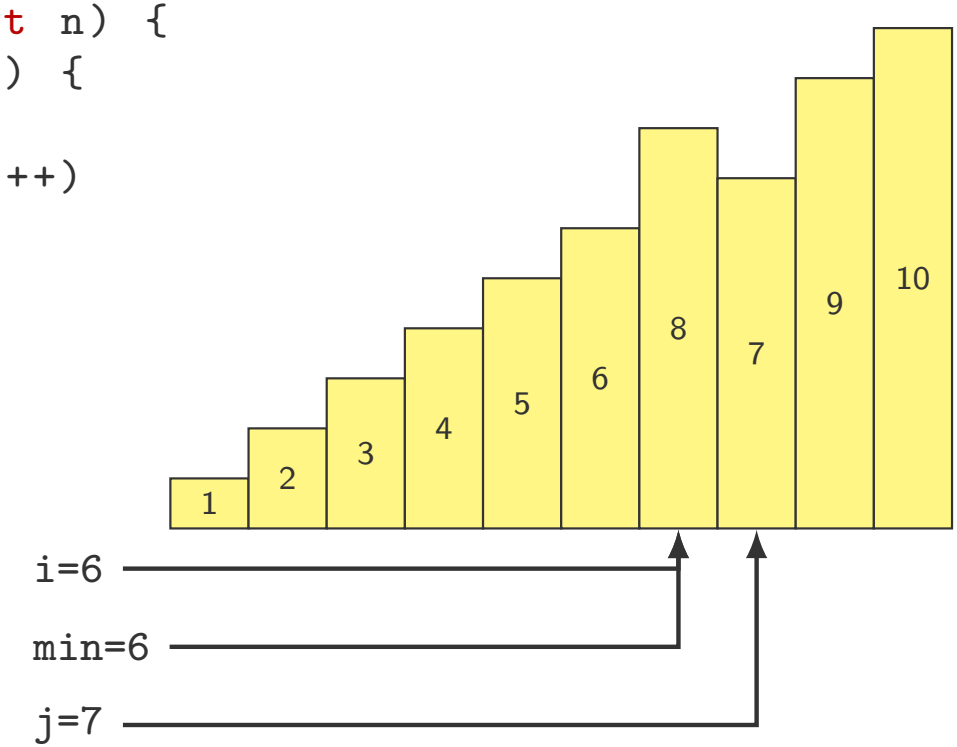
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



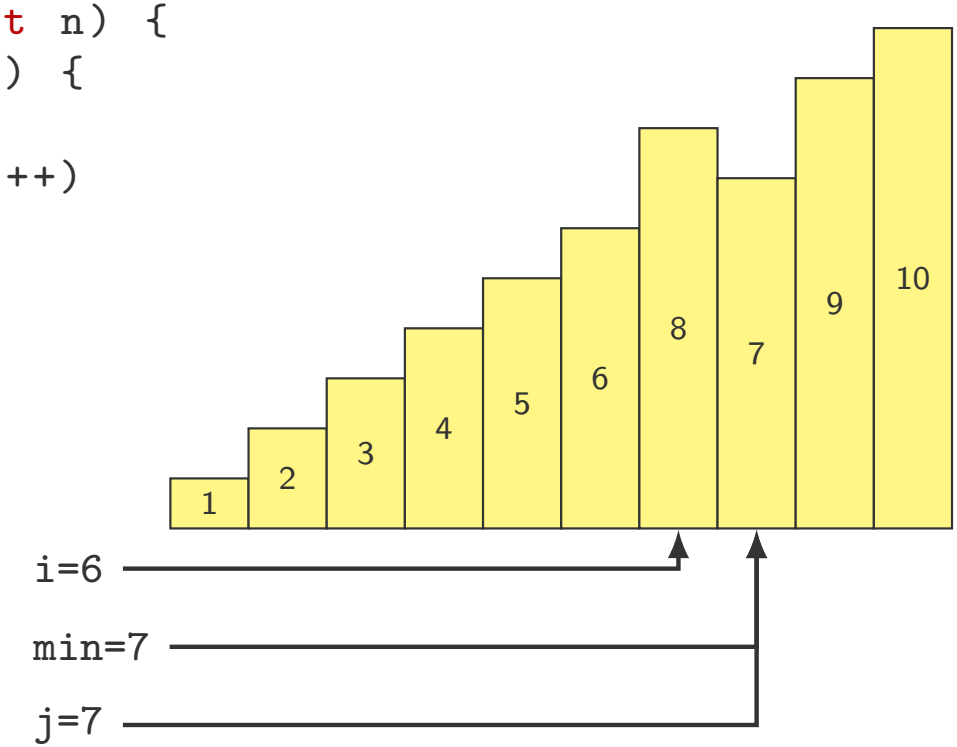
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



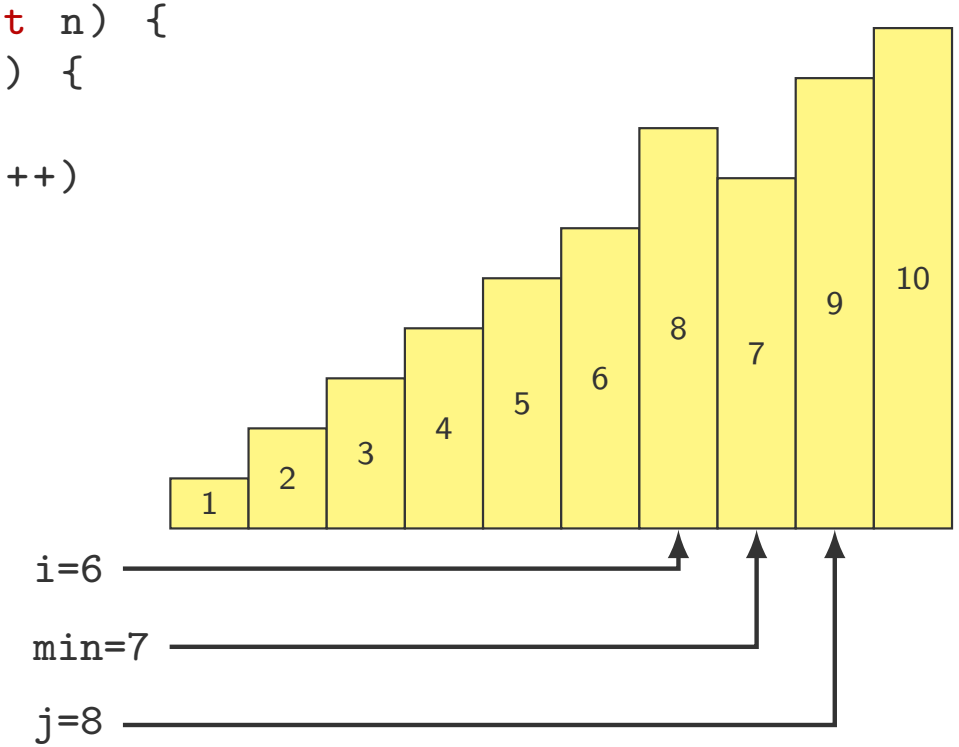
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



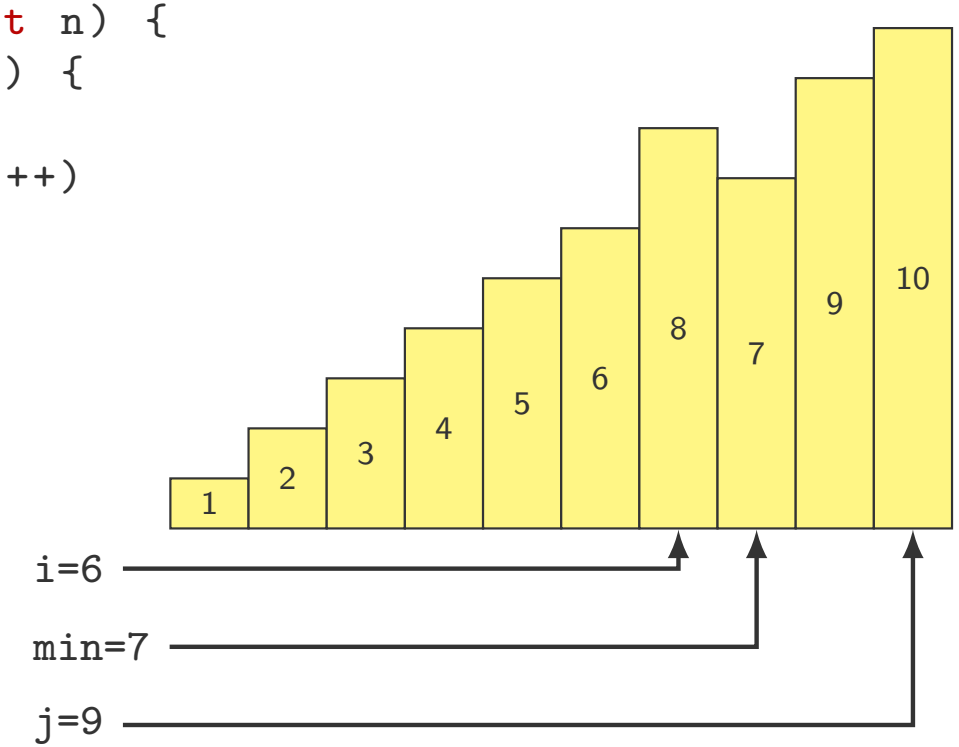
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



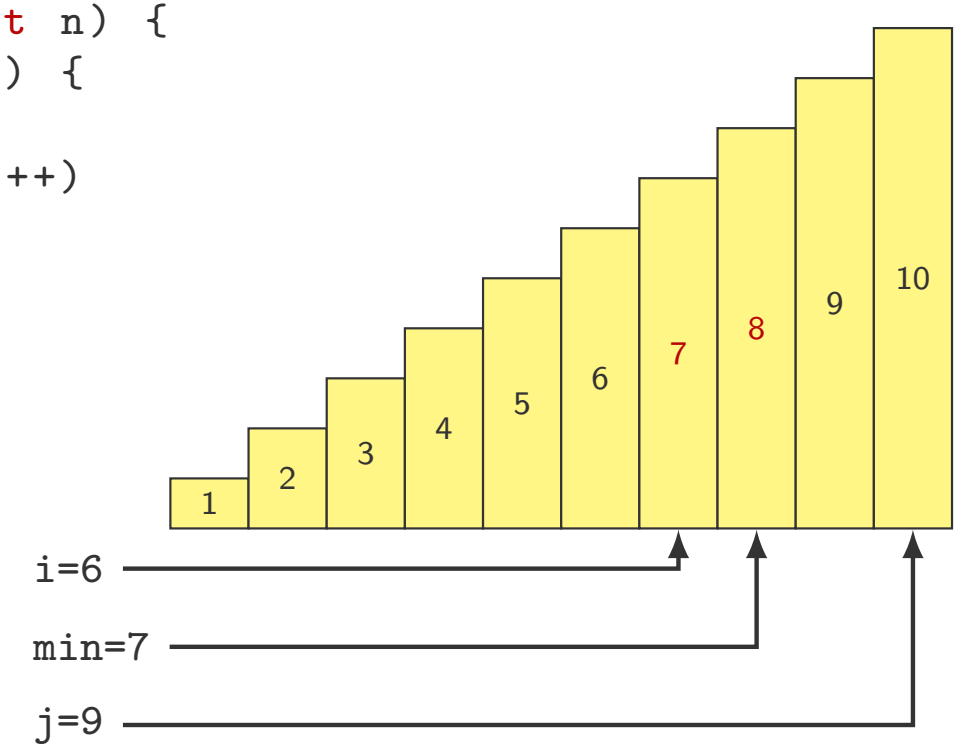
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



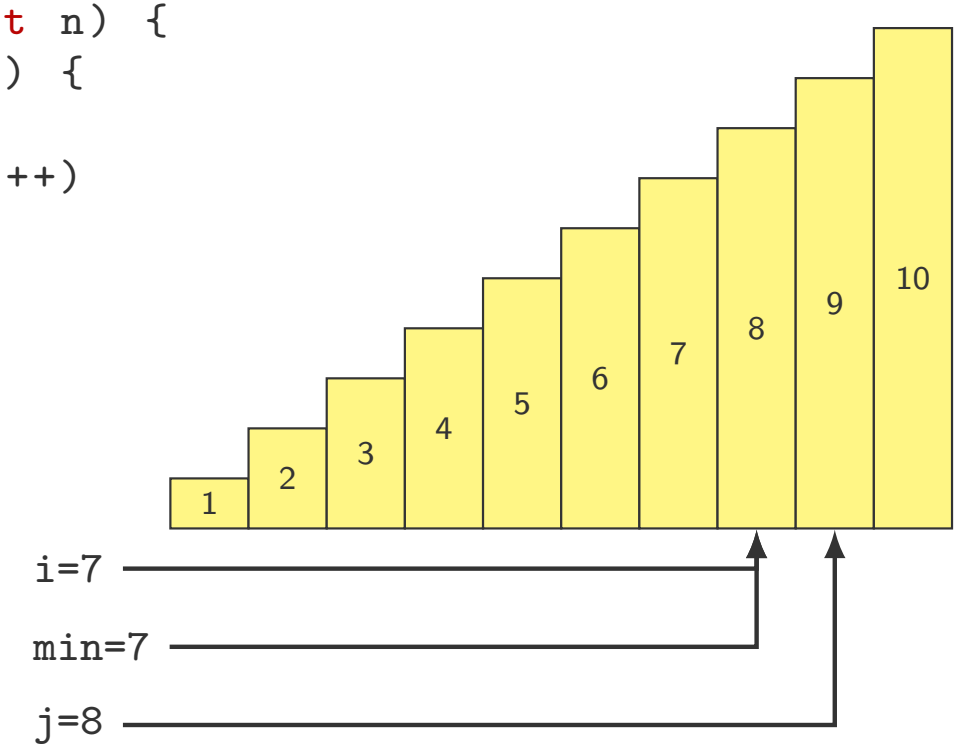
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



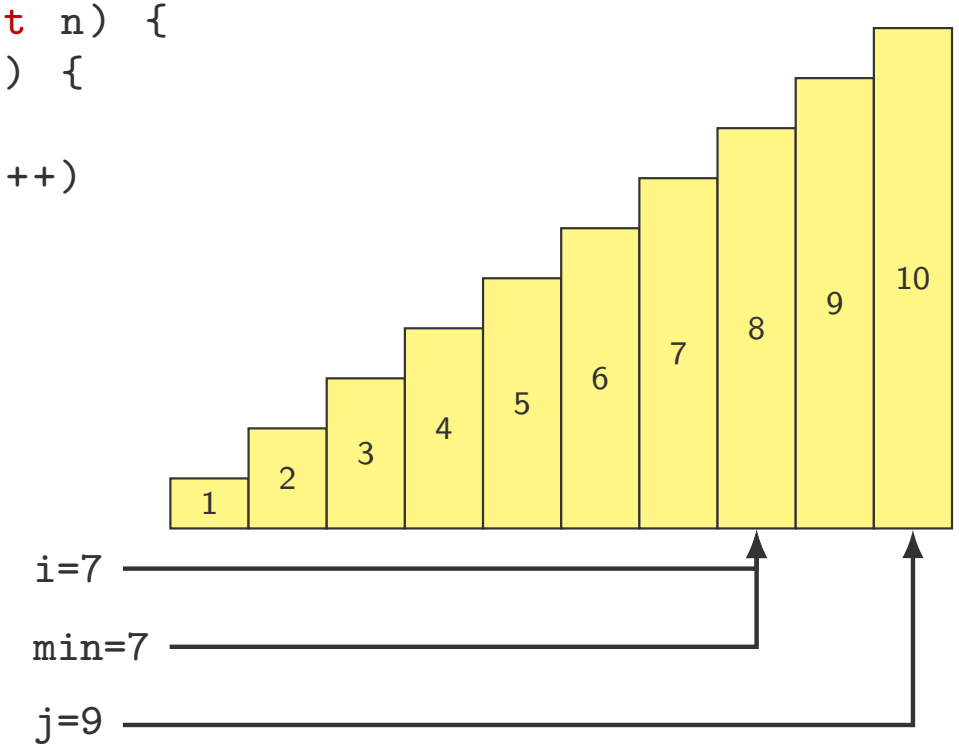
Ordenação por Seleção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



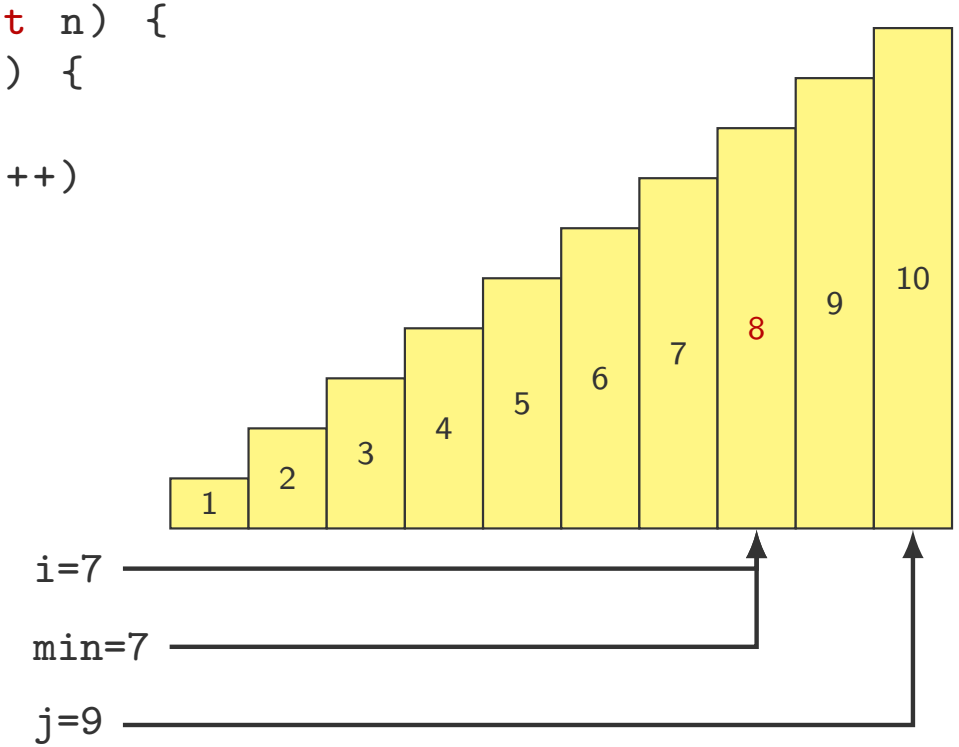
Ordenação por Seleção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



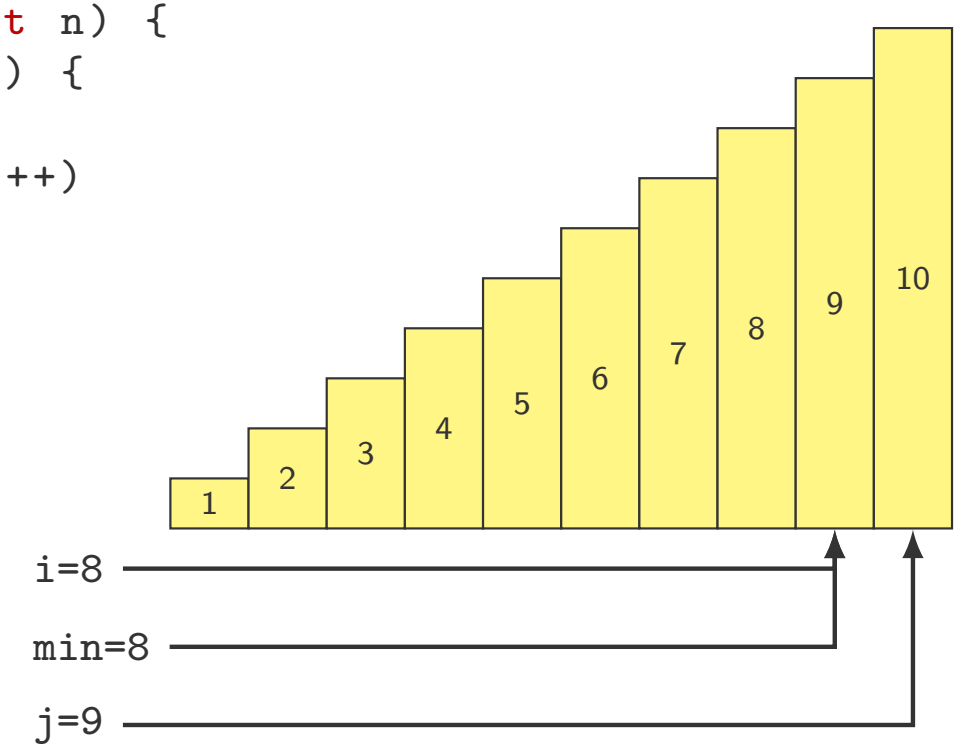
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



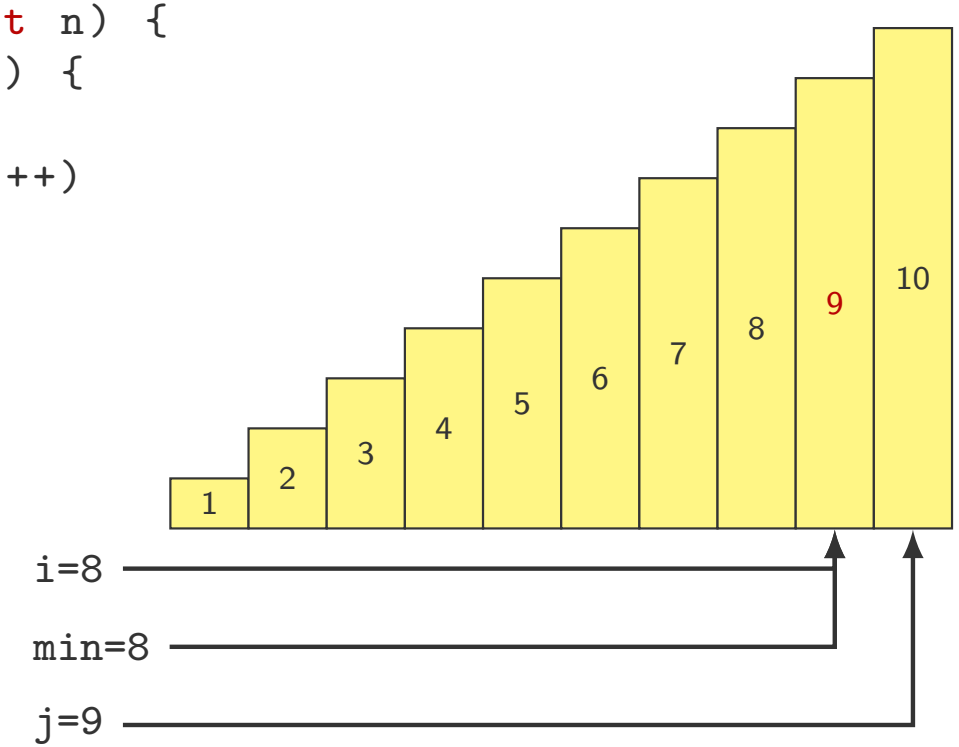
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



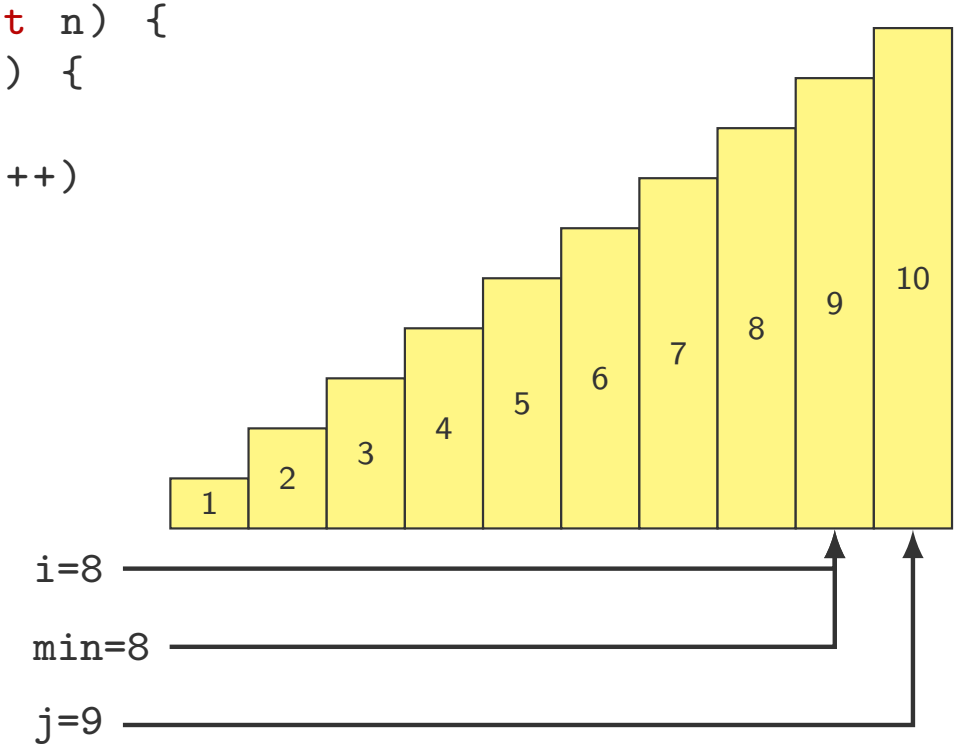
Ordenação por Seleção

Universidade Federal do Ceará
Campus Quixadá

Ideia:

- Trocar `v[0]` com o mínimo de `v[0], v[1], ..., v[n-1]`
- Trocar `v[1]` com o mínimo de `v[1], v[2], ..., v[n-1]`
- ...

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```



SelectionSort – Complexidade do algoritmo


```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

SelectionSort – Complexidade do algoritmo


```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

- número de comparações:
  $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2 = O(n^2)$

SelectionSort – Complexidade do algoritmo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

- número de comparações:
  $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2 = O(n^2)$

- número de trocas: $n - 1 = O(n)$

SelectionSort – Complexidade do algoritmo


```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

- número de comparações:
  $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2 = O(n^2)$
- número de trocas: $n - 1 = O(n)$
  - Muito bom quando trocas são muito caras

SelectionSort – Complexidade do algoritmo


```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

- número de comparações:
  $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2 = O(n^2)$
- número de trocas: $n - 1 = O(n)$
  - Muito bom quando trocas são muito caras
  - Porém, talvez seja melhor usar ponteiros nesse caso

Comparação entre os algoritmos

	Número de comparações		Número de trocas	
	Melhor caso	Pior caso	Melhor caso	Pior caso
Selection sort	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
Bubble sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$

- Para arrays de tamanho pequeno (≤ 100) esses algoritmos são boas escolhas
- Dos três algoritmos, insertion sort possui a melhor performance para a maioria dos arrays:
  - tira proveito de qualquer ordenação parcial que esteja no array
  - usa shifts, que são menos custosos que trocas

Algoritmos in Loco

Algoritmos in loco

- Um algoritmo in loco é um algoritmo que transforma a entrada sem usar estruturas de dados auxiliares.

Algoritmos in loco

- Um algoritmo in loco é um algoritmo que transforma a entrada sem usar estruturas de dados auxiliares.
  - uma quantidade constante e pequena de espaço de armazenamento extra é permitida (geralmente para variáveis auxiliares).

Algoritmos in loco

- Um algoritmo in loco é um algoritmo que transforma a entrada sem usar estruturas de dados auxiliares.
  - uma quantidade constante e pequena de espaço de armazenamento extra é permitida (geralmente para variáveis auxiliares).

- A entrada é geralmente sobrescrita pela saída conforme o algoritmo é executado. O algoritmo in loco atualiza a entrada apenas por meio da substituição ou troca de elementos.

Algoritmos in loco

- Um algoritmo in loco é um algoritmo que transforma a entrada sem usar estruturas de dados auxiliares.
  - uma quantidade constante e pequena de espaço de armazenamento extra é permitida (geralmente para variáveis auxiliares).

- A entrada é geralmente sobrescrita pela saída conforme o algoritmo é executado. O algoritmo in loco atualiza a entrada apenas por meio da substituição ou troca de elementos.

- Pergunta: BubbleSort, Insertion-Sort e Selection-Sort são algoritmos in loco?

Ordenação Estável

Ordenação estável

Definição

Um algoritmo de ordenação **estável** é aquele que, ao ordenar uma coleção de elementos, mantém a ordem relativa dos elementos iguais (ou seja, dos elementos que têm a mesma chave de comparação).

Em outras palavras: se dois elementos a e b têm o mesmo valor na chave usada para ordenação e a aparece antes de b na lista original, então, após a ordenação, a continuará aparecendo antes de b.

- Por exemplo, se o vetor tiver dois elementos de valor 13, um algoritmo de ordenação estável manterá o primeiro 13 antes do segundo.

Ordenação estável — Exemplo

- Exemplo: Suponha que os elementos de um vetor são pares da forma (d, m) que representam datas de um certo ano: a primeira componente representa o dia e a segunda o mês.

Ordenação estável — Exemplo

- Exemplo: Suponha que os elementos de um vetor são pares da forma (d, m) que representam datas de um certo ano: a primeira componente representa o dia e a segunda o mês.
- Suponha que o vetor está em ordem crescente das componentes d:

  (1,12), (7,12), (16,3), (25,9), (30,3), (30,6), (31,3).

Ordenação estável — Exemplo

- Exemplo: Suponha que os elementos de um vetor são pares da forma (d, m) que representam datas de um certo ano: a primeira componente representa o dia e a segunda o mês.
- Suponha que o vetor está em ordem crescente das componentes d:

  (1,12), (7,12), (16,3), (25,9), (30,3), (30,6), (31,3).

- Agora ordene o vetor pelas componentes m. Se usarmos um algoritmo de ordenação estável, o resultado estará em ordem cronológica:

  (16,3), (30,3), (31,3), (30,6), (25,9), (1,12), (7,12).

Ordenação estável — Exemplo

- Exemplo: Suponha que os elementos de um vetor são pares da forma (d, m) que representam datas de um certo ano: a primeira componente representa o dia e a segunda o mês.
- Suponha que o vetor está em ordem crescente das componentes d:

  (1,12), (7,12), (16,3), (25,9), (30,3), (30,6), (31,3).

- Agora ordene o vetor pelas componentes m. Se usarmos um algoritmo de ordenação estável, o resultado estará em ordem cronológica:

  (16,3), (30,3), (31,3), (30,6), (25,9), (1,12), (7,12).

- Se o algoritmo de ordenação não for estável, o resultado pode não ficar em ordem cronológica:

  (30,3), (16,3), (31,3), (30,6), (25,9), (7,12), (1,12).

Por que a estabilidade é importante?

Ordenações múltiplas:

- Se você ordena por um critério e depois por outro, a estabilidade garante que a ordenação anterior não seja perdida.

- **Exemplo:** ordenar funcionários primeiro por salário, depois por departamento.
  - Se o segundo algoritmo for estável, os funcionários de cada departamento estarão ordenados por ordem crescente de salário. Os departamentos estarão em ordem lexicográfica.

Exercício – Ordenação estável

- BubbleSort é um algoritmo estável?

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

Exercício – Ordenação estável

- BubbleSort é um algoritmo estável?

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

Exercício – Ordenação estável

- BubbleSort é um algoritmo estável?

```
1  void bubblesort(int A[], int n) {
2    for (int i = 0; i < n; i++) {
3      for (int j = n-1; j > i; j--) {
4        if (A[j] < A[j-1]) {
5            int aux = A[j];
6            A[j] = A[j-1];
7            A[j-1] = aux;
8        }
9      }
10   }
```

Sim!

Exercício – Ordenação estável

- Insertion-Sort é um algoritmo estável?

```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

Exercício – Ordenação estável

- Insertion-Sort é um algoritmo estável?

```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

Exercício – Ordenação estável

- Insertion-Sort é um algoritmo estável?

```
1  void insertionsort(int A[], int n) {
2      for (int j = 1; j < n; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= 0 && A[i] > key) {
6              A[i+1] = A[i];
7              i--;
8          }
9          A[i+1] = key;
10     }
11 }
```

Sim!

Exercício – Ordenação estável

- Selection-Sort é um algoritmo estável?

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

Exercício – Ordenação estável

- Selection-Sort é um algoritmo estável?

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

Exercício – Ordenação estável

- Selection-Sort é um algoritmo estável?

```
1  void selectionsort(int A[], int n) {
2    for (int i = 0; i < n-1; i++) {
3      int indexMin = i;
4      for (int j = i+1; j < n; j++)
5        if(A[j] < A[indexMin])
6          indexMin = j;
7      int aux = A[i];
8      A[i] = A[indexMin];
9      A[indexMin] = aux;
10   }
11 }
```

Não! Por exemplo, teste a sequência 443

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM