Campus de Frutal

UEMG UNIVERSIDADE DO ESTADO DE MINAS GERAIS

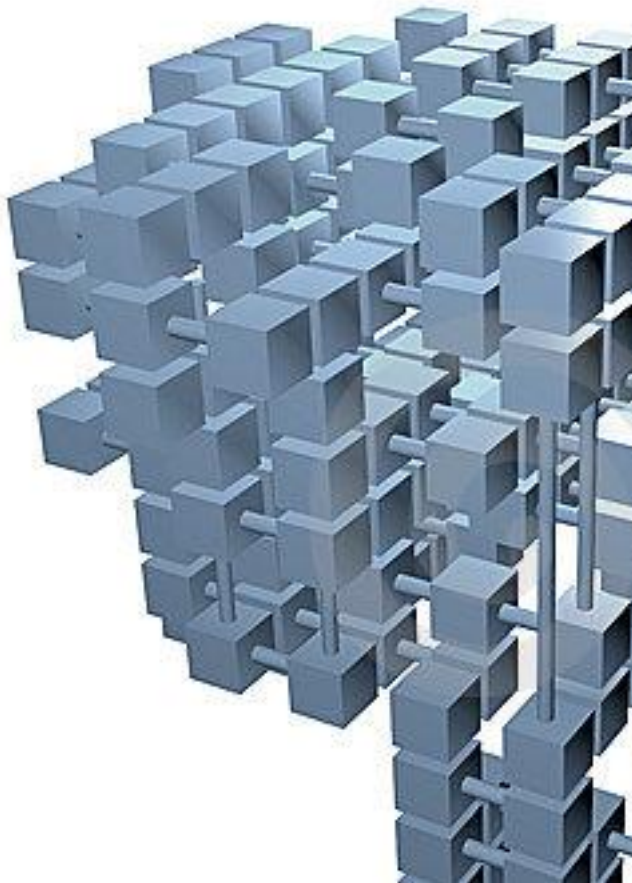
Sistemas de Informaçao

Estrutura de Dados

II

Ordenações

Seleção/Inserção/Troca



Prof. Ivan José dos Reis Filho

ivanfilhoreis@gmail.com

Conteúdo da Aula

- Ordenação por Seleção
- por Inserção
- Ordenação por Troca

Ordenação (Sorting)

- Entrada:
  - Uma sequência de
    n números a0 , a1 , a2 , …, an-1
- Saída:
  - Uma permutação da sequência de entrada
    a'0 , a'1 ,a'2 , …, a'n-1 ,
    onde a'0 ≤ a'1 ≤ a'2 ≤ … ≤ a'n-1

Por exemplo: (uma **instancia do problema)**

- Entrada: 3, 9, 10, 7, 2, 5, 8, 9, 0, 1
- Saída: 0, 1, 2, 3, 5, 7, 8, 9, 9, 10

Algumas considerações

Os algoritmos a seguir são:

– Comparativos

Aplicação dos operadores: **<, >, ≥, = e ≠**

–Sem conhecimento "prévio" dos dados

Não se assume o tipo, a distribuição ou o intervalo dos dados

– Ordenação por trocas

Os elementos são dispostos em uma sequência, que pode ser implícita pela memória linear

Principais Elementos Comparativos

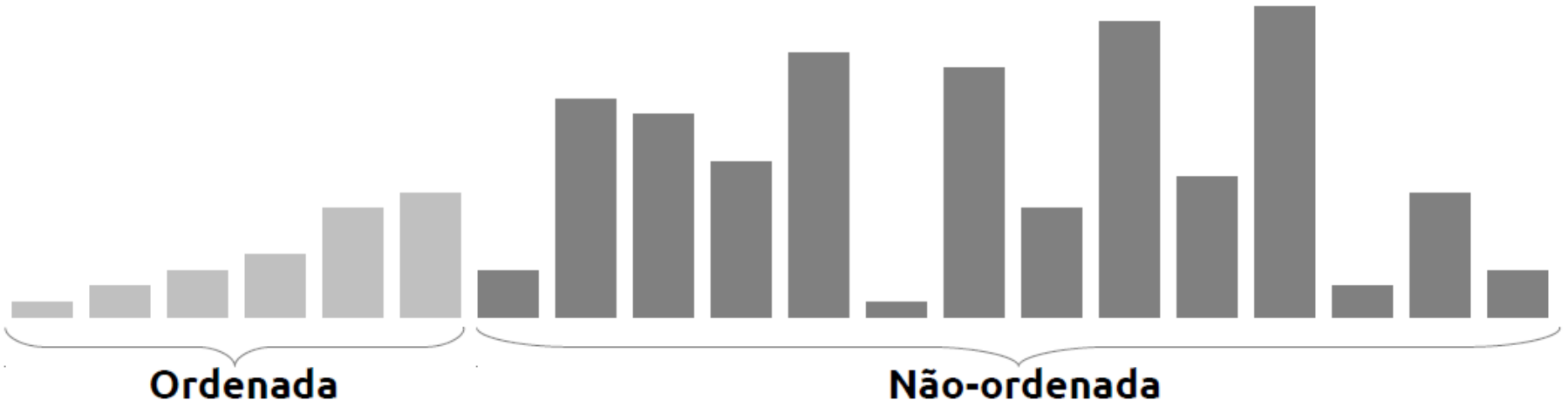
- Selection Sort
- Insertion Sort
- Bubble Sort

} Elementares

- Shell Sort
- Merge Sort
- Quick Sort
- Heap Sort

Selection Sort

- Divisão dos dados em duas sequências: ordenada e não-ordenada.
- Iteração: procurar pelo **menor elemento da** sequência não-ordenada e concatená-lo na sequência ordenada



Selection Sort

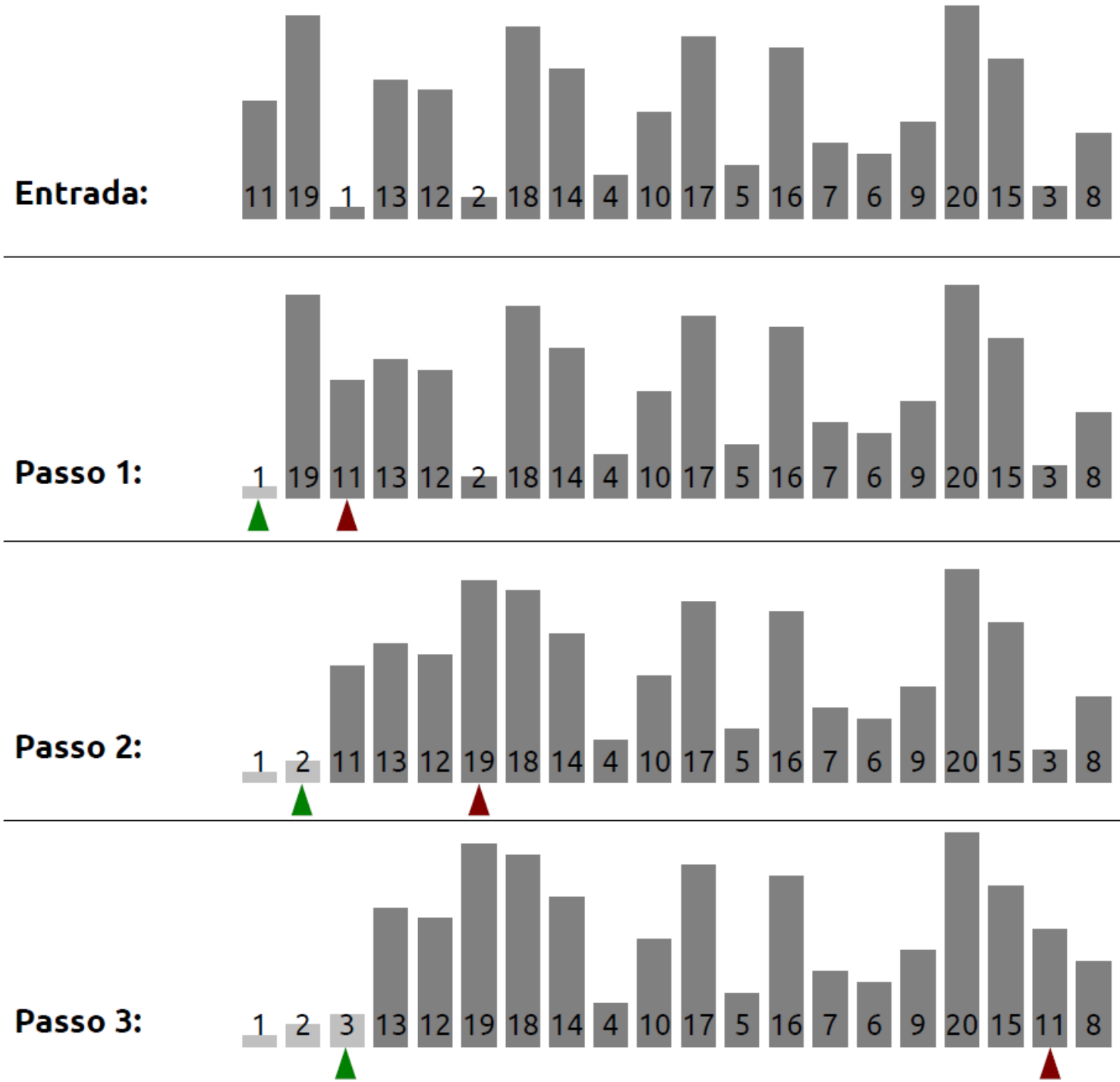
```
template <class Item>
void selection(Item vetor[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int min = i;

        for (int j = i+1; j < n; j++)
            if (vetor[j] < vetor[min])
                min = j;

        swap(vetor[i], vetor[min]);

    }
}
```

i
controla a iteração,
índice da sequência ordenada

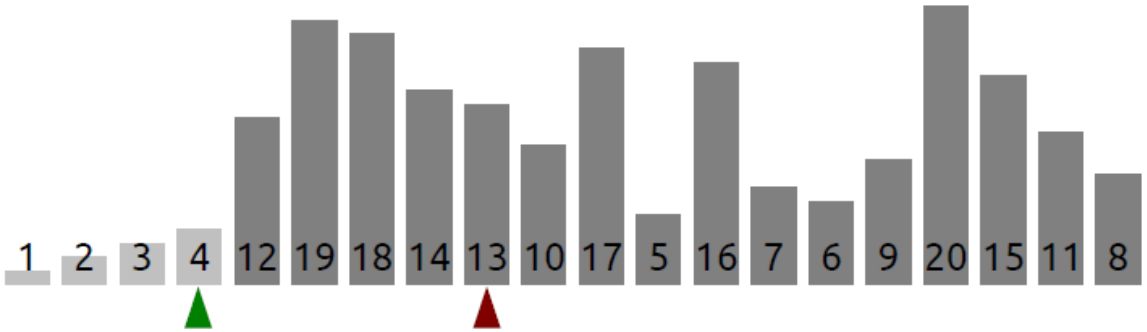
j
controla a busca pelo valor mínimo, índice da sequência não-ordenada

min
Índice do menor elemento da sequência não-rdenada

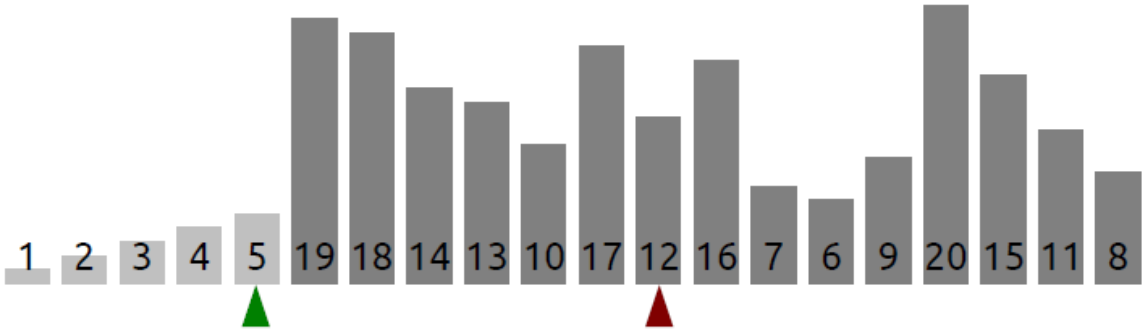
```
template <class Item>
  void swap(Item &A, Item &B)
    { Item t = A ; A = B; B = t; }
```


Entrada:
11 19 1 13 12 2 18 14 4 10 17 5 16 7 6 9 20 15 3 8
Passo 1:
1 19 11 13 12 2 18 14 4 10 17 5 16 7 6 9 20 15 3 8
Passo 2:
1 2 11 13 12 19 18 14 4 10 17 5 16 7 6 9 20 15 3 8
Passo 3:
1 2 3 13 12 19 18 14 4 10 17 5 16 7 6 9 20 15 11 8

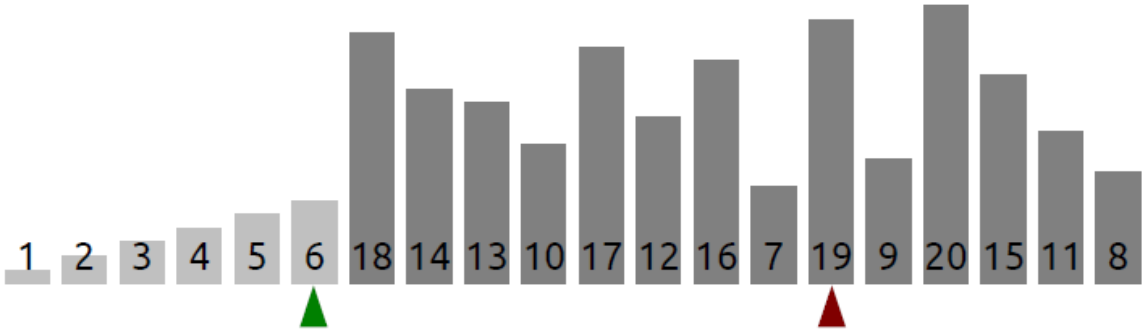
Passo 4:



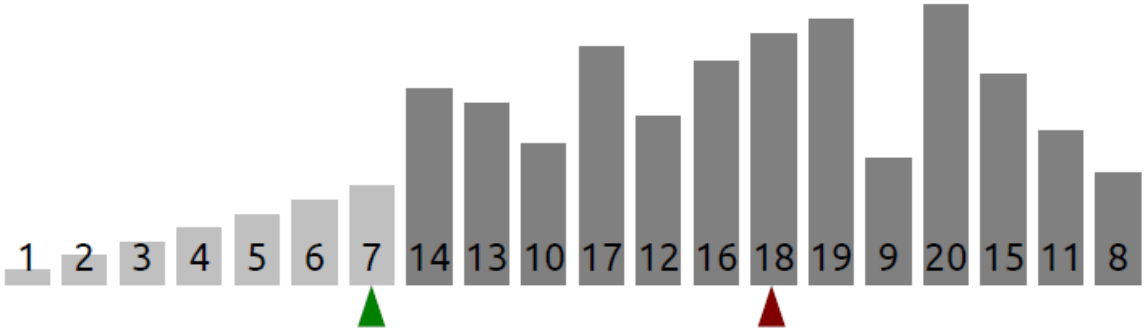
Passo 5:



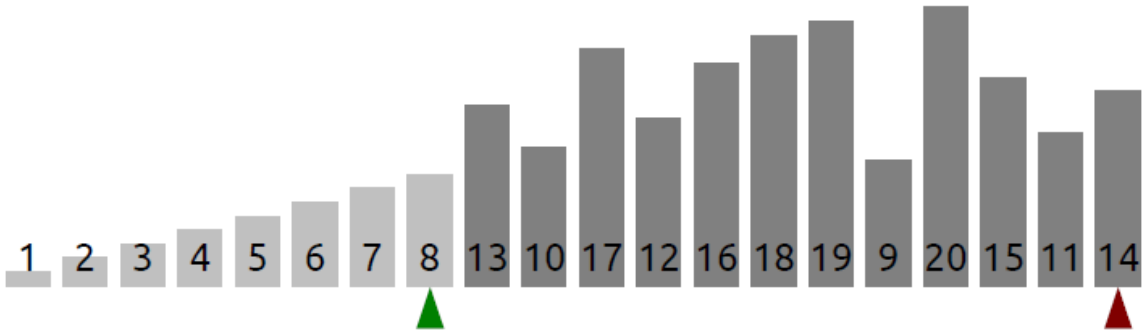
Passo 6:



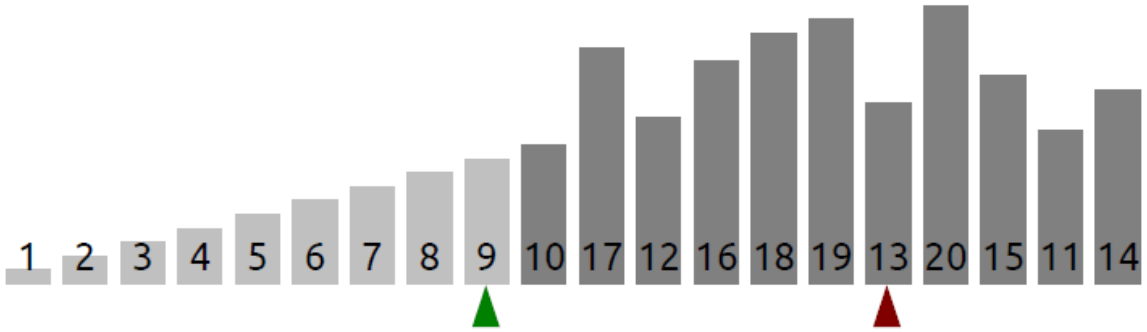
Passo 7:



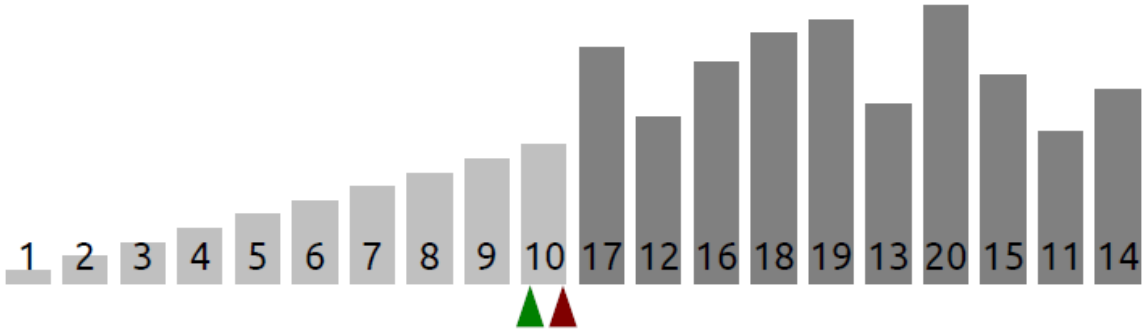
Passo 8:



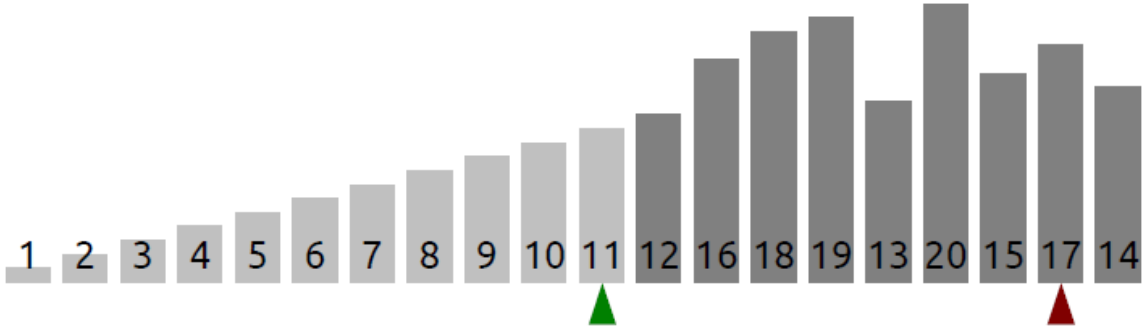
Passo 9:



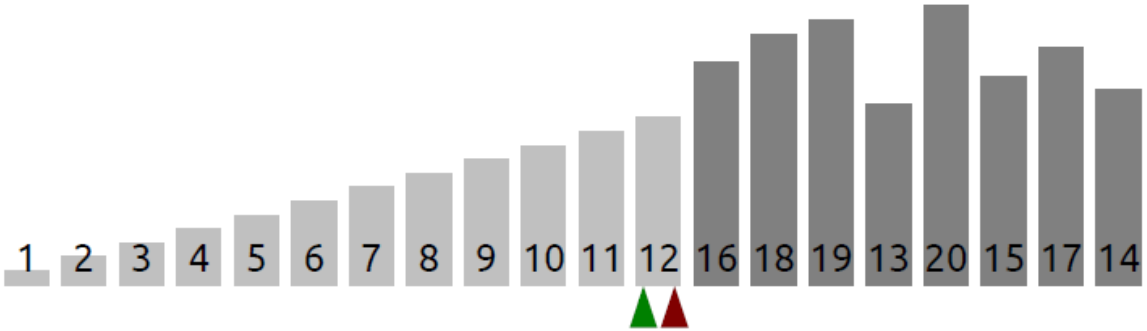
Passo 10:



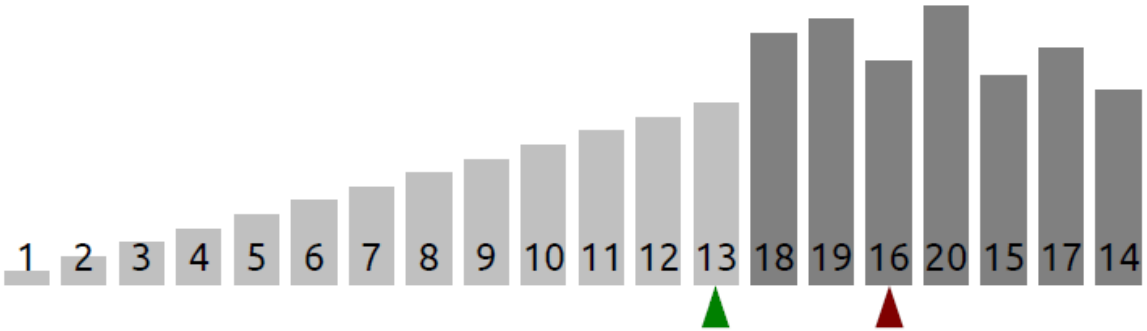
Passo 11:



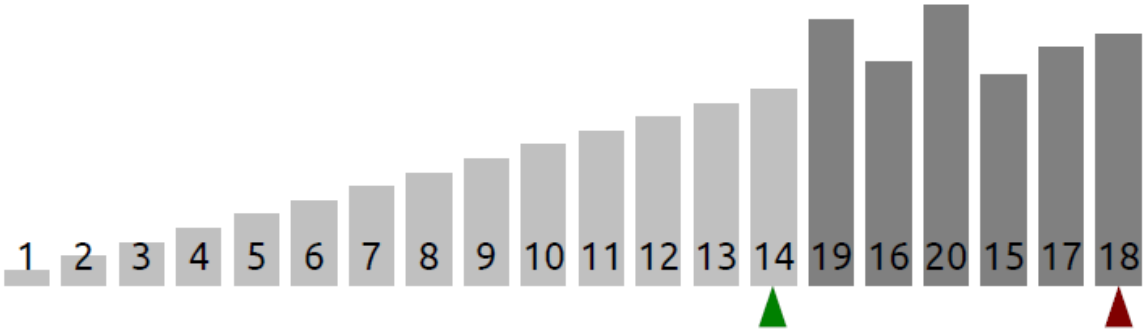
Passo 12:



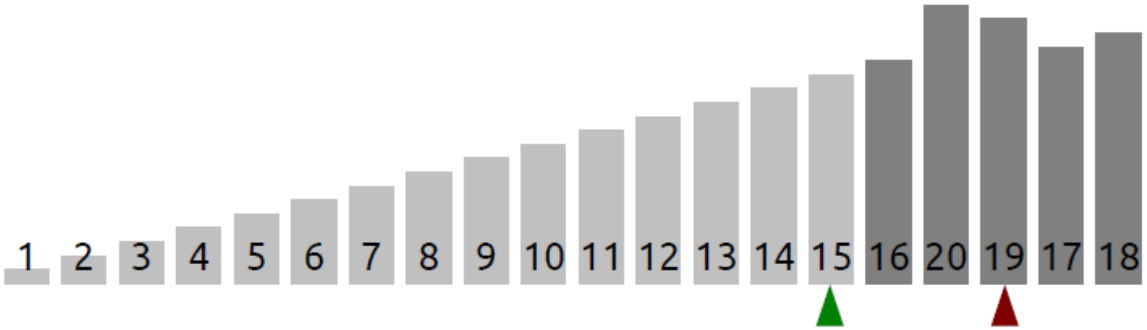
Passo 13:



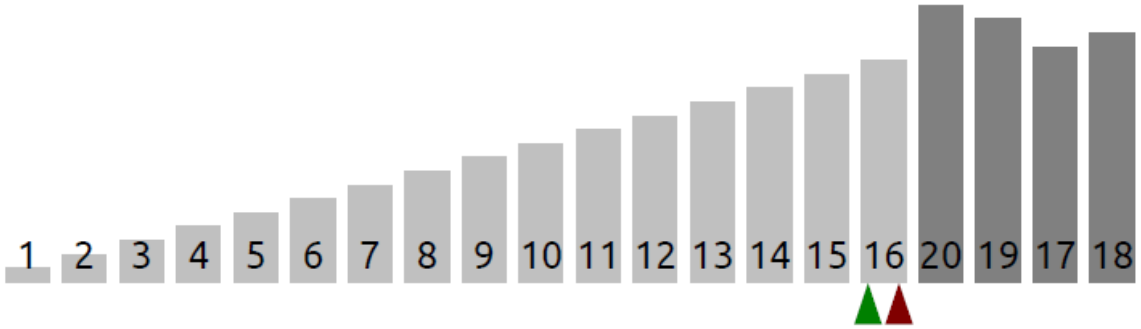
Passo 14:



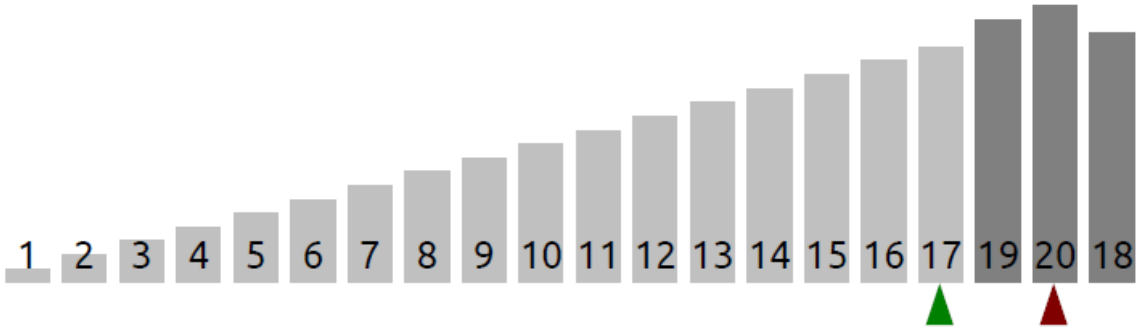
Passo 15:



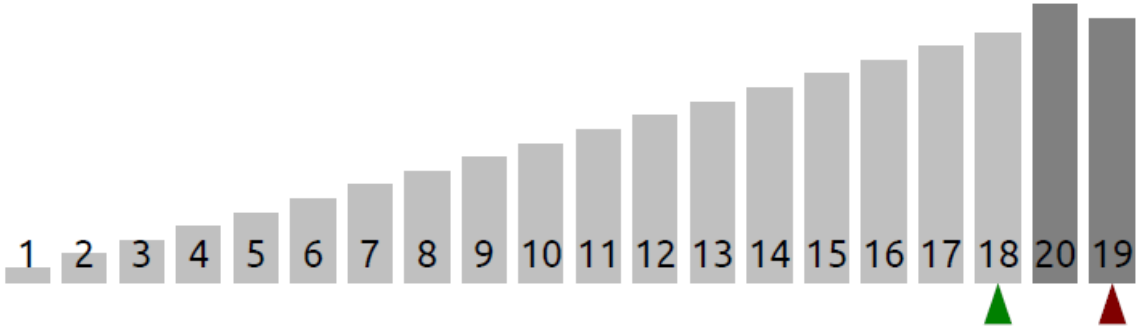
Passo 16:



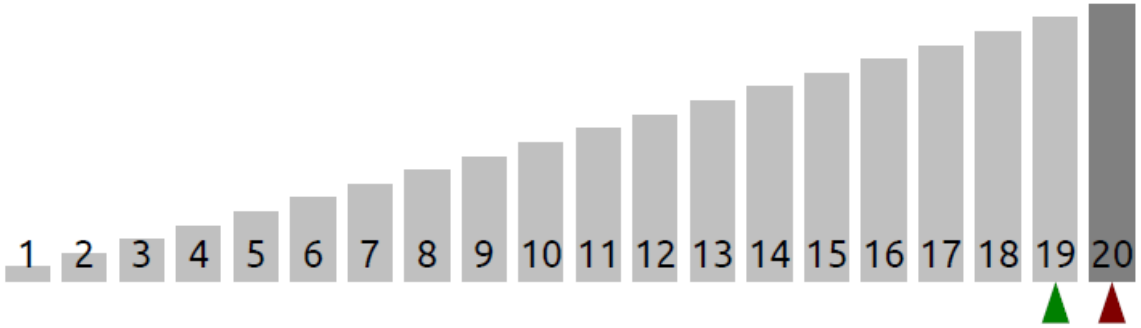
Passo 17:



Passo 18:



Passo 19:



Selection Sort

- Quantas Comparações são executadas?
- Quantas Trocas são executadas?

```
template <class Item>
void selection(Item vetor[], int n)
{
    for (int i = 0; i < n; i++)
   {
        int min = i;

        for (int j = i+1; j < n; j++)
            if (vetor[j] < vetor[min])
                min = j;

        swap(vetor[i], vetor[min]);

    }
}
```

Selection Sort

- Quantas comparações são executadas?
- Quantas trocas são executadas?

Sempre!
O número de comparações ou trocas não dependem dos valores dos dados.

n-1, n-2, n-3, ..., 1

n x 1

```
template <class Item>
void selection(Item vetor[], int n)
{
    for (int i = 0; i < n; i++)
   {
        int min = i;

        for (int j = i+1; j < n; j++)
            if (vetor[j] < vetor[min])
                min = j;

        swap(vetor[i], vetor[min]);

    }
}
```

Selection Sort

- Os valores dos dados não interferem na execução do algoritmo.
- Crescimento do número de trocas em relação ao tamanho de entrada: **linear.**
- Crescimento do número de comparações em relação ao tamanho de entrada: **quadrático.**
- Crescimento do uso de memória em relação ao tamanho da entrada: **constante.**

DÚVIDAS?

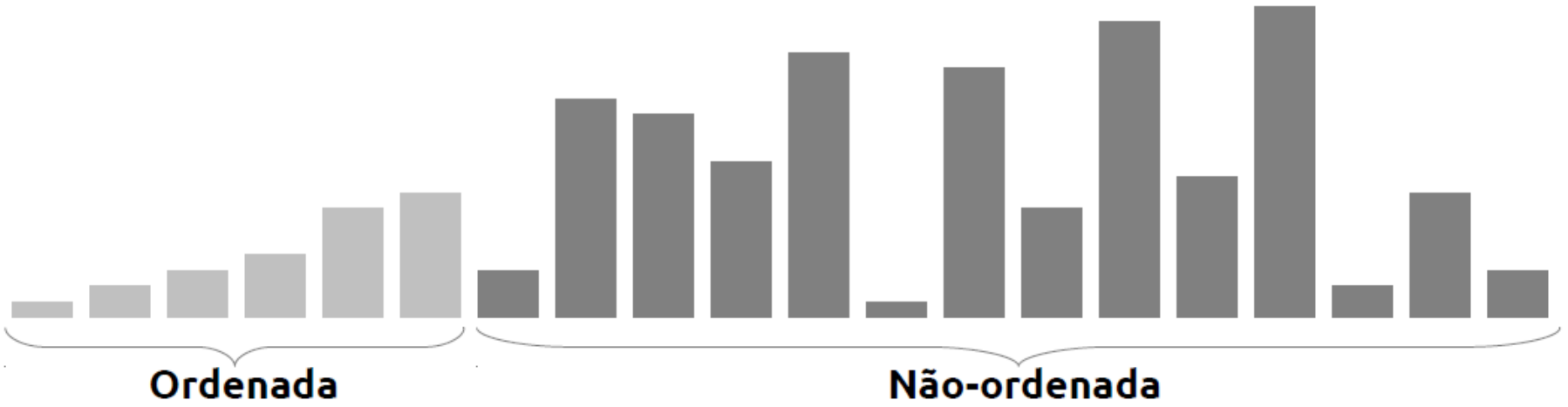
Prof. Ivan José dos Reis Filho

Insertion Sort

- Divisão dos dados em duas sequências:

Ordenada e não ordenada

- Interação: Inserir o primeiro elemento da sequência não ordenada na sequência ordenada.



Insertion Sort

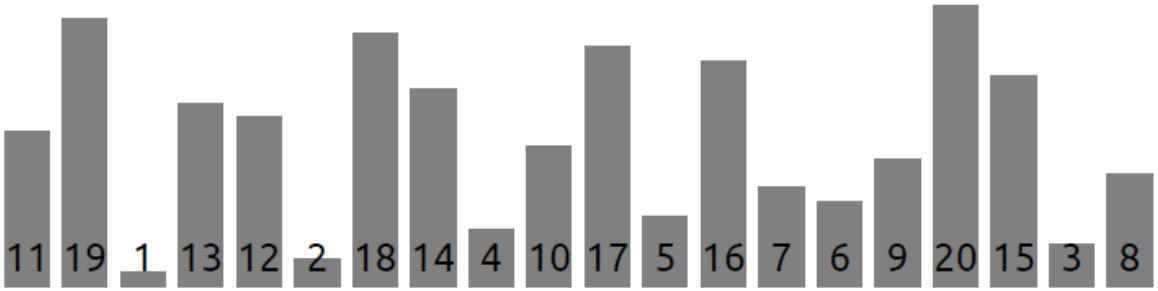
```
template <class Item>
void insertion(Item vetor[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int j = i;
        while ( j > 0 && vetor[j-1] > vetor[j])
        {
            swap(vetor[j-1], vetor[j]);
            j--;
        }
    }
}
```

i
controla a iteração, índice da sequência ordenada

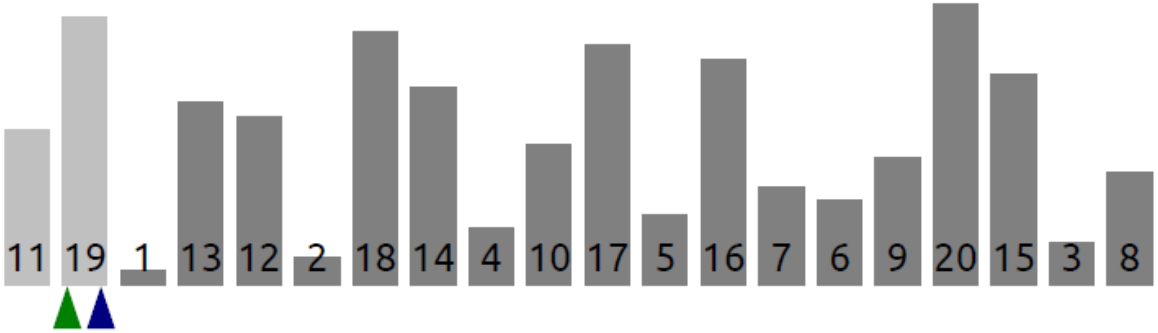
j
controla a inserção do elemento da sequência não-ordenada na sequência ordenada

```
template <class Item>
  void swap(Item &A, Item &B)
    { Item t = A ; A = B; B = t; }
```

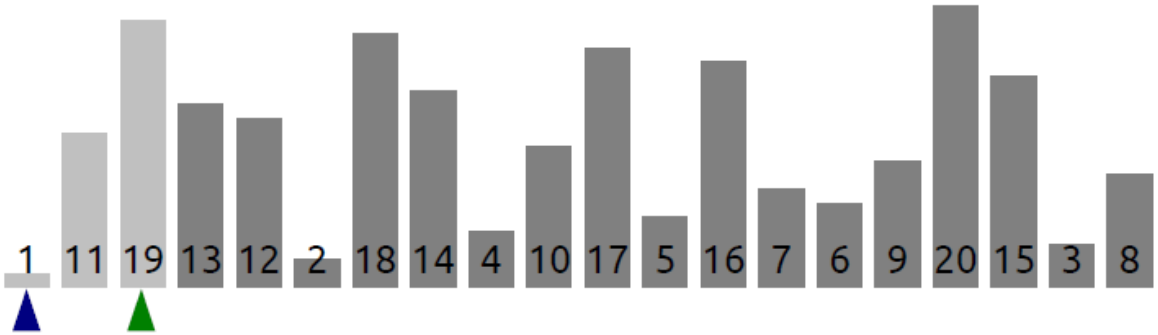
Entrada:



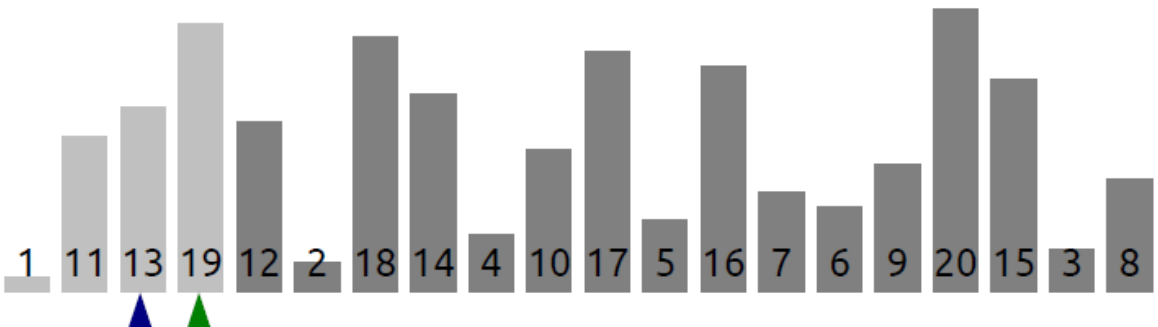
Passo 1:



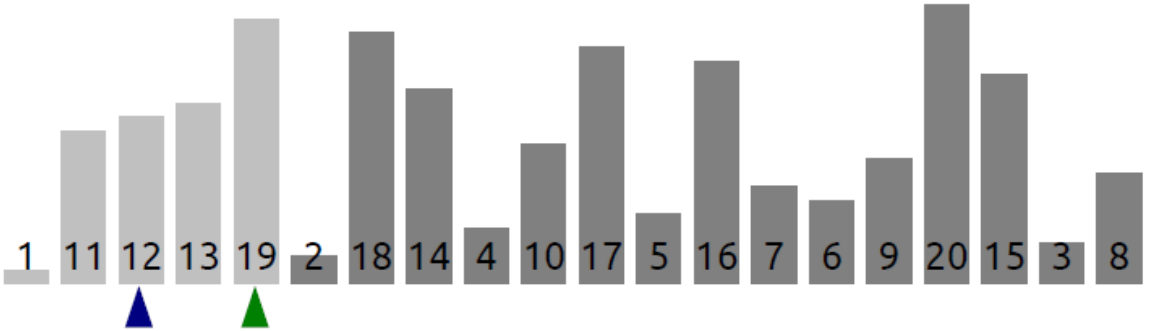
Passo 2:



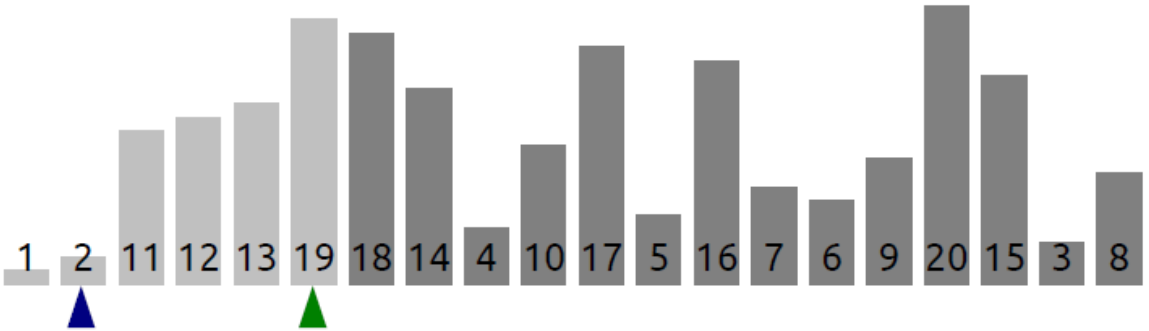
Passo 3:



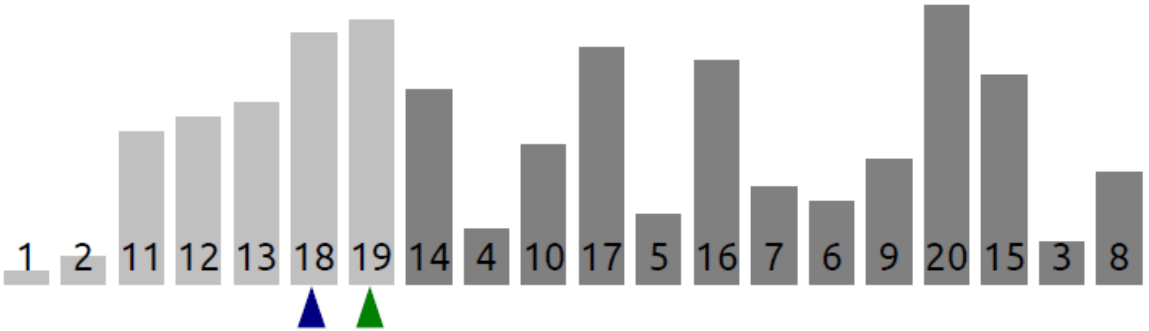
Passo 4:



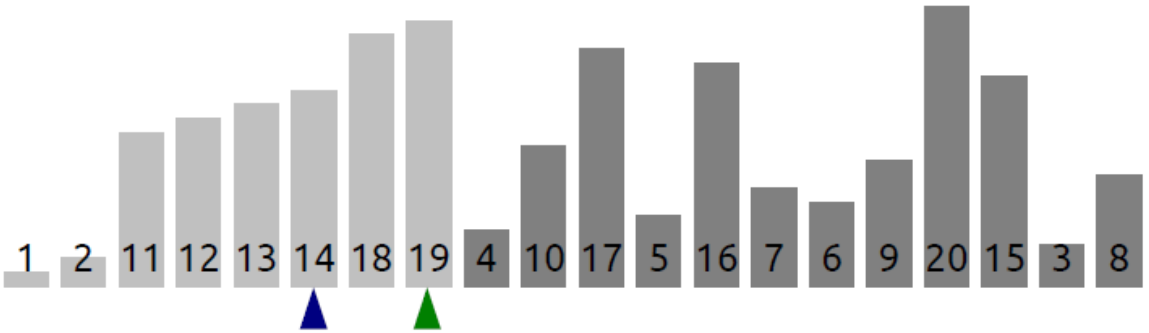
Passo 5:



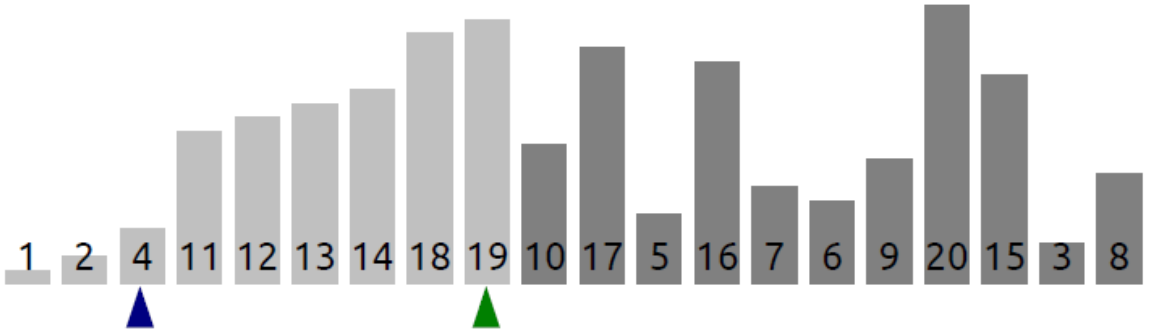
Passo 6:



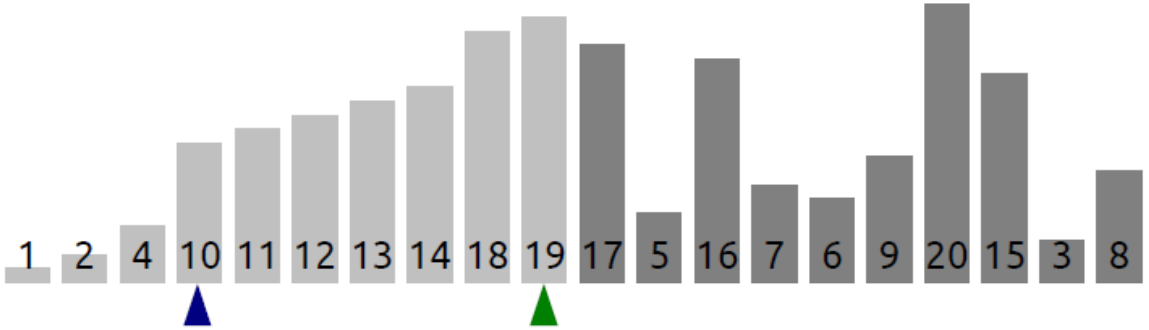
Passo 7:



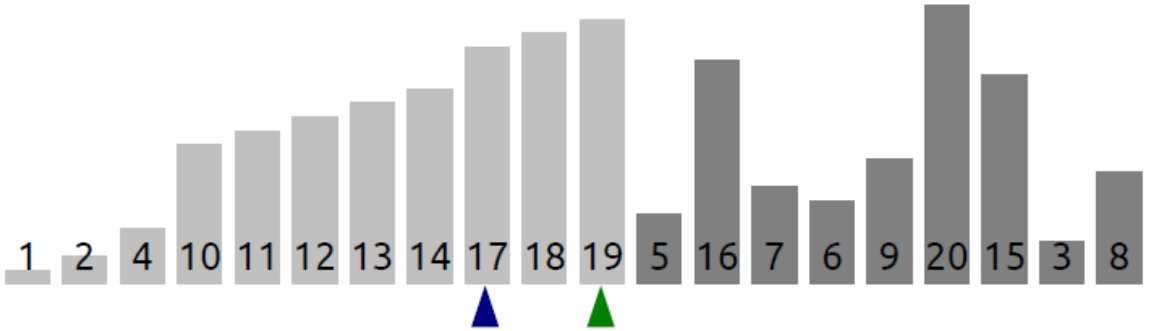
Passo 8:



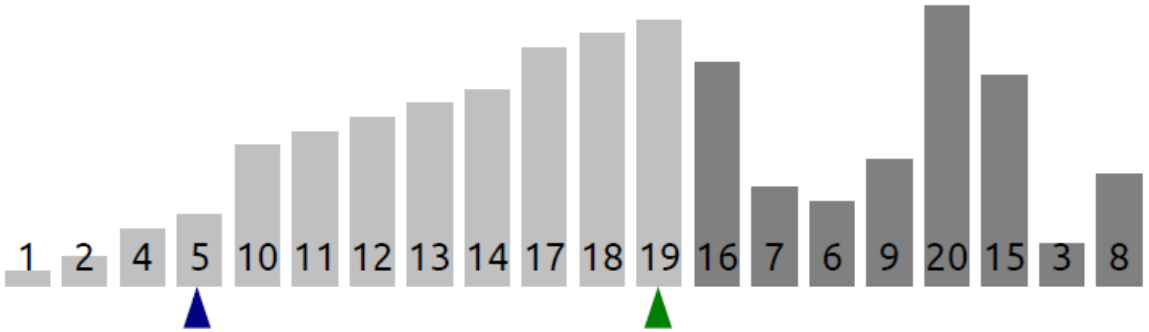
Passo 9:



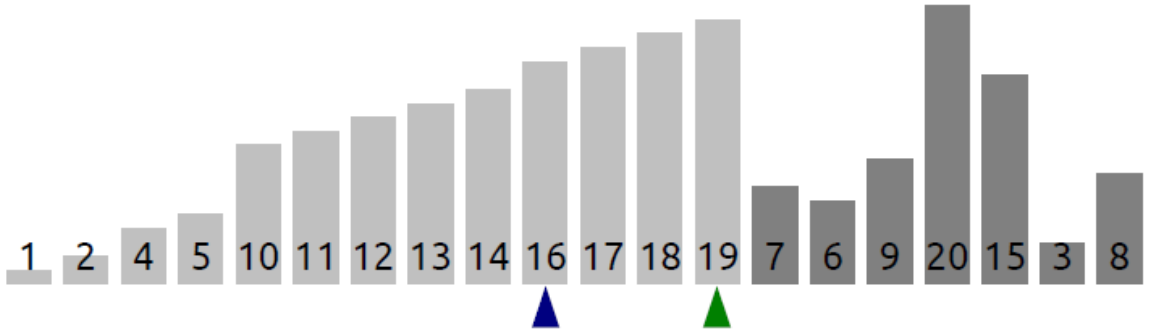
Passo 10:



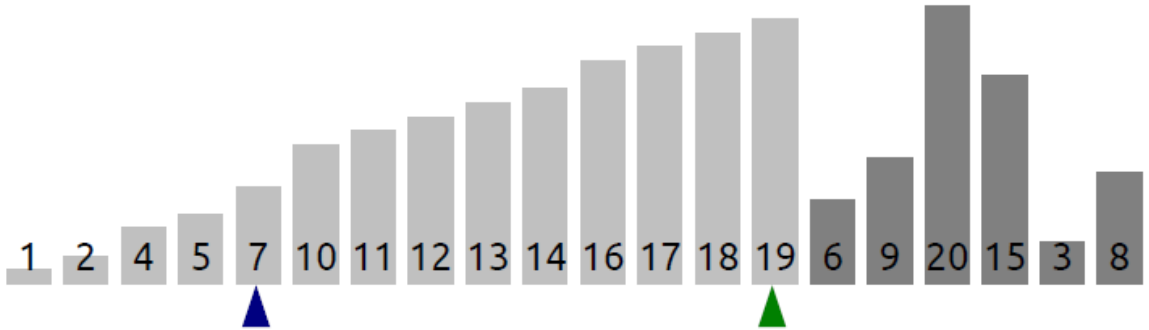
Passo 11:



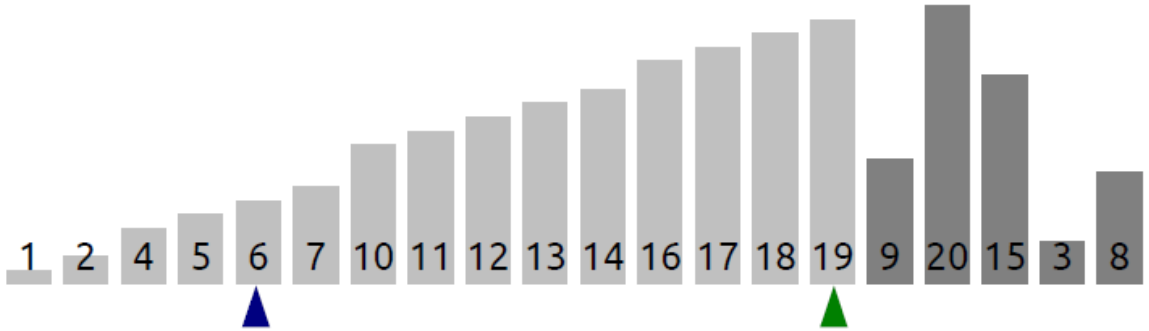
Passo 12:



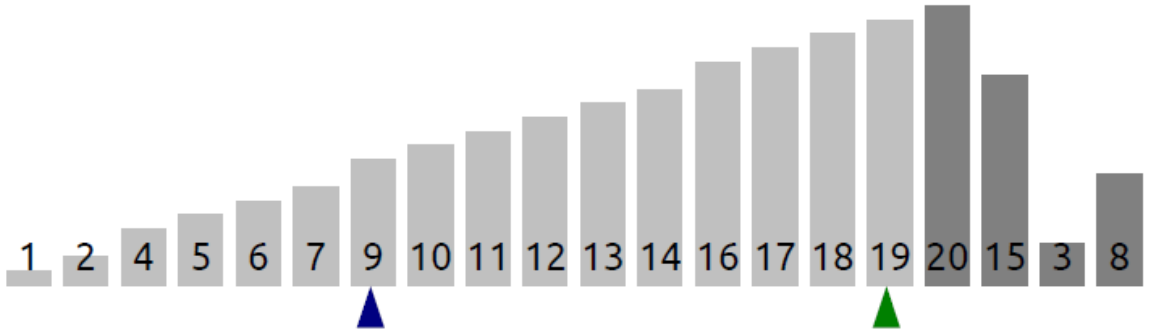
Passo 13:



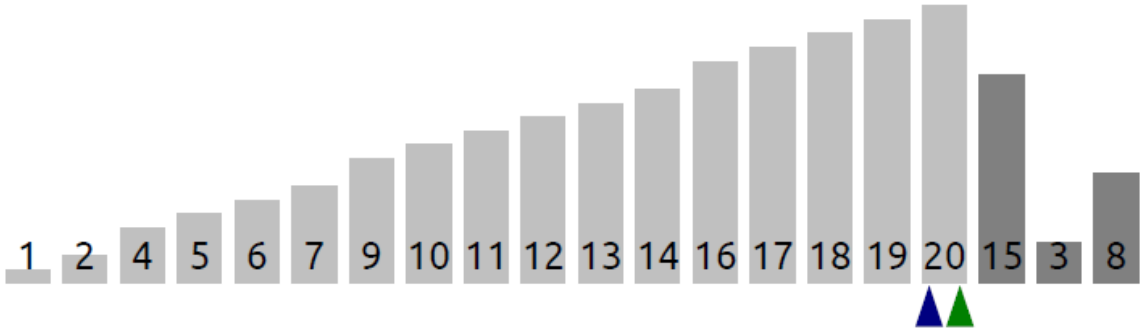
Passo 14:



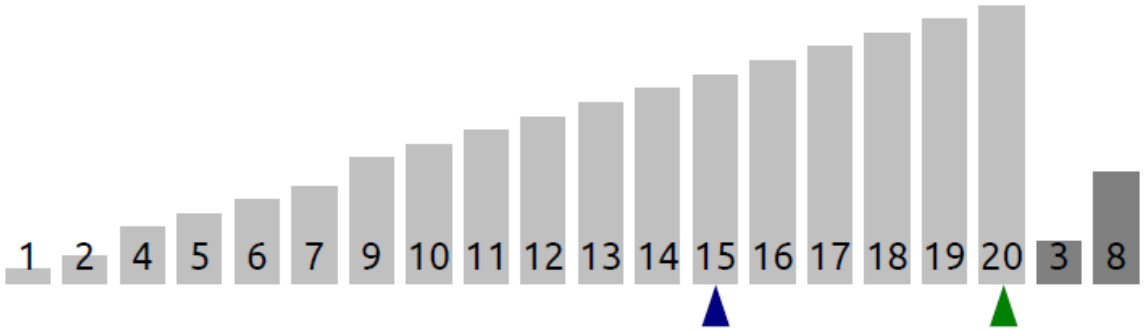
Passo 15:



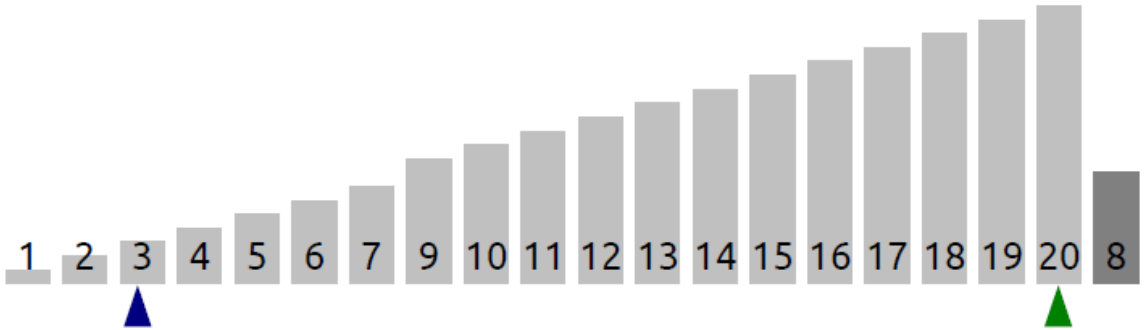
Passo 16:



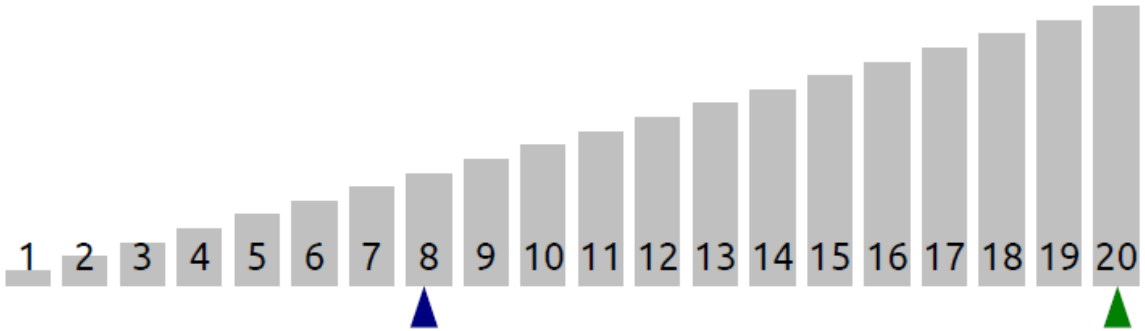
Passo 17:



Passo 18:



Passo 19:



Insertion Sort

- Quantas operações são executadas?
- Quantas trocas são executadas?
- É estável?
- Quantidade de Memória?

```
template <class Item>
void insertion(Item vetor[], int n)
{
     for (int i = 1; i < n; i++)
     {
          int j = i;
          while ( j > 0 && vetor[j-1] > vetor[j])
          {
               swap(vetor[j-1], vetor[j]);
               j--;
          }
     }
}
```

Insertion Sort

- Melhor Caso: Vetor quase ordenado
- Comparações: Linear
- Trocas Constantes

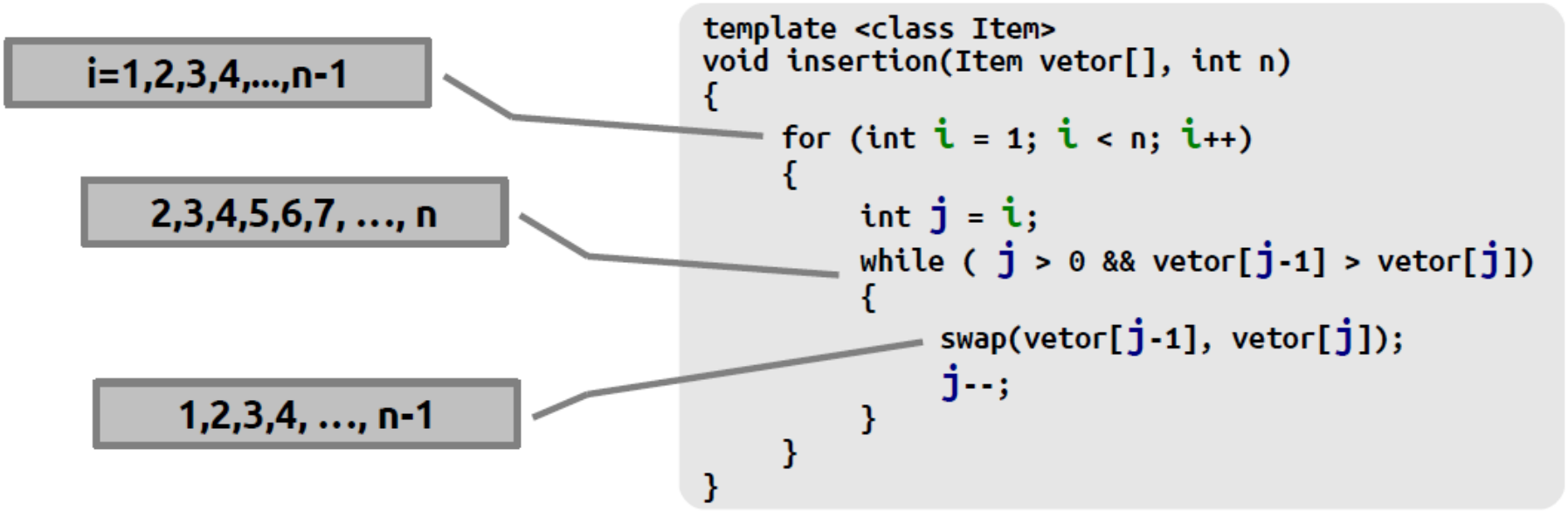
i=1,2,3,4,...,n-1

Nunca é executado!

```
template <class Item>
void insertion(Item vetor[], int n)
{
     for (int i = 1; i < n; i++)
     {
          int j = i;
          while ( j > 0 && vetor[j-1] > vetor[j])
          {
               swap(vetor[j-1], vetor[j]);
               j--;
          }
     }
}
```

Insertion Sort

- Pior Caso: Vetor inversamente ordenado
- Comparações: Quadrática
- Trocas: Quadrática

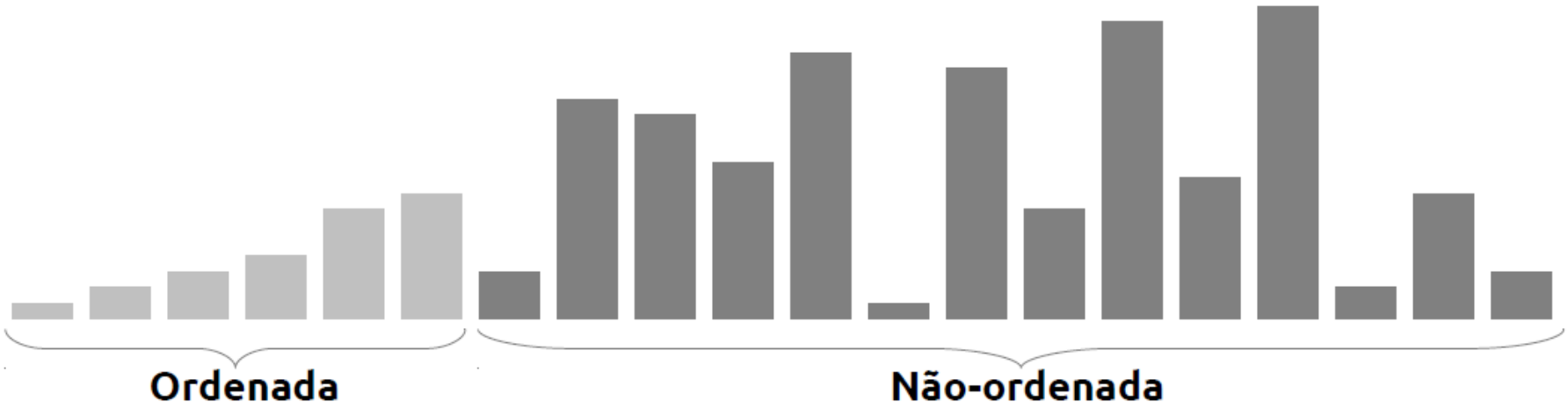


DÚVIDAS?

Prof. Ivan José dos Reis Filho

Bubble Sort

- Aplicação sucessiva de comparações entre vizinhos (na prática também separa a sequência em duas partes: ordenada e não-ordenada).
- Iteração: percorrer toda a sequência não-ordenada comparando todos os vizinhos e trocando de posição quando necessário, no final, o menor elemento poderá ser concatenado na sequência ordenada.



Bubble Sort

```
template <class Item>
void bubble(Item vetor[], int n)
{
    bool swapped;
    int i = 0;
    do {
        swapped = false;
        for( int j = n-1; j > i; j--)
        {
            if ( vetor[j-1] > vetor[j])
            {
                swap(vetor[j-1], vetor[j]);
                swapped = true;
            }
        }
        i++;
    } while( swapped && i < n );
}
```

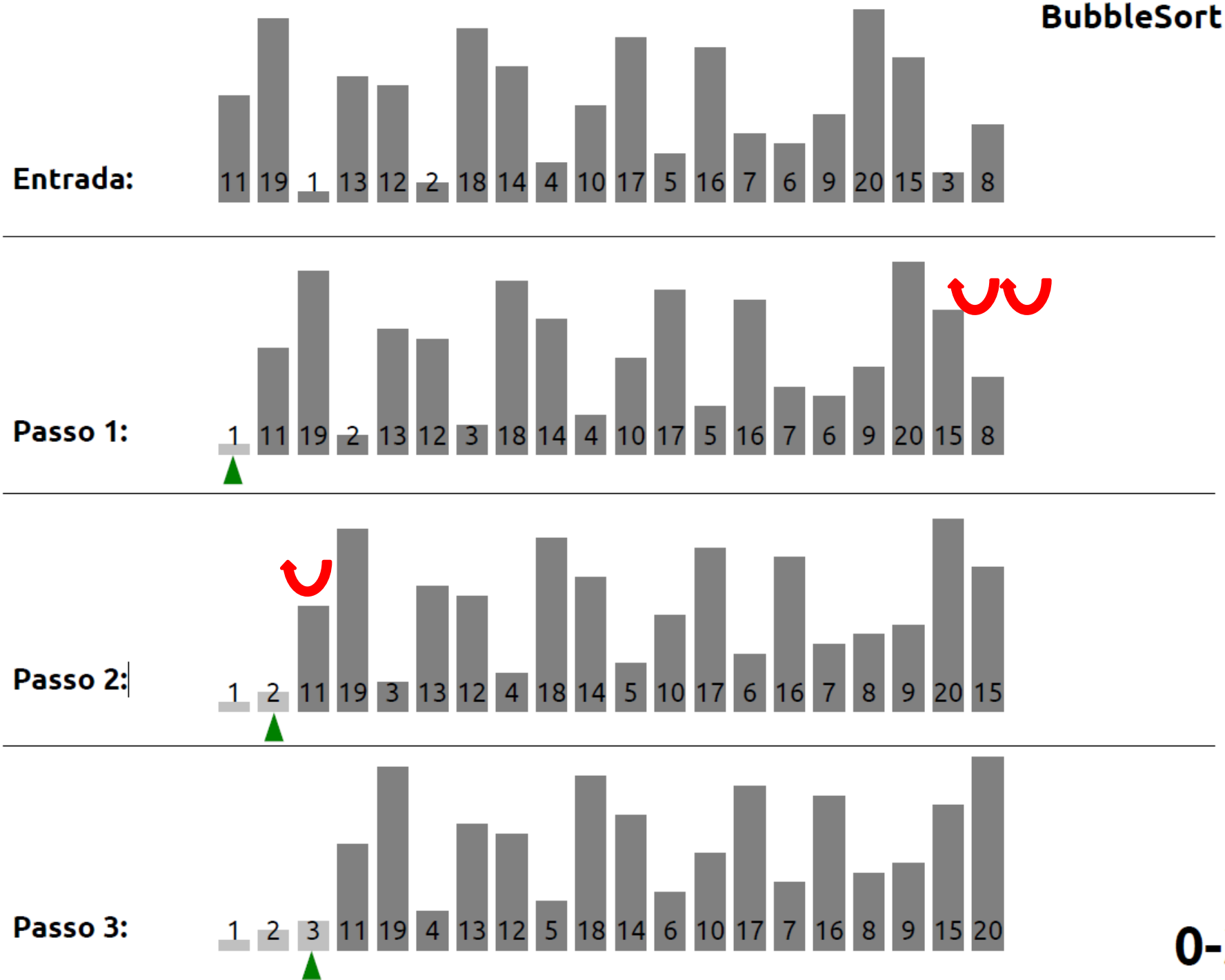
i
controla a iteração, índice da sequência ordenada

j
controla a varredura na sequência não-ordenada

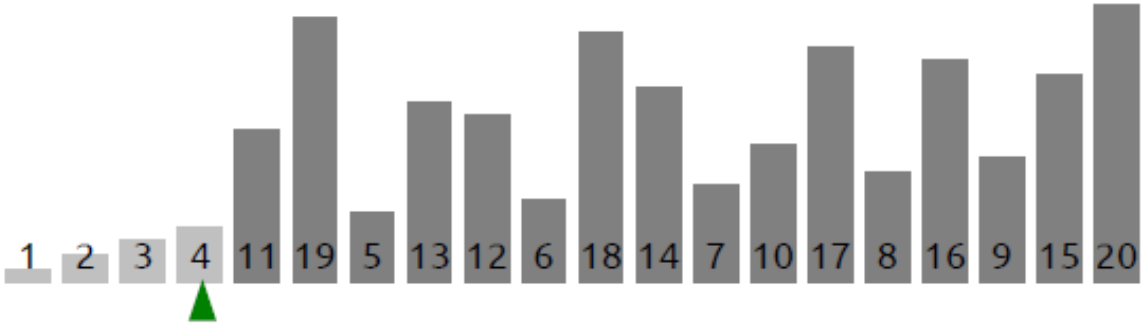
swapped
marca se houve ou não trocas

```
template <class Item>
  void swap(Item &A, Item &B)
    { Item t = A ; A = B; B = t; }
```

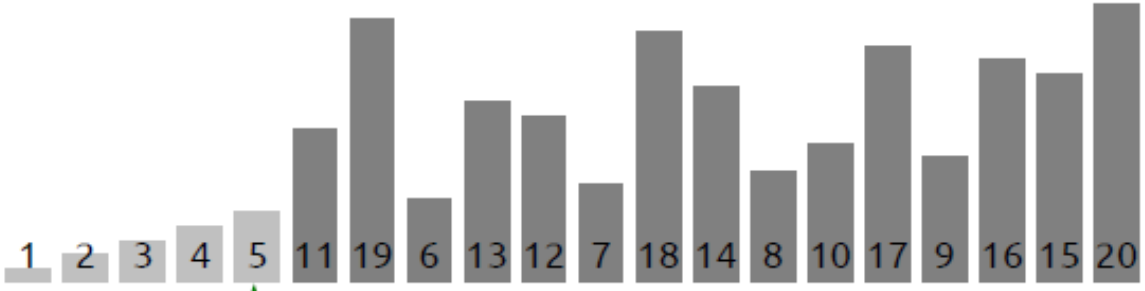
BubbleSort


Entrada:
11 19 1 13 12 2 18 14 4 10 17 5 16 7 6 9 20 15 3 8
Passo 1:
1 11 19 2 13 12 3 18 14 4 10 17 5 16 7 6 9 20 15 8
Passo 2:
1 2 11 19 3 13 12 4 18 14 5 10 17 6 16 7 8 9 20 15
Passo 3:
1 2 3 11 19 4 13 12 5 18 14 6 10 17 7 16 8 9 15 20

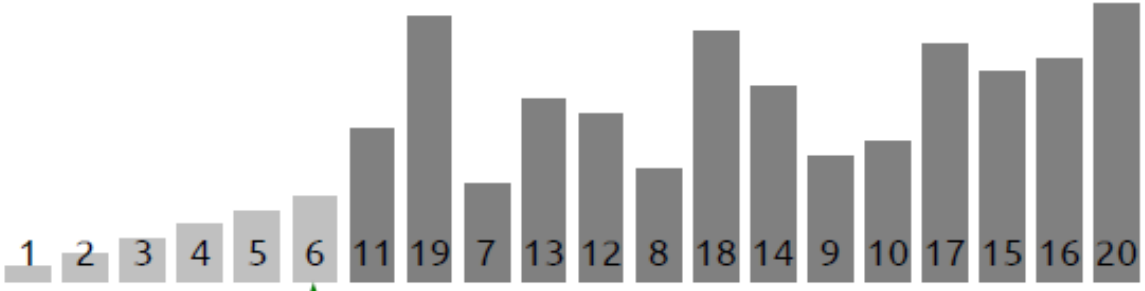
Passo 4:



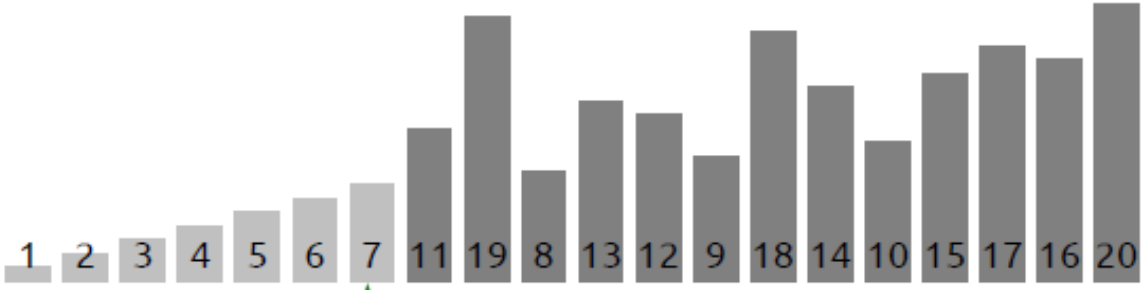
Passo 5:



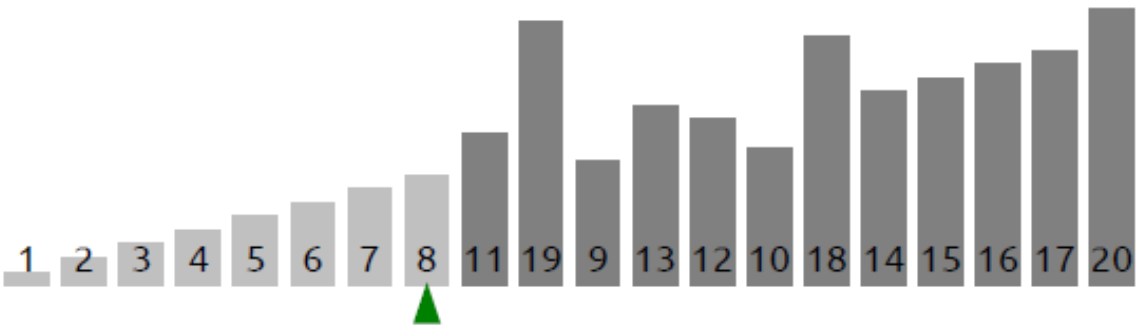
Passo 6:



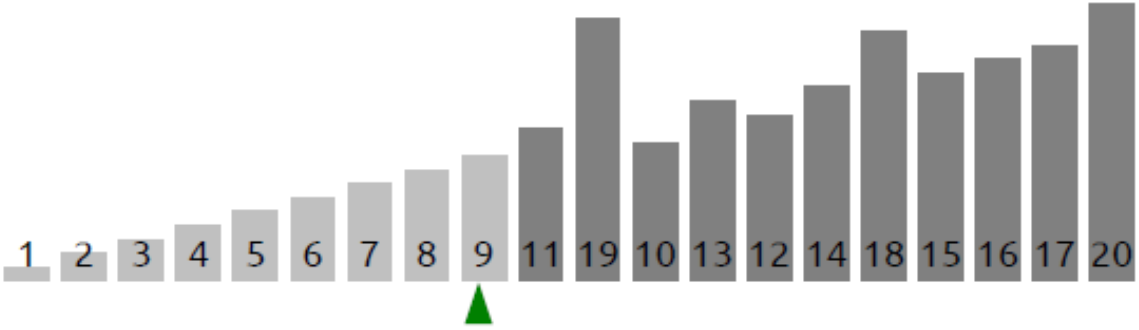
Passo 7:



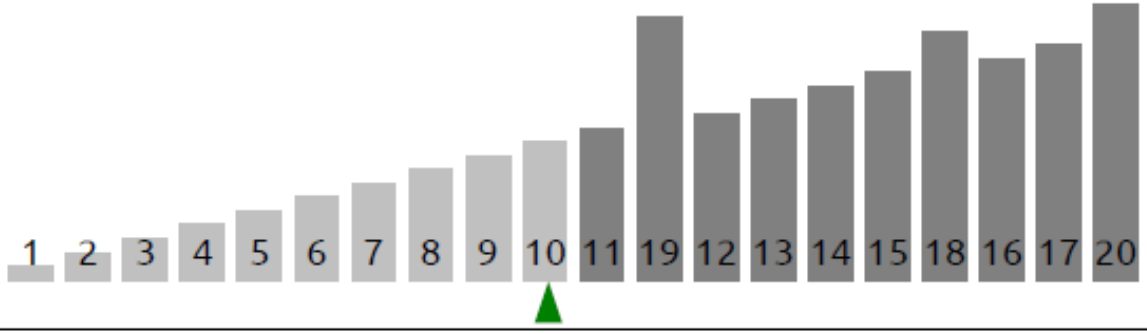
Passo 8:



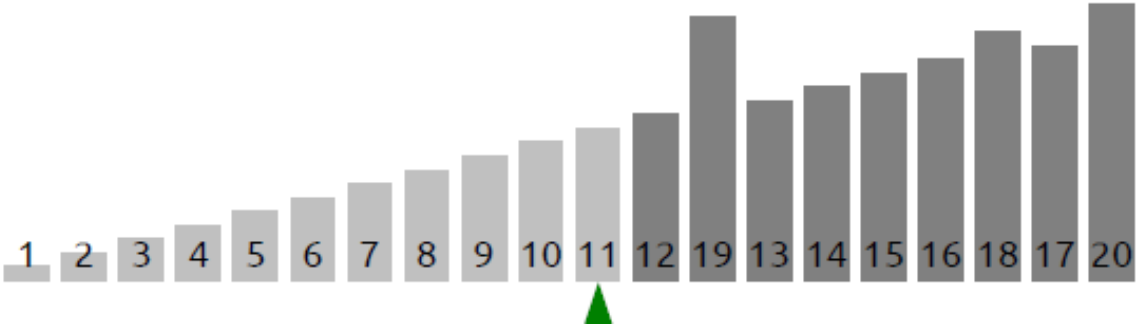
Passo 9:



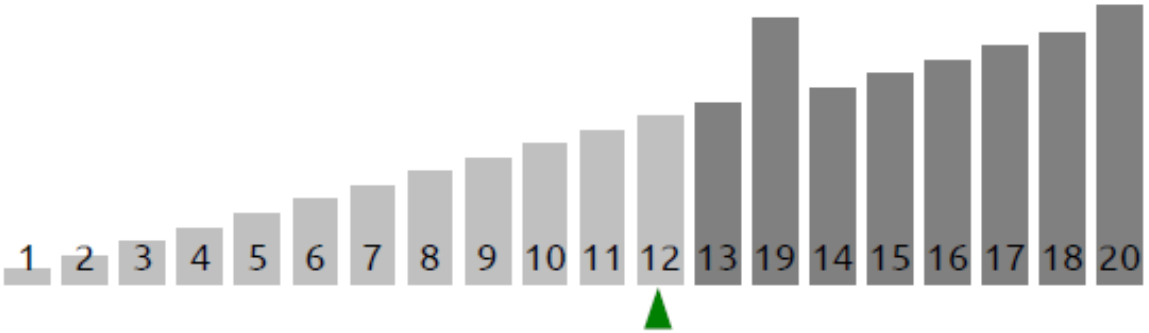
Passo 10:



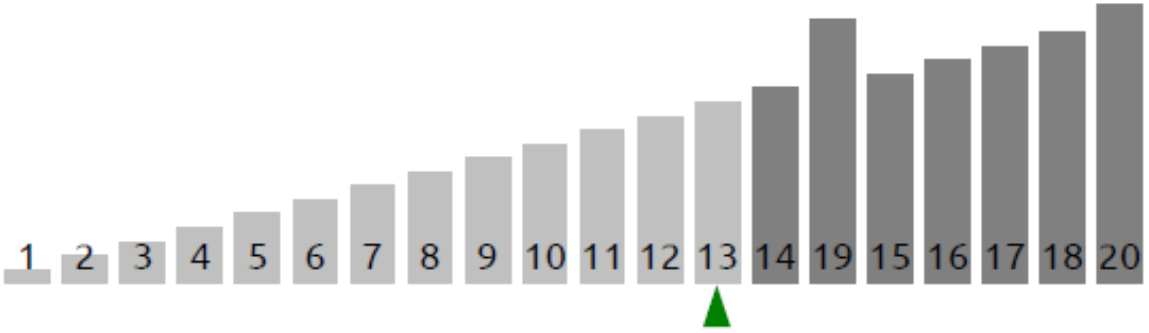
Passo 11:



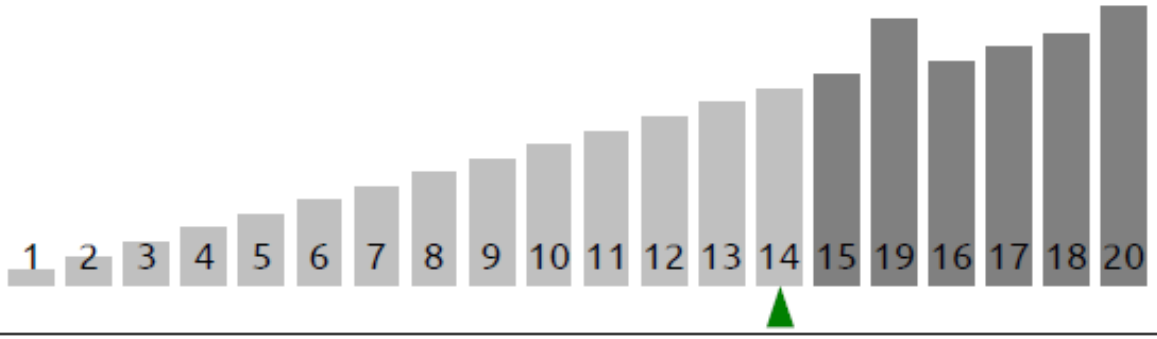
Passo 12:



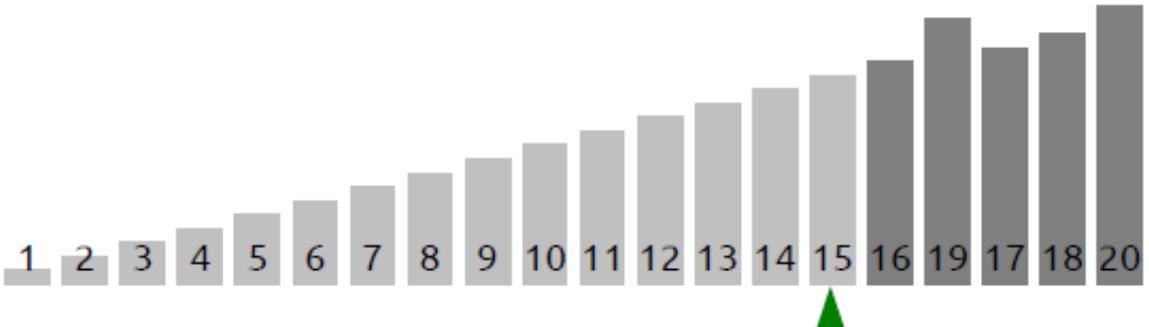
Passo 13:



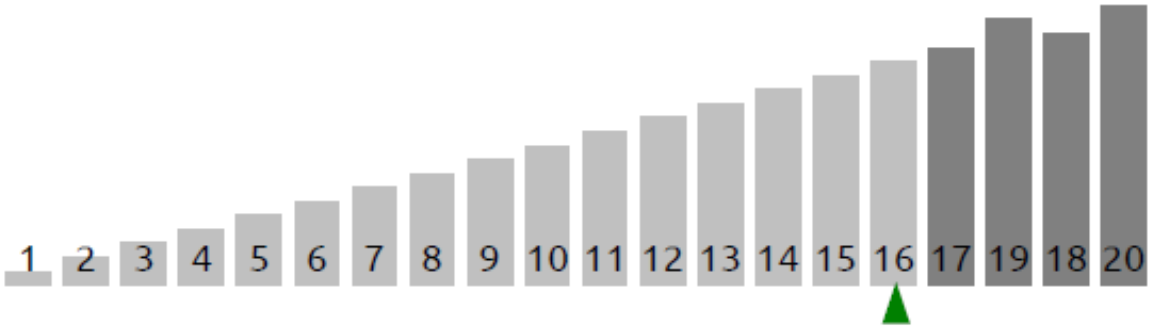
Passo 14:



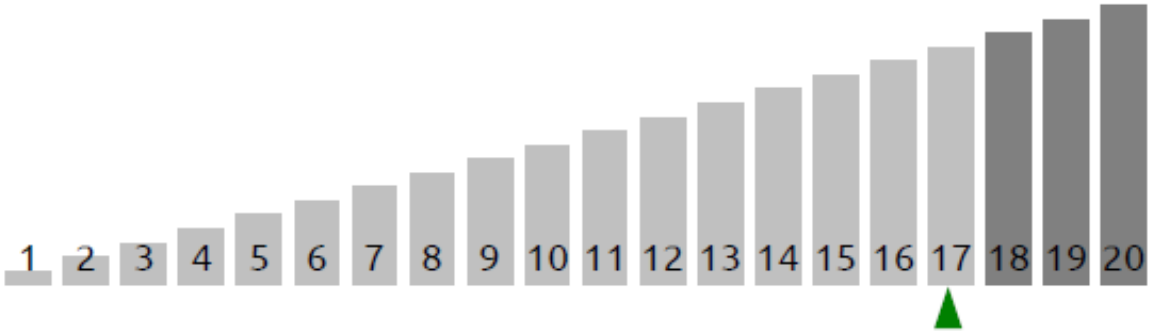
Passo 15:



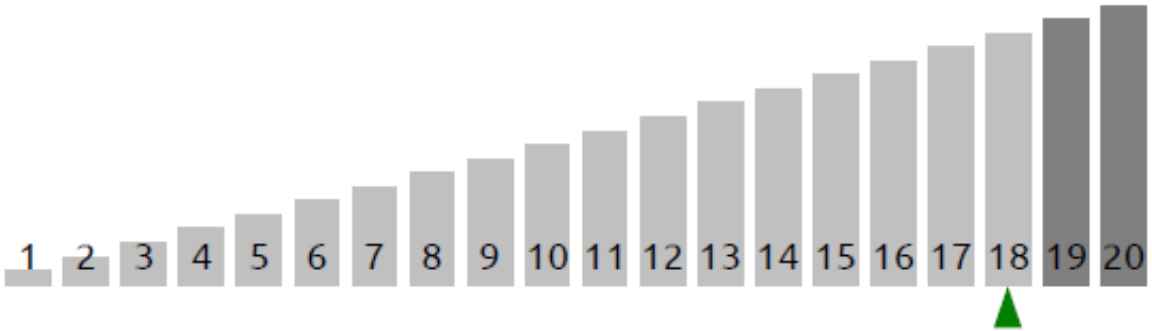
Passo 16:



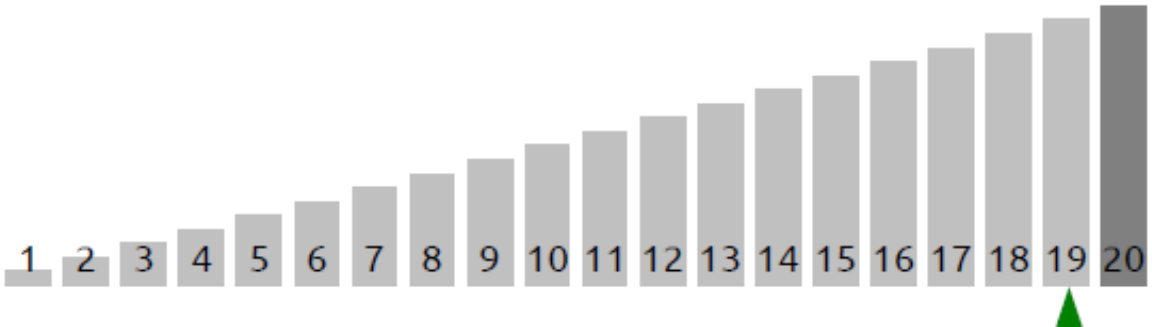
Passo 17:



Passo 18:



Passo 19:



Bubble Sort

- Quantas comparações são executadas?
- Quantas trocas são executadas?

```
template <class Item>
void bubble(Item vetor[], int n)
{
      bool swapped;
      int i = 0;
      do {
            swapped = false;
            for( int j = n-1; j > i; j--)
            {
                  if ( vetor[j-1] > vetor[j])
                  {
                        swap(vetor[j-1], vetor[j]);
                        swapped = true;
                  }
            }
            i++;
      } while( swapped && i < n );
}
```

Bubble Sort

- Melhor caso: Vetor já ordenado

Apenas i=0

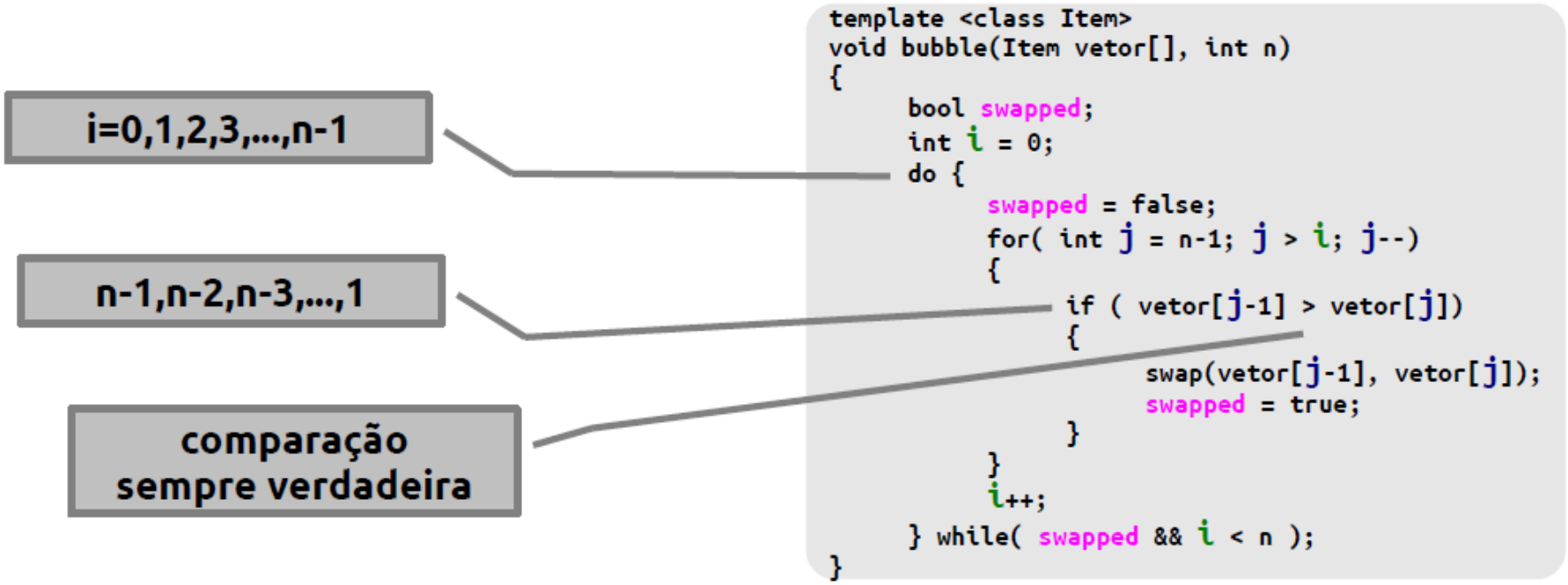
j = n-1, n-2, ..., 1

Nunca é executado!

```
template <class Item>
void bubble(Item vetor[], int n)
{
      bool swapped;
      int i = 0;
      do {
            swapped = false;
            for( int j = n-1; j > i; j--)
            {
                  if ( vetor[j-1] > vetor[j])
                  {
                        swap(vetor[j-1], vetor[j]);
                        swapped = true;
                  }
            }
            i++;
      } while( swapped && i < n );
}
```

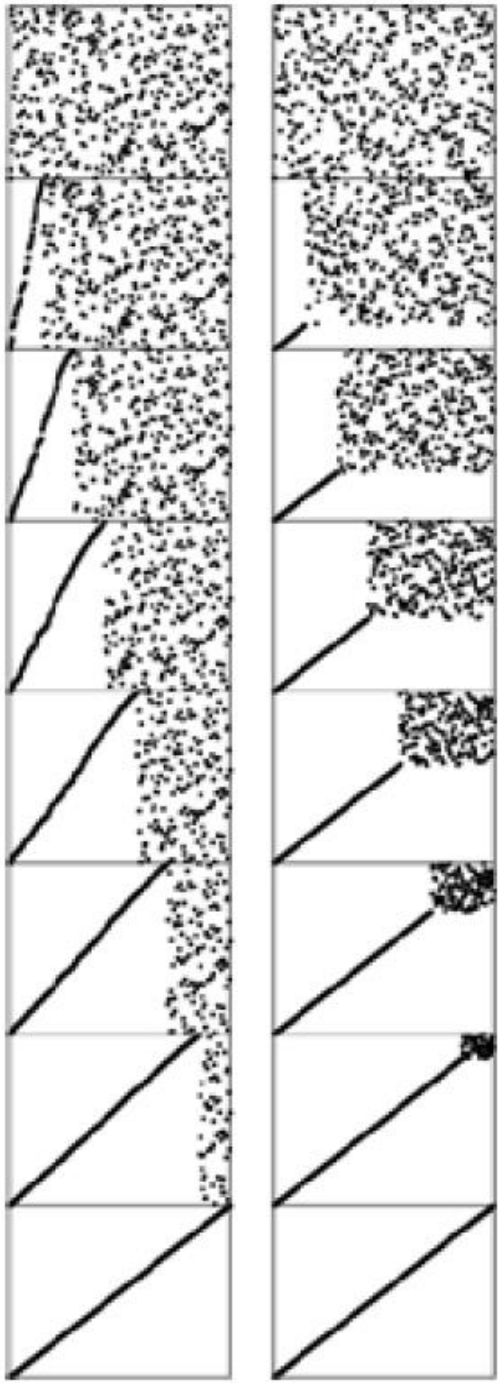
Bubble Sort

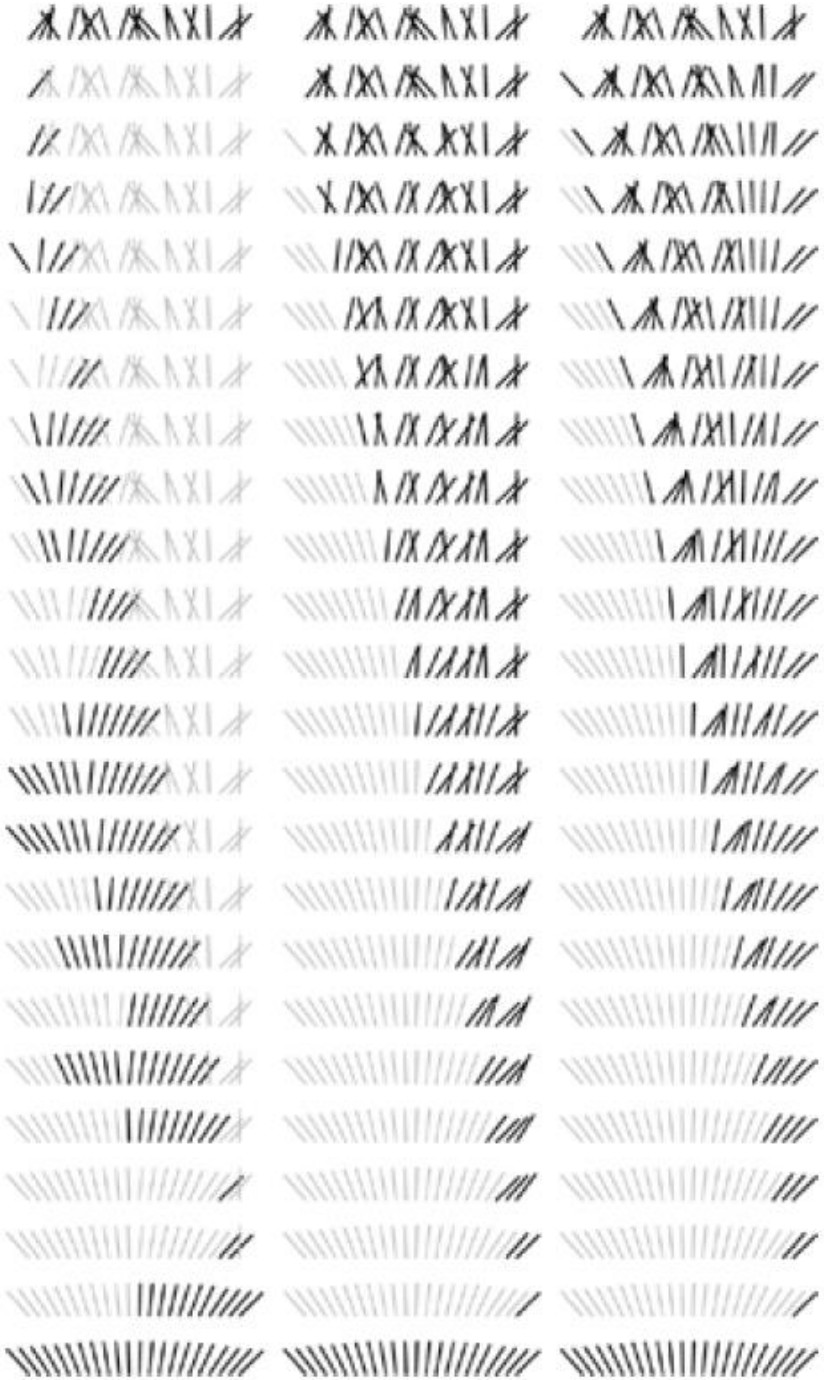
- Pior caso: Vetor inversamente ordenado



DÚVIDAS?

Prof. Ivan José dos Reis Filho





Trabalho!!!