Ordenação: algoritmos elementares

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2022

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:

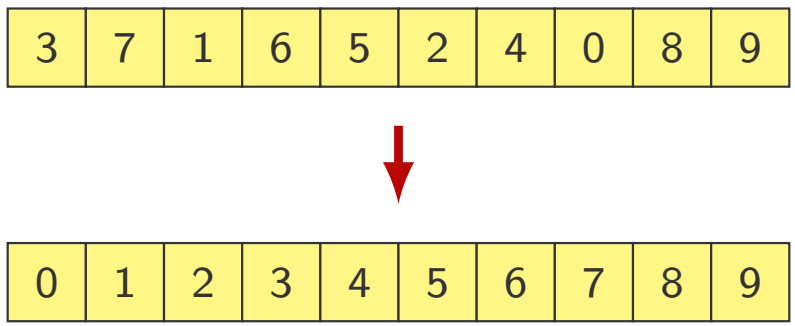
(1) inserir os dados na estrutura respeitando sua ordem.

(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:

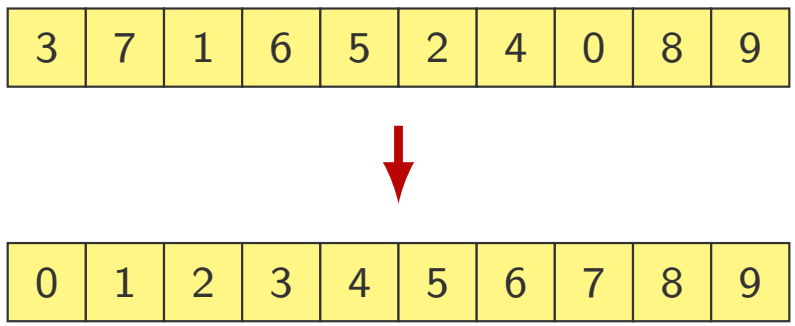
- rearranjar os elementos de um vetor $A[0 \ldots n-1]$ de tal modo que ele se torne crescente, ou seja, de modo que $A[0] \leq \cdots \leq A[n-1]$.



Ordenação

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

Ideia:

BubbleSort – Ordenação por flutuação

Ideia:

- do fim para o começo, vamos trocando pares invertidos

BubbleSort – Ordenação por flutuação

Ideia:

- do fim para o começo, vamos trocando pares invertidos
- eventualmente, encontramos o menor elemento

BubbleSort – Ordenação por flutuação

Ideia:

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

BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

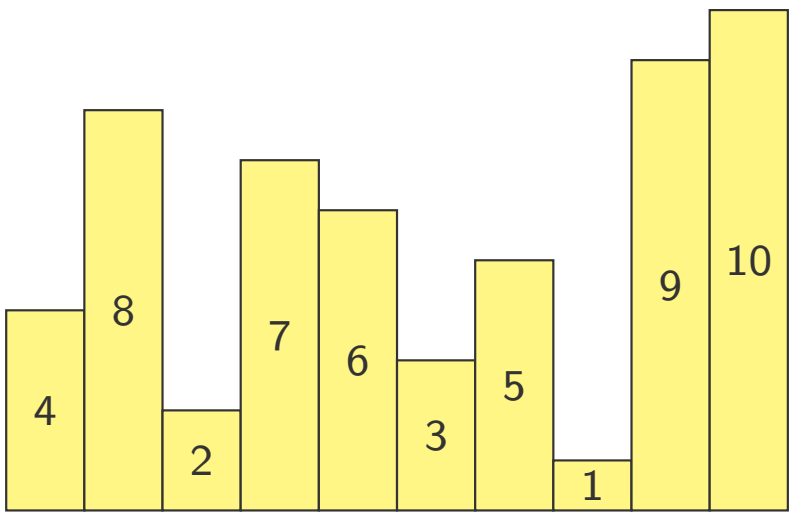
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



i

j

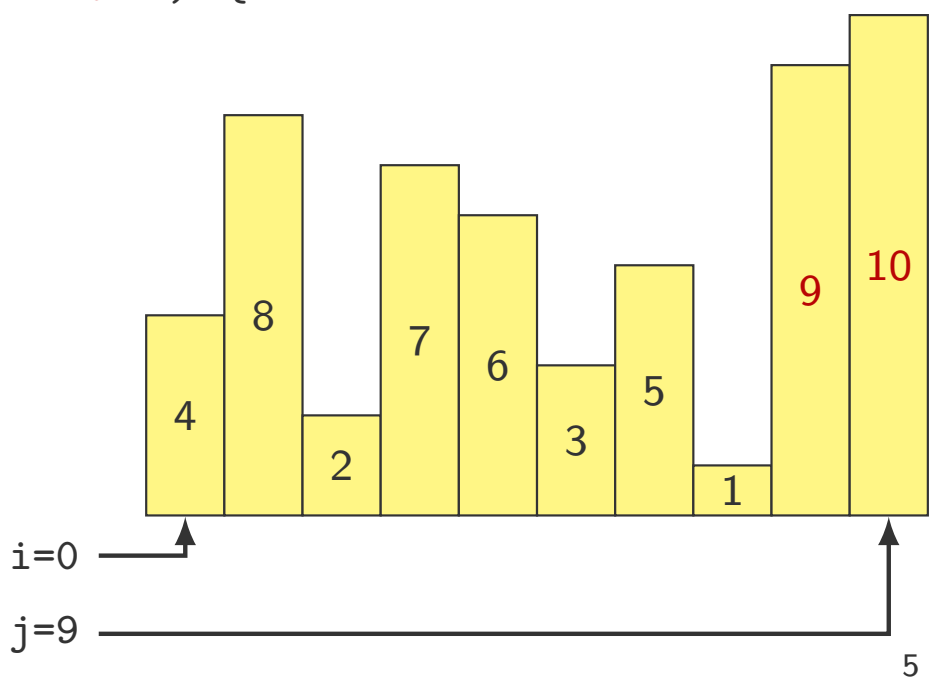
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

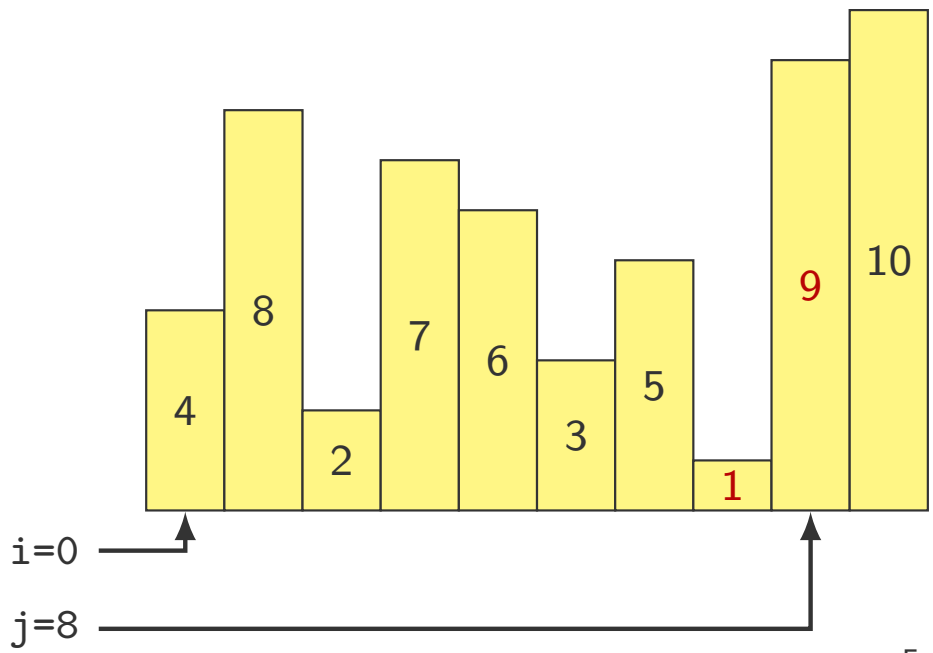


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

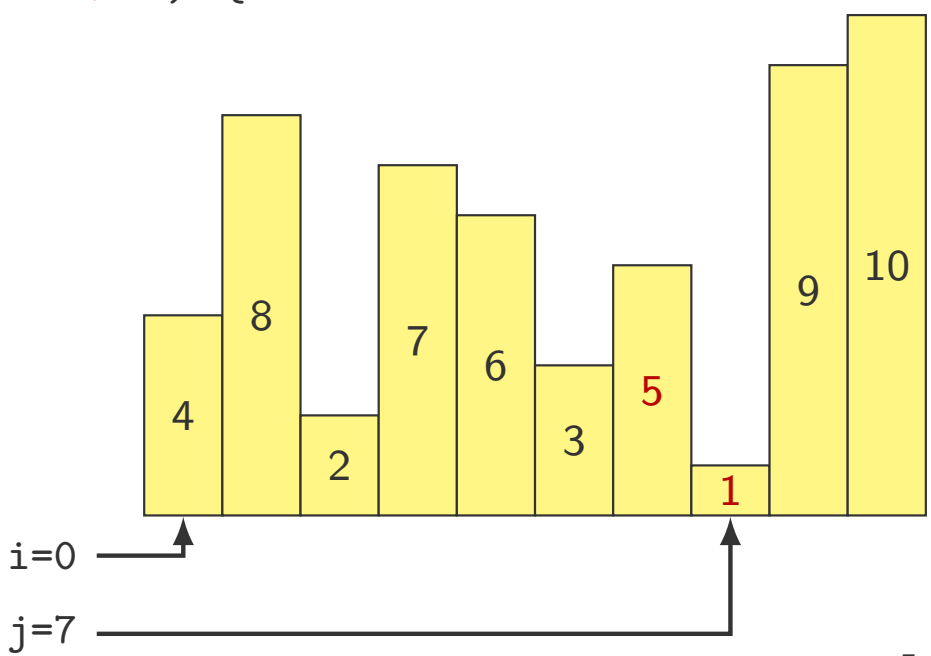


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

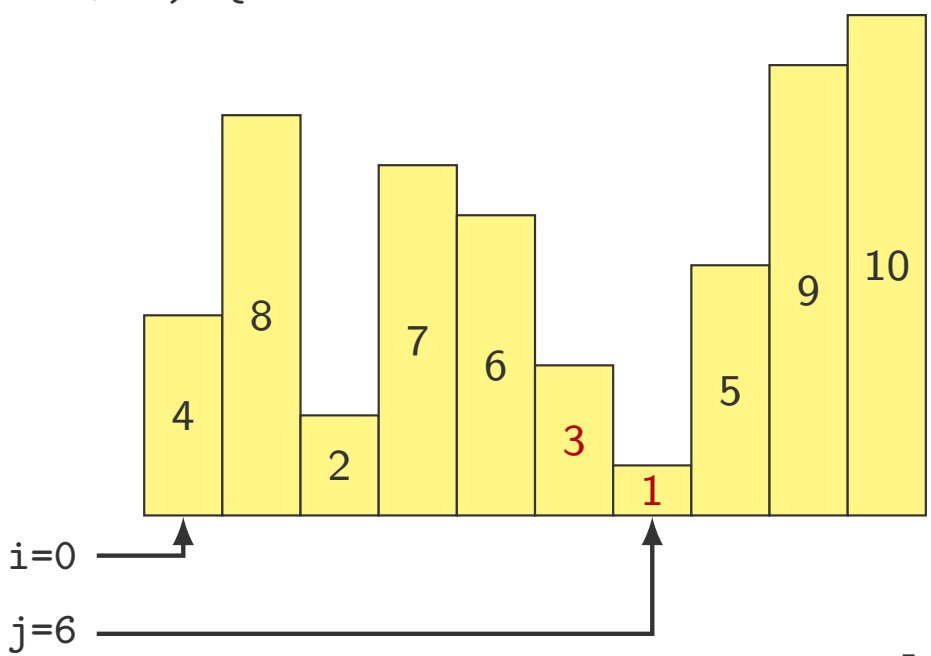


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



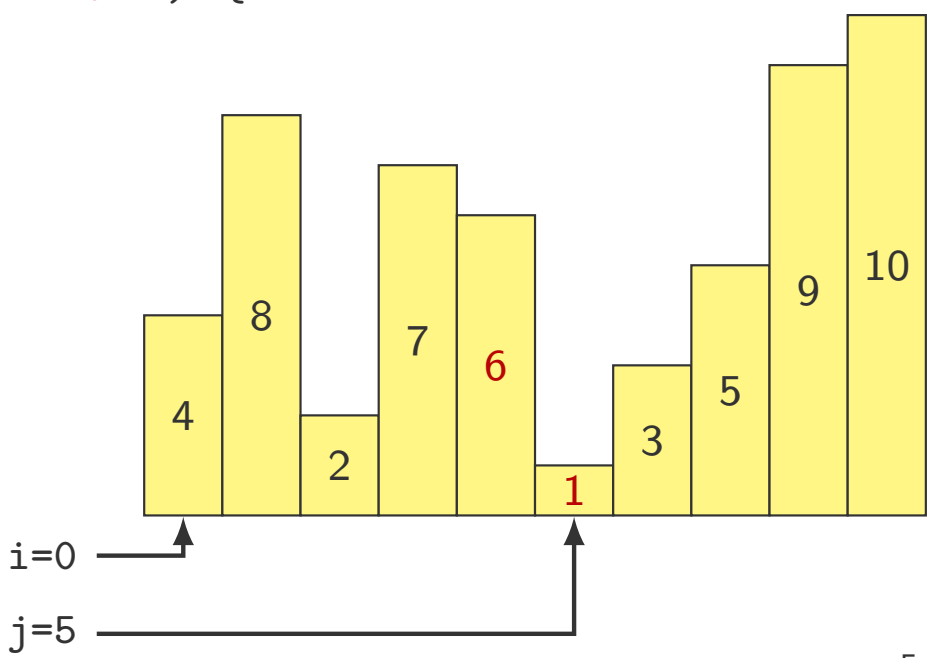
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

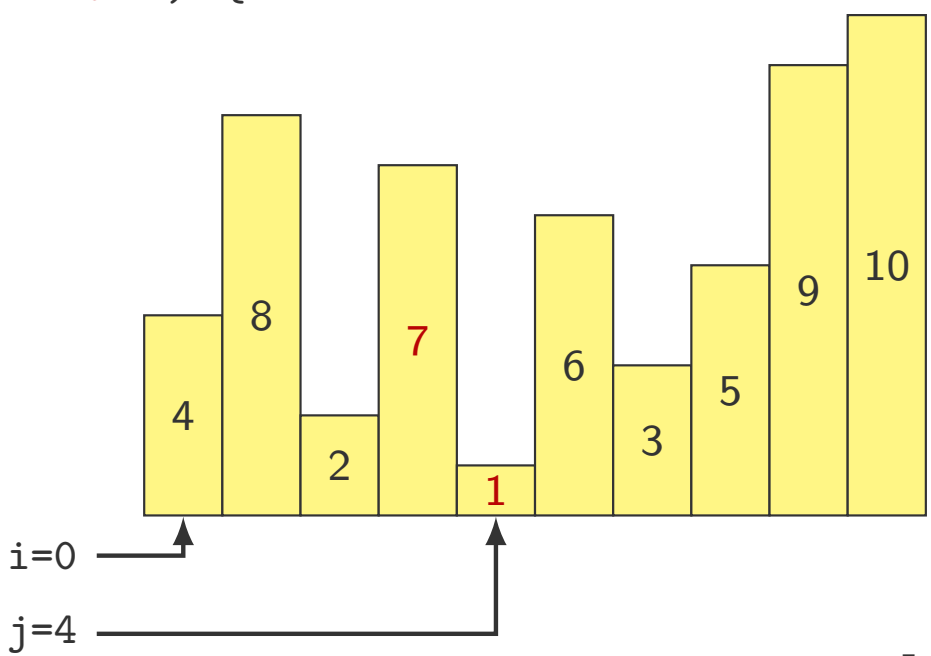


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



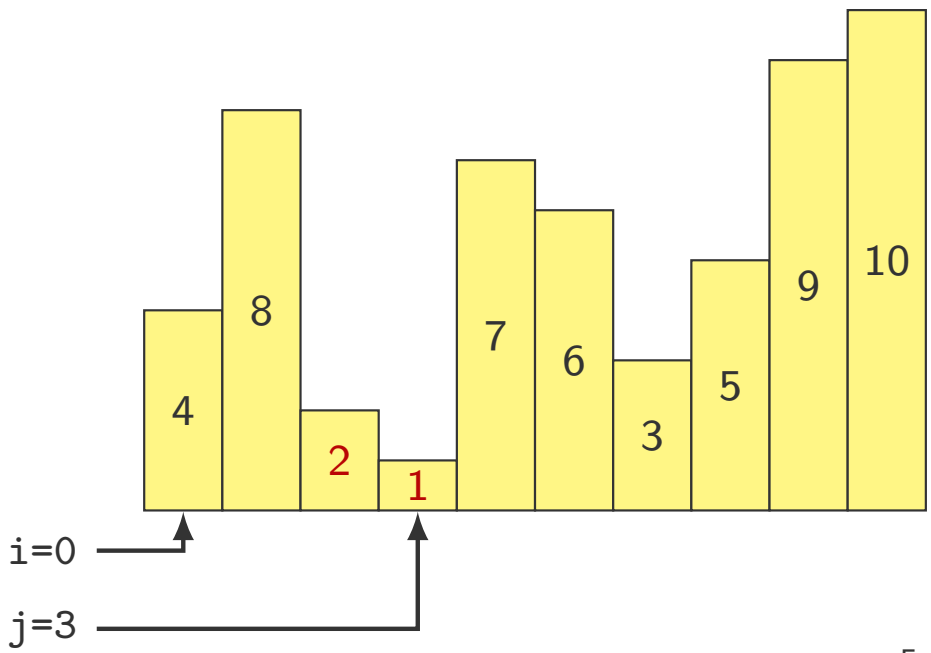
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

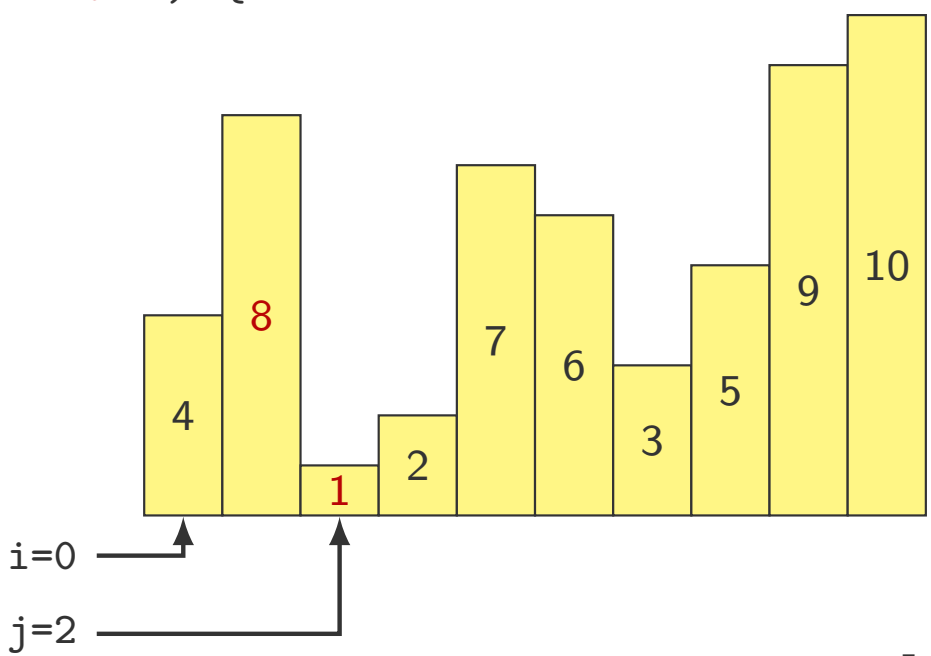


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

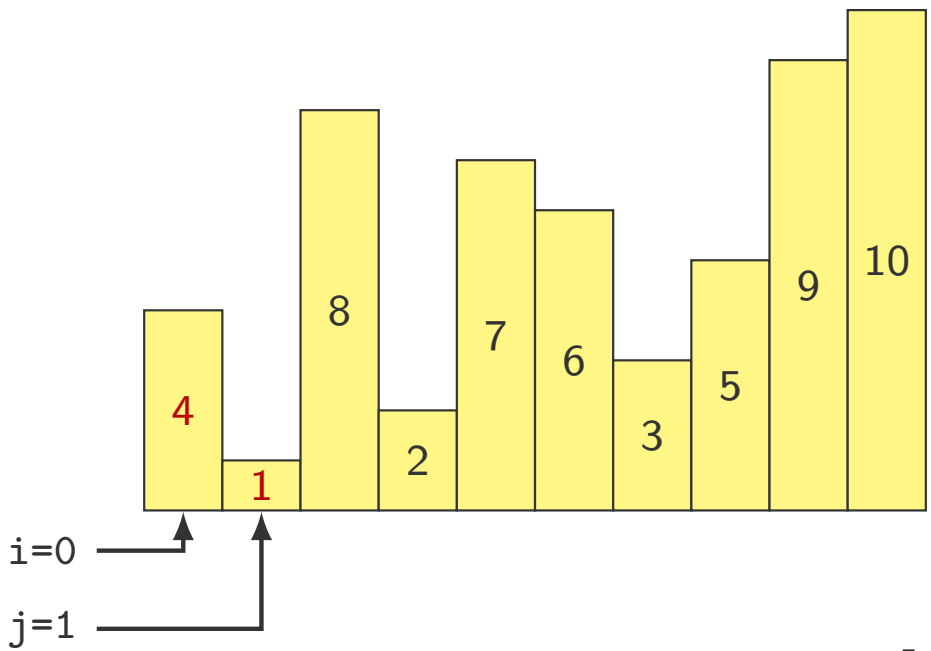


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



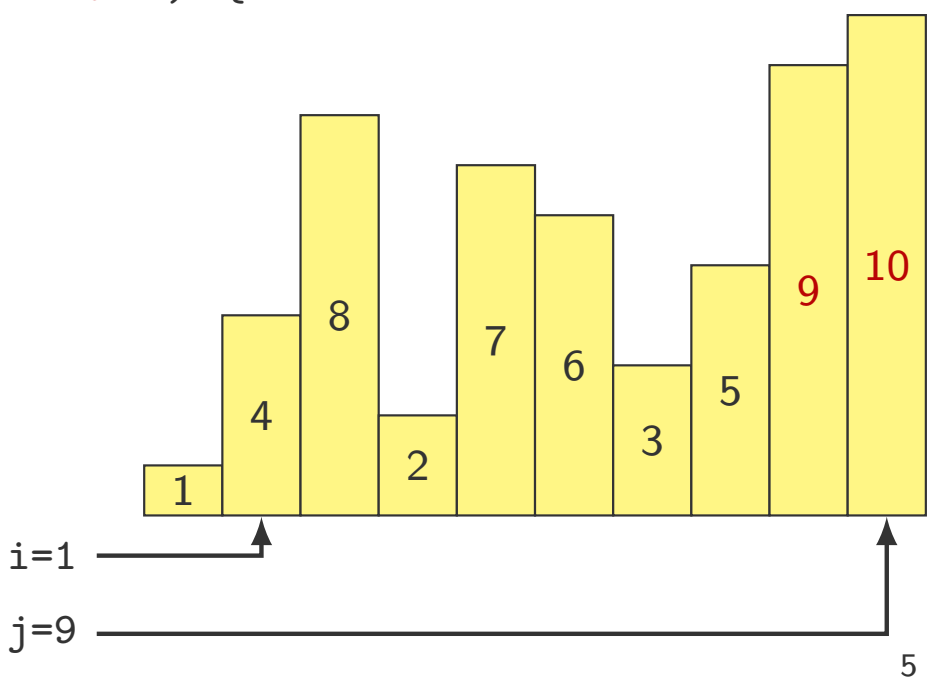
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



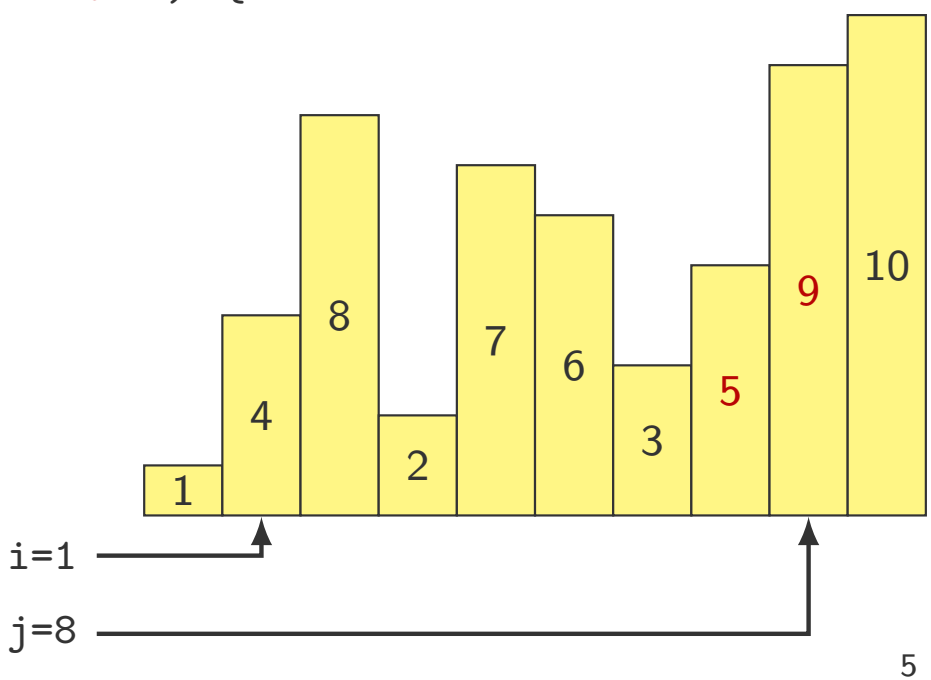
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



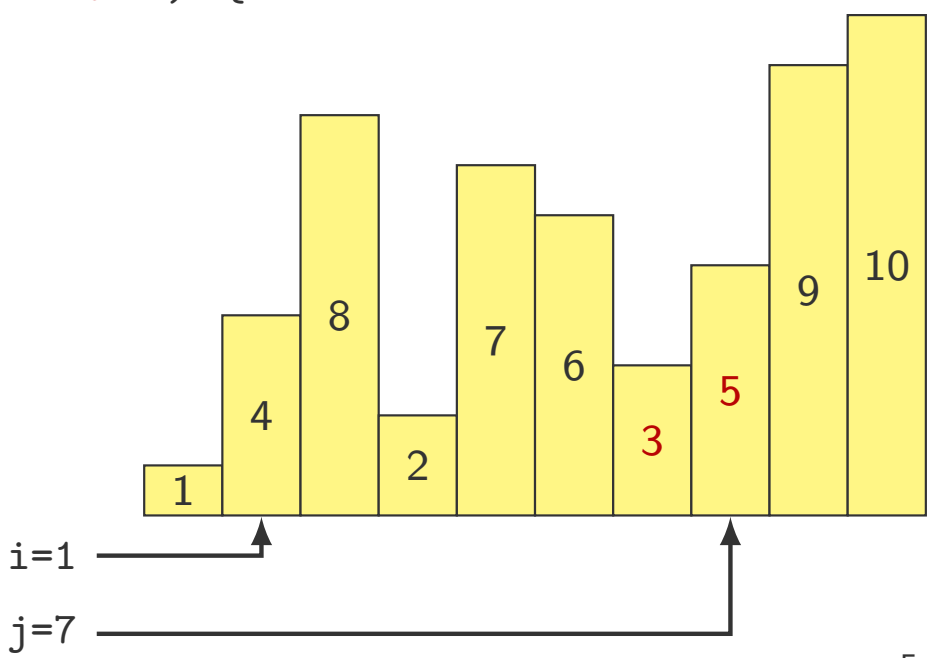
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



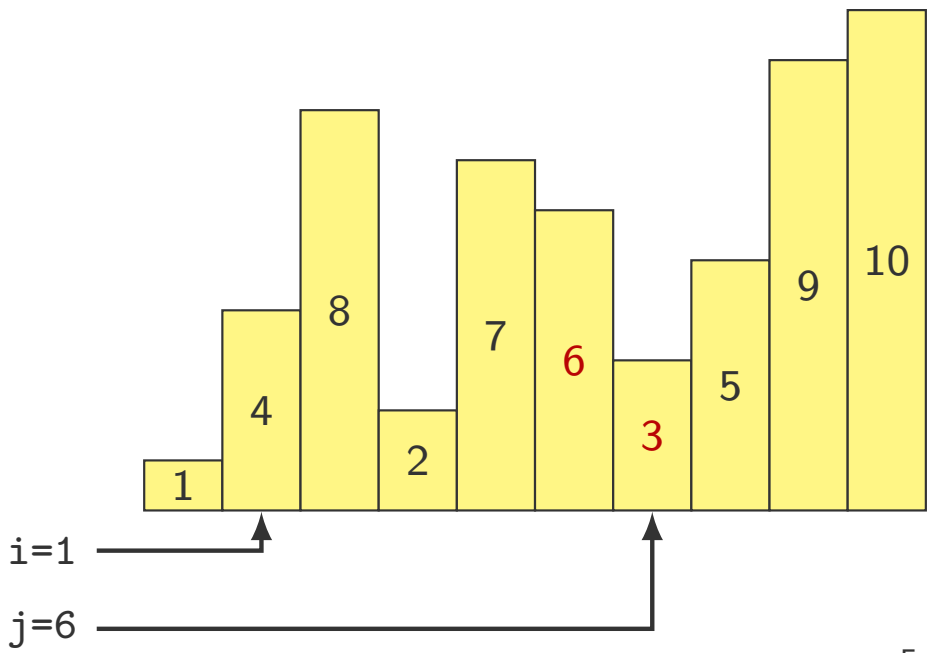
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

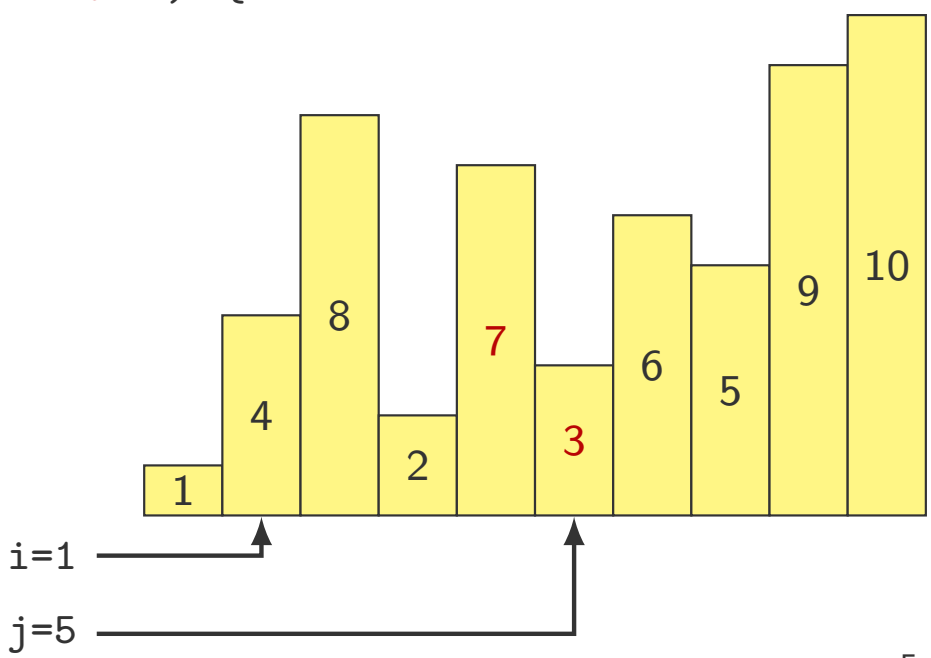


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

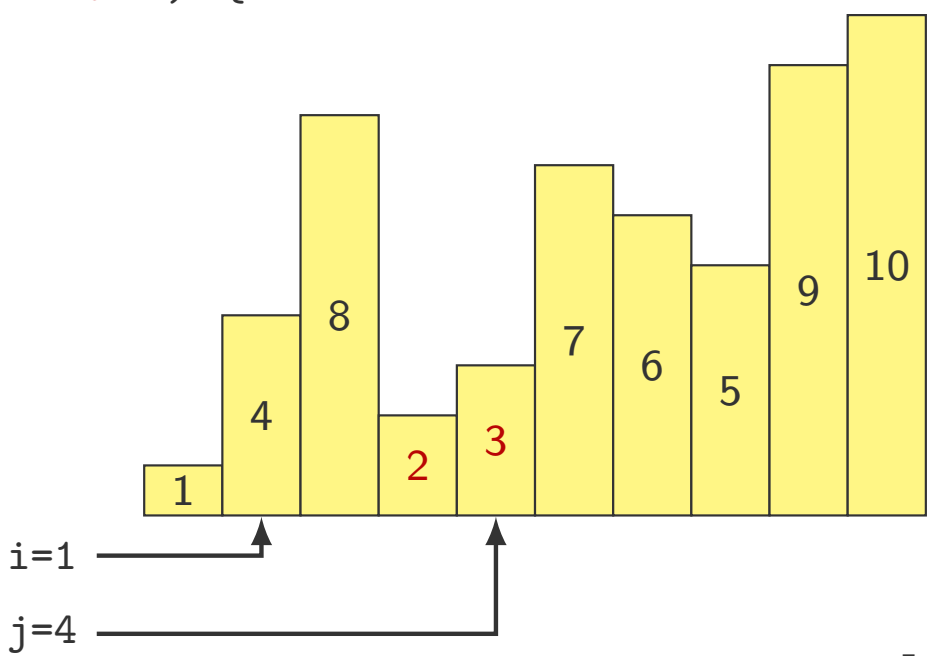


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

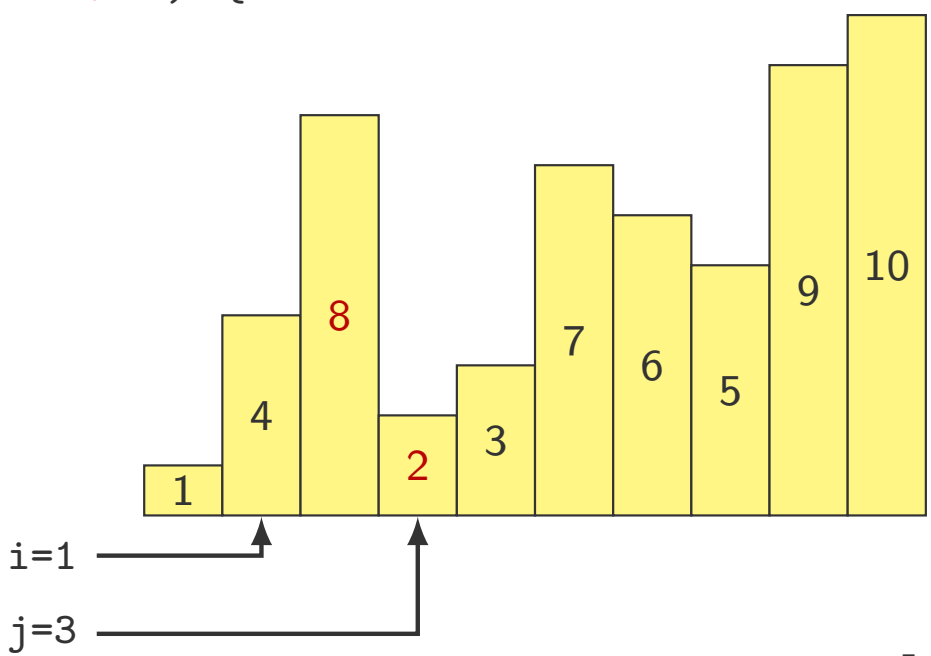


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

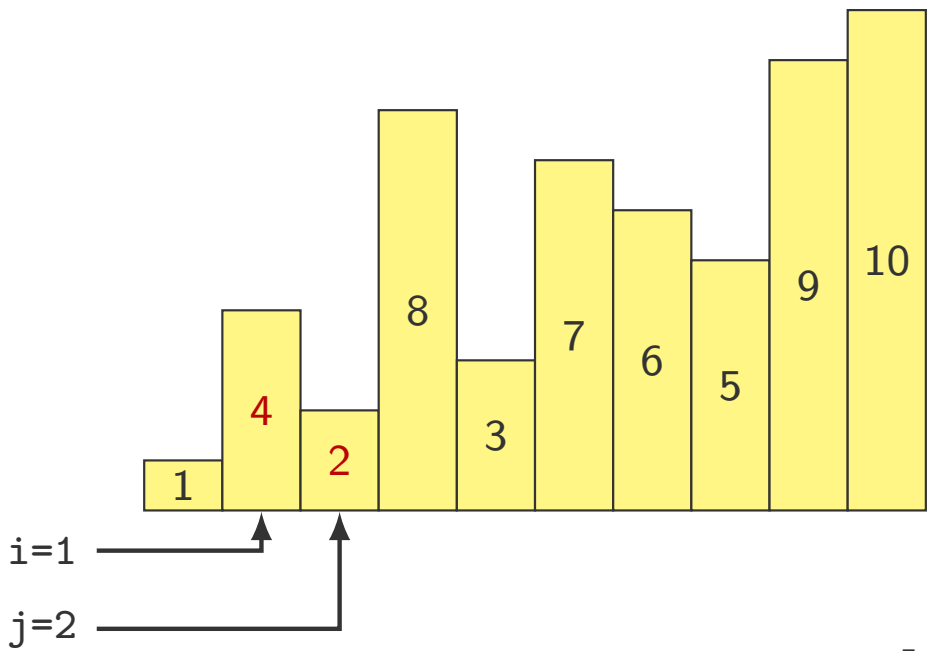


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



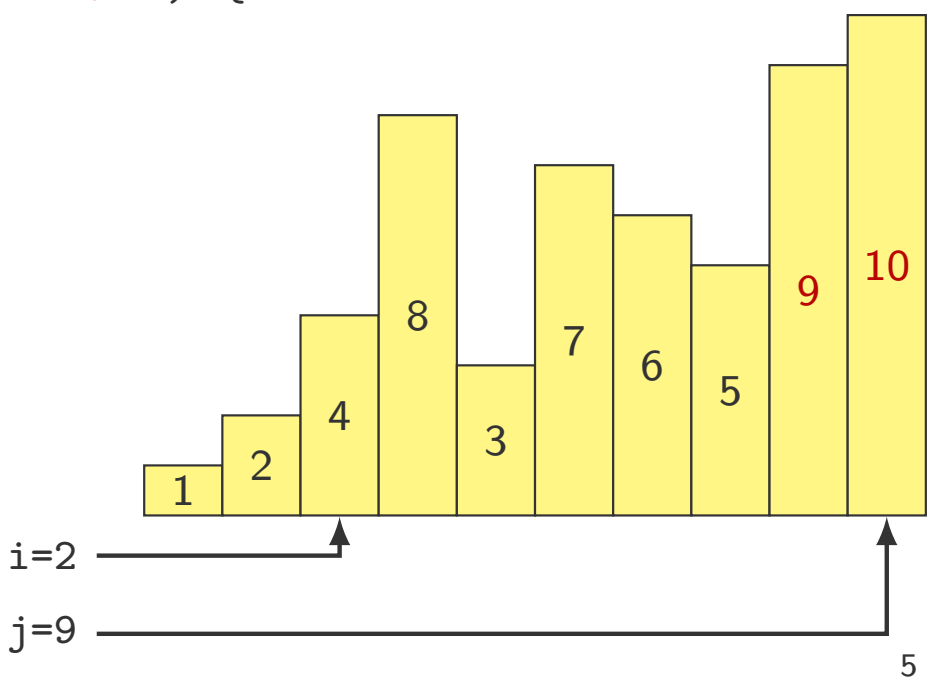
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

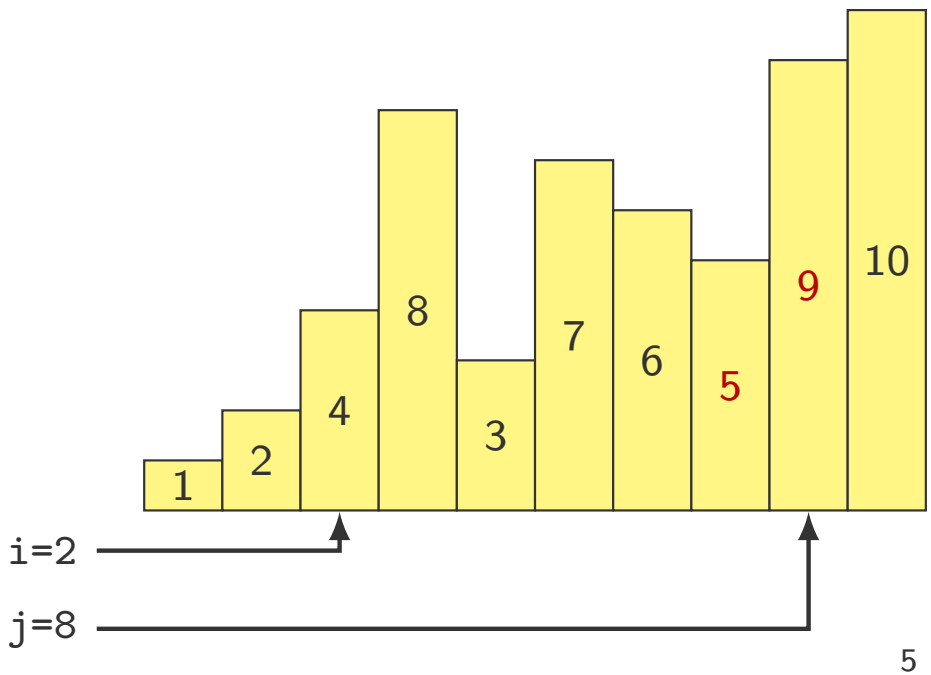


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

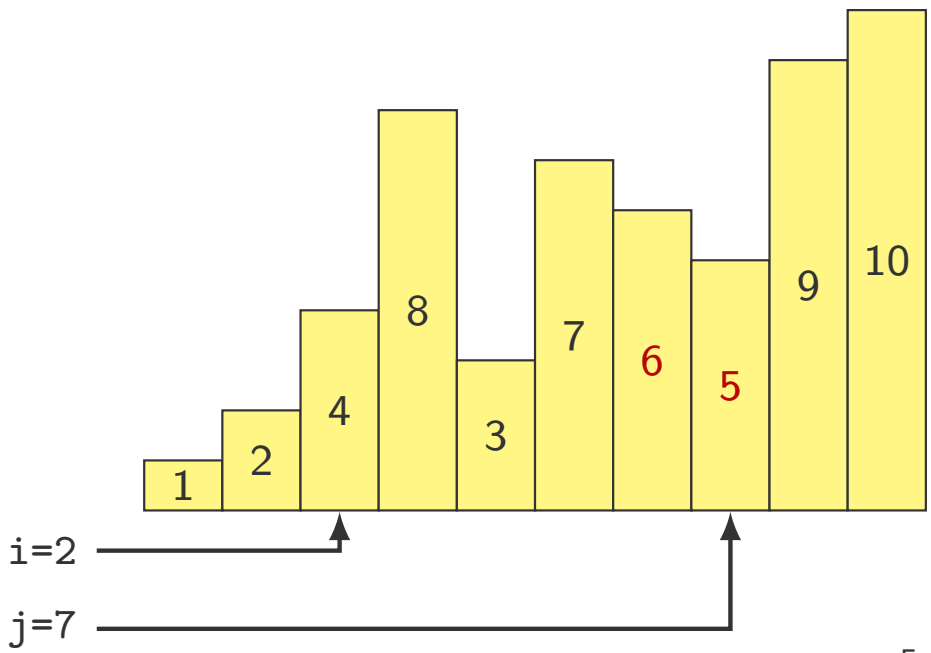


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



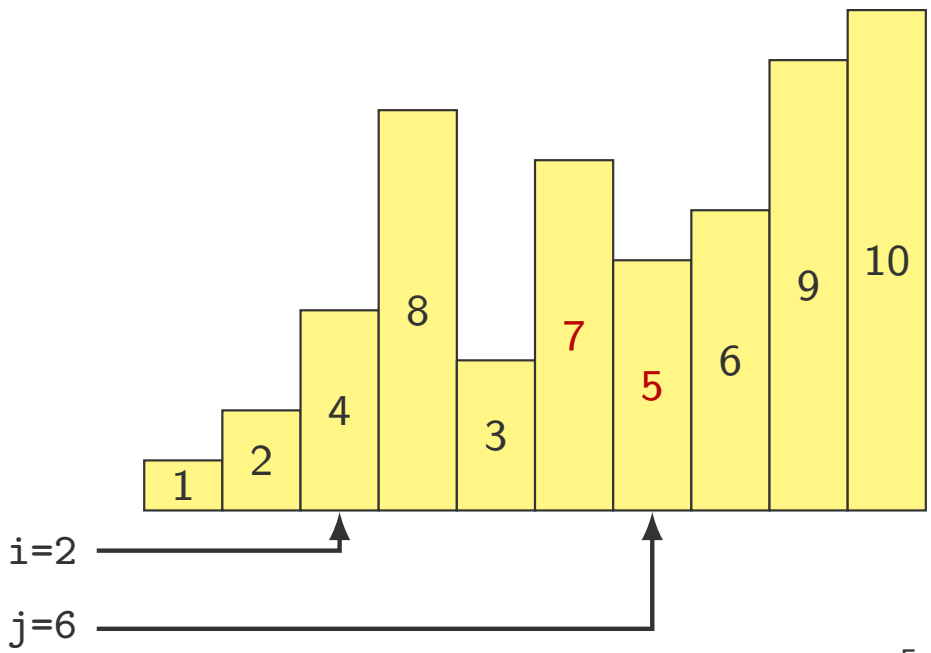
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



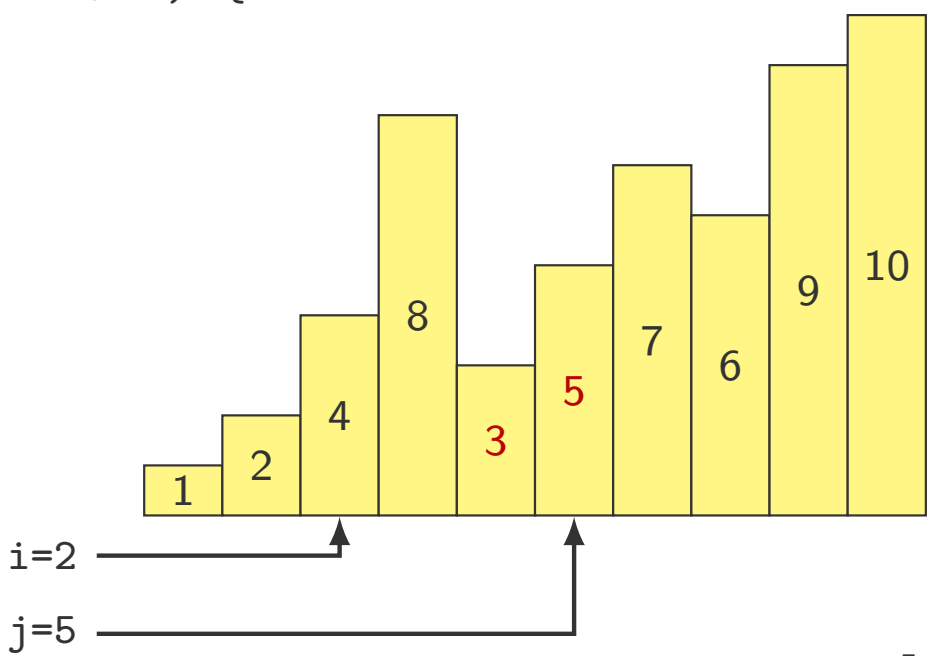
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



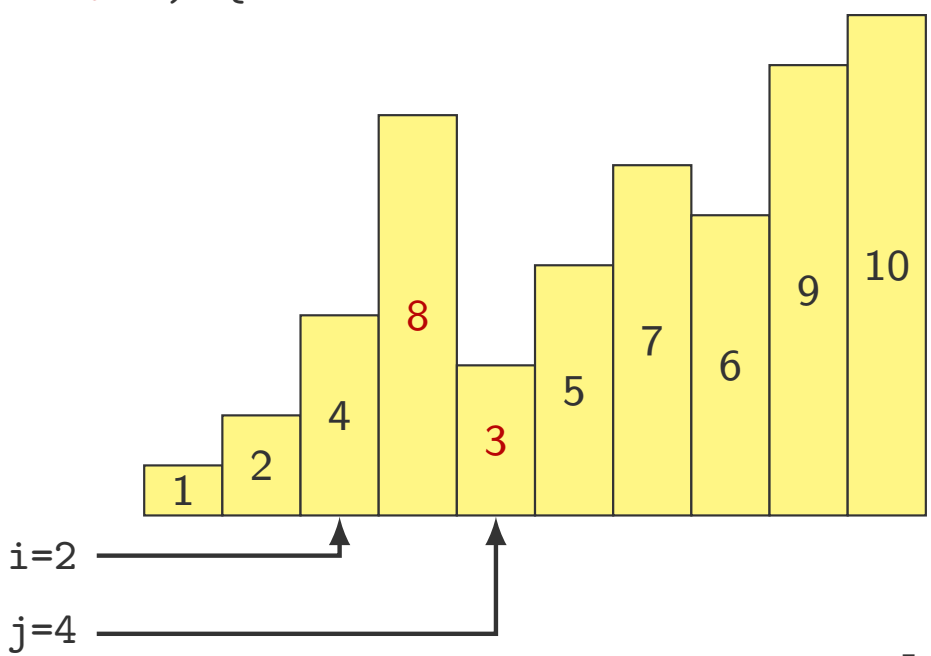
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



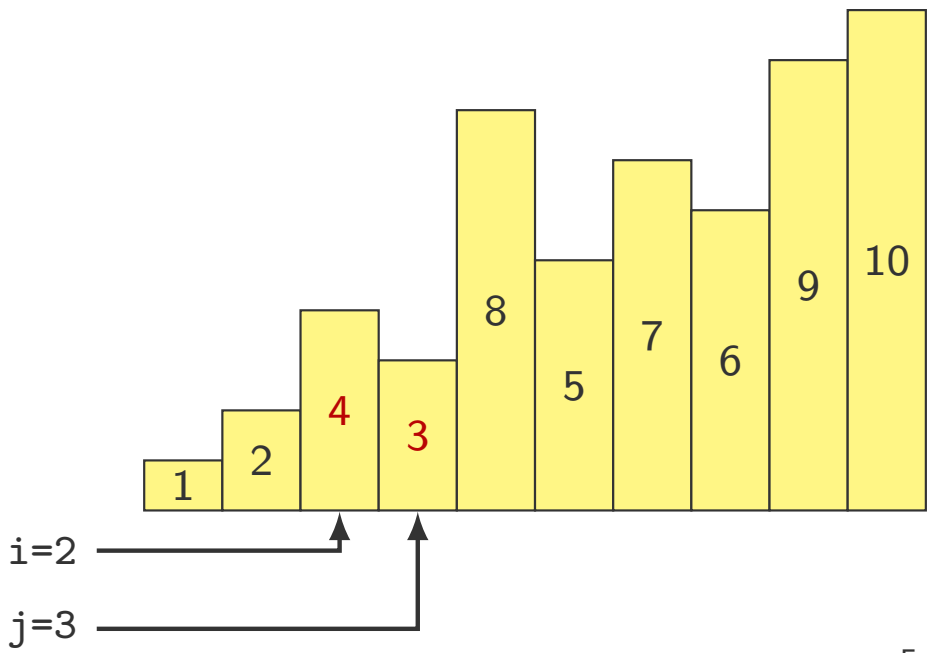
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



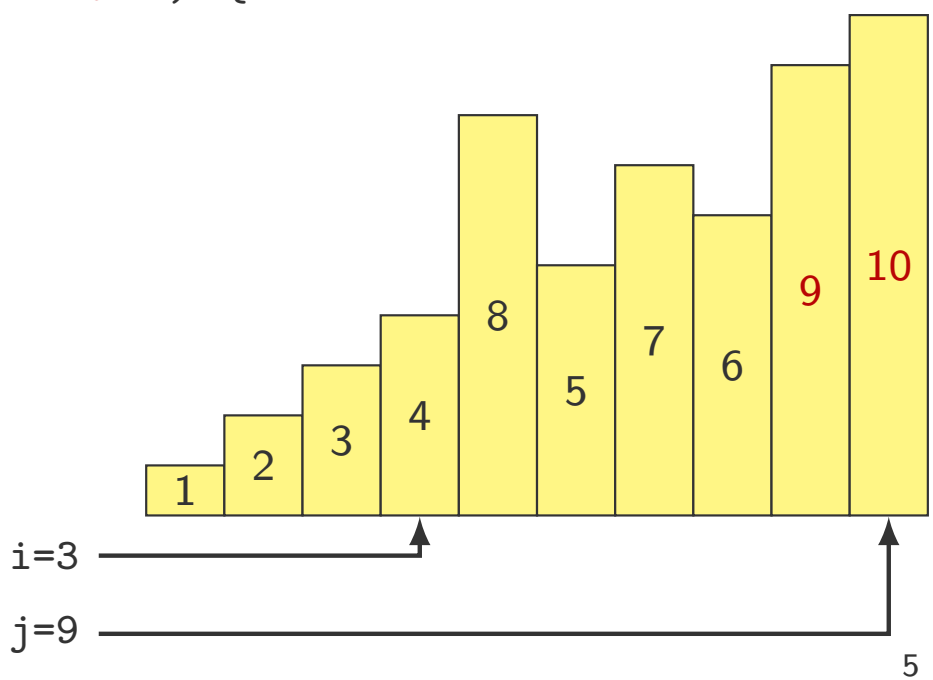
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



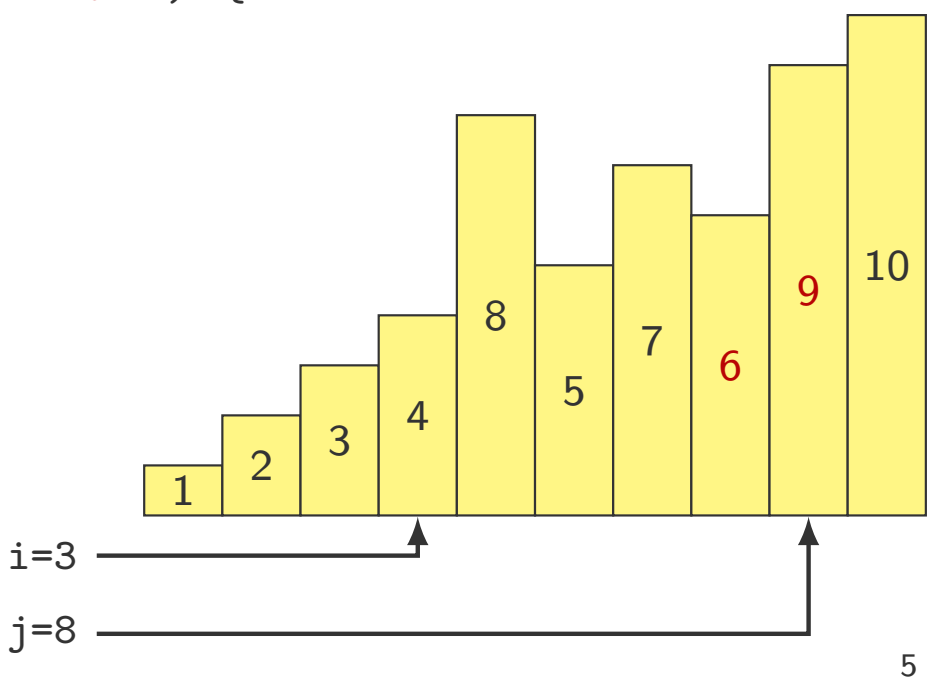
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



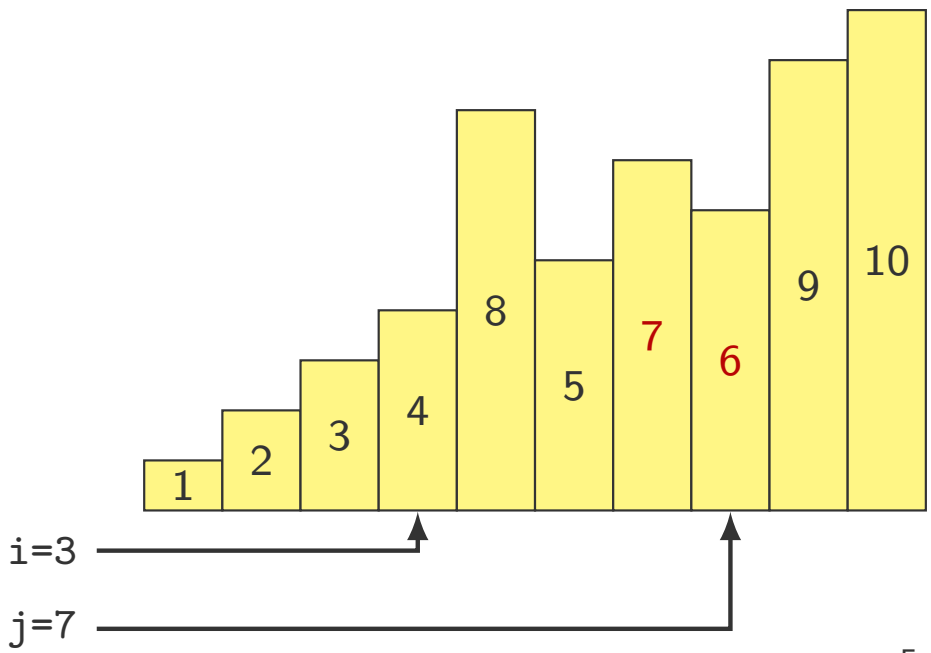
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



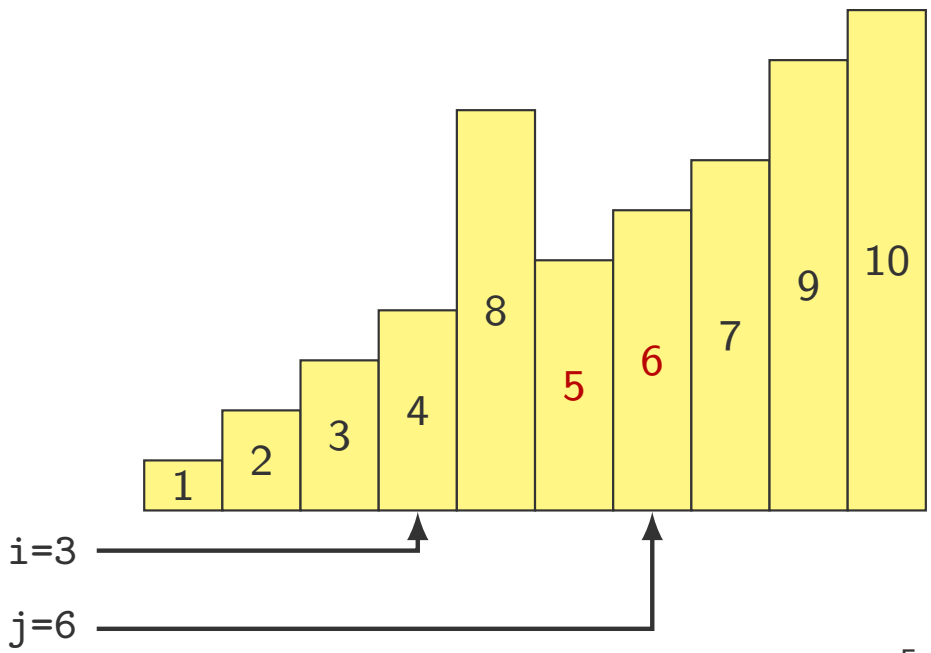
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



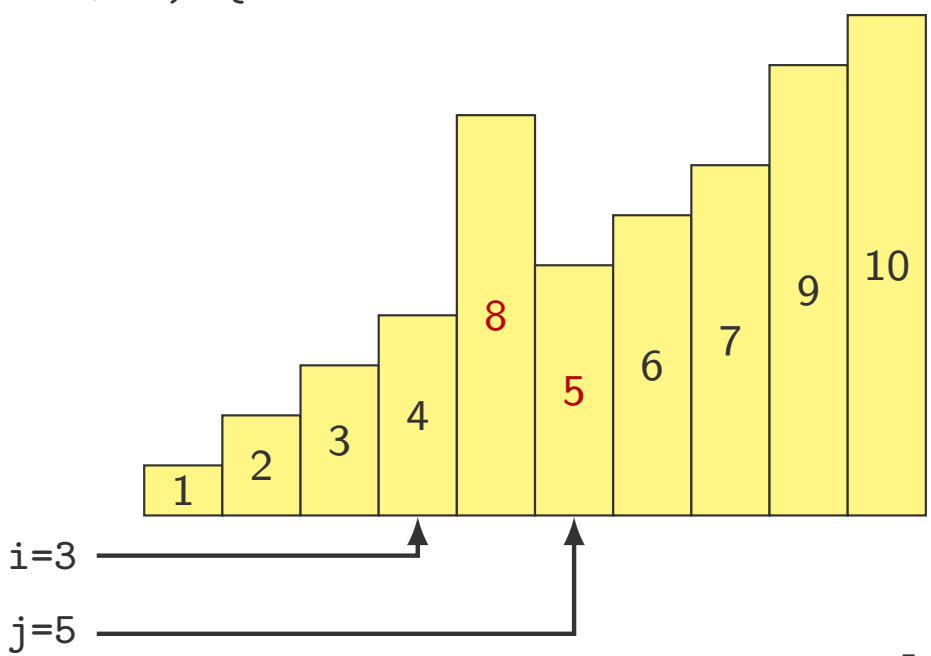
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



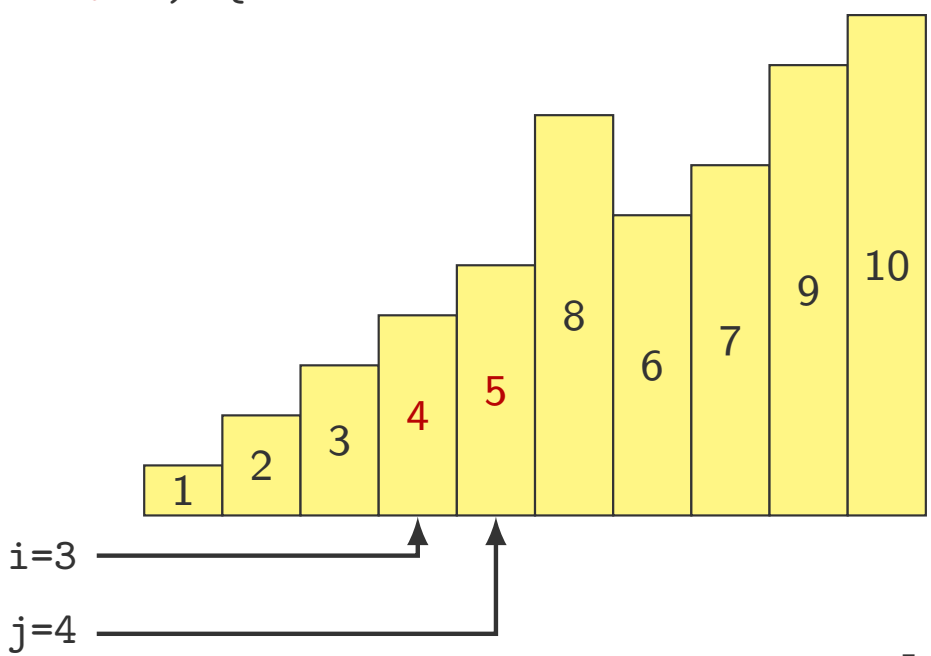
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



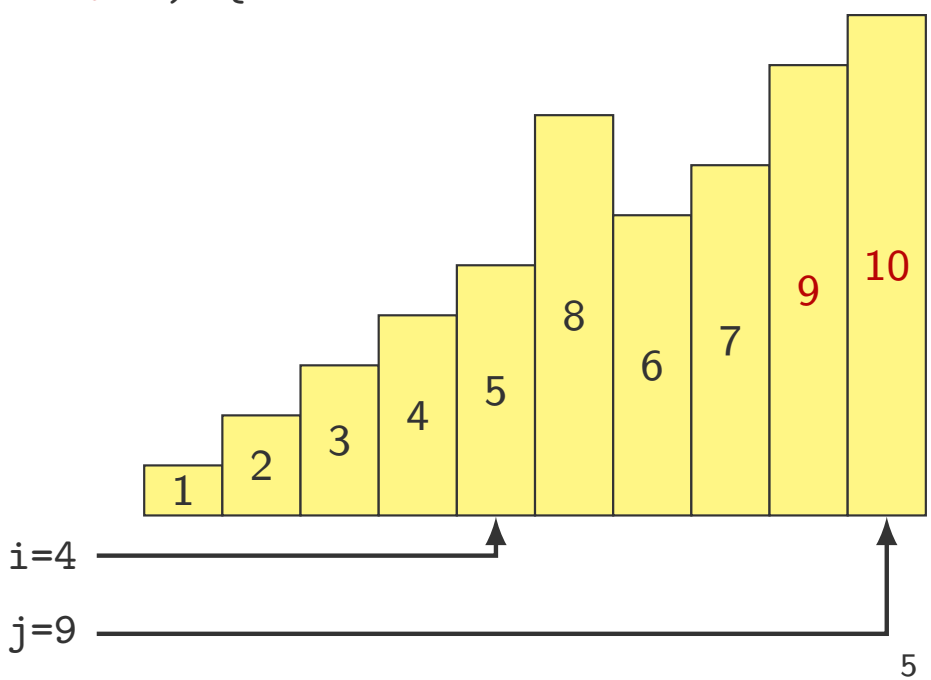
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



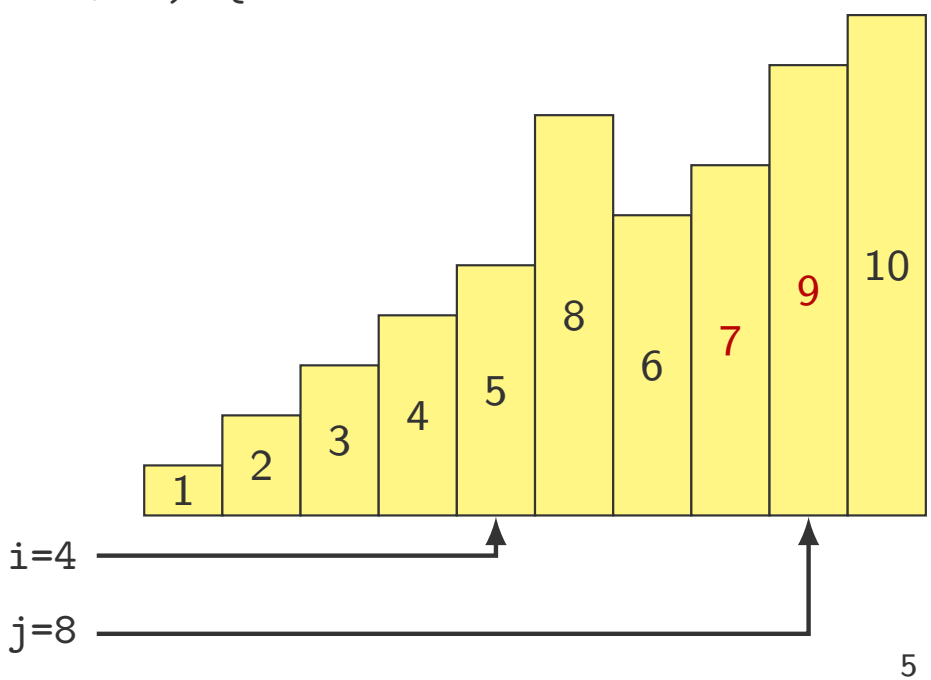
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



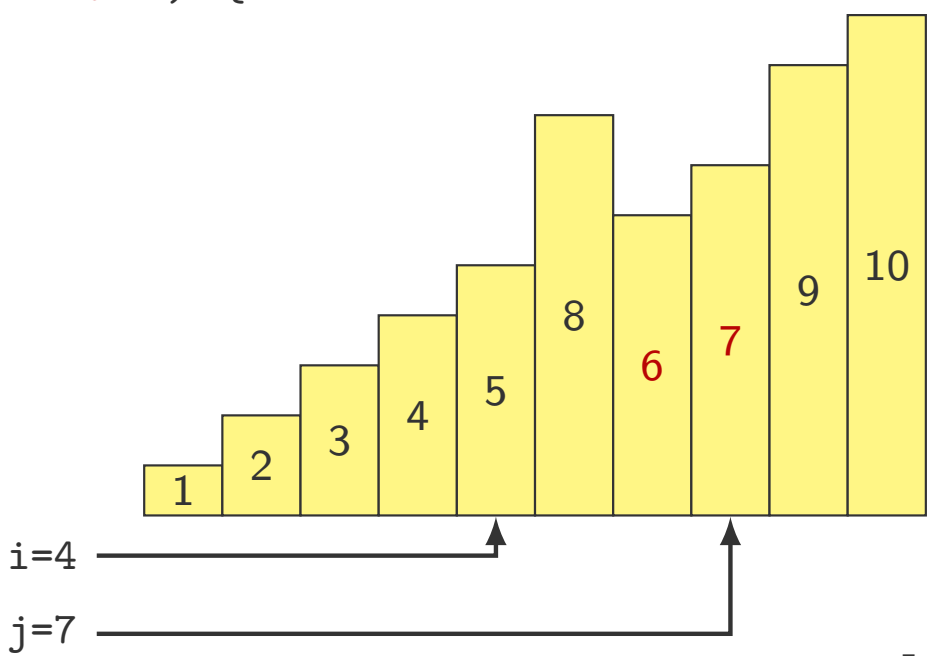
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



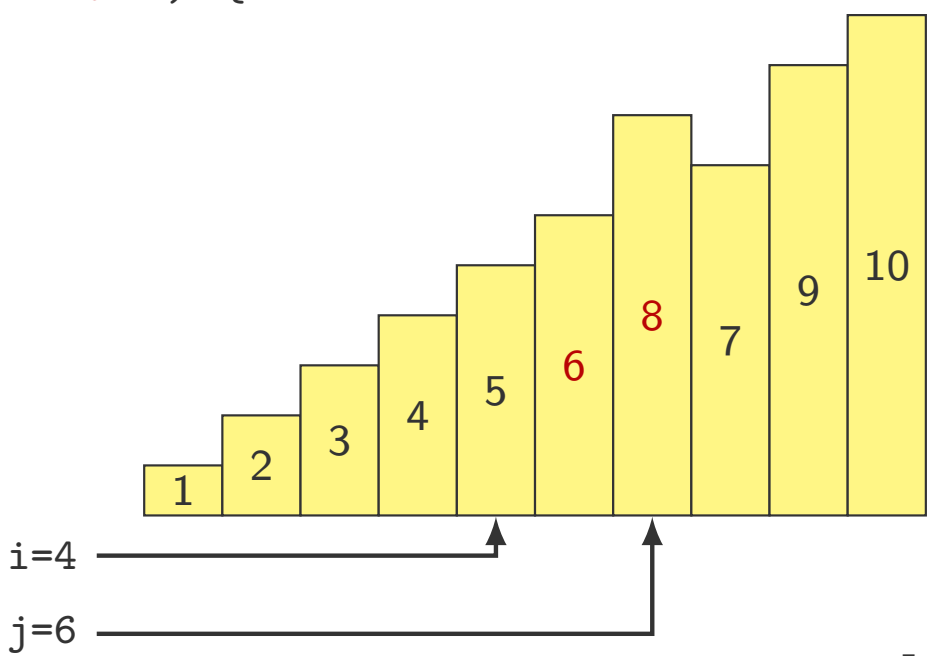
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



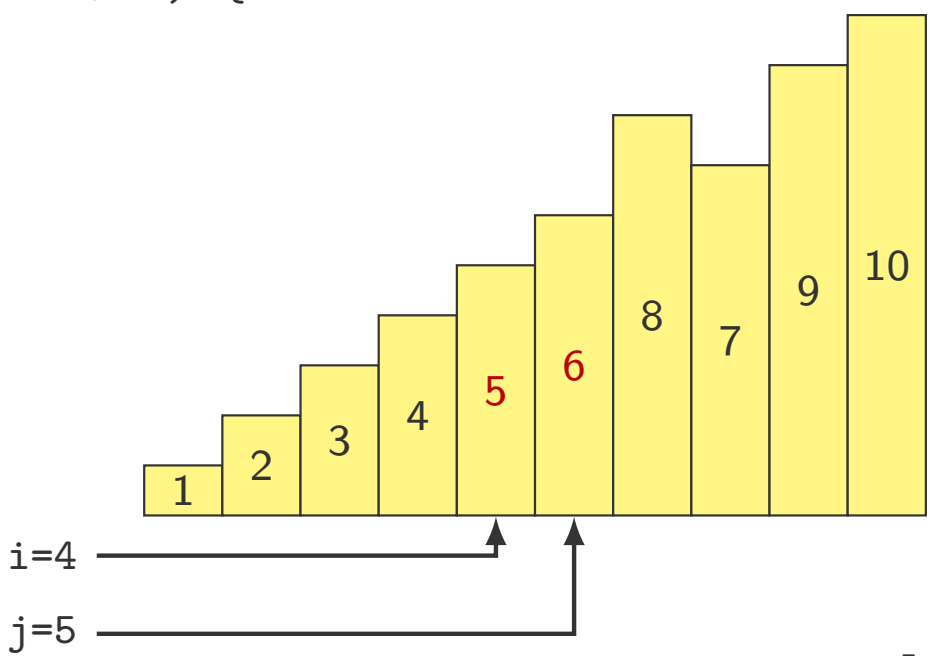
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



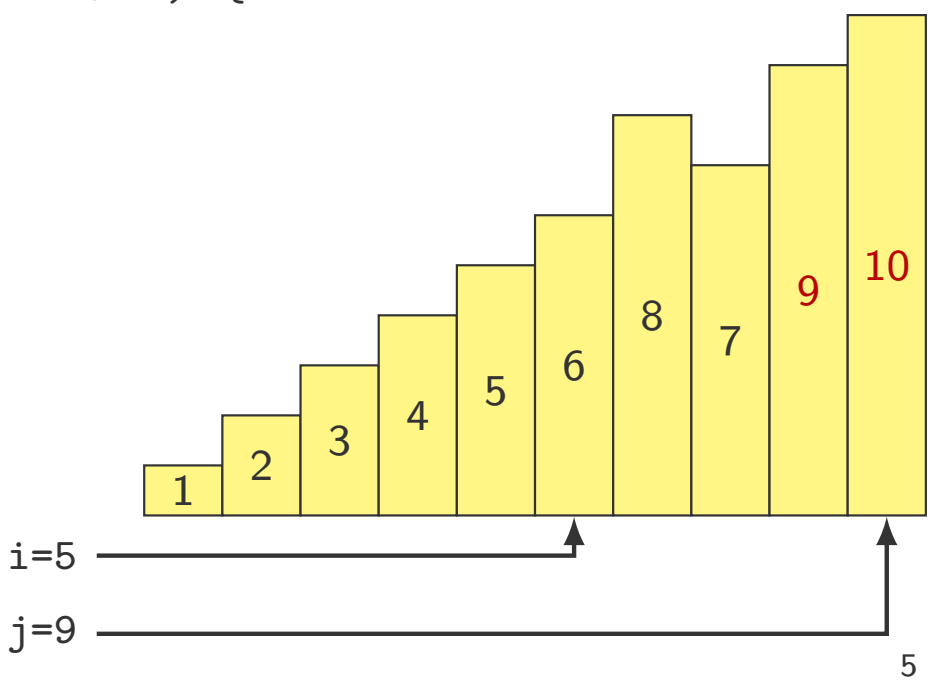
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



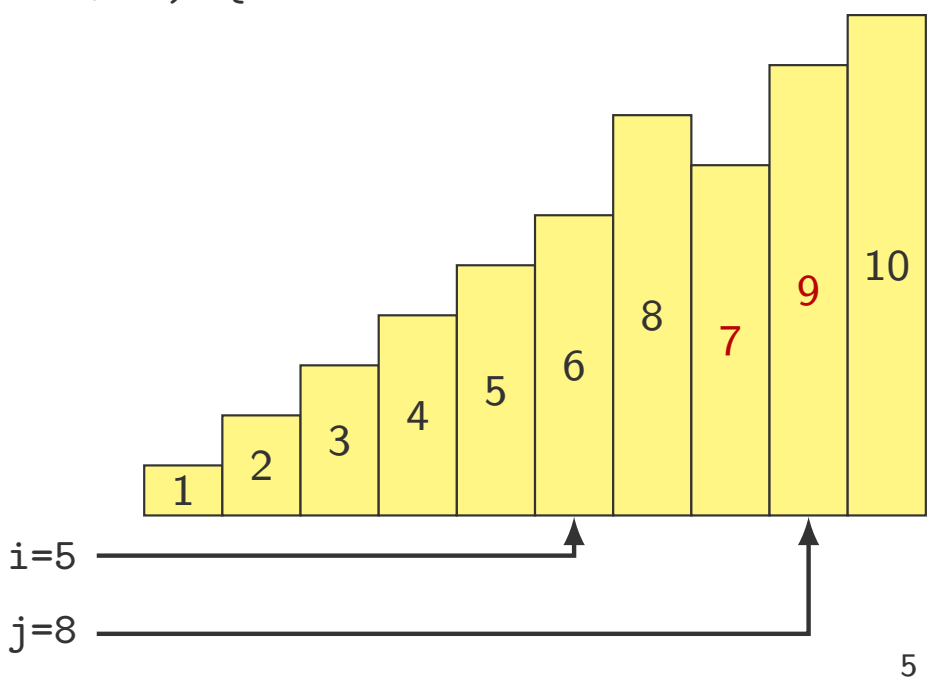
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



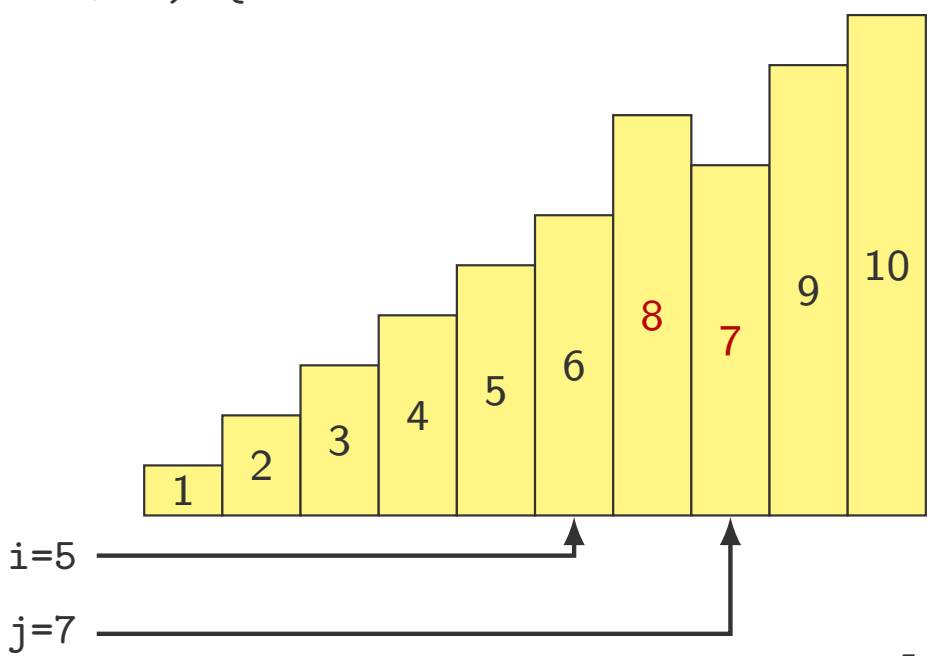
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



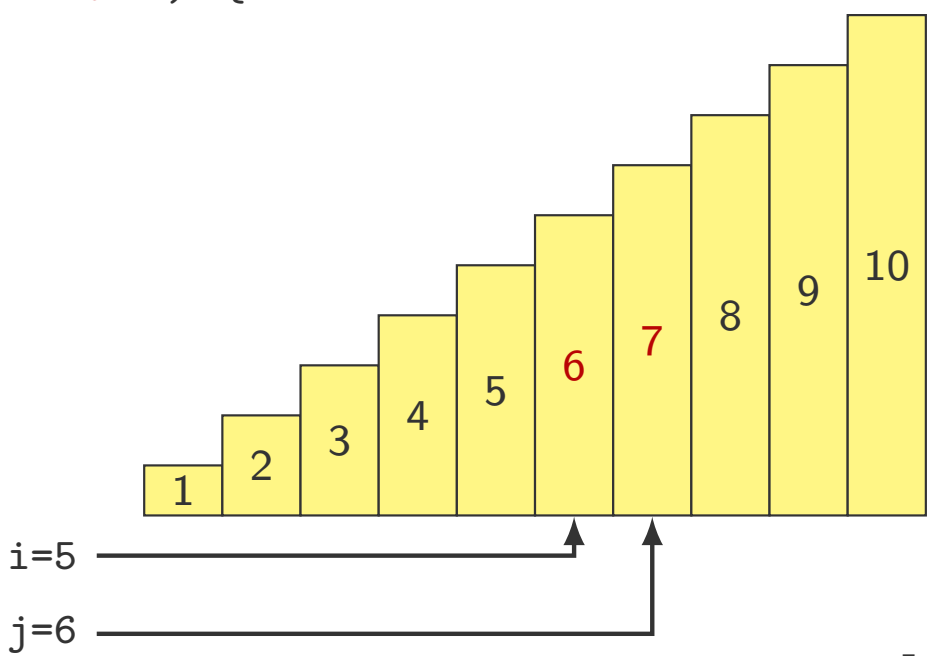
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



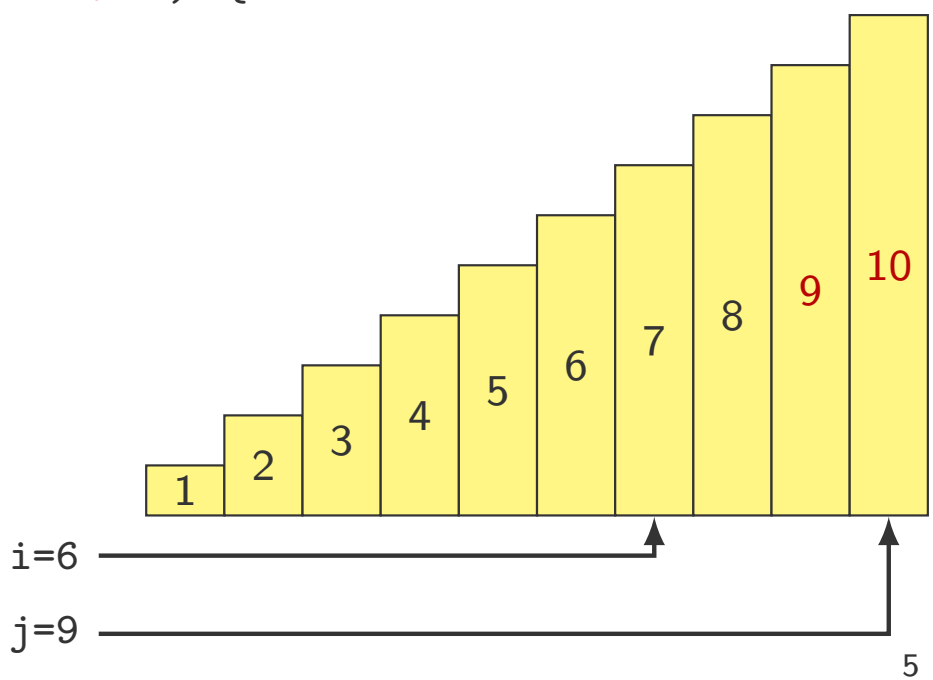
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

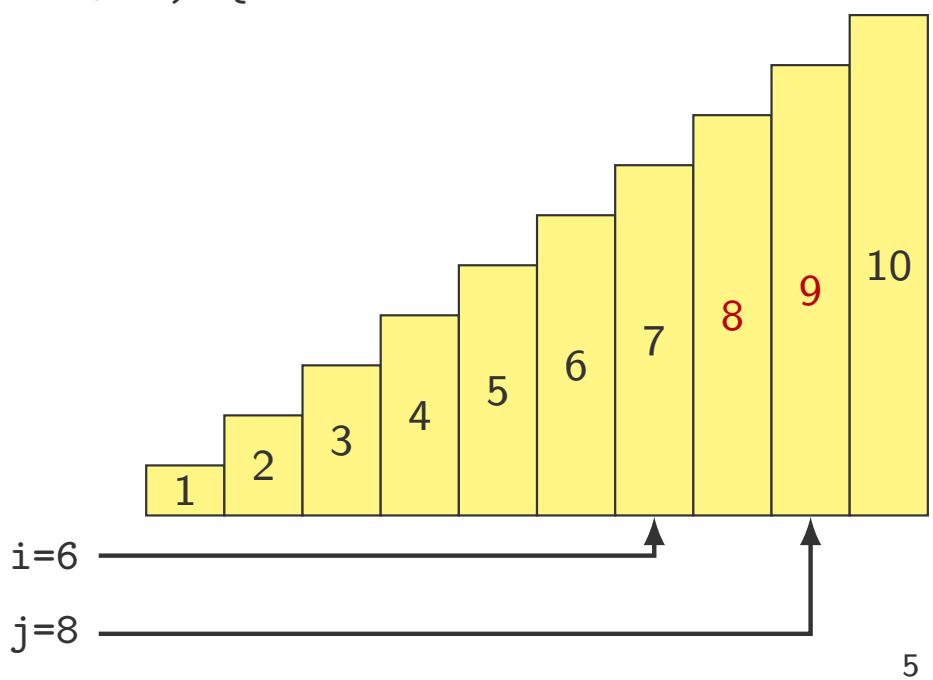


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



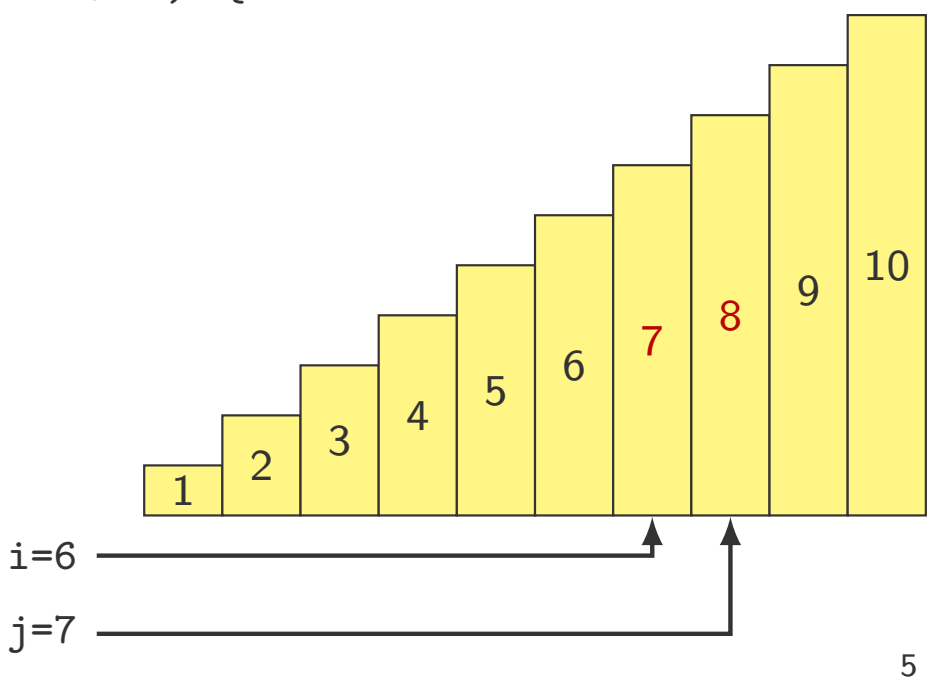
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



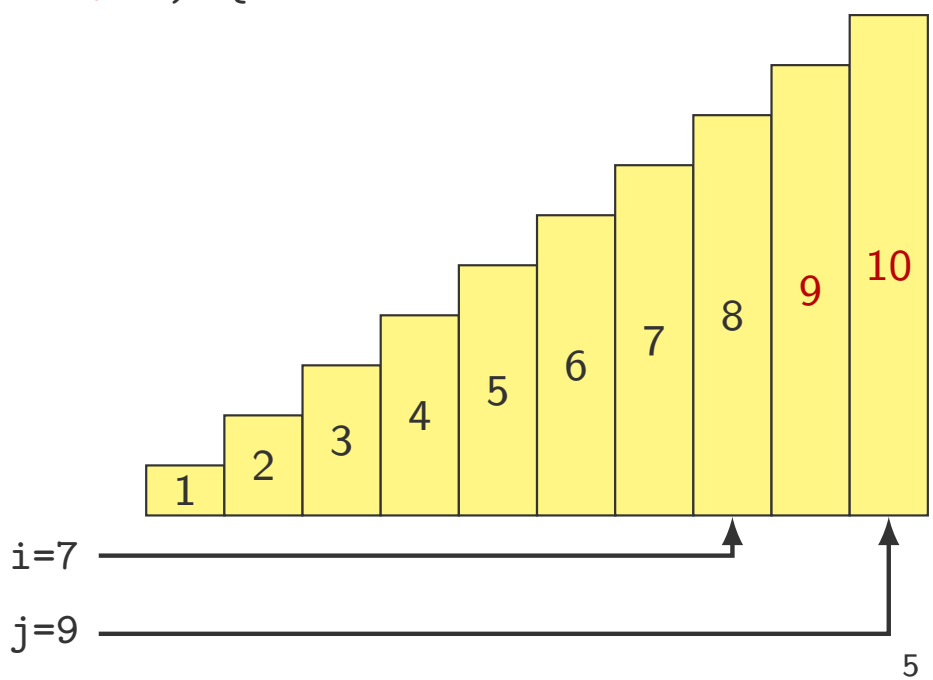
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

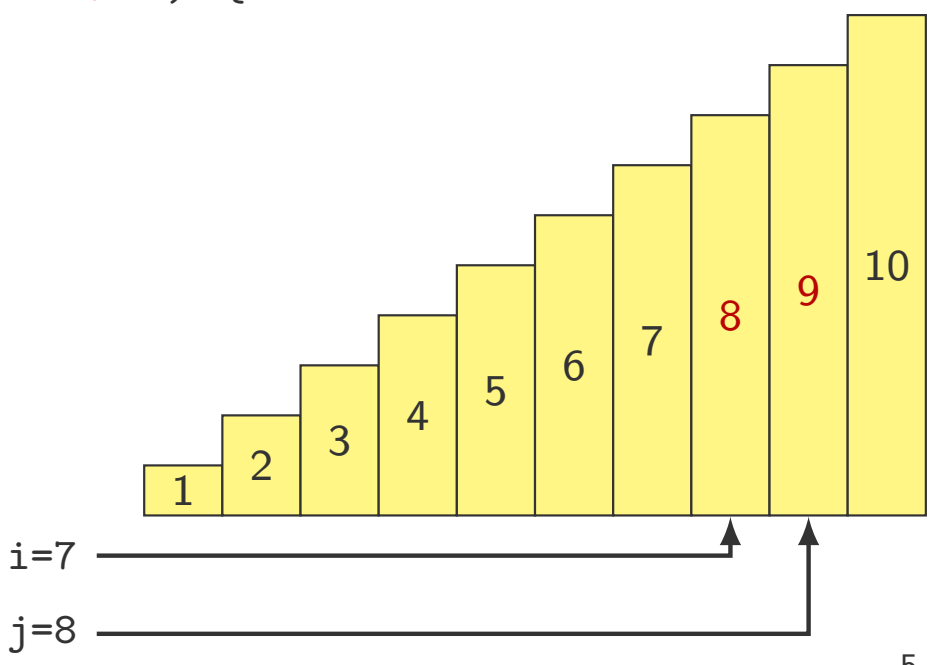


BubbleSort – Ordenação por flutuação

Ideia:

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

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



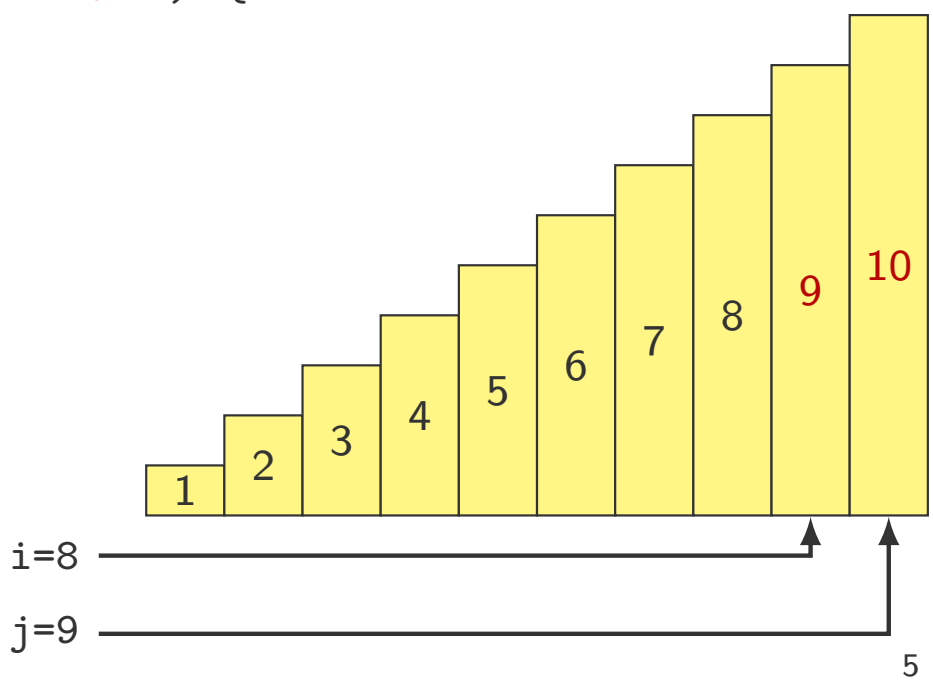
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



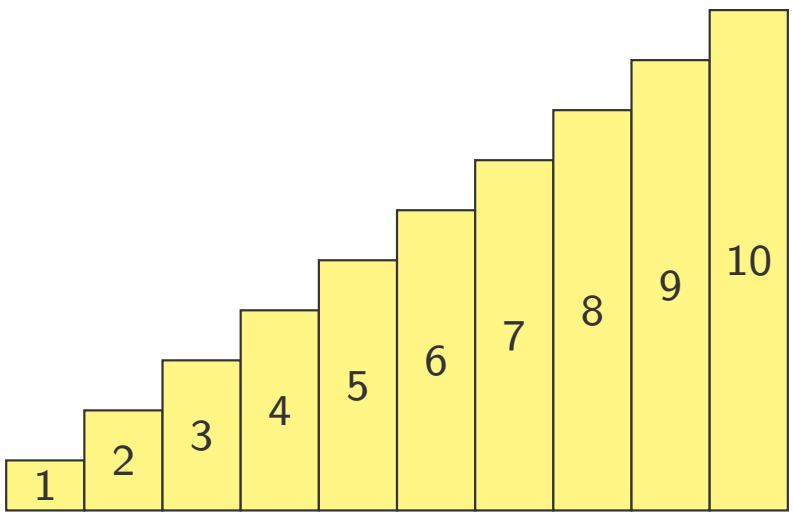
BubbleSort – Ordenação por flutuação

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 os elementos que estiverem antes

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```



i

j

BubbleSort — Complexidade do algoritmo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

BubbleSort — Complexidade do algoritmo


```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

No pior caso toda comparação gera uma troca:

BubbleSort — Complexidade do algoritmo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

No pior caso toda comparação gera uma troca:

- comparações: $n(n-1)/2 = O(n^2)$

BubbleSort — Complexidade do algoritmo

Universidade Federal do Ceará
Campus Quixadá

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

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

Universidade Federal do Ceará
Campus Quixadá

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

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 l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

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 l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

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 l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

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 l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

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:

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

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: $O(n)$

Parando quando não há mais trocas

Se não aconteceu nenhuma troca, podemos parar o algoritmo

```
1  void bubblesort_v2(int A[], int l, int r) {
2      bool trocou = true;
3      for (int i = l; i < r && trocou; i++){
4          trocou = false;
5          for (int j = r; j > i; j--)
6              if (A[j] < A[j-1]) {
7                  std::swap(A[j],A[j-1]);
8                  trocou = true;
9              }
10     }
11 }
```

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: $O(1)$

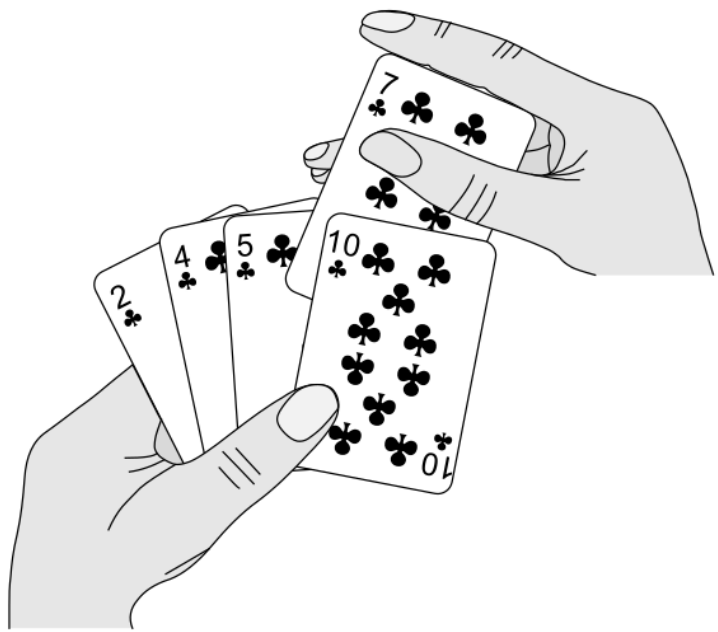
InsertionSort

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.

Ordenação por Inserção

Algoritmo

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

Testar com o vetor: $[4, 8, 2, 7, 6, 3, 5, 1, 9, 10]$

InsertionSort - Complexidade do algoritmo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void insertionsort(int A[], int l, int r) {
2      for (int j = l+1; j <= r; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= l && 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 l, int r) {
2      for (int j = l+1; j <= r; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= l && 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


```
1  void insertionsort(int A[], int l, int r) {
2      for (int j = l+1; j <= r; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= l && 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 l, int r) {
2      for (int j = l+1; j <= r; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= l && 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 6 é igual a $\sum_{j=1}^{n-1} j$ no pior caso.

InsertionSort - Complexidade do algoritmo


```
1  void insertionsort(int A[], int l, int r) {
2      for (int j = l+1; j <= r; j++) {
3          int key = A[j];
4          int i = j-1;
5          while(i >= l && 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 6 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

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```

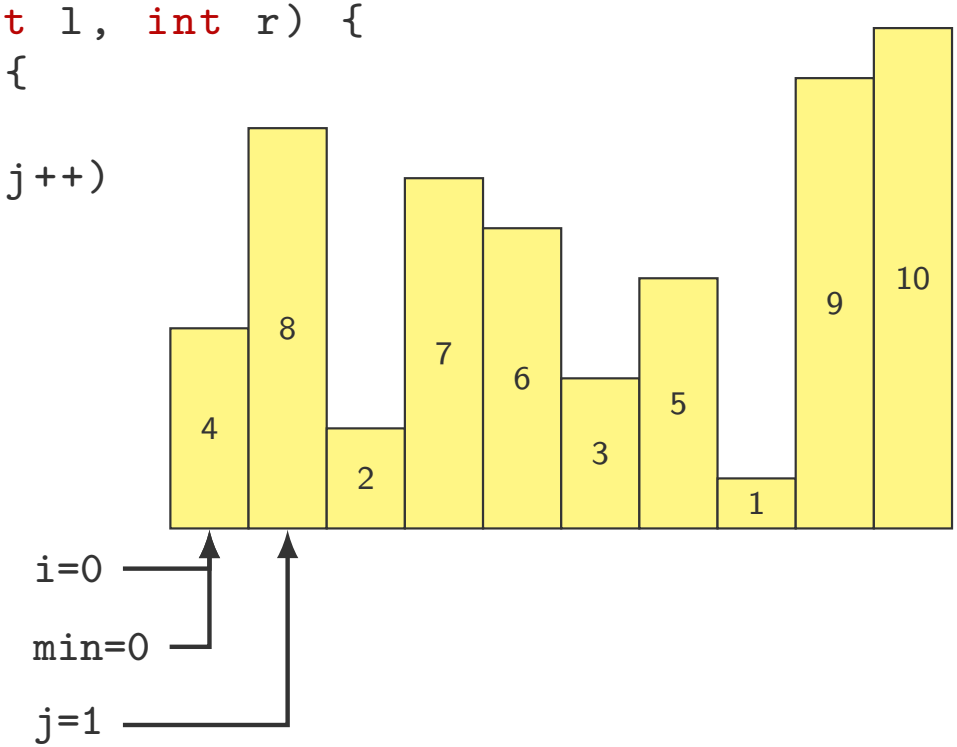
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



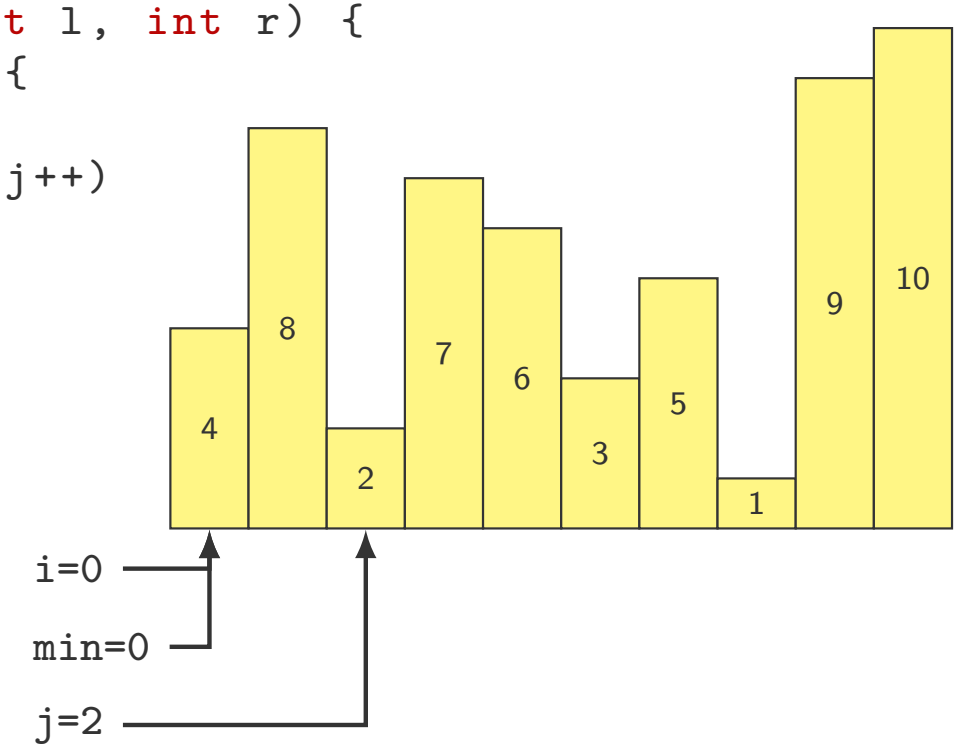
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



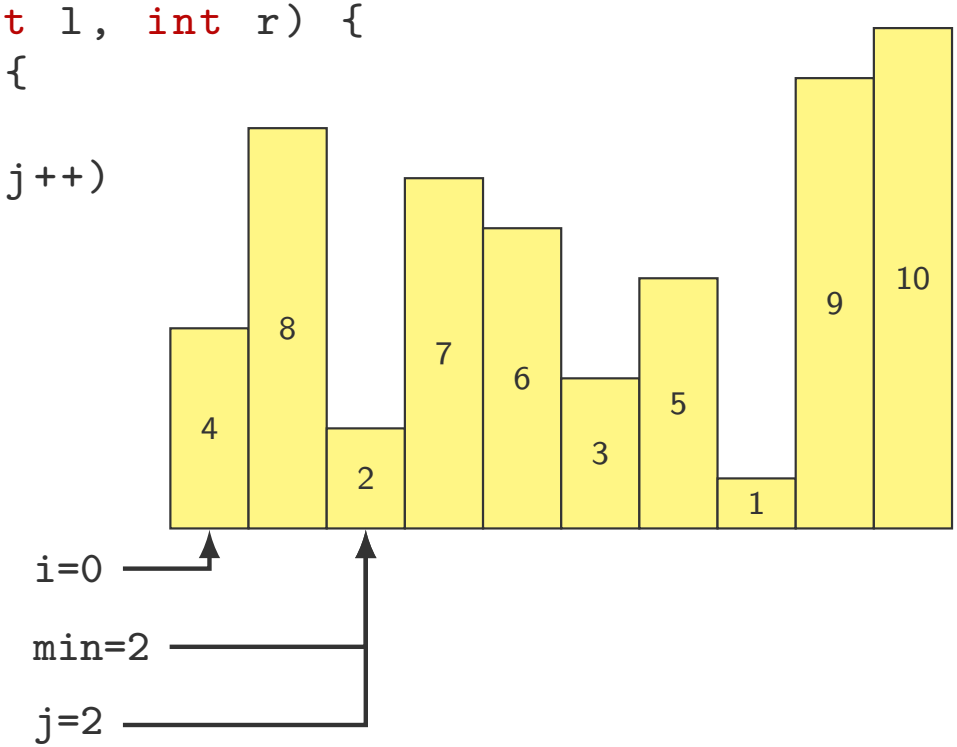
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



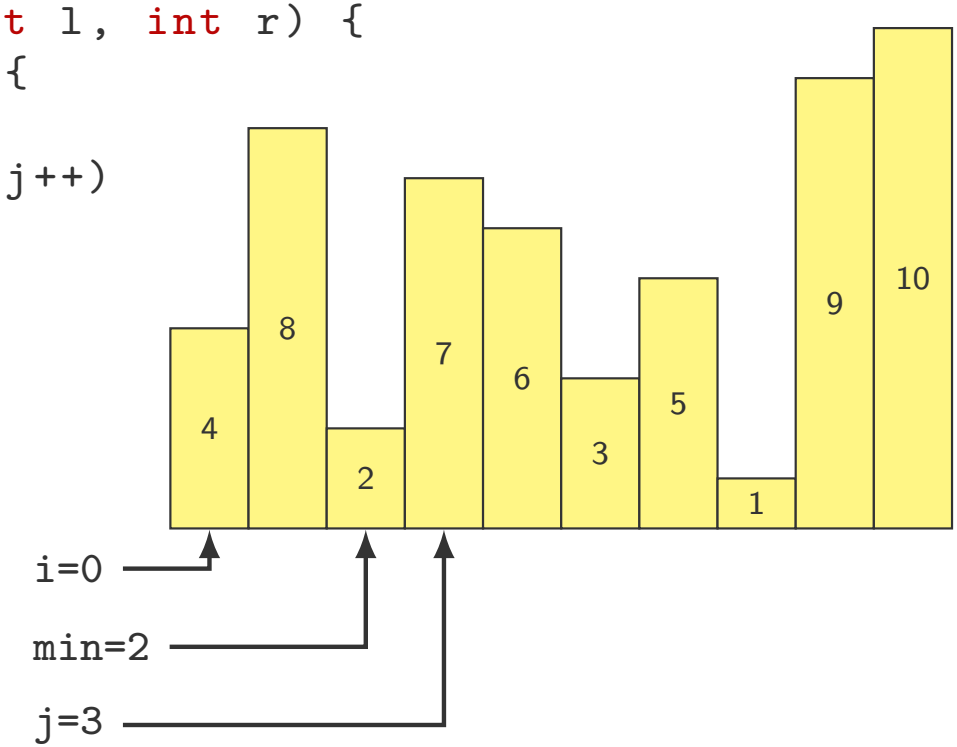
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



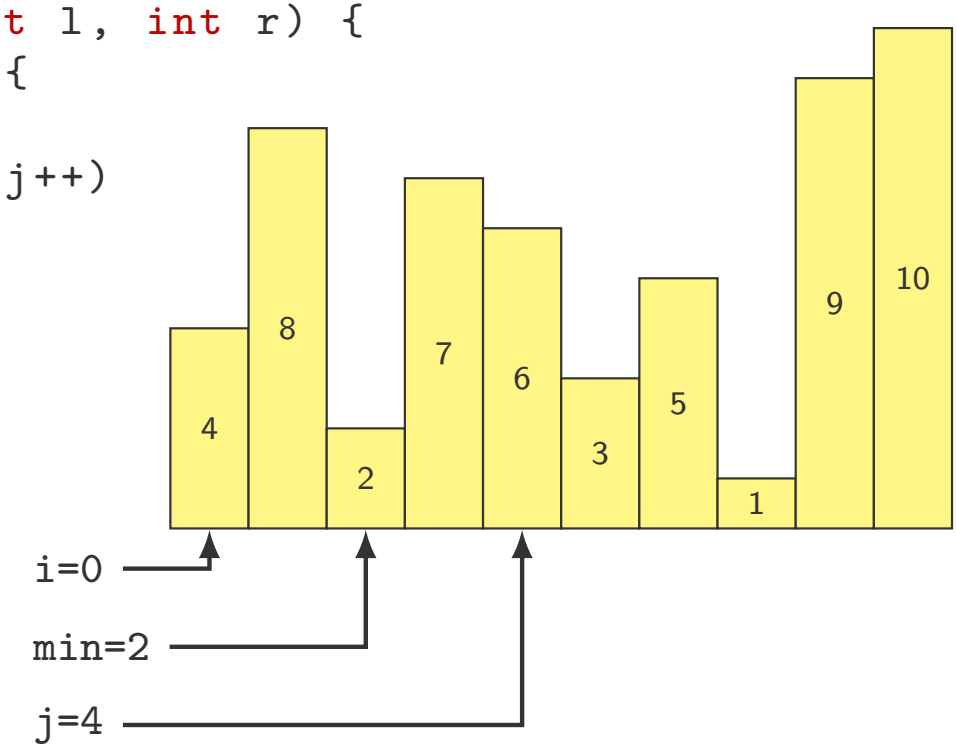
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



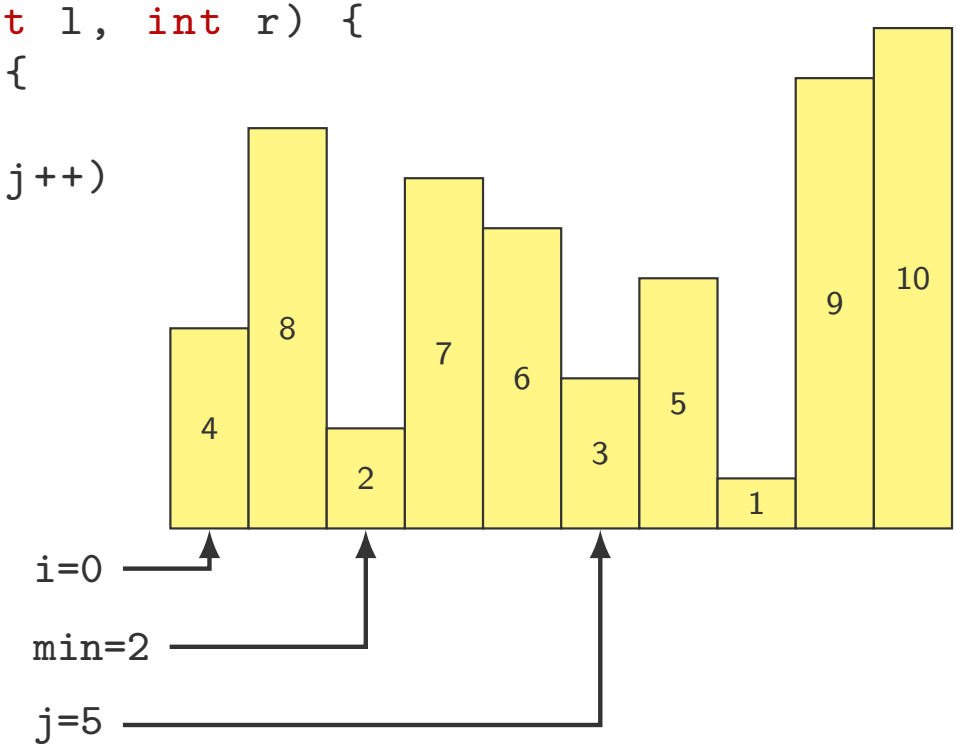
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



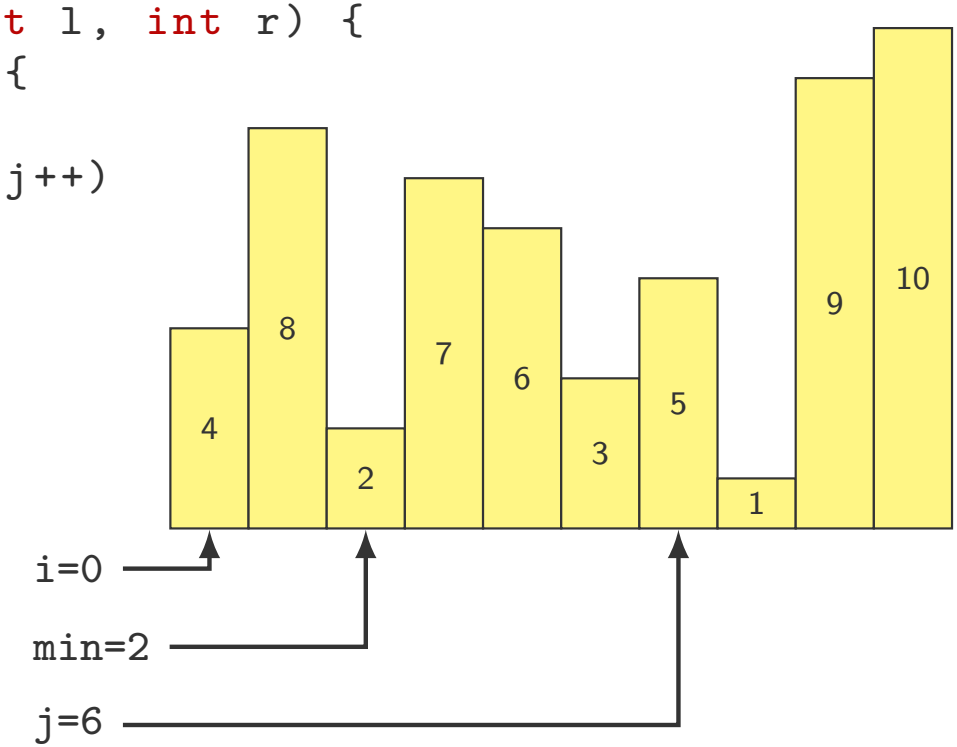
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



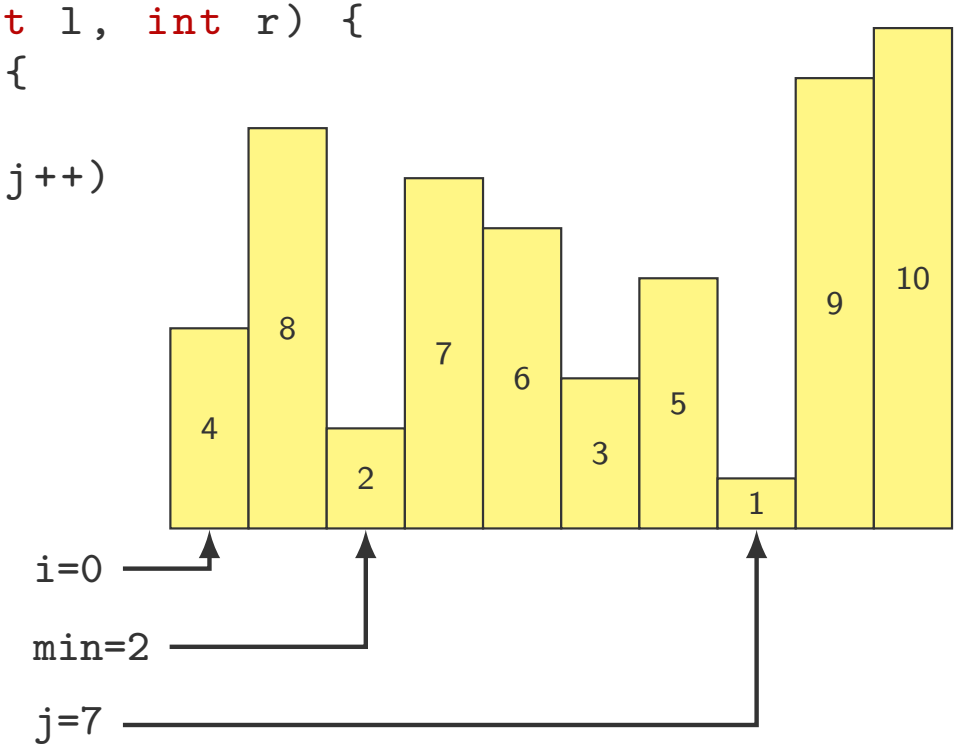
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



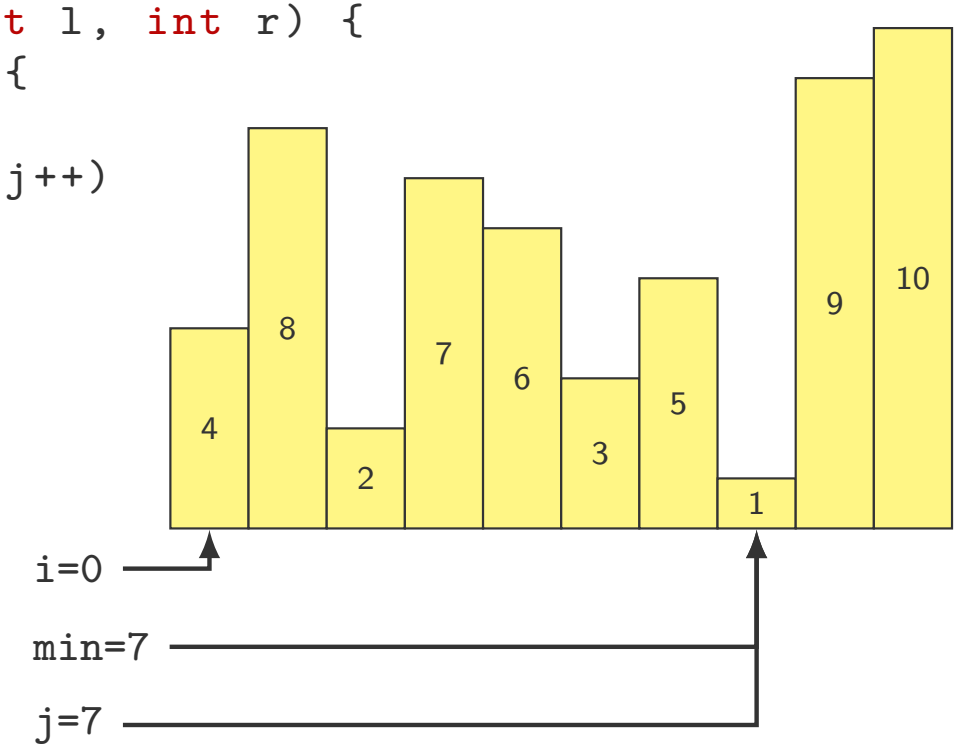
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



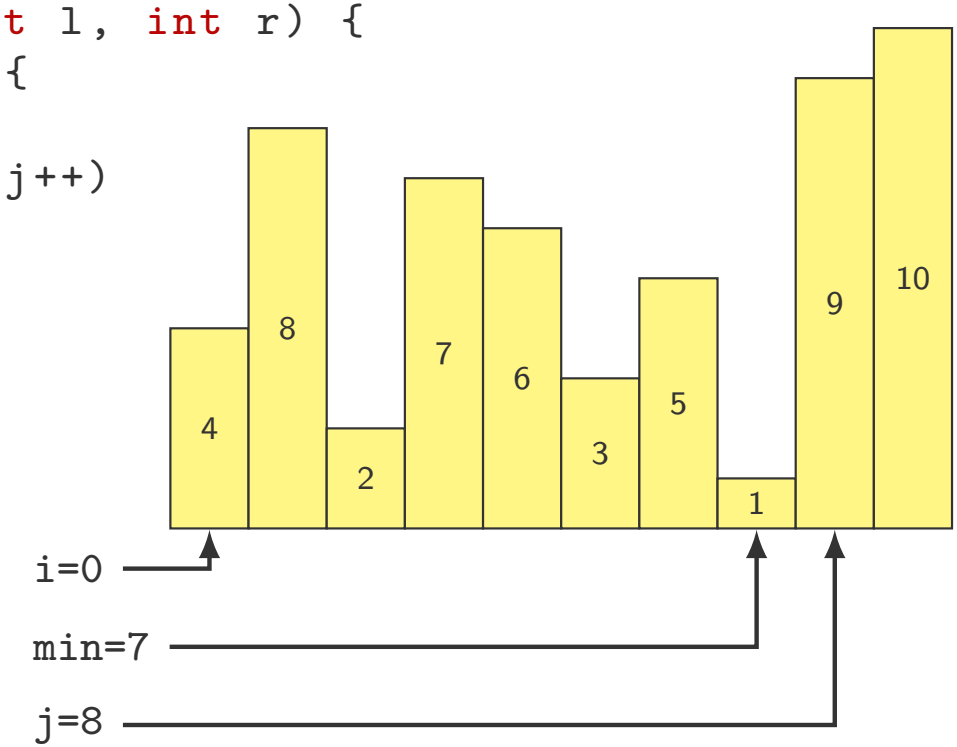
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



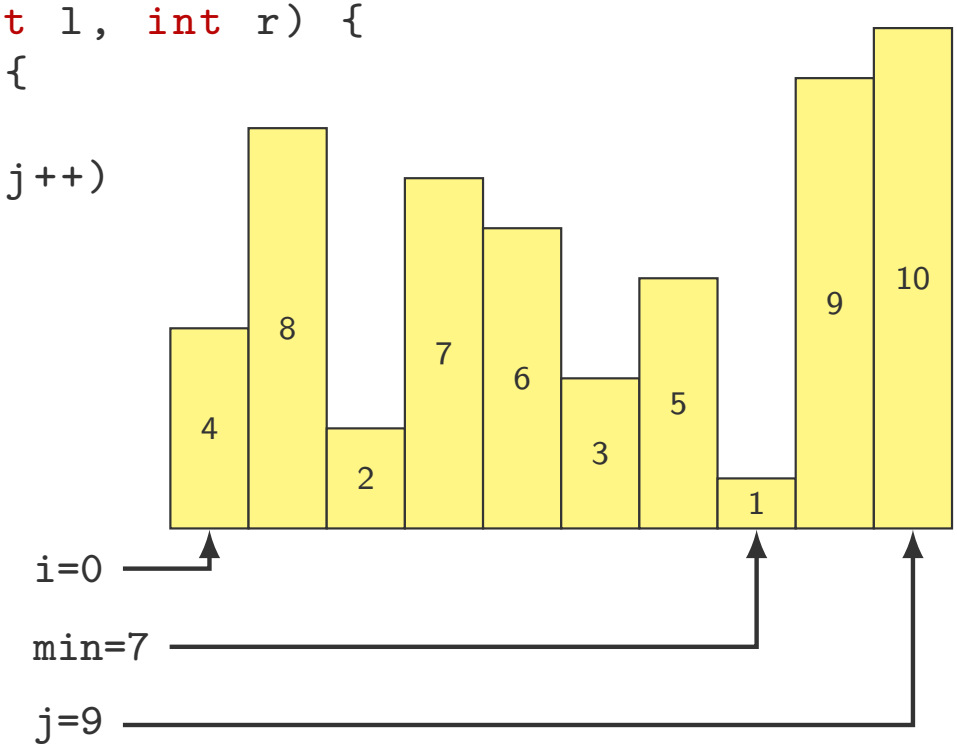
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



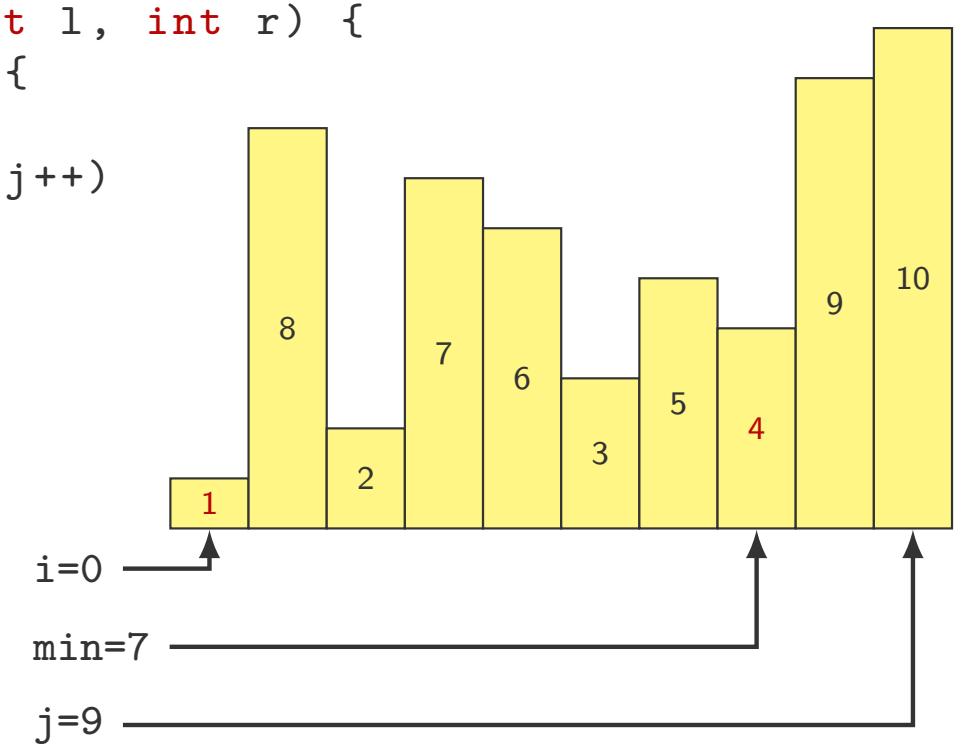
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



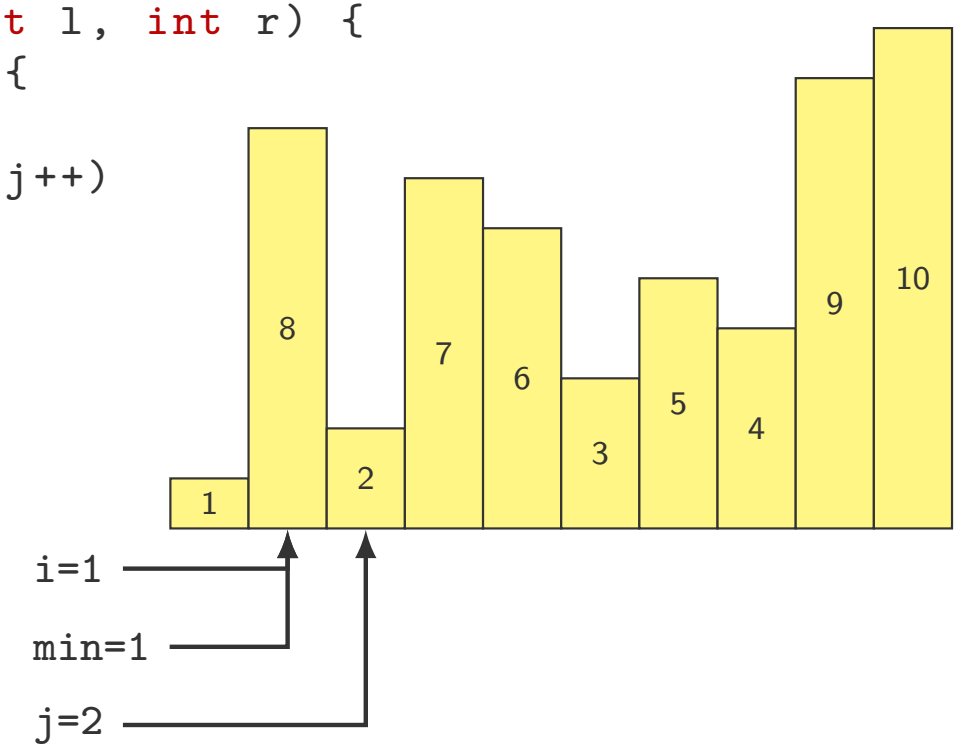
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



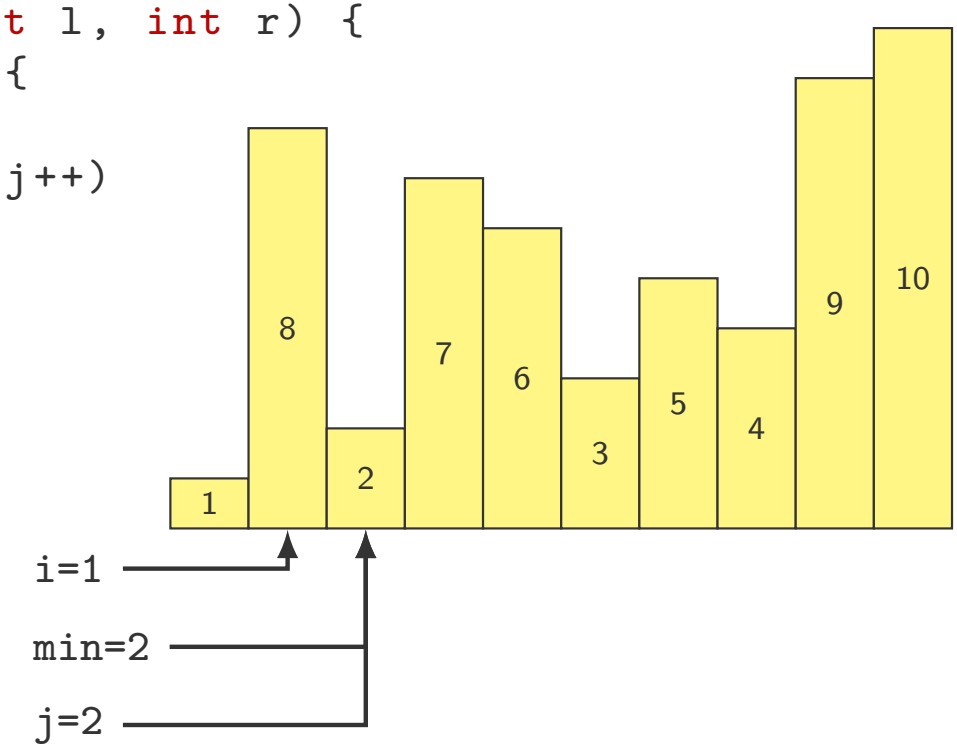
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



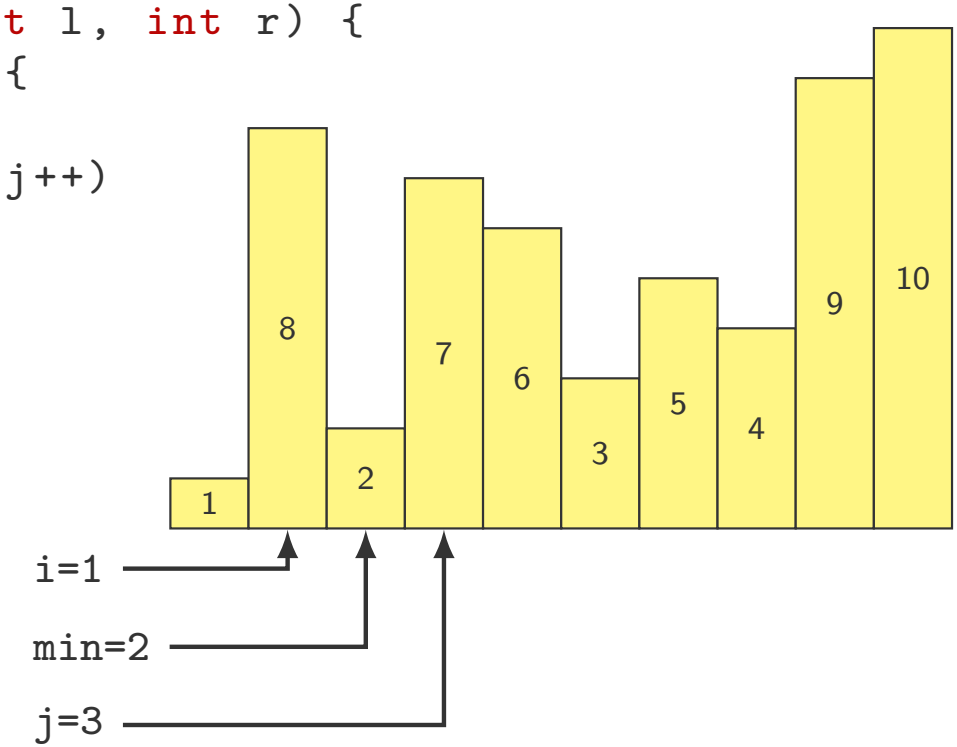
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



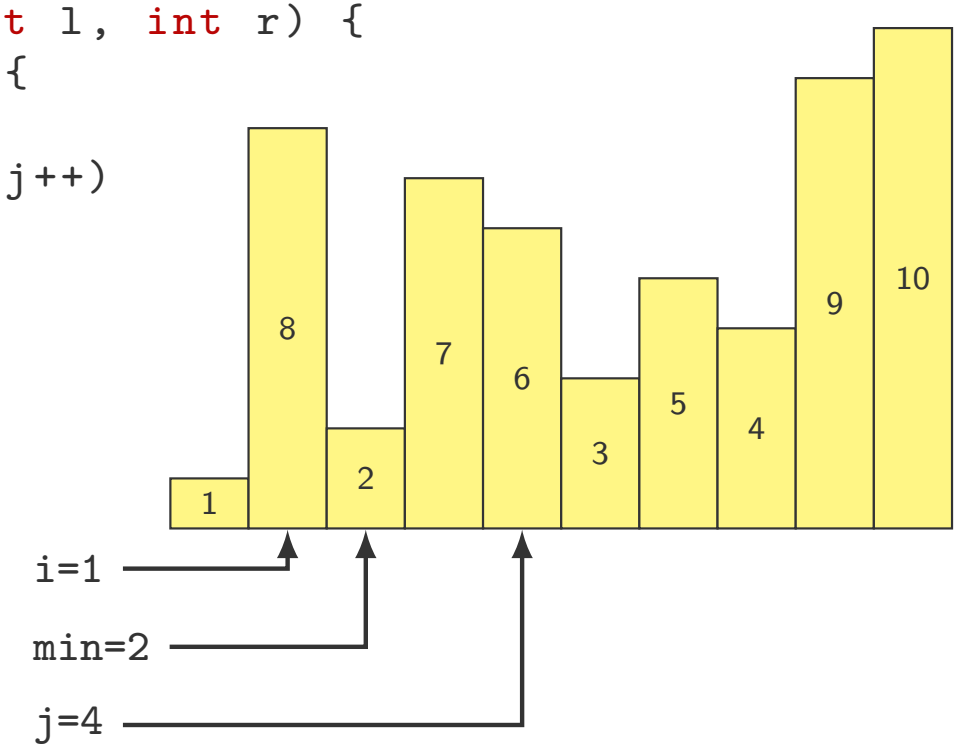
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



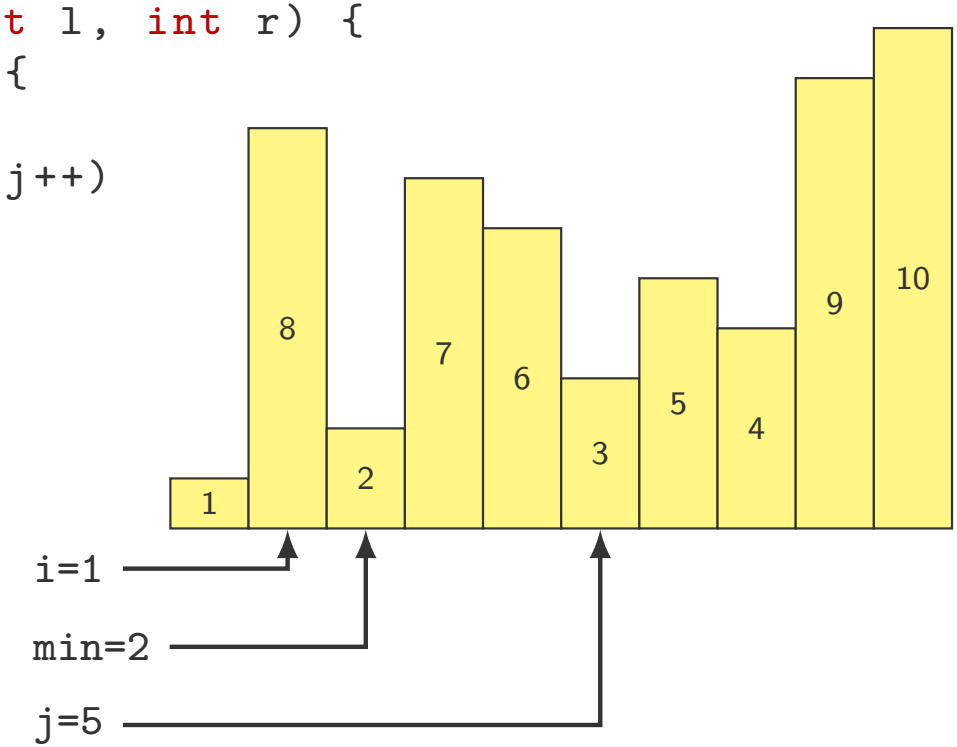
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



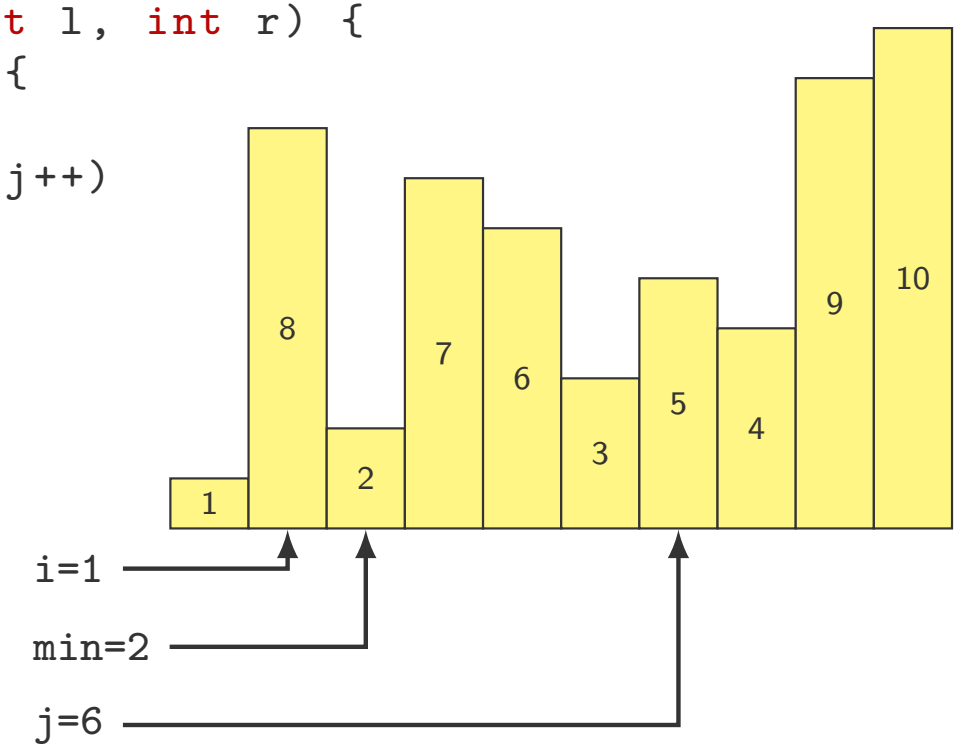
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



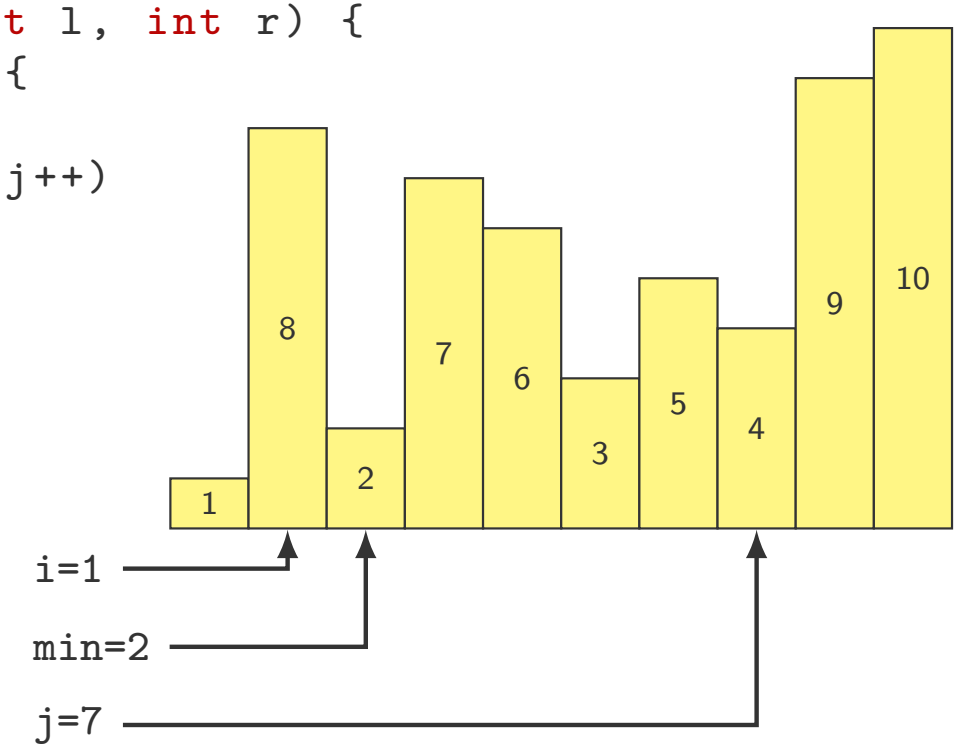
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



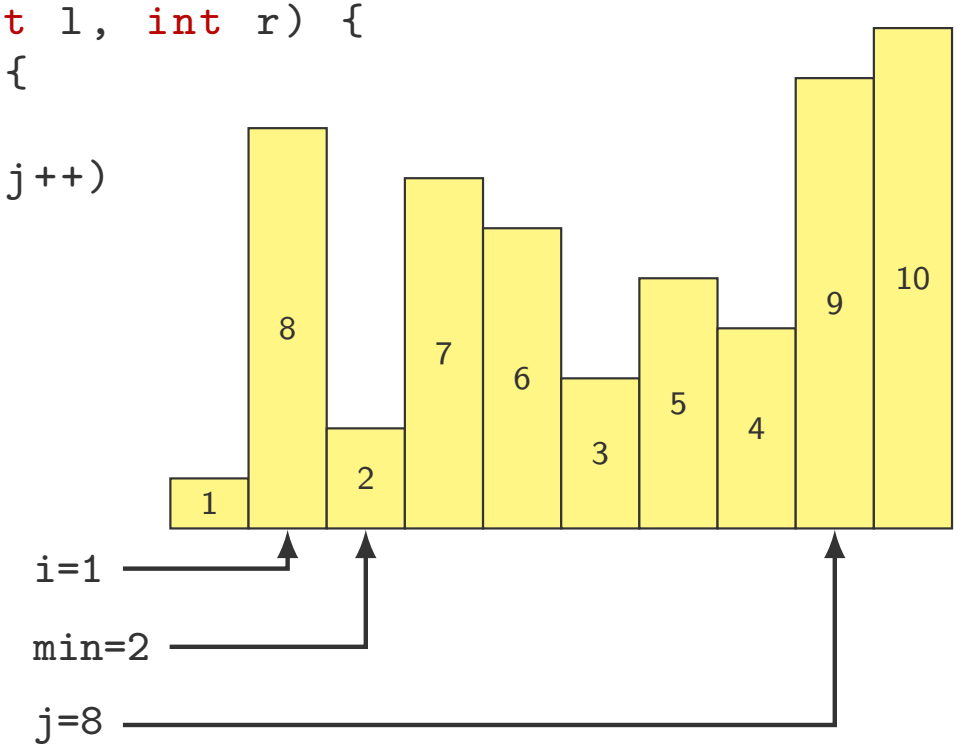
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



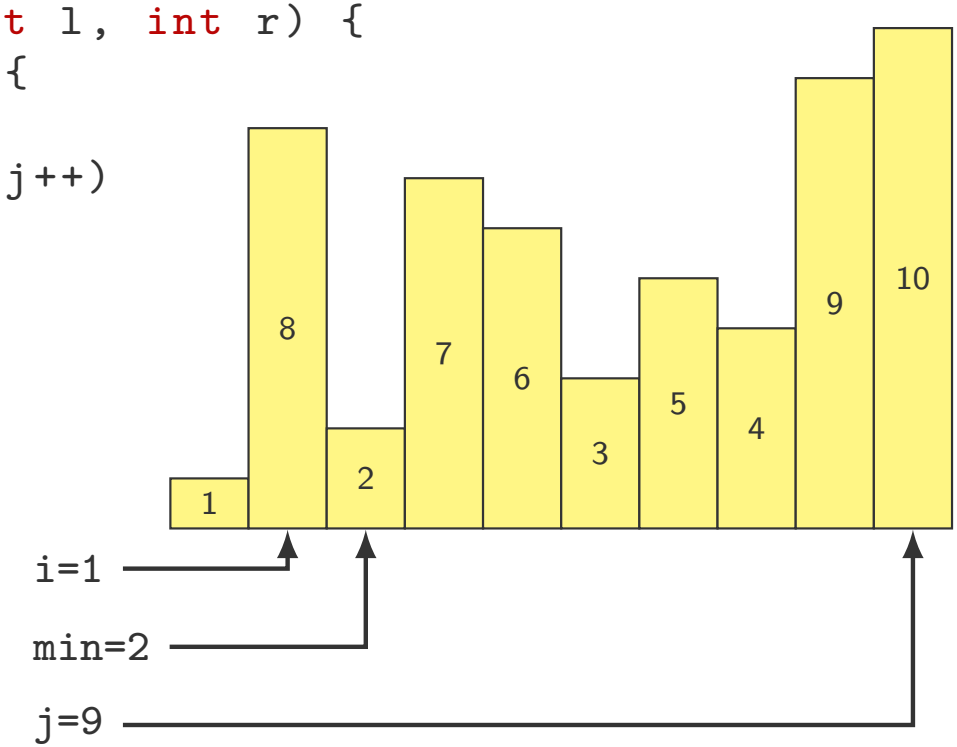
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



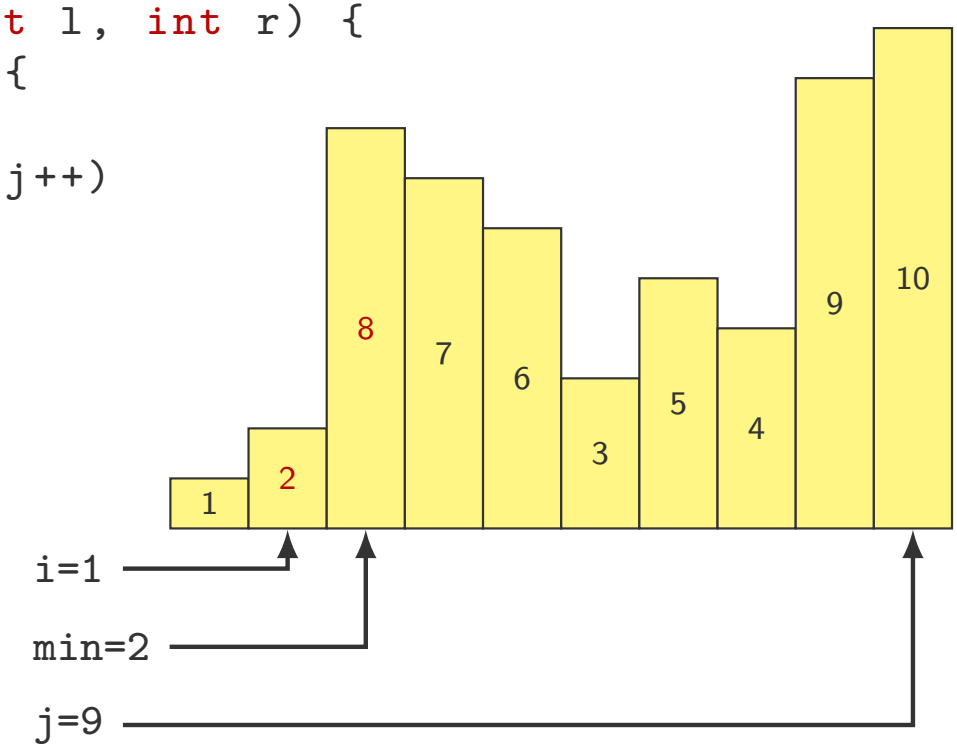
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



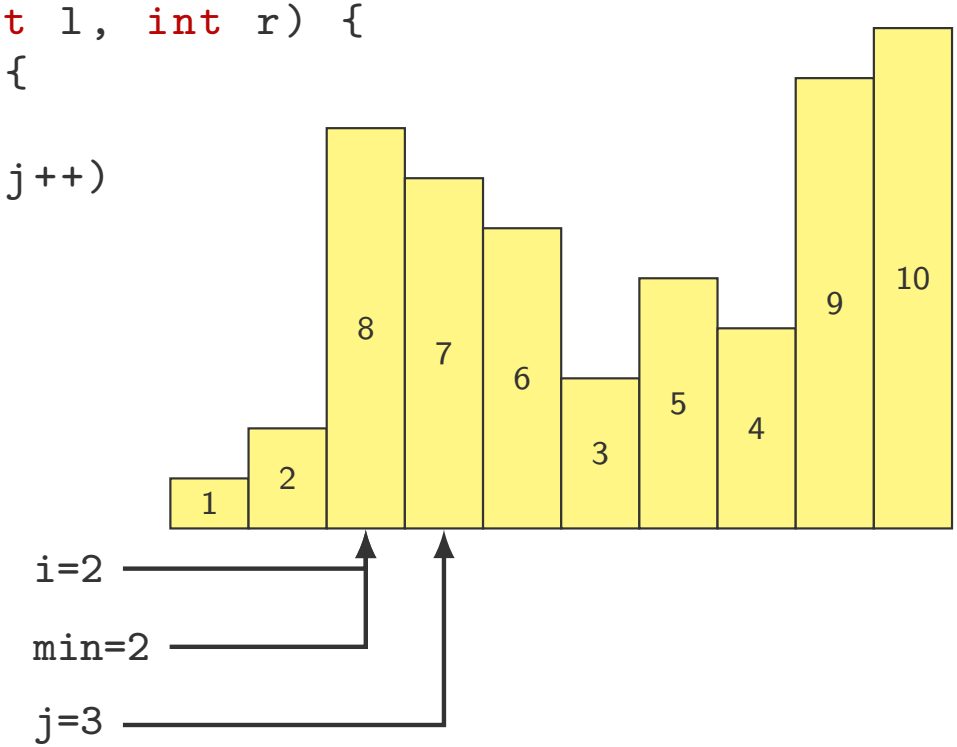
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



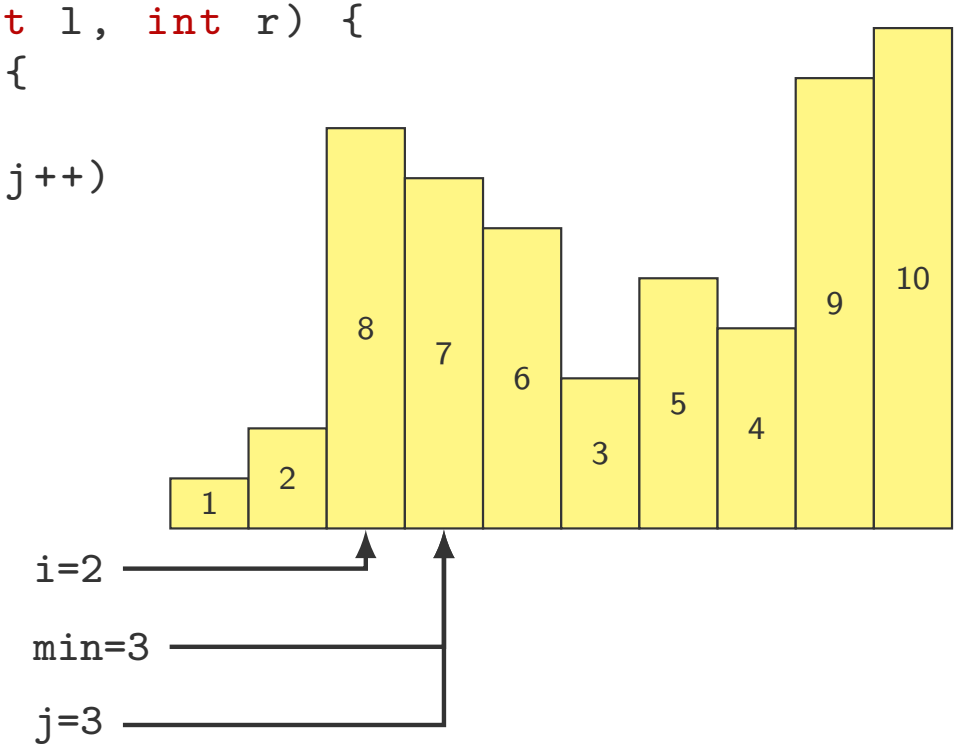
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



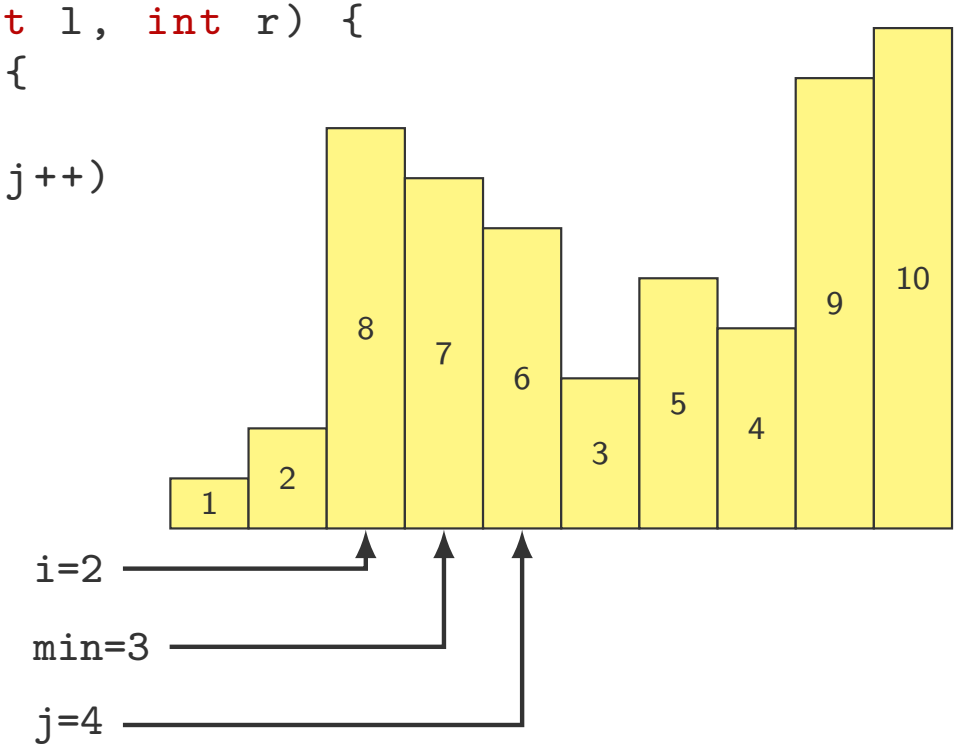
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



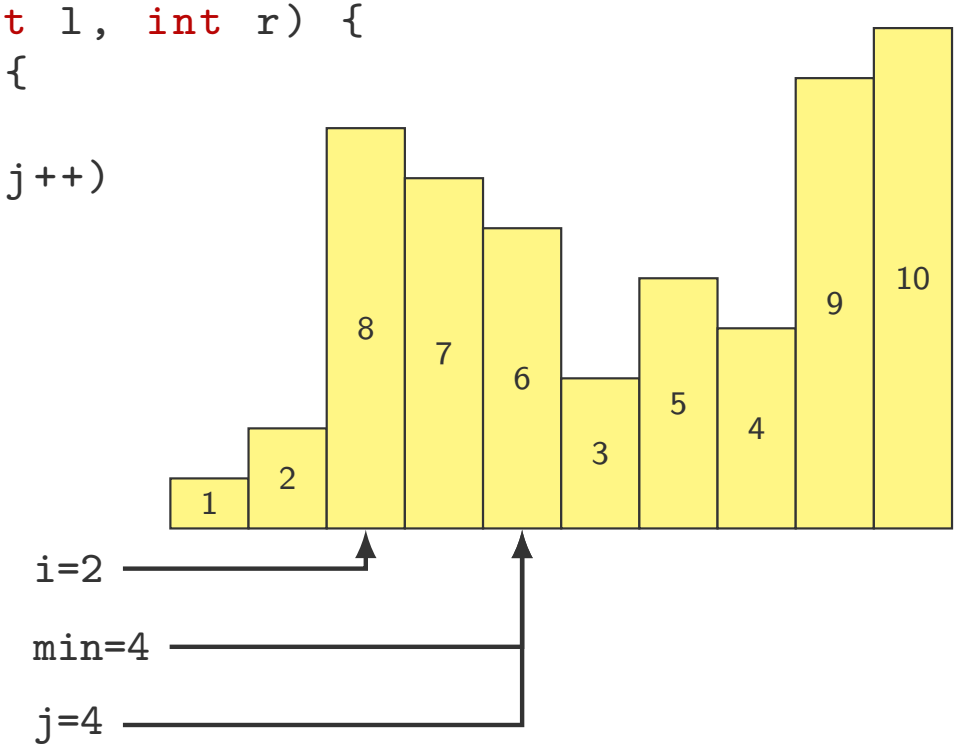
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



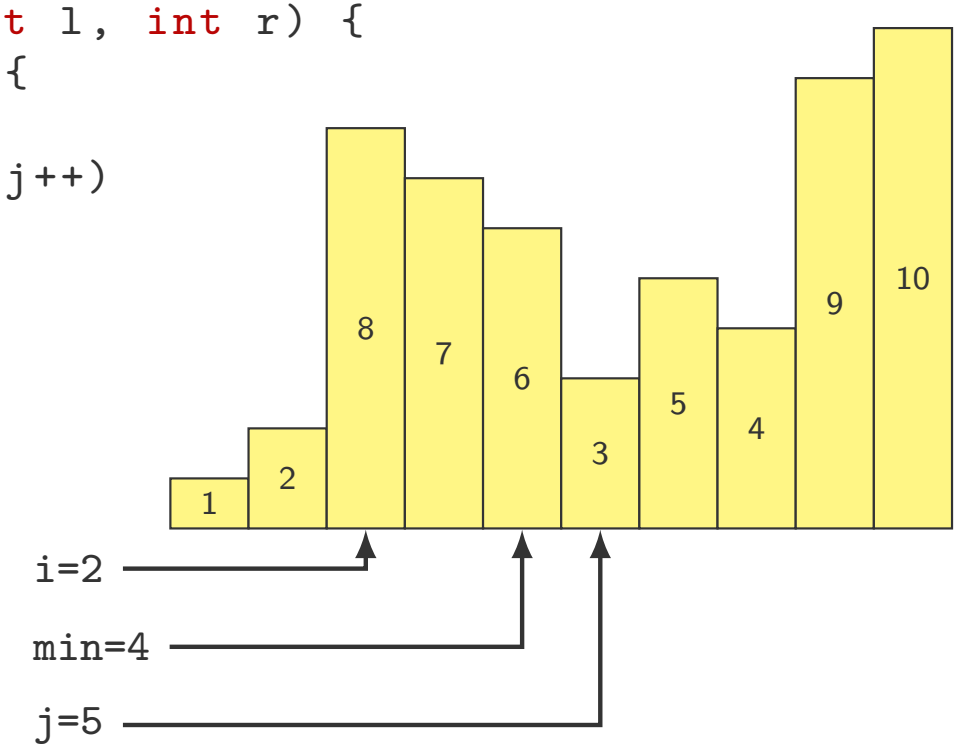
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



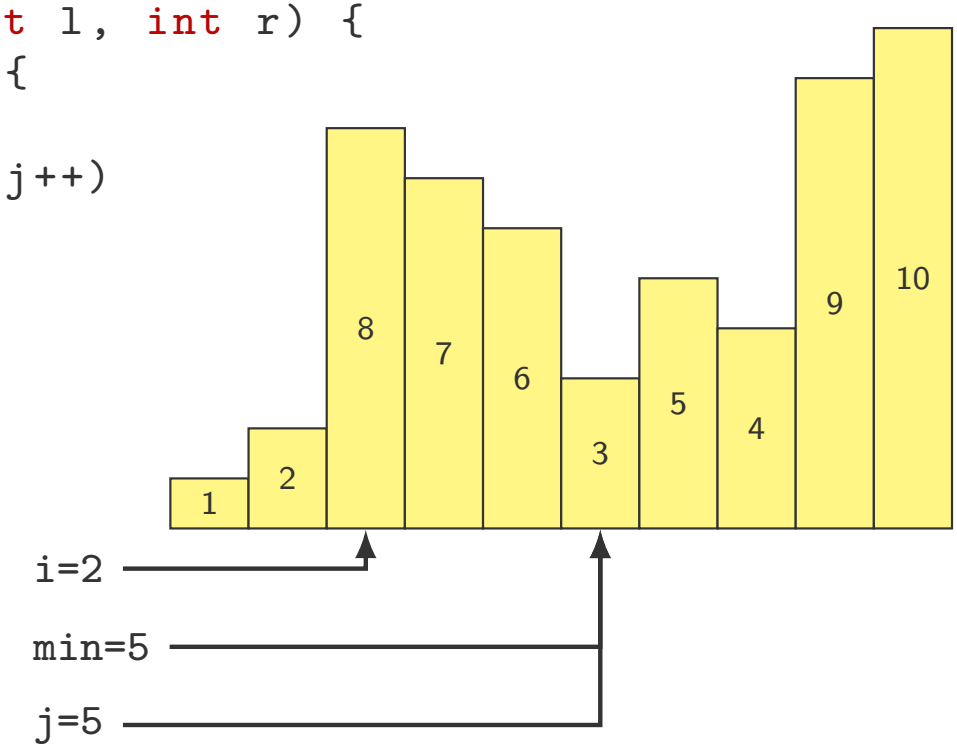
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



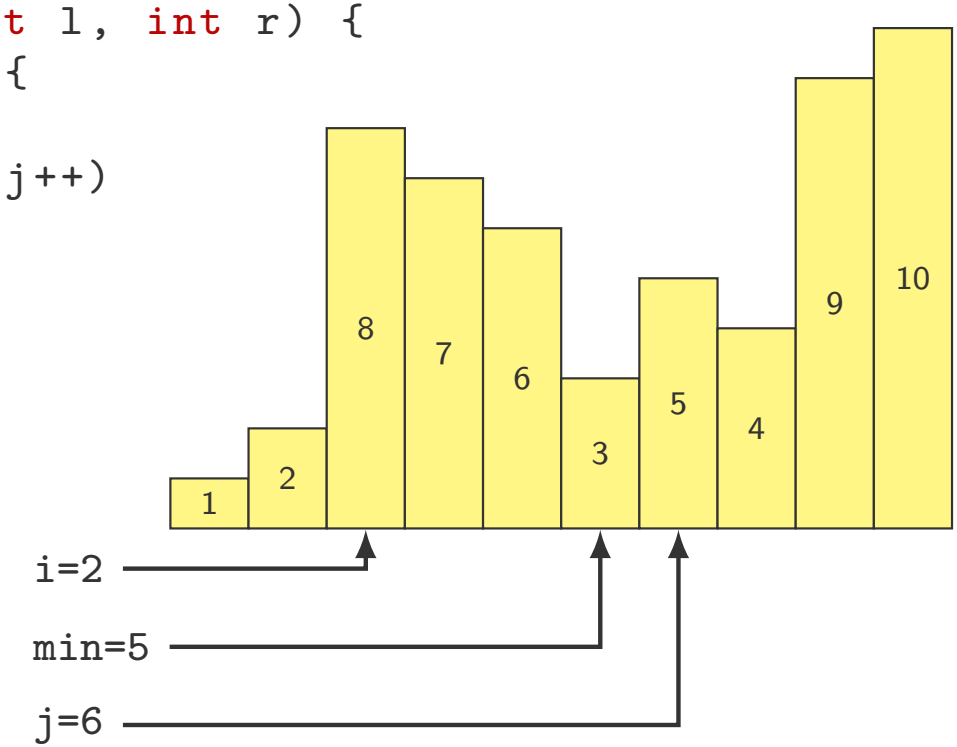
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



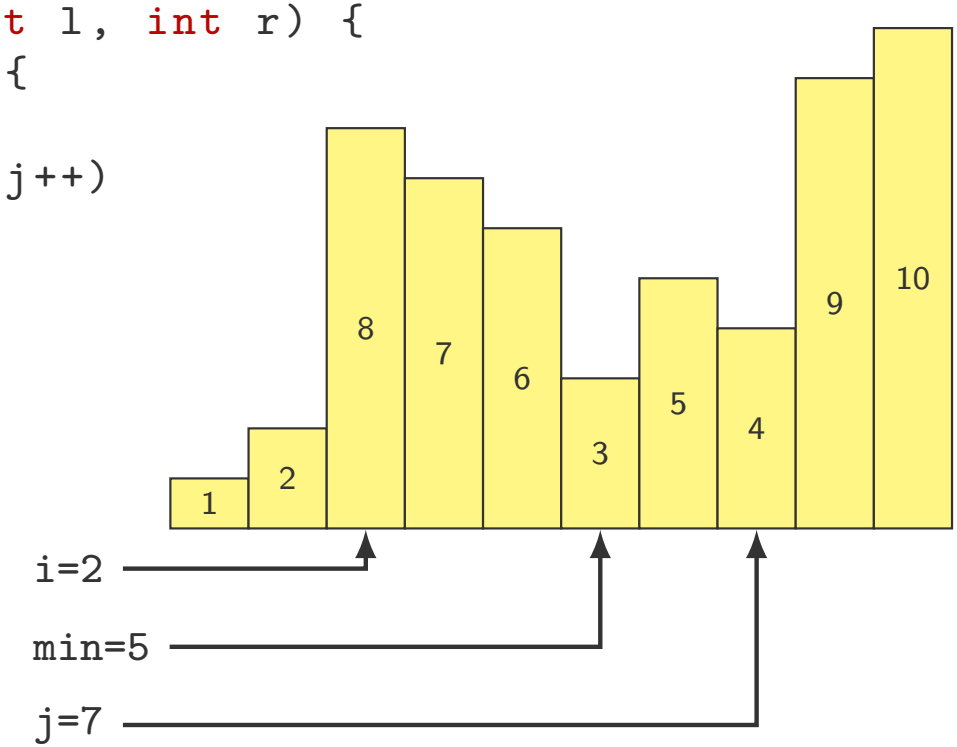
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



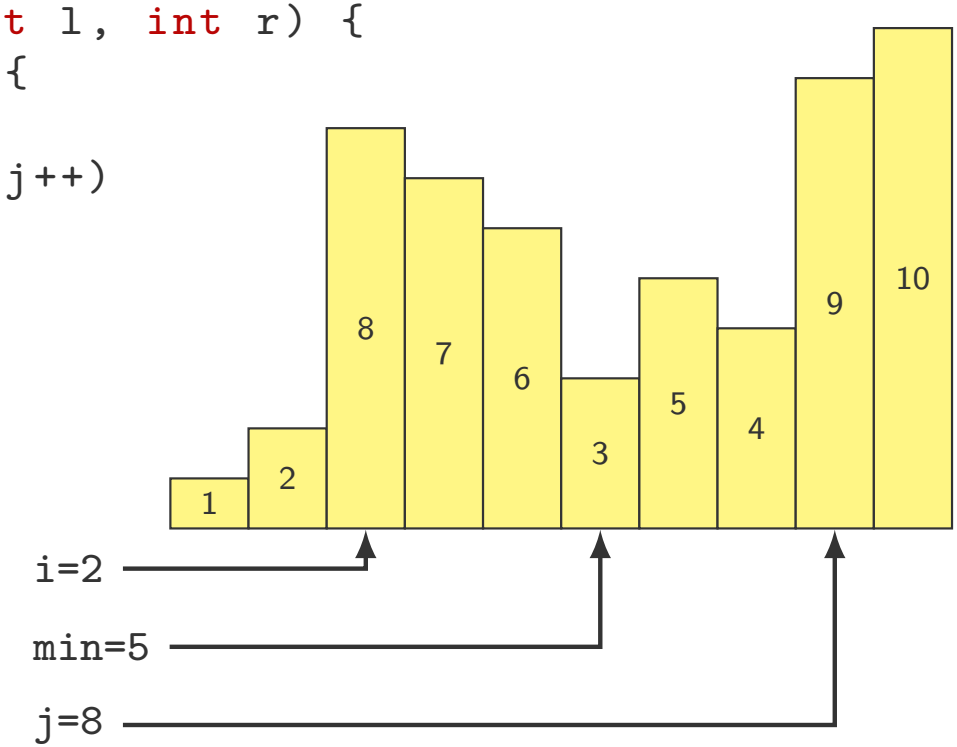
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



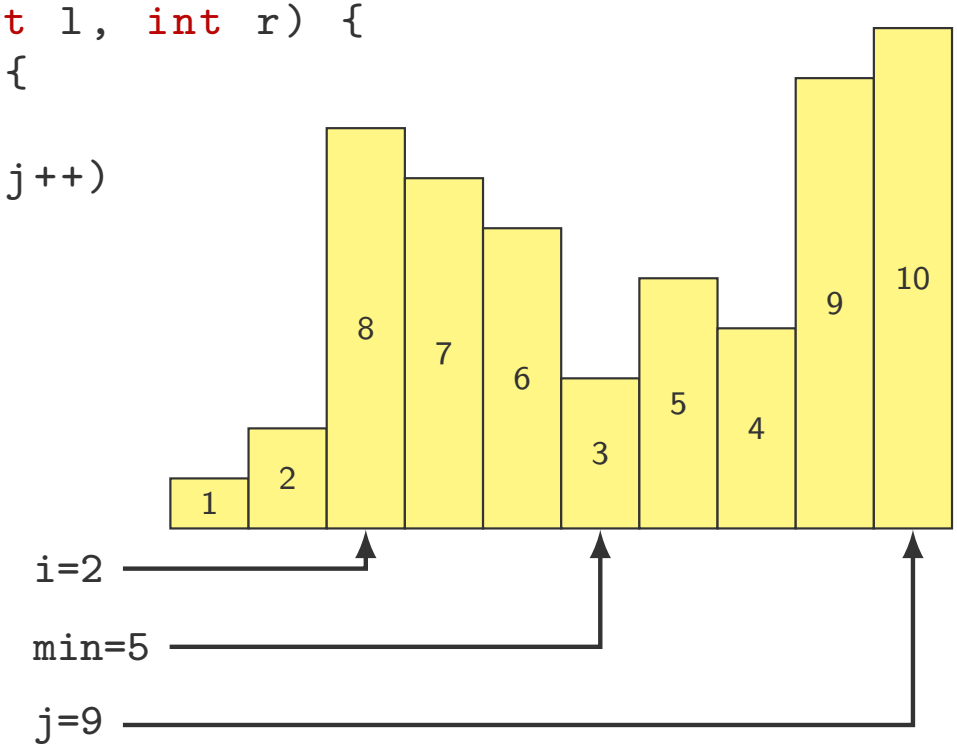
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



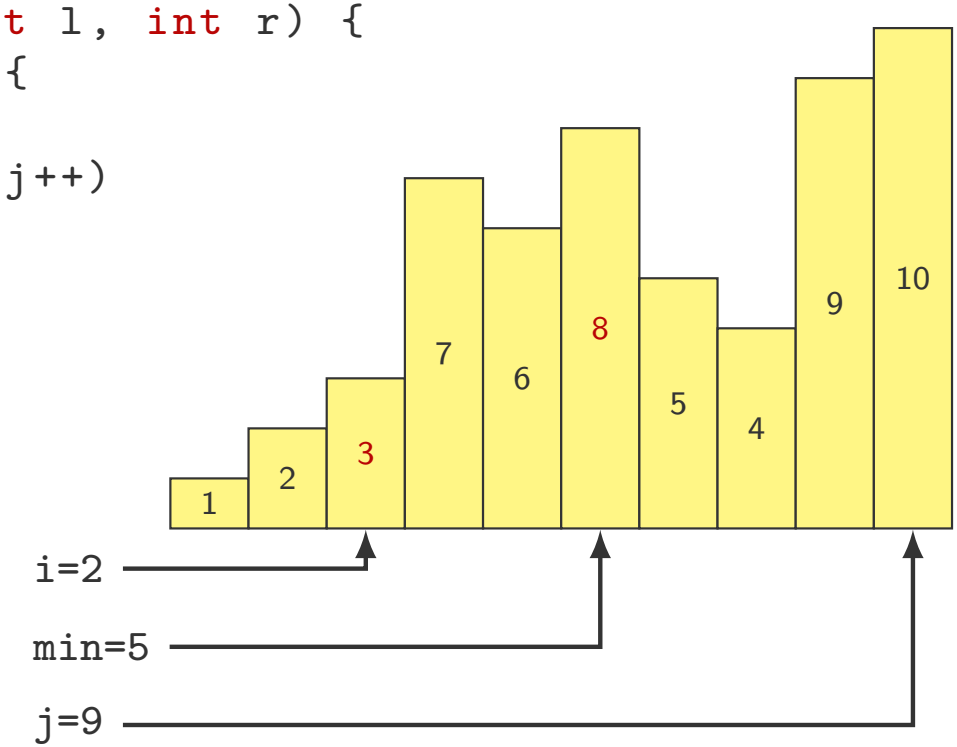
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



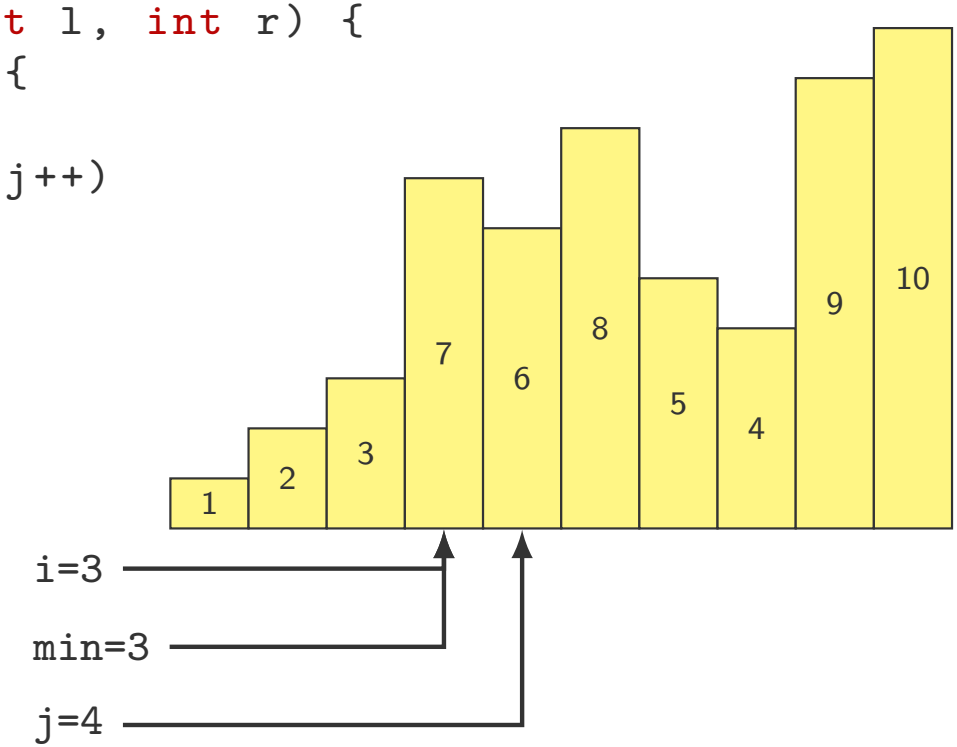
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



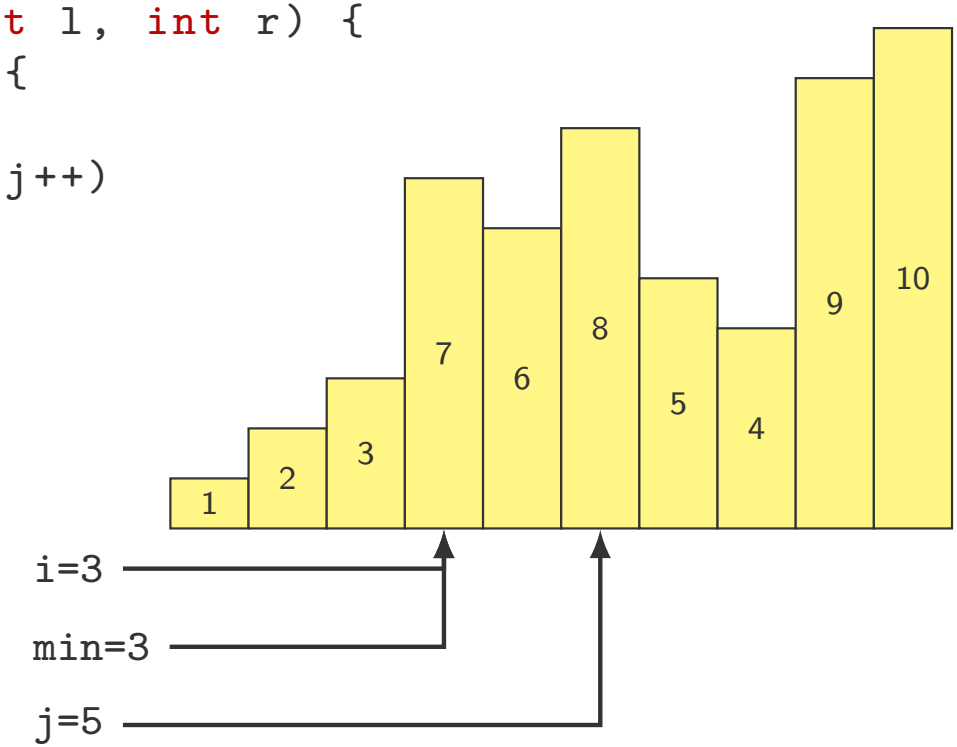
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



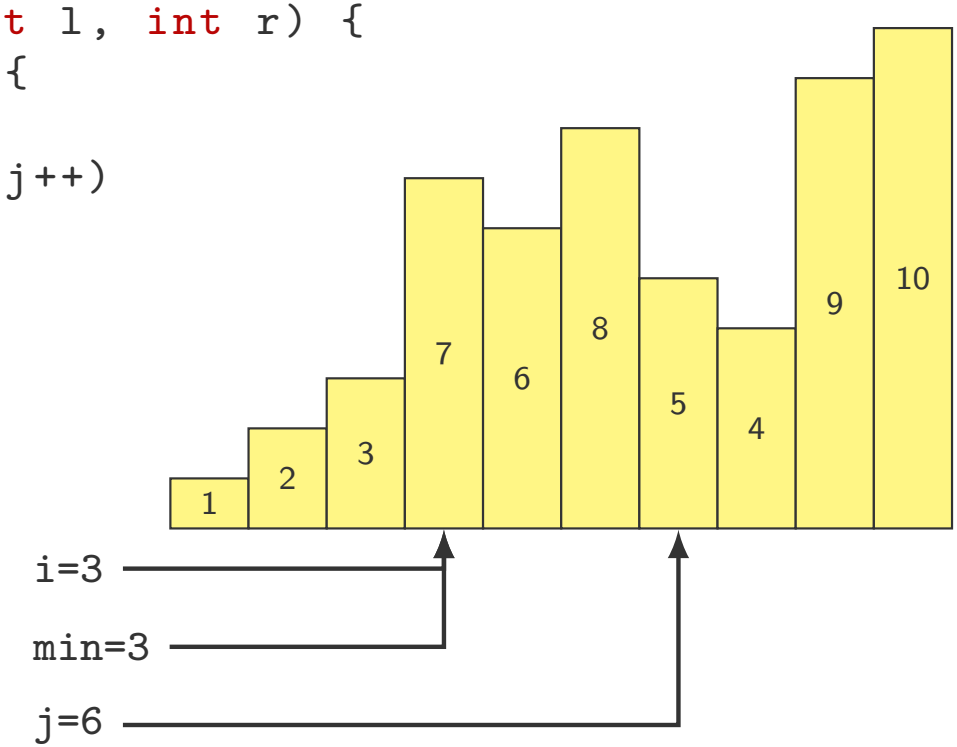
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



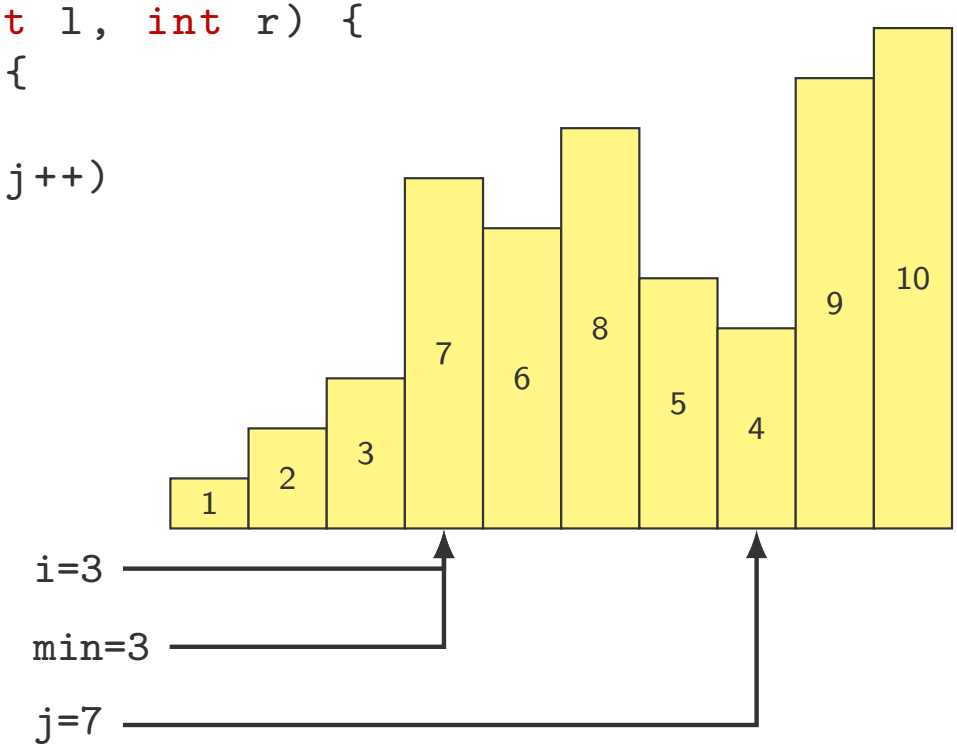
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```

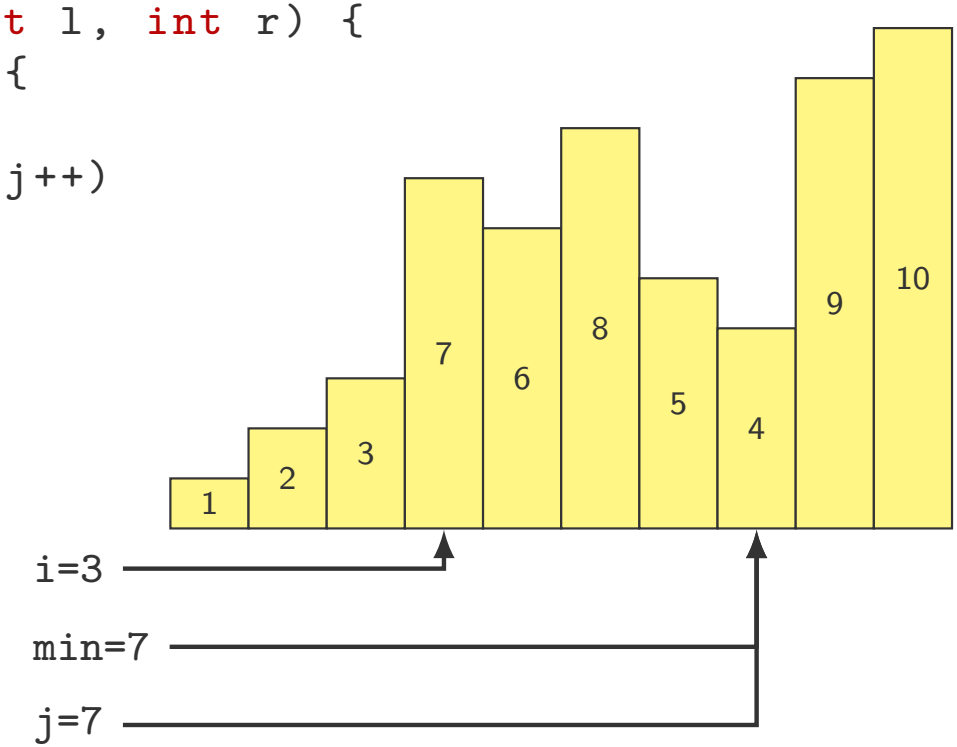


Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```

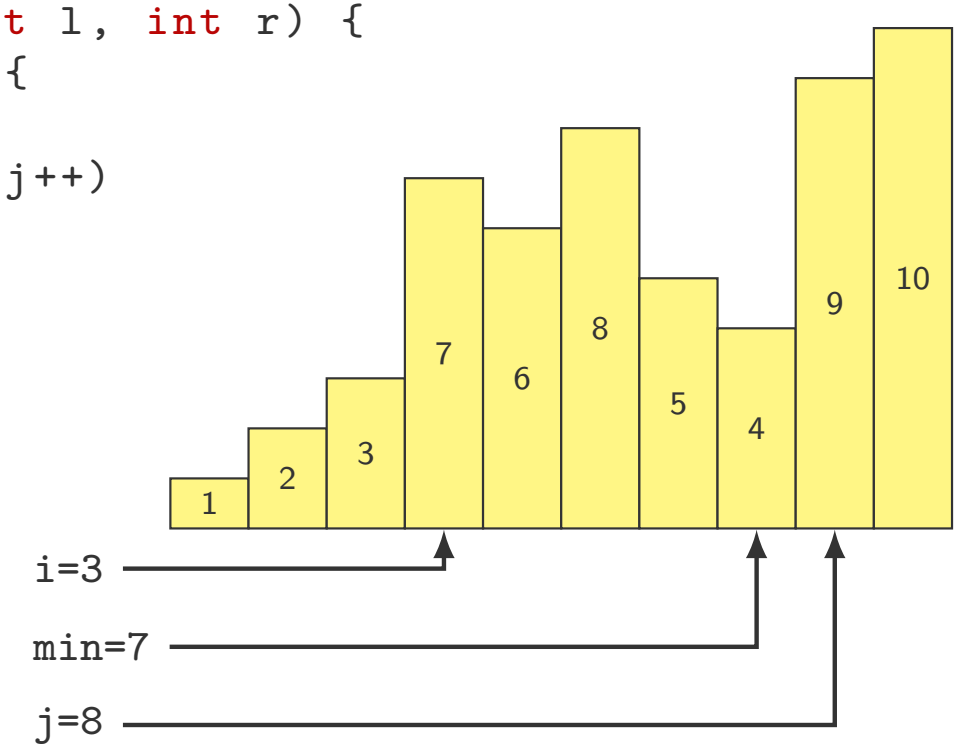


Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



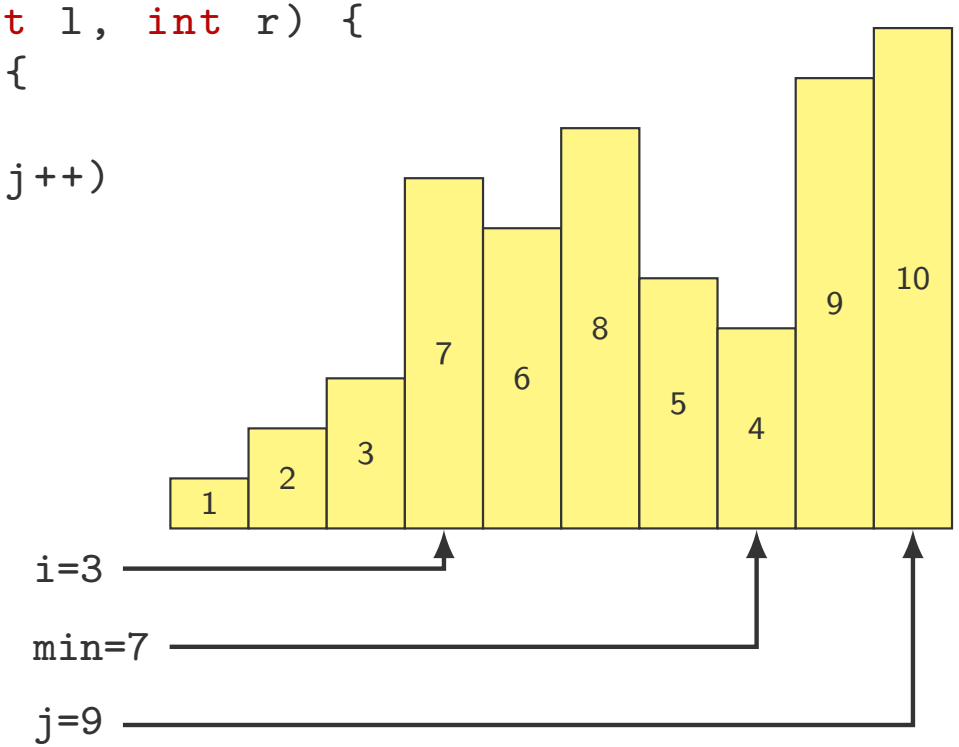
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



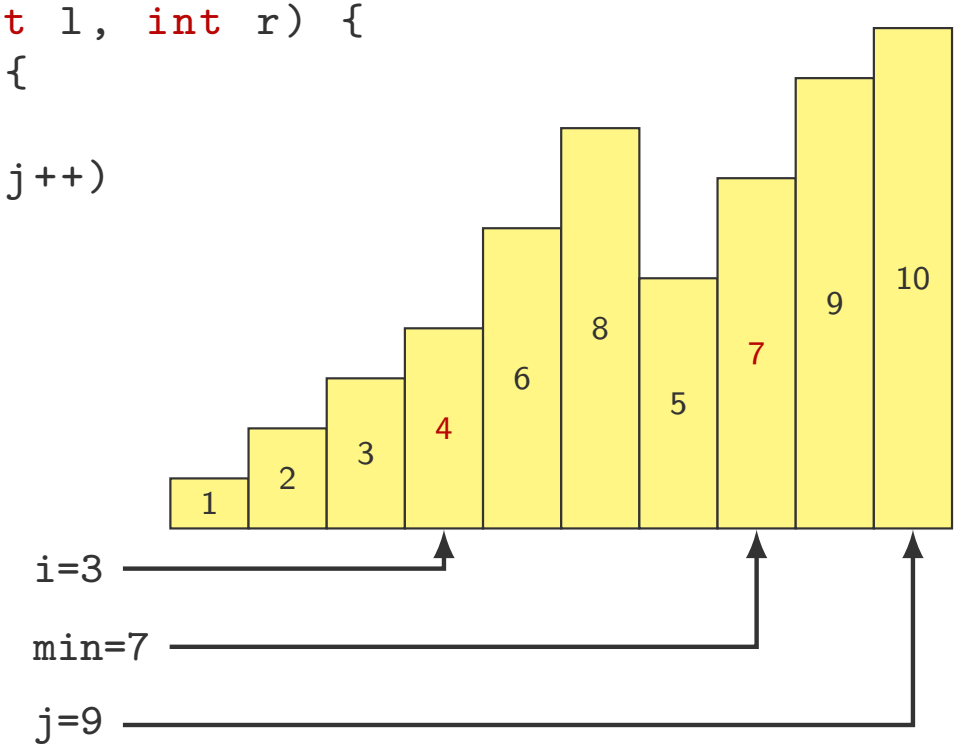
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



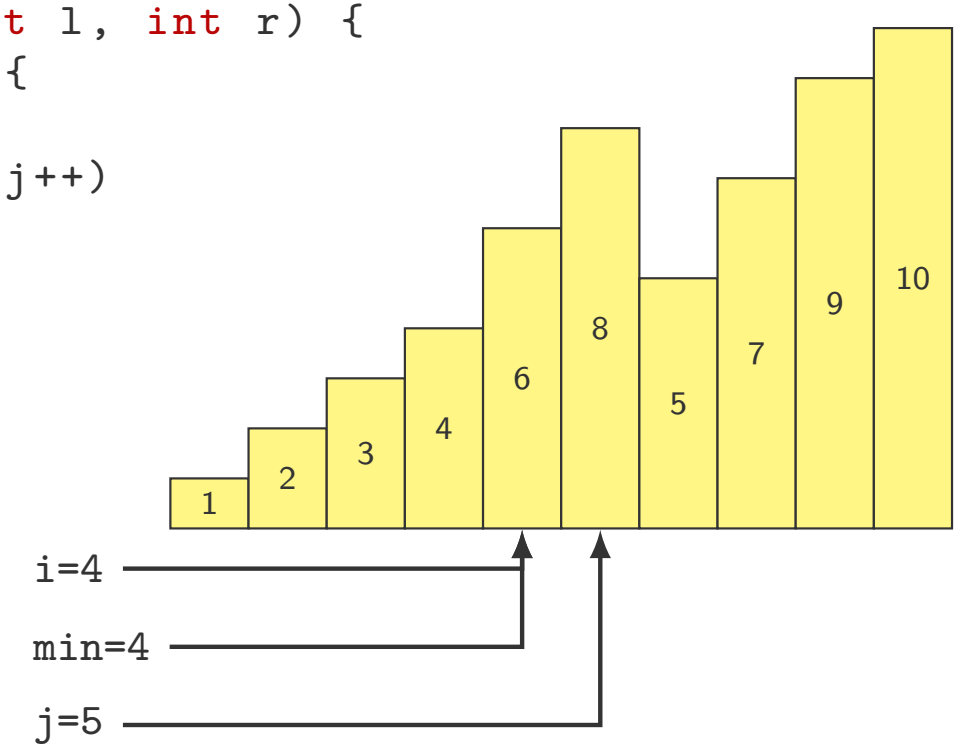
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



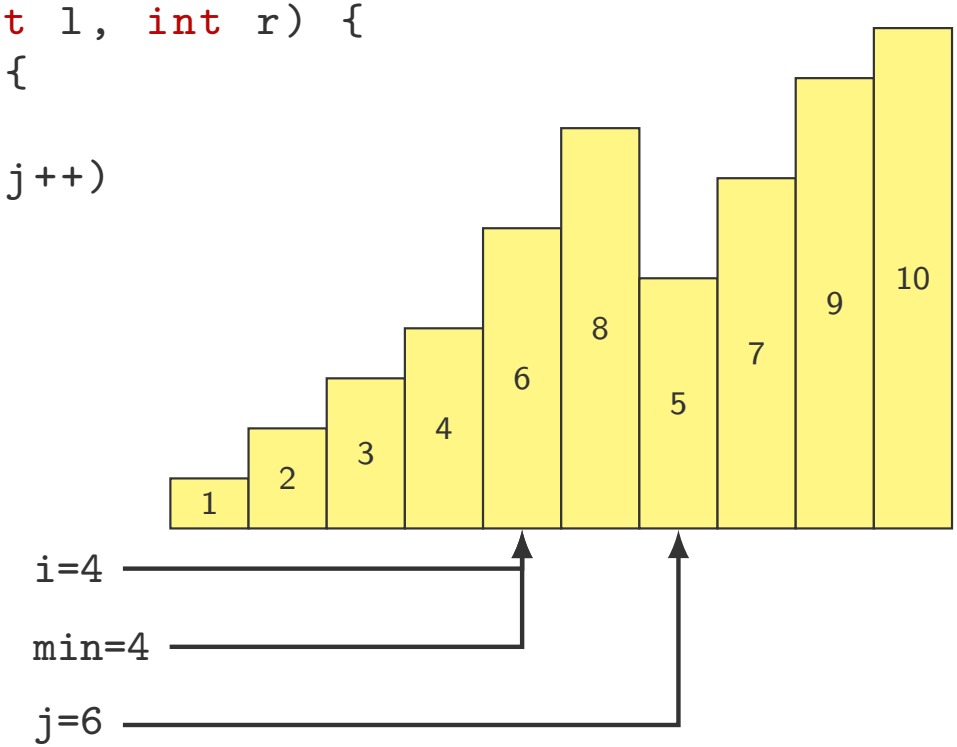
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



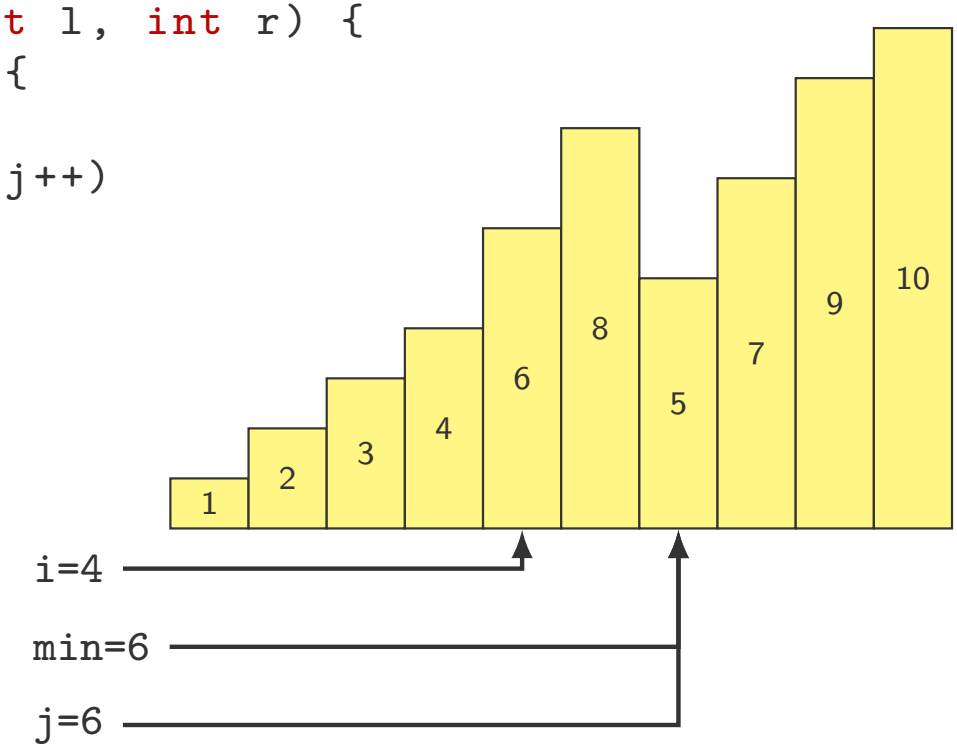
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



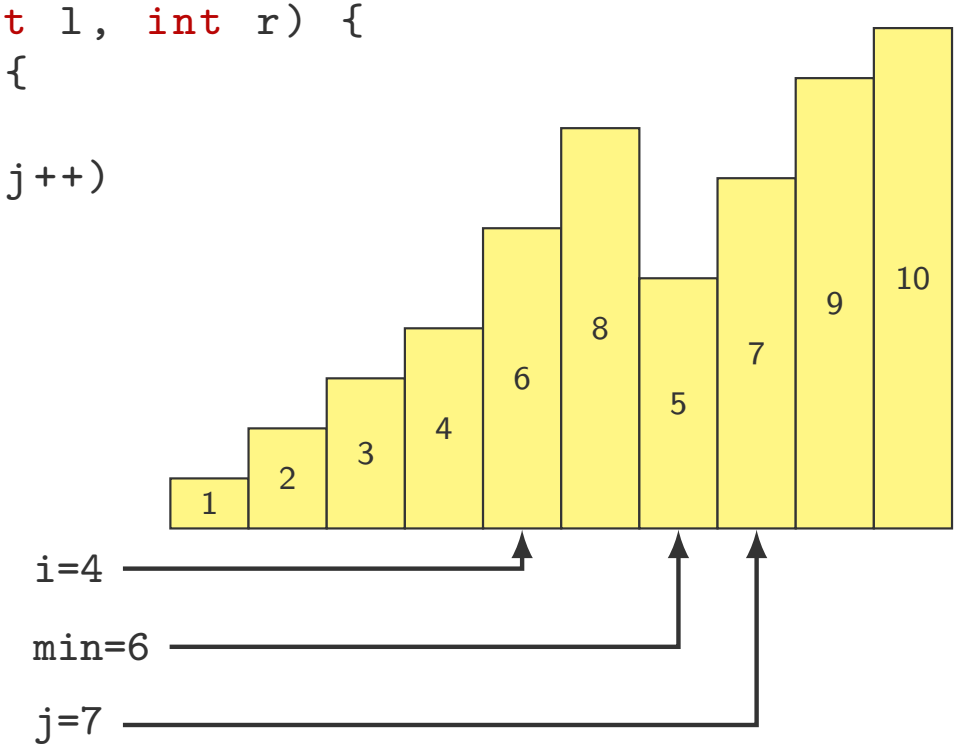
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



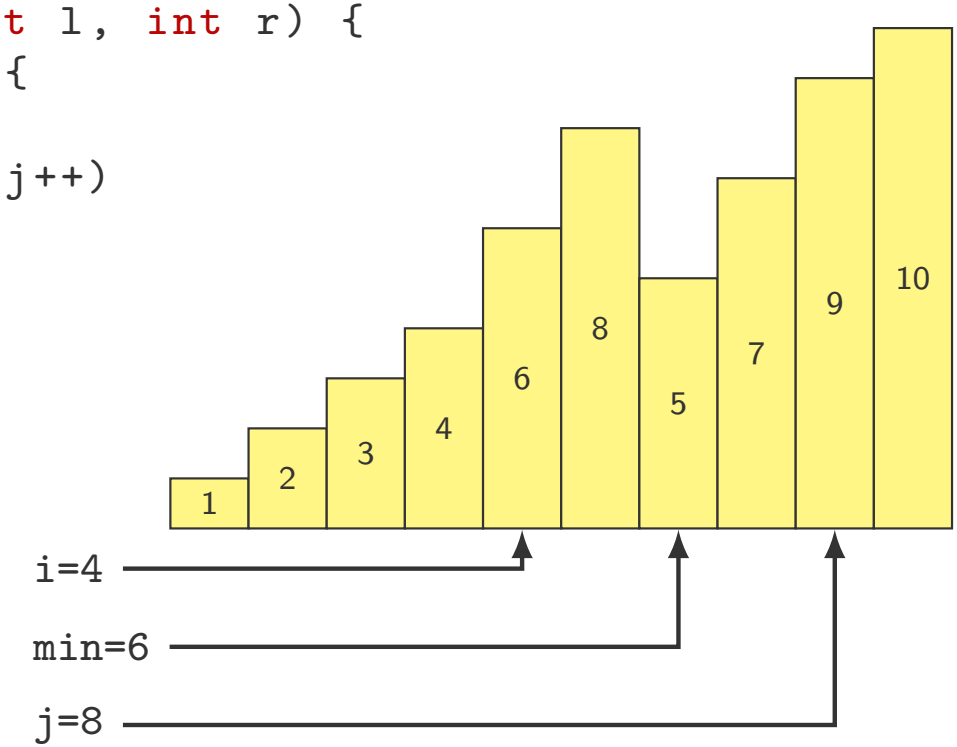
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



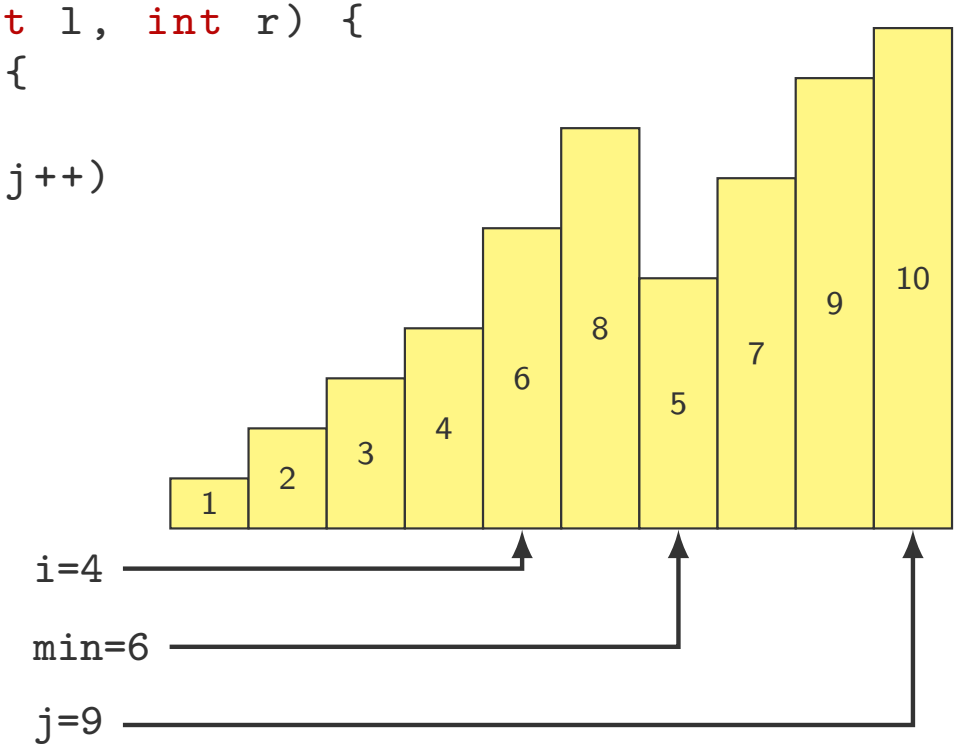
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



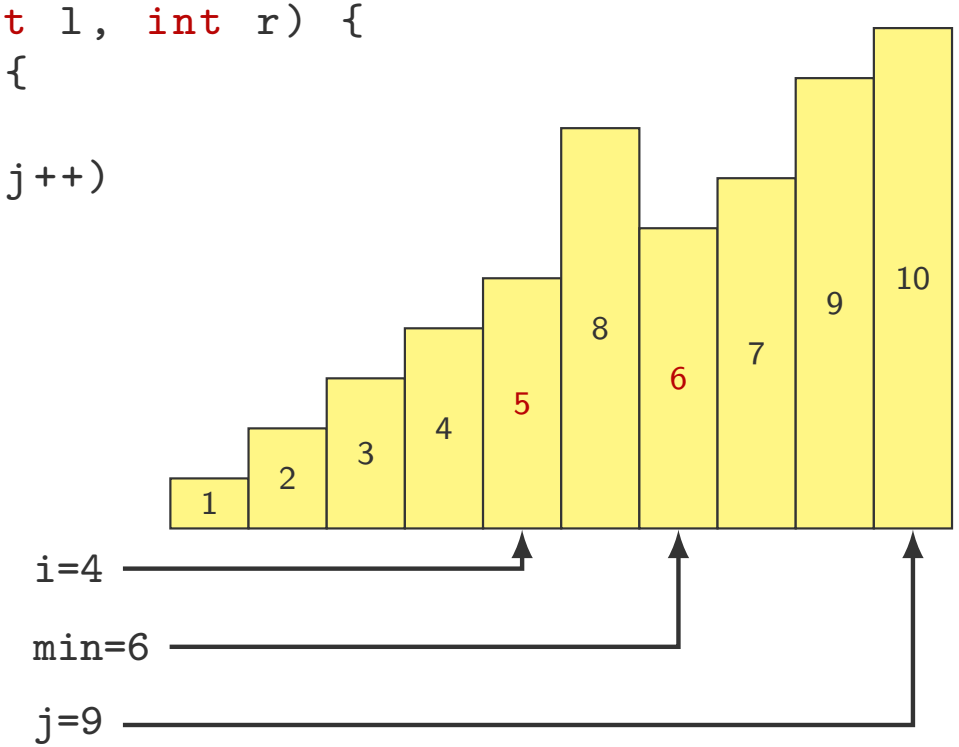
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



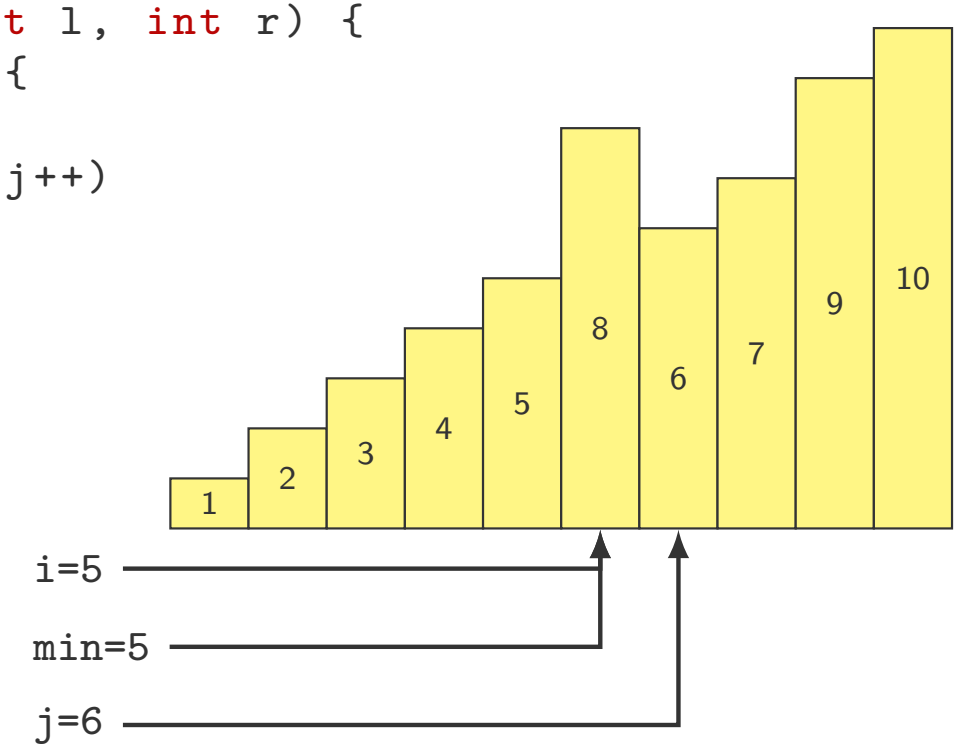
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



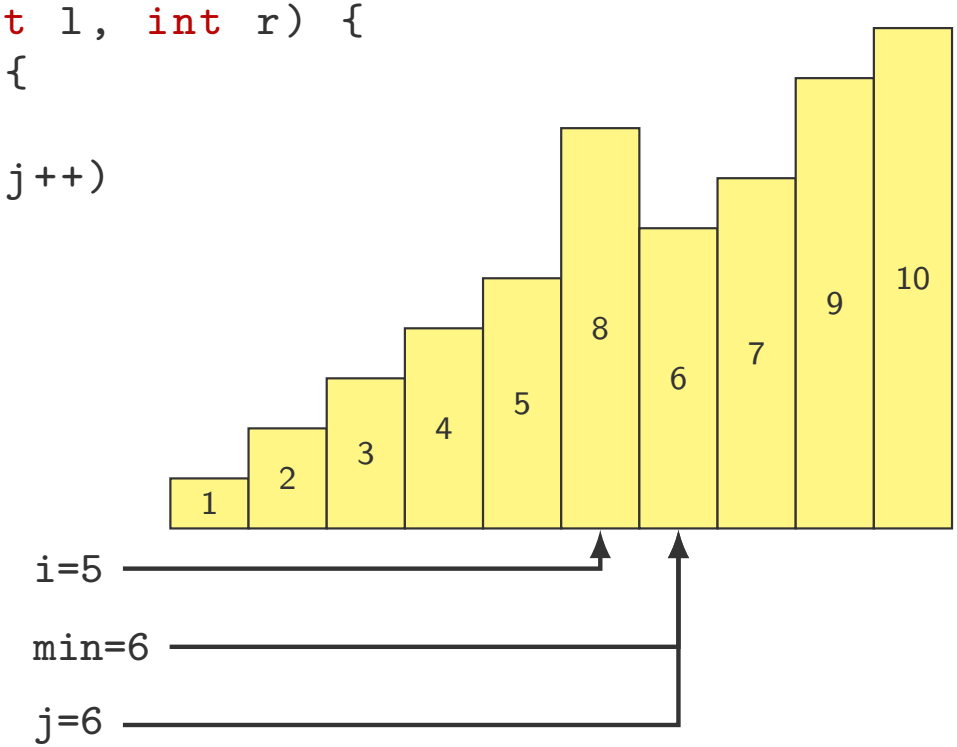
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



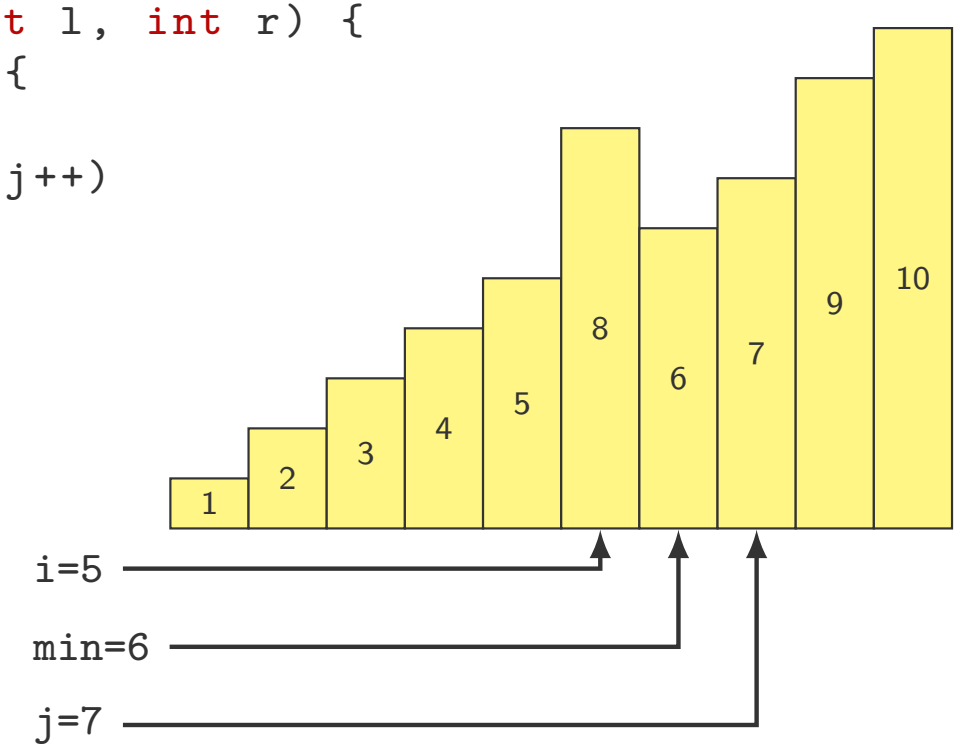
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



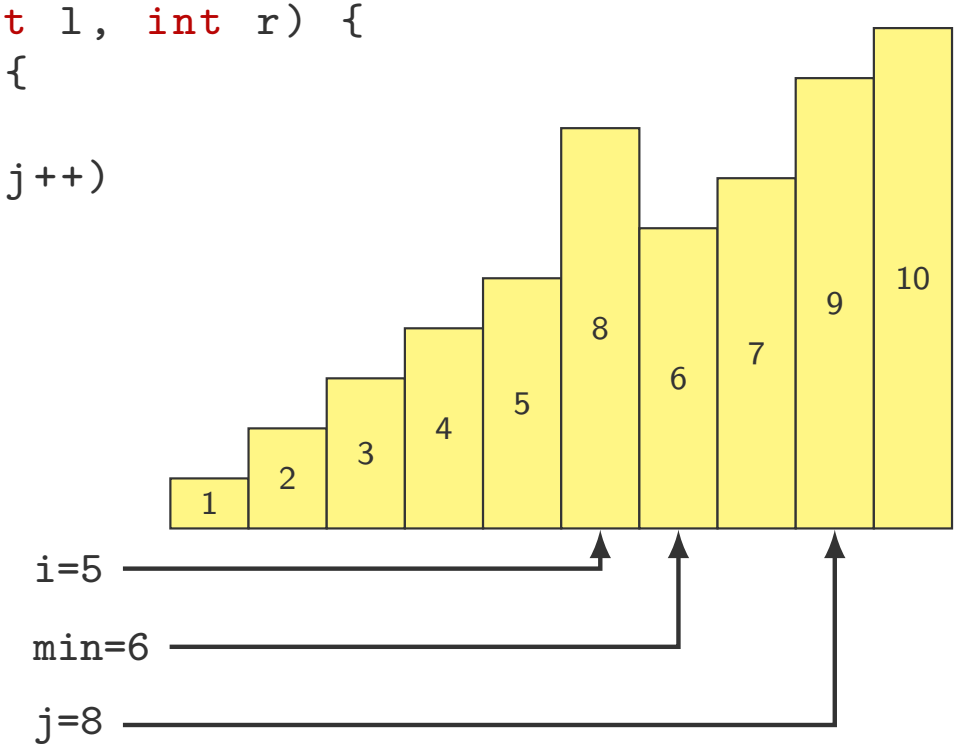
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



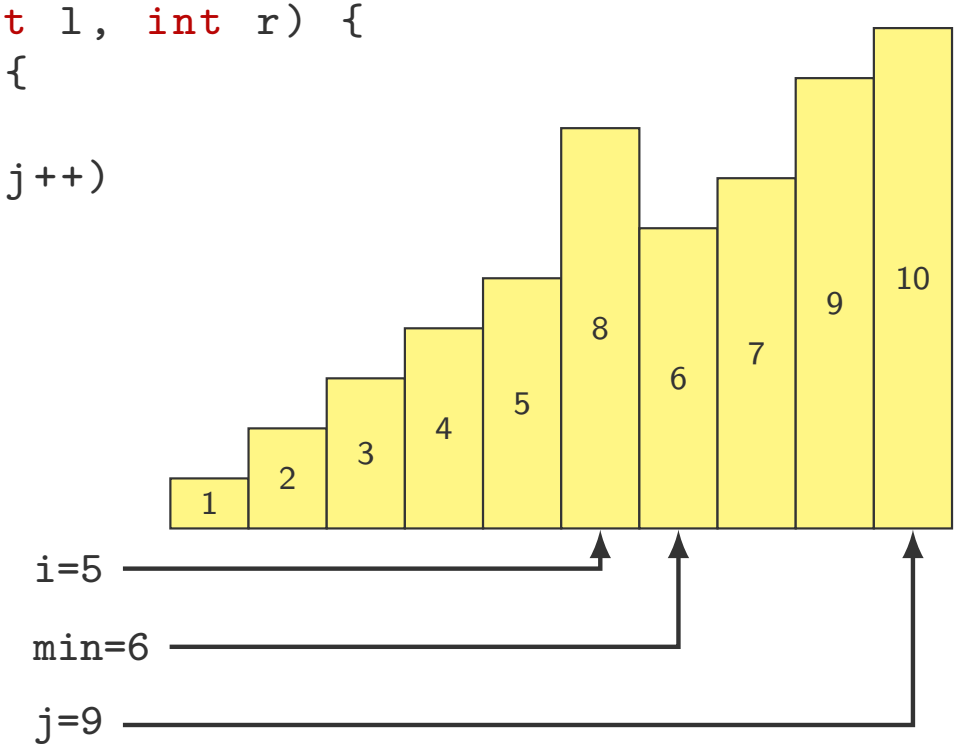
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



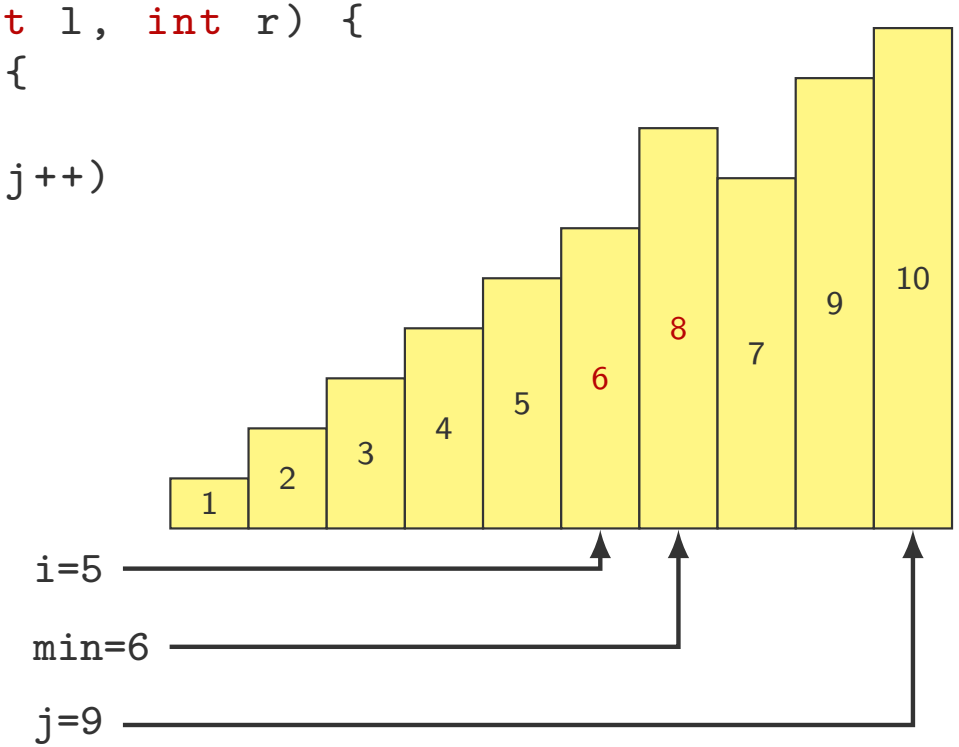
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



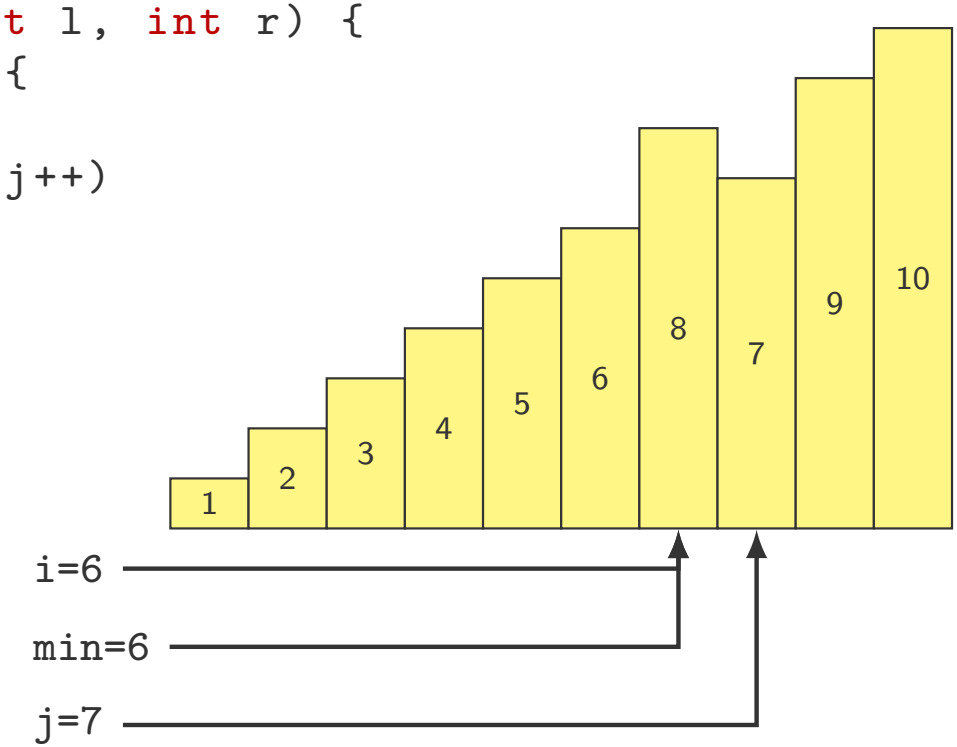
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



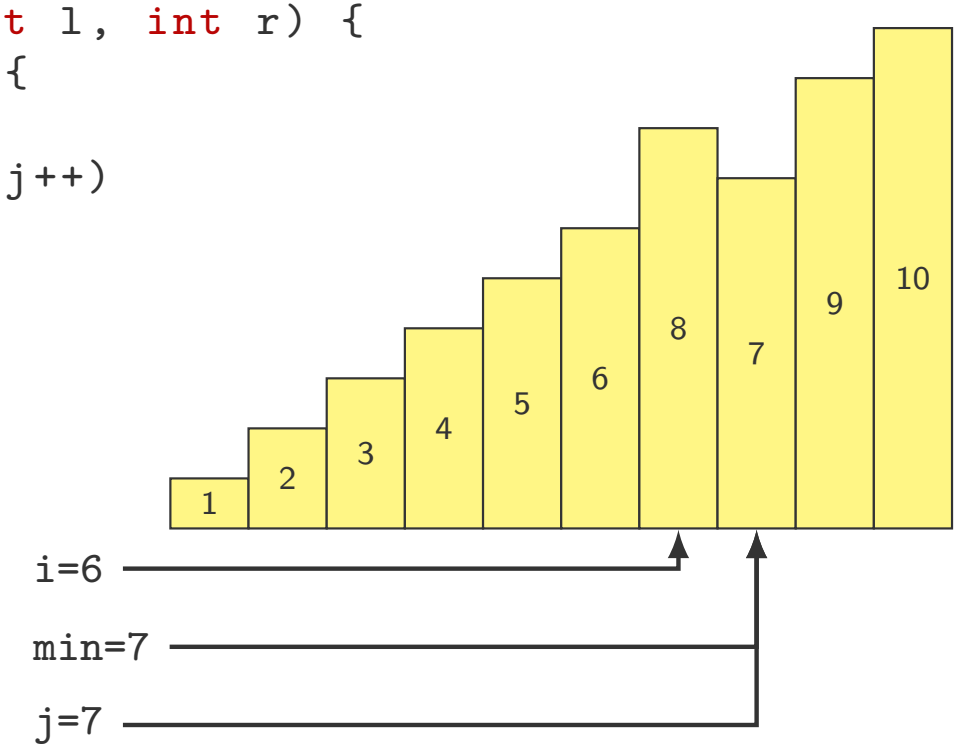
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



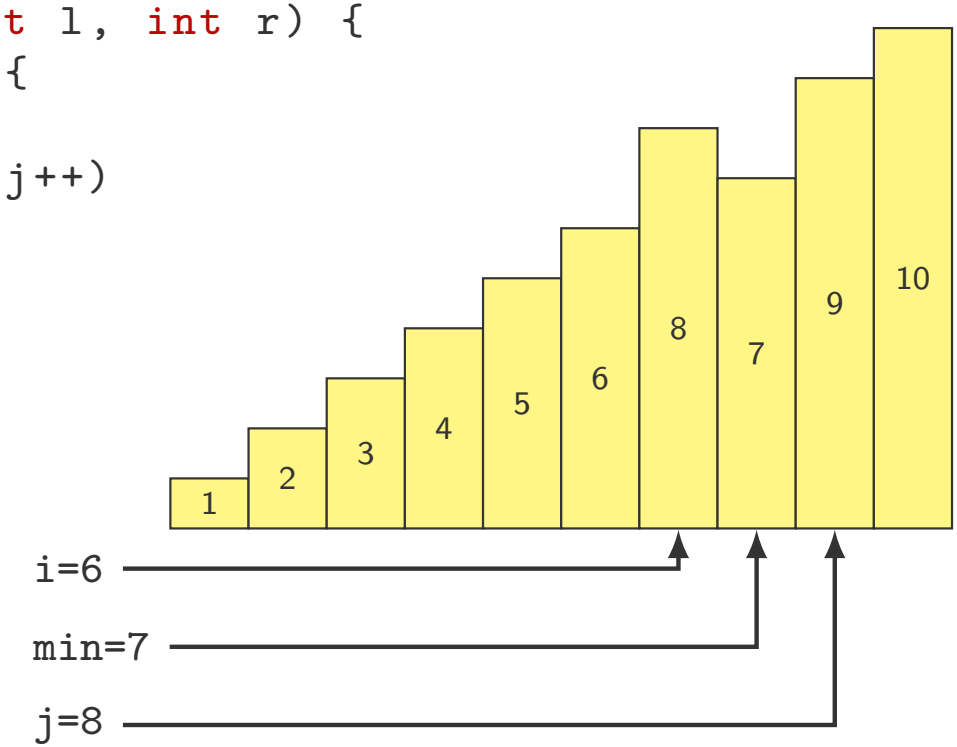
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



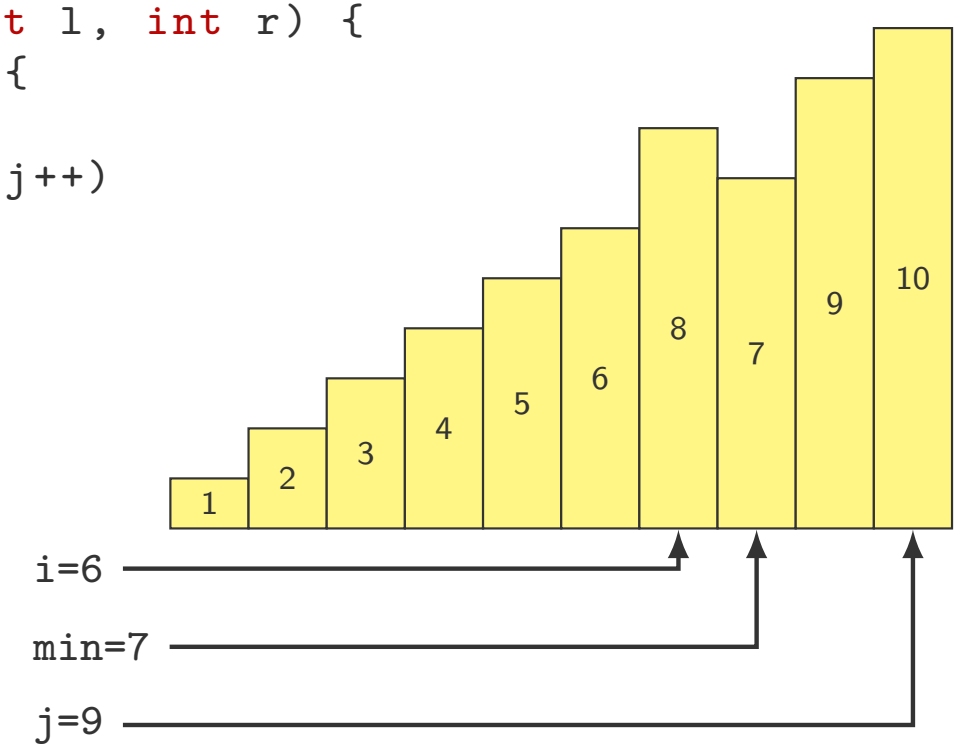
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



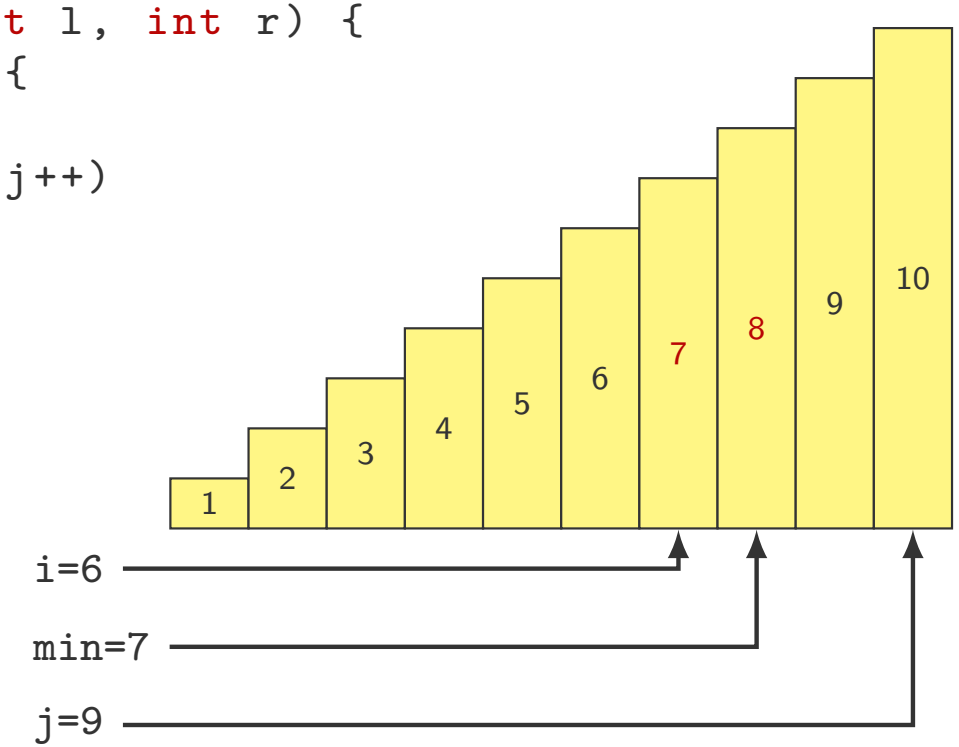
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



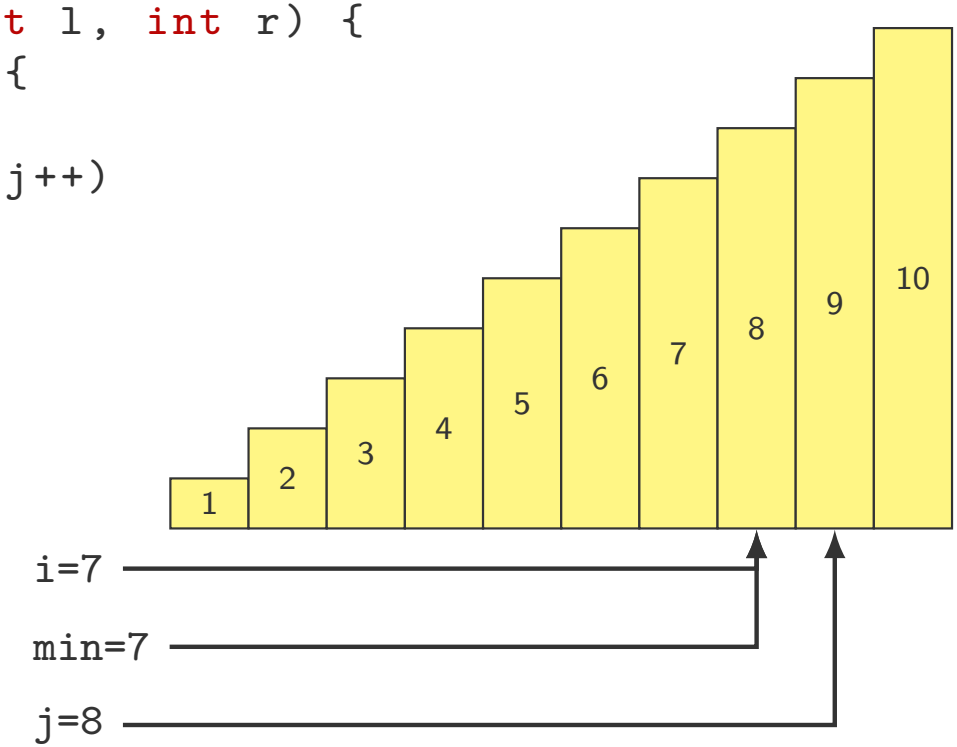
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



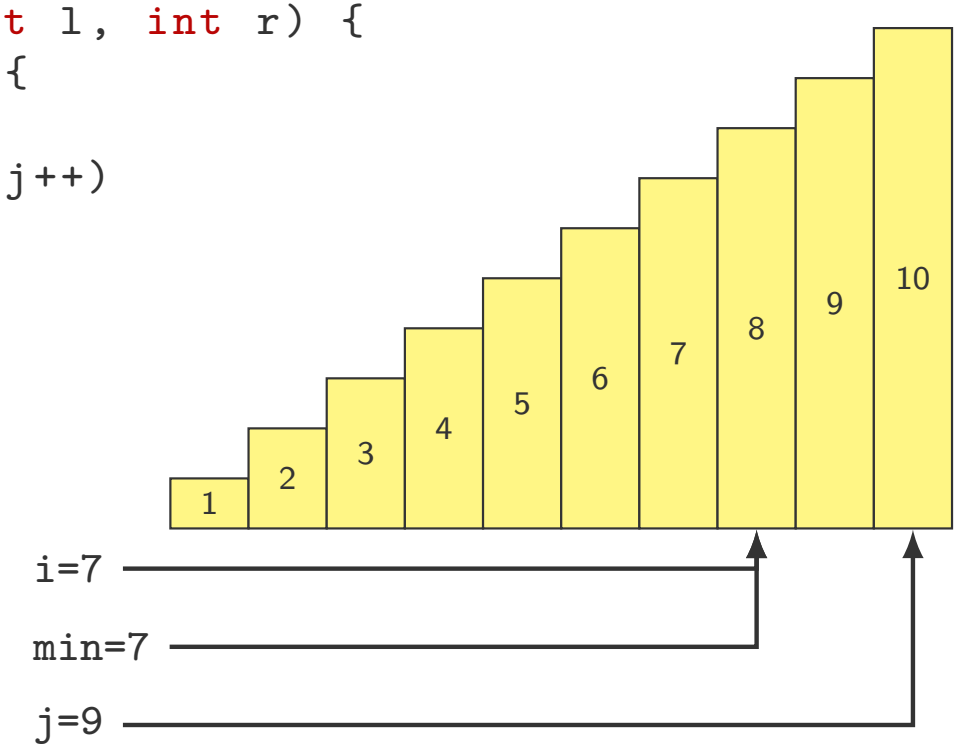
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



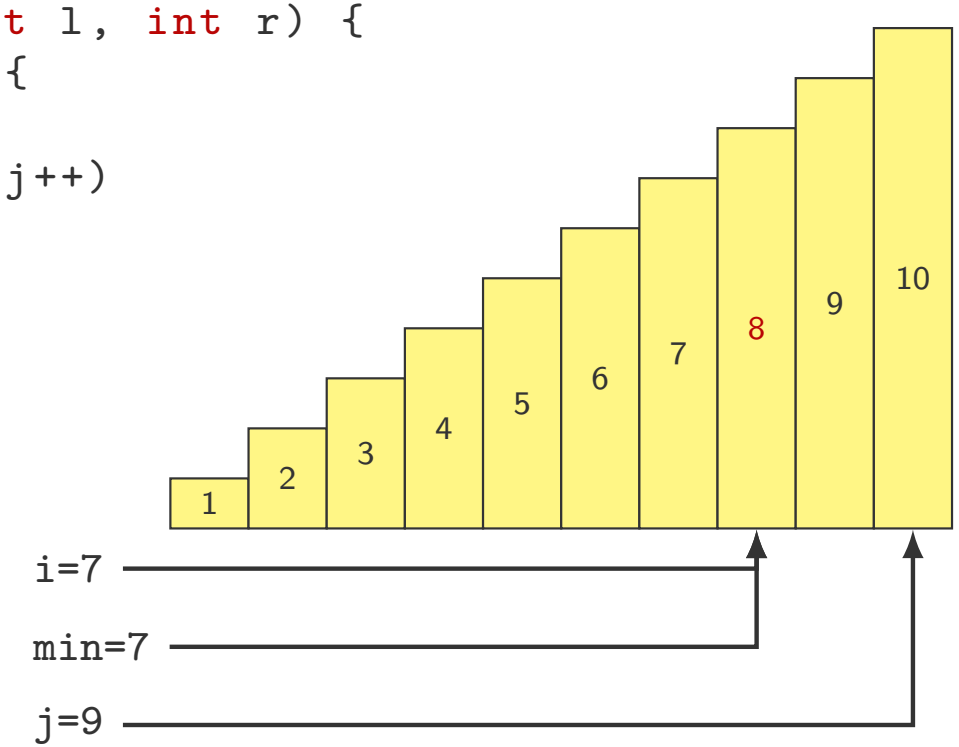
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



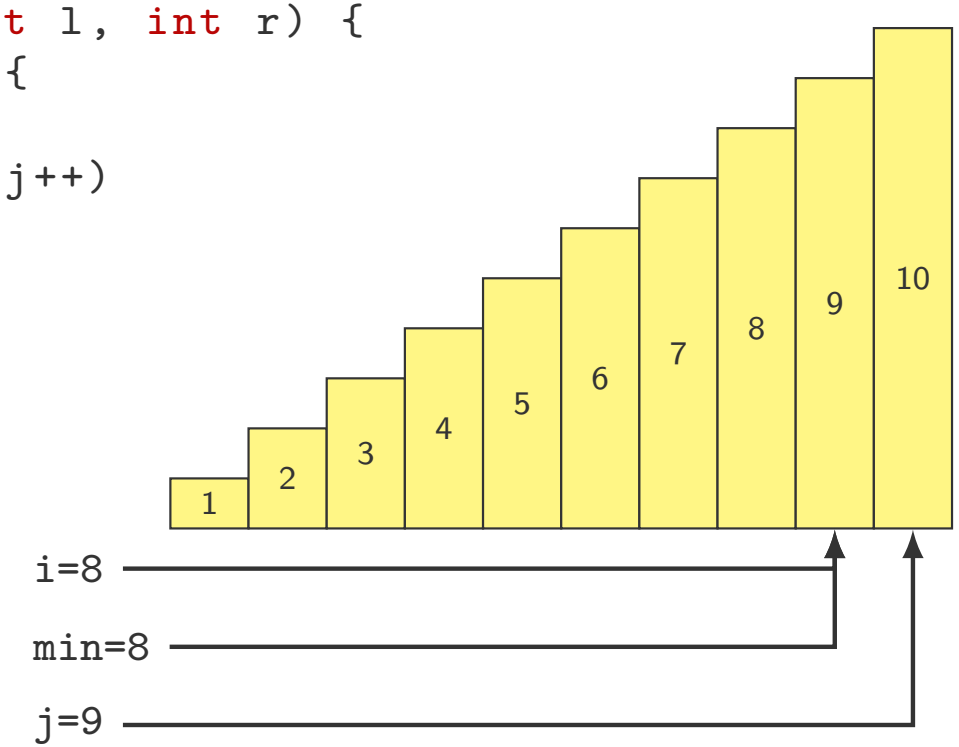
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



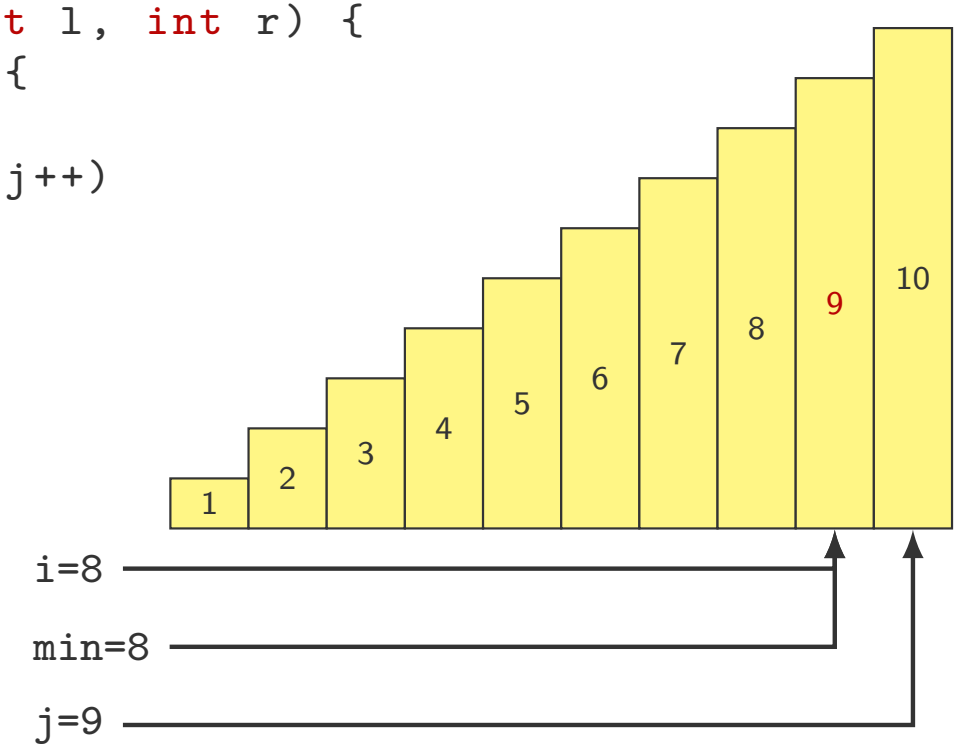
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 }
```



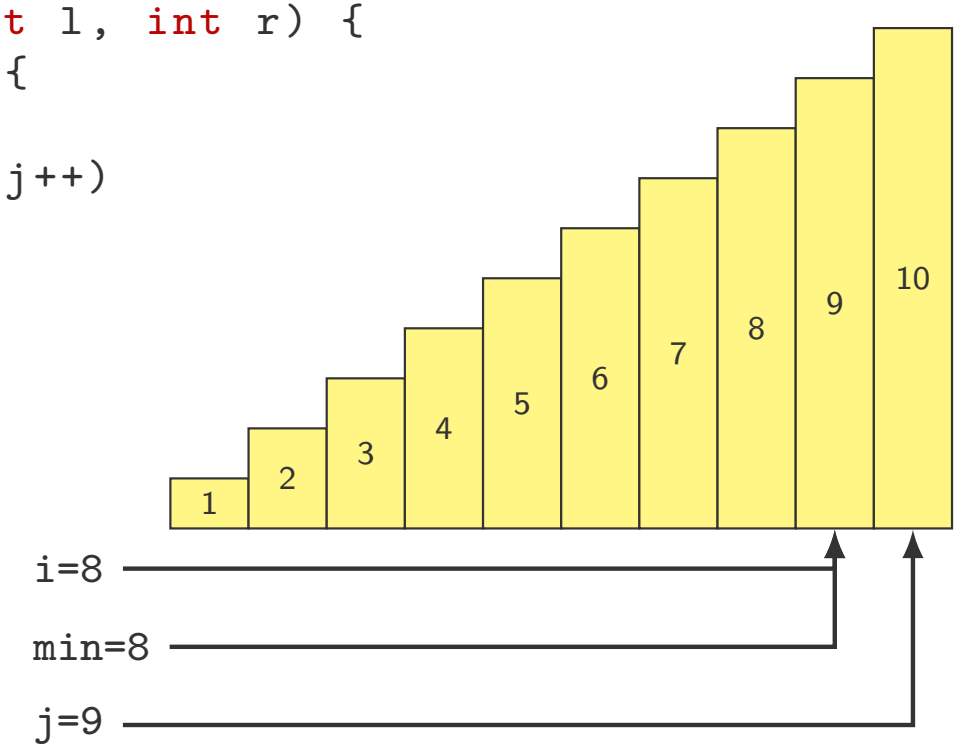
Ordenação por Seleção

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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void selectionsort(int A[], int l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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


```
1  void selectionsort(int A[], int l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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 l, int r) {
2    for (int i = l; i < r; i++) {
3      int indexMin = i;
4      for (int j = i+1; j <= r; 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

- Um algoritmo de ordenação é estável se não altera a posição relativa de elementos que têm um mesmo valor.

Ordenação estável

- Um algoritmo de ordenação é estável se não altera a posição relativa de elementos que têm um mesmo valor.

- 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

- Um algoritmo de ordenação é estável se não altera a posição relativa de elementos que têm um mesmo valor.

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

- 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

- 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

- 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

- 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).

Exercício – Ordenação estável

- BubbleSort é um algoritmo estável?

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

Exercício – Ordenação estável

- BubbleSort é um algoritmo estável?

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

Exercício – Ordenação estável

- BubbleSort é um algoritmo estável?

```
1 void bubblesort(int A[], int l, int r) {
2   for (int i = l; i < r; i++)
3     for (int j = r; 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 }
```

Sim!

Exercício – Ordenação estável

- Insertion-Sort é um algoritmo estável?

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

Exercício – Ordenação estável

- Selection-Sort é um algoritmo estável?

```
1    for (int i = l; i < r; i++) {
2      int indexMin = i;
3      for (int j = i+1; j <= r; j++)
4        if(A[j] < A[indexMin])
5          indexMin = j;
6      int aux = A[i];
7      A[i] = A[indexMin];
8      A[indexMin] = aux;
9    }
10 }
```

Exercício – Ordenação estável

- `Selection-Sort` é um algoritmo estável?

```
1    for (int i = l; i < r; i++) {
2      int indexMin = i;
3      for (int j = i+1; j <= r; j++)
4        if(A[j] < A[indexMin])
5          indexMin = j;
6      int aux = A[i];
7      A[i] = A[indexMin];
8      A[indexMin] = aux;
9    }
10 }
```

Não! Por exemplo, teste a sequência 443

Funções de Ordenação do C++

Biblioteca `algorithm` — `sort()`

A biblioteca padrão do C++ possui o cabeçalho `algorithm` que possui o seguinte template de função:

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );
```

- `RandomIt` é qualquer iterador de acesso aleatório, que é um iterador que pode mover-se para frente, para trás e em incrementos maiores que 1.
  - `first` é um iterador para o primeiro elemento.
  - `last` aponta para uma posição após o último elemento.
- Pode ser aplicada a: arrays, `std::vector` e `std::deque`
- O contêiner `list` suporta apenas iteradores bidirecionais, portanto fornecem sua própria função-membro `sort`.

Exemplo


```cpp
#include <iostream> // ex01.cpp
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    int arr[] {34,22,9,4,11,55};

    vector<int> vec {77,66,88,55,22,11};

    sort(begin(arr), end(arr));

    sort(vec.begin(), vec.end());

    for(const auto& e : arr) cout << e << " ";
    cout << endl;

    for(const auto& e : vec) cout << e << " ";
    cout << endl;
}
```

Biblioteca `algorithm` — `sort()`

`algorithm` também fornece a seguinte sobrecarga da função sort:

```
template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );
```

- Na função acima, `Compare` é qualquer função com a seguinte assinatura:

```
bool Compare(const Type1 &a, const Type2 &b);
```

  - Esta função retorna `true` se e somente se o primeiro argumento for menor que o segundo.
  - Os tipos `Type1` e `Type2` devem ser tais que um objeto do tipo `RandomIt` possa ser desreferenciado e então implicitamente convertido em ambos.

Exemplo


```
1  #include <iostream> // ex02.cpp
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  int main() {
7      vector<int> vec {77,66,88,55,22,11};
8
9      // ordena em ordem decrescente
10     sort(vec.begin(), vec.end(), greater<int>());
11
12     for(const auto& e : vec)
13         cout << e << " ";
14     cout << endl;
15 }
```

Biblioteca algorithm — stable_sort()

O cabeçalho algorithm também fornece as seguintes funções de ordenação estáveis:

```
template< class RandomIt >
void stable_sort( RandomIt first, RandomIt last );

template< class RandomIt, class Compare >
void stable_sort(RandomIt first, RandomIt last, Compare c);
```

Exemplo


```
1  #include <iostream> // ex03.cpp
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  struct Data {
7      int dia;
8      int mes;
9  };
10
11 bool cmp_d(const Data &d1, const Data &d2) {
12     return d1.dia < d2.dia;
13 }
14
15 bool cmp_m(const Data &d1, const Data &d2) {
16     return d1.mes < d2.mes;
17 }
```

Exemplo (cont)


```
18 int main() {
19     vector<Data> vec {{1,12}, {7,12}, {16,3},
20         {25,9}, {30,3}, {30,6}, {31,3}};
21
22     stable_sort(vec.begin(), vec.end(), cmp_d);
23     stable_sort(vec.begin(), vec.end(), cmp_m);
24
25     for(const auto& e : vec)
26         cout << "(" << e.dia << "," << e.mes << ") ";
27     cout << endl;
28 }
```

Biblioteca `algorithm` — `is_sorted()`

O cabeçalho `algorithm` também fornece as seguintes funções para testar se um array ou contêiner sequencial está ordenado:

```
template< class ForwardIt >
bool is_sorted(ForwardIt first, ForwardIt last);

template< class ForwardIt, class Compare >
bool is_sorted(ForwardIt first,ForwardIt last,Compare comp);
```

- Essas funções checam se os elementos no intervalo [first, last) estão ordenados em ordem crescente.

Exemplo


```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  int main() {
7      vector<int> v1 {1,2,3,4,5,6,7,8,9};
8      vector<int> v2 {1,2,3,4,5,7,6,8,9};
9
10     if(is_sorted(v1.begin(), v1.end()))
11         cout << "v1 is sorted\n";
12     else
13         cout << "v1 is not sorted\n";
14
15     if(is_sorted(v2.begin(), v2.end()))
16         cout << "v2 is sorted\n";
17     else
18         cout << "v2 is not sorted\n";
19 }
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM