ALGORITMOS E ESTRUTURA DE DADOS

Curso de Engenharia de Software
Lucas Sampaio Leite

O problema da ordenação

- Definição: Dada uma coleção de elementos, o problema da ordenação consiste em reorganizá-los de acordo com um critério estabelecido, geralmente em ordem crescente ou decrescente.

O problema da ordenação

INSTITUTO FEDERAL
Pernambuco

- Definição: Dada uma coleção de elementos, o problema da ordenação consiste em reorganizá-los de acordo com um critério estabelecido, geralmente em ordem crescente ou decrescente.
- Operação fundamental na computação:
  - Nos primórdios da computação era de conhecimento comum que até 30% de todos os ciclos de processamento era gasto com ordenação.[1]
  - Hoje, essa proporção é menor, devido à eficiência dos algoritmos de ordenação.[1]

[1]Robert Sedgewick and Kevin Wayne. 2011. *Algorithms* (4th ed.). Addison-Wesley Professional

Bubble sort

INSTITUTO FEDERAL
Pernambuco

- O Bubble Sort é um algoritmo de ordenação que percorre repetidamente a coleção, comparando elementos adjacentes e trocando suas posições sempre que estiverem na ordem incorreta. A cada passagem, os maiores (ou menores) elementos são deslocados gradualmente para suas posições finais.
- O nome Bubble Sort ("Ordenação por Bolha") vem do fato de que os maiores elementos "flutuam" gradualmente para o final da coleção, assim como bolhas sobem à superfície da água.

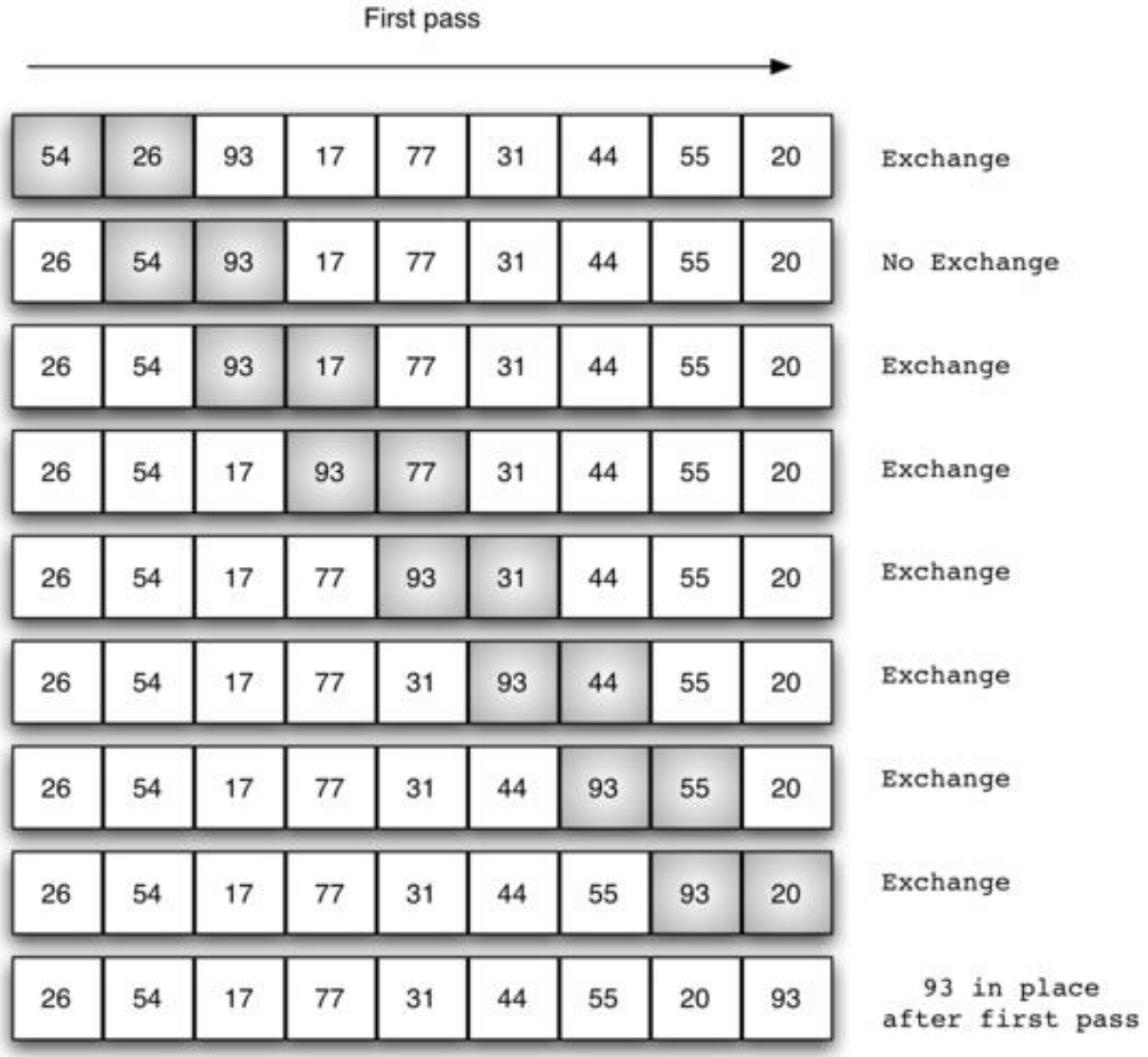
Bubble sort

INSTITUTO FEDERAL
Pernambuco

- Etapas:
    1. Compare dois elementos adjacentes.
    2. Se o elemento da esquerda for maior que o da direita, troque-os de posição.
    3. Continue esse processo para todos os pares adjacentes da coleção.
    4. Ao final de cada passagem, pelo menos um elemento é posicionado corretamente.
    5. Repita as passagens até que nenhuma troca seja realizada, indicando que a coleção está ordenada.

Bubble sort

- Exemplo (primeira passagem):



Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OBubbleSort.html#lst-bubble

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
j = 0

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
j = 0

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
j = 1

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
j = 1

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	10	1	2	7	3

tamanho = 7
i = 0
j = 1

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	10	1	2	7	3

tamanho = 7
i = 0
j = 2

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	10	1	2	7	3

tamanho = 7
i = 0
j = 2

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	10	2	7	3

tamanho = 7
i = 0
j = 2

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	10	2	7	3

tamanho = 7
i = 0
j = 3

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	10	2	7	3

tamanho = 7
i = 0
j = 3

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	10	7	3

tamanho = 7
i = 0
j = 3

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	10	7	3

tamanho = 7
i = 0
j = 4

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	10	7	3

tamanho = 7
i = 0
j = 4

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	10	3

tamanho = 7
i = 0
j = 4

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	10	3

tamanho = 7
i = 0
j = 5

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	10	3

tamanho = 7
i = 0
j = 5

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	3	10

tamanho = 7
i = 0
j = 5

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	10	3

tamanho = 7
i = 0
j = 6 ✖

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	3	10

tamanho = 7
i = 1
j = 0

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	3	10

tamanho = 7
i = 1
j = 0

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	3	10

tamanho = 7
i = 1
j = 1

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	8	1	2	7	3	10

tamanho = 7
i = 1
j = 1

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	8	2	7	3	10

tamanho = 7
i = 1
j = 1

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	8	2	7	3	10

tamanho = 7
i = 1
j = 2

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	8	2	7	3	10

tamanho = 7
i = 1
j = 2

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	8	7	3	10

tamanho = 7
i = 1
j = 2

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	8	7	3	10

tamanho = 7
i = 1
j = 3

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	8	7	3	10

tamanho = 7
i = 1
j = 3

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	7	8	3	10

tamanho = 7
i = 1
j = 3

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	7	8	3	10

tamanho = 7
i = 1
j = 4

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	7	8	3	10

tamanho = 7
i = 1
j = 4

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	7	3	8	10

tamanho = 7
i = 1
j = 4

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	7	3	8	10

tamanho = 7
i = 1
j = 5 ✖

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

5	1	2	7	3	8	10

tamanho = 7
i = 2
j = 0

Continua até a última passagem!!!

Bubble sort

INSTITUTO FEDERAL
Pernambuco

```c
void bubbleSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j + 1]) {
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}
```

O que acontece se o vetor já estiver ordenado no momento da chamada da função bubleSort?

Bubble sort

```c
void bubbleSortModified(int vetor[], int tamanho) {

    int trocou;

    for (int i = 0; i < tamanho - 1; i++) {

        trocou = 0;

        for (int j = 0; j < tamanho - i - 1; j++) {

            if (vetor[j] > vetor[j + 1]) {

                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;

                trocou = 1;
            }
        }

        if (trocou == 0)
            break;
    }
}
```

Bubble sort

INSTITUTO FEDERAL
Pernambuco

- Vantagens:
  - Fácil de compreender e implementar.
  - Baixo uso de memória: Realiza a ordenação no próprio vetor.
  - Com uma pequena otimização, pode identificar que a coleção já está ordenada quando nenhuma troca é realizada durante uma passagem.
- Desvantagens:
  - Baixa eficiência: Realiza muitas comparações e trocas desnecessárias.
  - Complexidade quadrática: Possui complexidade $O(n^2)$.

Selection sort

INSTITUTO FEDERAL
Pernambuco

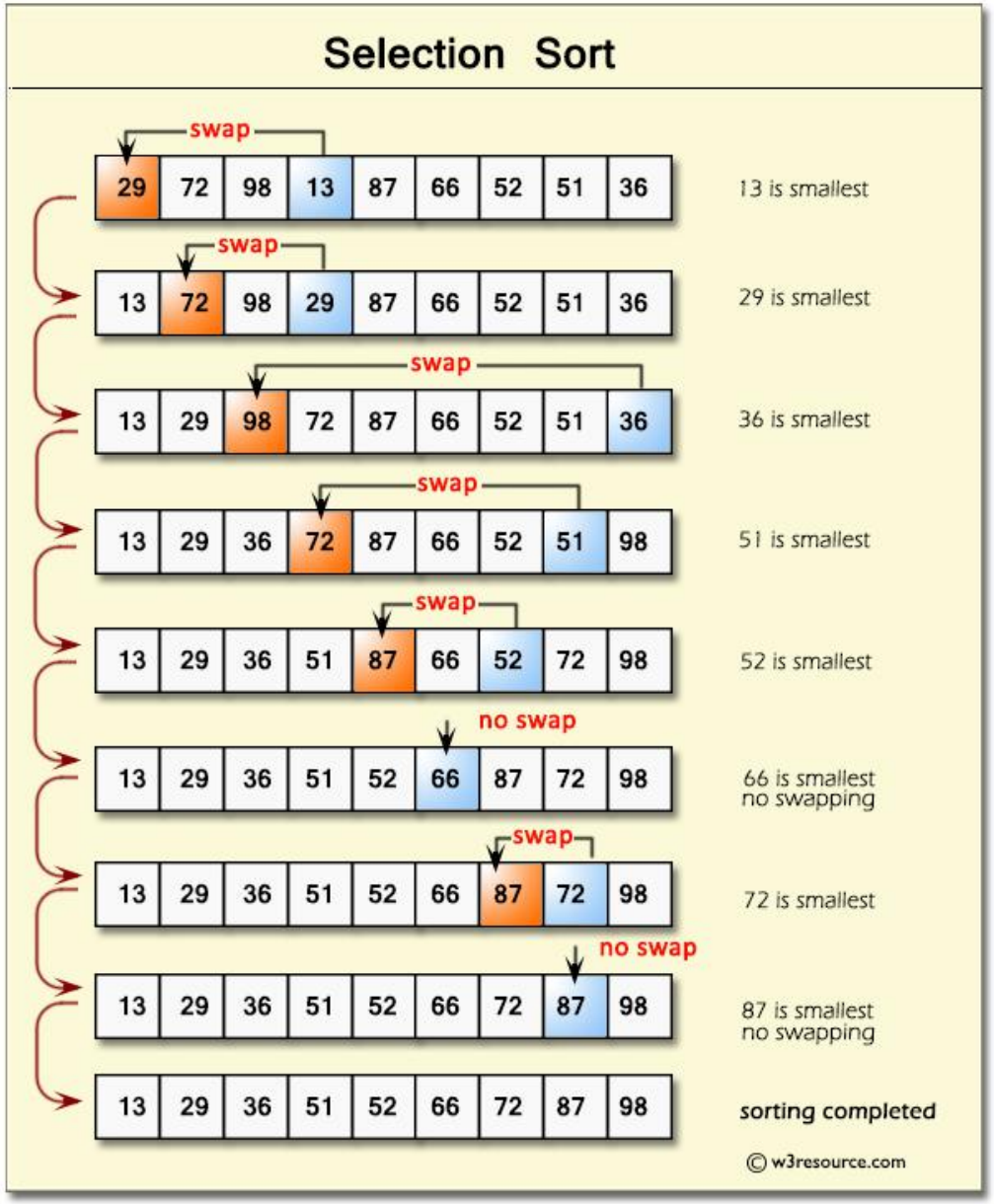
- O Selection Sort ordena a coleção selecionando, a cada passo, o menor (ou maior) elemento da parte ainda não ordenada e colocando-o em sua posição correta.
- Diferentemente do Bubble Sort, o Selection Sort realiza apenas uma troca por passagem, após identificar o menor (ou maior) elemento da região não ordenada da coleção. Isso reduz a quantidade de trocas realizadas pelo algoritmo.

Selection sort

INSTITUTO FEDERAL
Pernambuco

- Etapas:
    1. Considere a primeira posição da coleção como a posição a ser preenchida.
    2. Procure o menor elemento entre os elementos ainda não ordenados.
    3. Troque esse elemento com o elemento da posição atual.
    4. Avance para a próxima posição da coleção.
    5. Repita o processo até que todos os elementos estejam ordenados.

Selection sort



Fonte da imagem: https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-11.php

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

INSTITUTO FEDERAL
Pernambuco

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 0
j = 1

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 0
j = 1

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 0
j = 2

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 0
j = 2

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 0
j = 3

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 0
j = 3

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 3

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 4

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 4

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 5

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 5

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 6

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 6

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 7 ✖

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 0
menor = 3
j = 7

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 0
menor = 3
j = 7

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 1
j = 2

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 1
j = 2

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 2
j = 2

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 2
j = 3

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 2
j = 3

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 3
j = 3

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 3
j = 4

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 3
j = 4

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 4

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 5

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 5

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 6

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 6

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 7 ✖

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	10	8	5	2	7	3

tamanho = 7
i = 1
menor = 4
j = 7

Selection sort

```c
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	2	8	5	10	7	3

tamanho = 7
i = 1
menor = 4
j = 7

Selection sort

```
void selectionSort(int vetor[], int tamanho) {

    for (int i = 0; i < tamanho - 1; i++) {

        int menor = i;

        for (int j = i + 1; j < tamanho; j++) {

            if (vetor[j] < vetor[menor]) {
                menor = j;
            }
        }

        if (menor != i) {
            int temp = vetor[i];
            vetor[i] = vetor[menor];
            vetor[menor] = temp;
        }
    }
}
```

1	2	8	5	10	7	3

tamanho = 7
i = 2
menor = 2
j = 3

Continua até a última passagem!!!

Selection sort

INSTITUTO FEDERAL
Pernambuco

- Vantagens:
  - Fácil de compreender e implementar.
  - Poucas trocas por passagem.
  - Baixo uso de memória: Realiza a ordenação no próprio vetor.
  - Desempenho previsível.
- Desvantagens:
  - Baixa eficiência.
  - Grande número de comparações: Mesmo que a coleção já esteja ordenada, o algoritmo continua realizando todas as comparações previstas.
  - Complexidade quadrática: Possui complexidade $O(n^2)$.

Insertion sort

INSTITUTO FEDERAL
Pernambuco

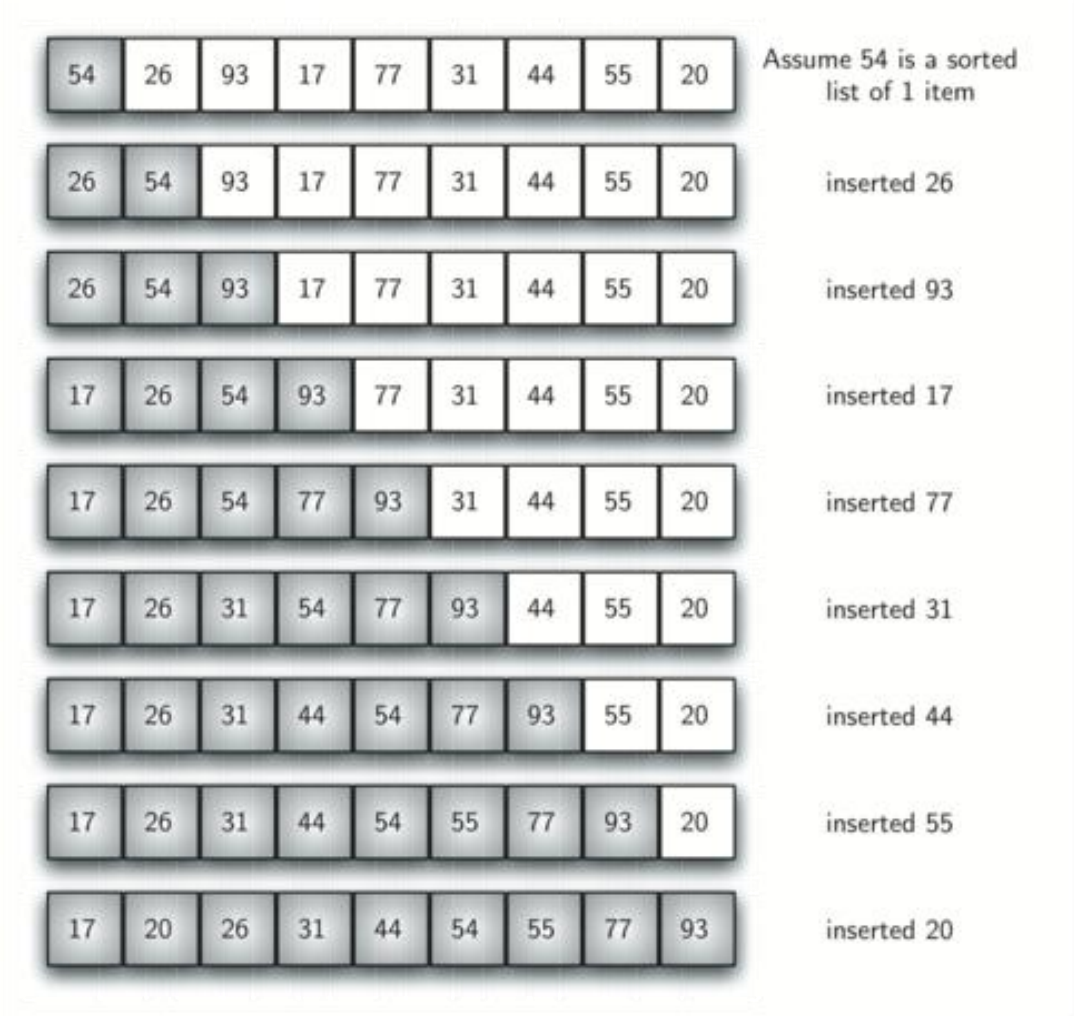
- O Insertion Sort é um algoritmo de ordenação que constrói a sequência ordenada gradualmente. A cada passo, um elemento é retirado da parte não ordenada da coleção e inserido na posição correta da parte já ordenada.
- A ideia é semelhante à forma como organizamos cartas em uma mão de baralho: cada nova carta é inserida na posição adequada entre as cartas já organizadas.

Insertion sort

INSTITUTO FEDERAL
Pernambuco

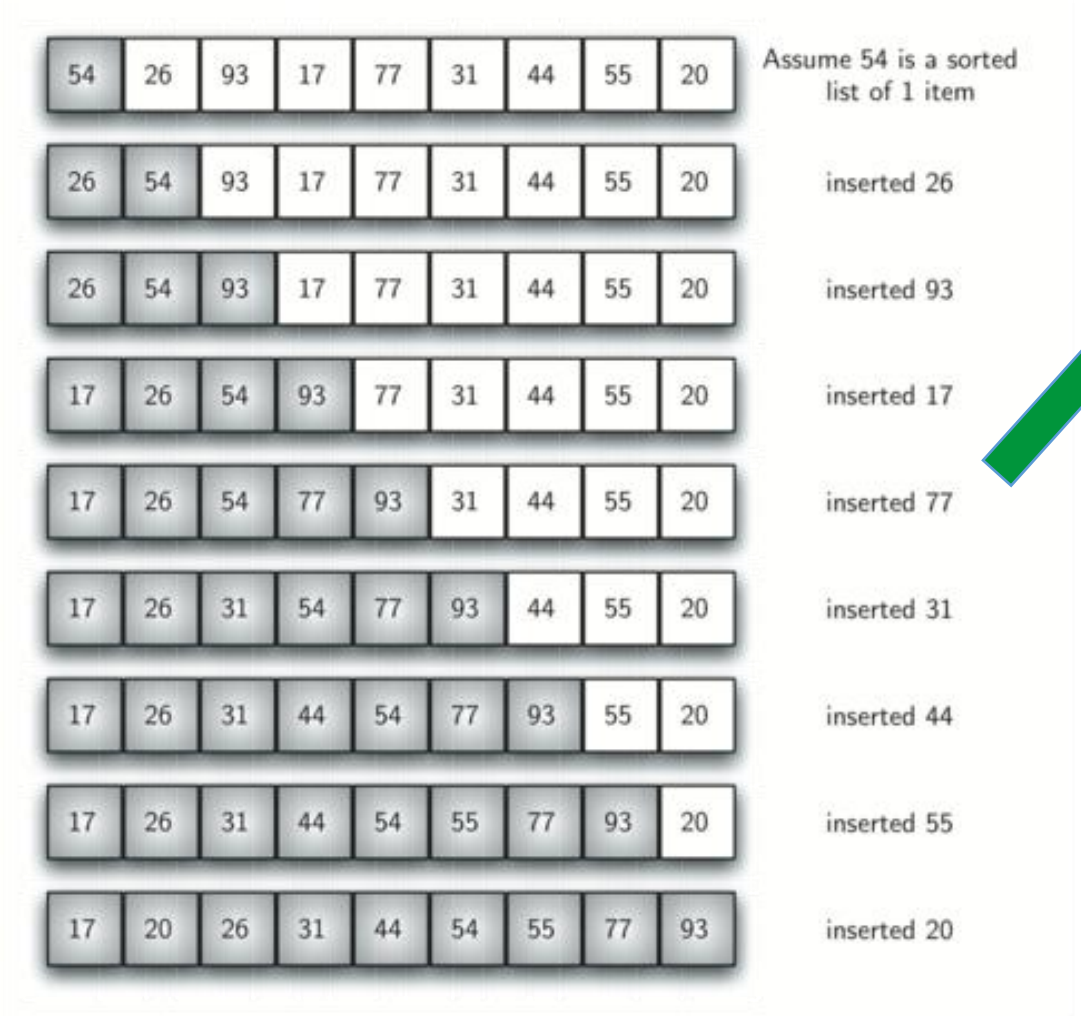
- Etapas:
    1. Considere o primeiro elemento da coleção como uma subsequência já ordenada.
    2. Selecione o próximo elemento da parte não ordenada.
    3. Compare esse elemento com os elementos da subsequência ordenada, da direita para a esquerda.
    4. Desloque para a direita os elementos maiores que o elemento selecionado.
    5. Insira o elemento na posição correta.
    6. Repita os passos 2 a 5 até que todos os elementos tenham sido inseridos na subsequência ordenada.

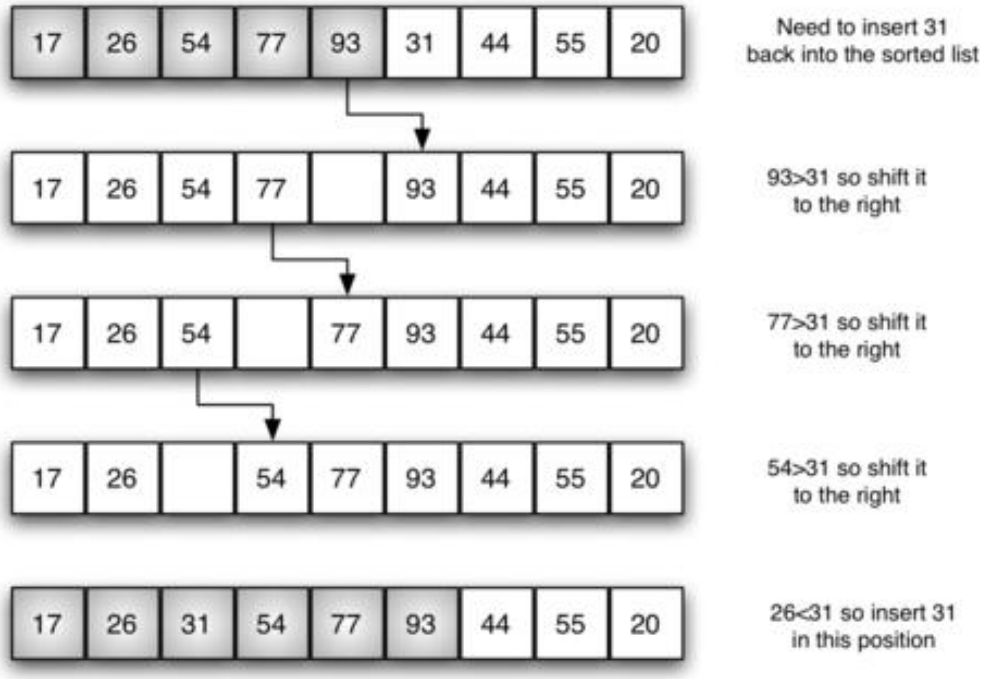
Insertion sort



Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/AOrdenacaoPorInsercao.html

Insertion sort





Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/AOrdenacaoPorInsercao.html

Insertion sort

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

INSTITUTO FEDERAL
Pernambuco

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	8	1	2	7	3

tamanho = 7

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 1
chave = 10
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 1
chave = 10
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 1
chave = 10
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 2
chave = 8
j = 1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	8	1	2	7	3

tamanho = 7
i = 2
chave = 8
j = 1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	10	1	2	7	3

tamanho = 7
i = 2
chave = 8
j = 1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	10	1	2	7	3

tamanho = 7
i = 2
chave = 8
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	10	10	1	2	7	3

tamanho = 7
i = 2
chave = 8
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	10	1	2	7	3

tamanho = 7
i = 2
chave = 8
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	10	1	2	7	3

tamanho = 7
i = 3
chave = 1
j = 2

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	10	1	2	7	3

tamanho = 7
i = 3
chave = 1
j = 2

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	10	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 2

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	10	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	10	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	8	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	8	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	8	8	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	5	8	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = 0

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	5	8	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = -1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

5	5	8	10	2	7	3

tamanho = 7
i = 3
chave = 1
j = -1 ✖

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

tamanho = 7
i = 3
chave = 1
j = -1

Insertion sort

INSTITUTO FEDERAL
Pernambuco

```
void insertionSort(int vetor[], int tamanho) {

    for (int i = 1; i < tamanho; i++) {

        int chave = vetor[i];
        int j = i - 1;

        while (j >= 0 && vetor[j] > chave) {
            vetor[j + 1] = vetor[j];
            j--;
        }

        vetor[j + 1] = chave;
    }
}
```

1	5	8	10	2	7	3

tamanho = 7
i = 4
chave = 2
j = 3

Continua até a última passagem!!!

Insertion sort

INSTITUTO FEDERAL
Pernambuco

- Vantagens:
  - Fácil de compreender e implementar.
  - Eficiente para coleções pequenas: Apresenta bom desempenho quando o número de elementos é reduzido.
  - Adaptativo: aproveita coleções já ordenadas ou quase ordenadas.
  - Baixo uso de memória: Realiza a ordenação no próprio vetor.
- Desvantagens:
  - Muitas movimentações: Em vetores muito desordenados, pode ser necessário deslocar vários elementos para inserir cada novo item na posição correta.
  - Complexidade quadrática: Possui complexidade $O(n^2)$.

Merge sort

INSTITUTO FEDERAL
Pernambuco

- O Merge Sort é um algoritmo de ordenação baseado na estratégia de divisão e conquista (divide and conquer).
  - O Merge Sort divide o problema em subproblemas menores, resolve cada um deles e depois combina os resultados para obter a solução final.
- Ele funciona dividindo repetidamente a coleção em partes menores até que cada parte contenha apenas um elemento. Em seguida, essas partes são combinadas (merge) de forma ordenada, produzindo gradualmente uma coleção totalmente ordenada.
- O Merge Sort é conhecido por sua eficiência em grandes conjuntos de dados e possui uma complexidade de tempo médio e pior caso de O(n*logn), onde n representa o número de elementos a serem ordenados.

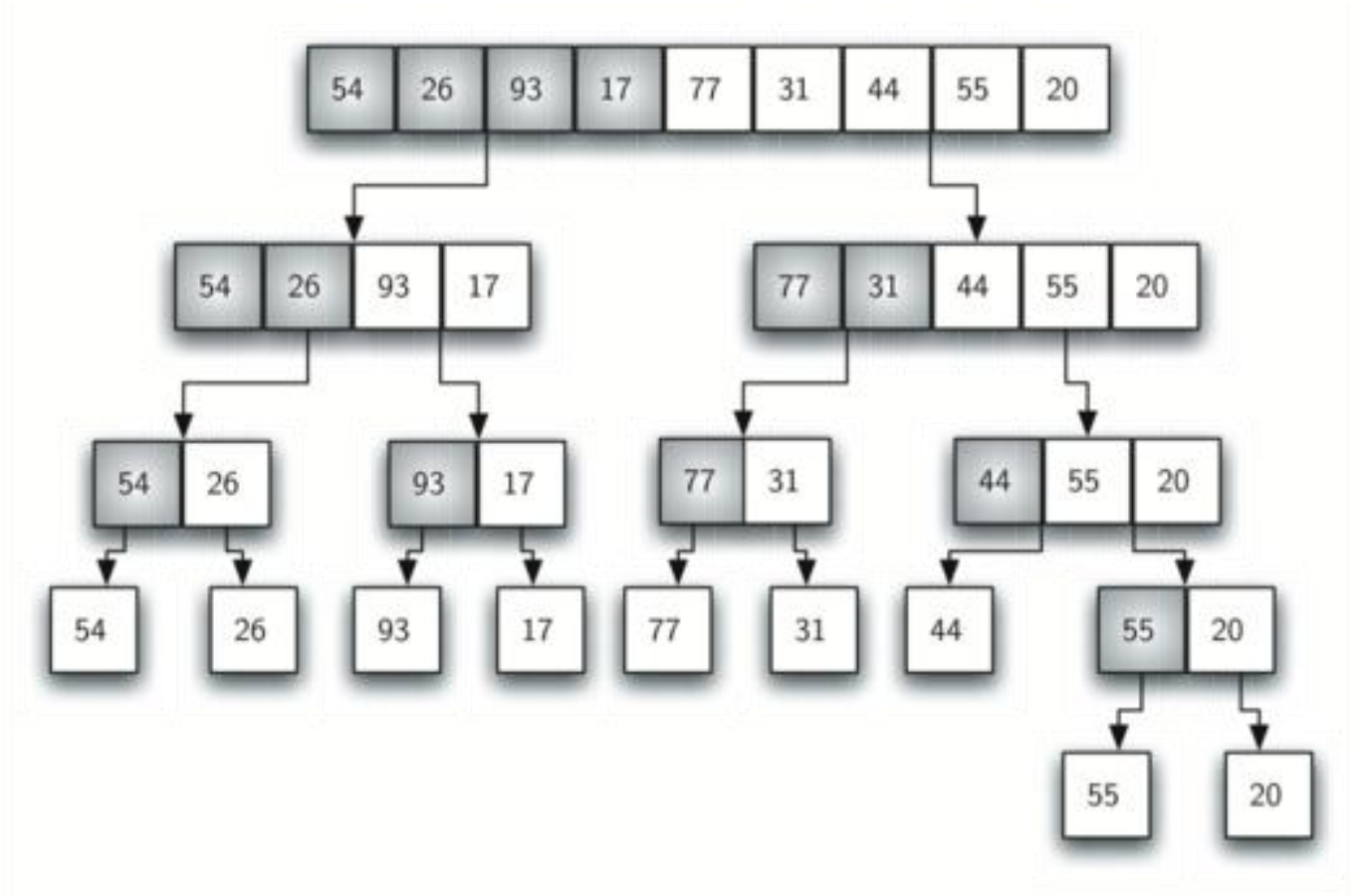
Merge sort

INSTITUTO FEDERAL
Pernambuco

- Etapas:
    1. Divida a coleção em duas metades aproximadamente iguais.
    2. Repita a divisão recursivamente para cada metade até que cada subcoleção contenha apenas um elemento.
    3. Intercale as subcoleções, comparando seus elementos e formando uma nova sequência ordenada.
    4. Copie os elementos ordenados para a coleção original.
    5. Repita a intercalação das subcoleções já ordenadas até obter uma única coleção completamente ordenada.

Merge sort

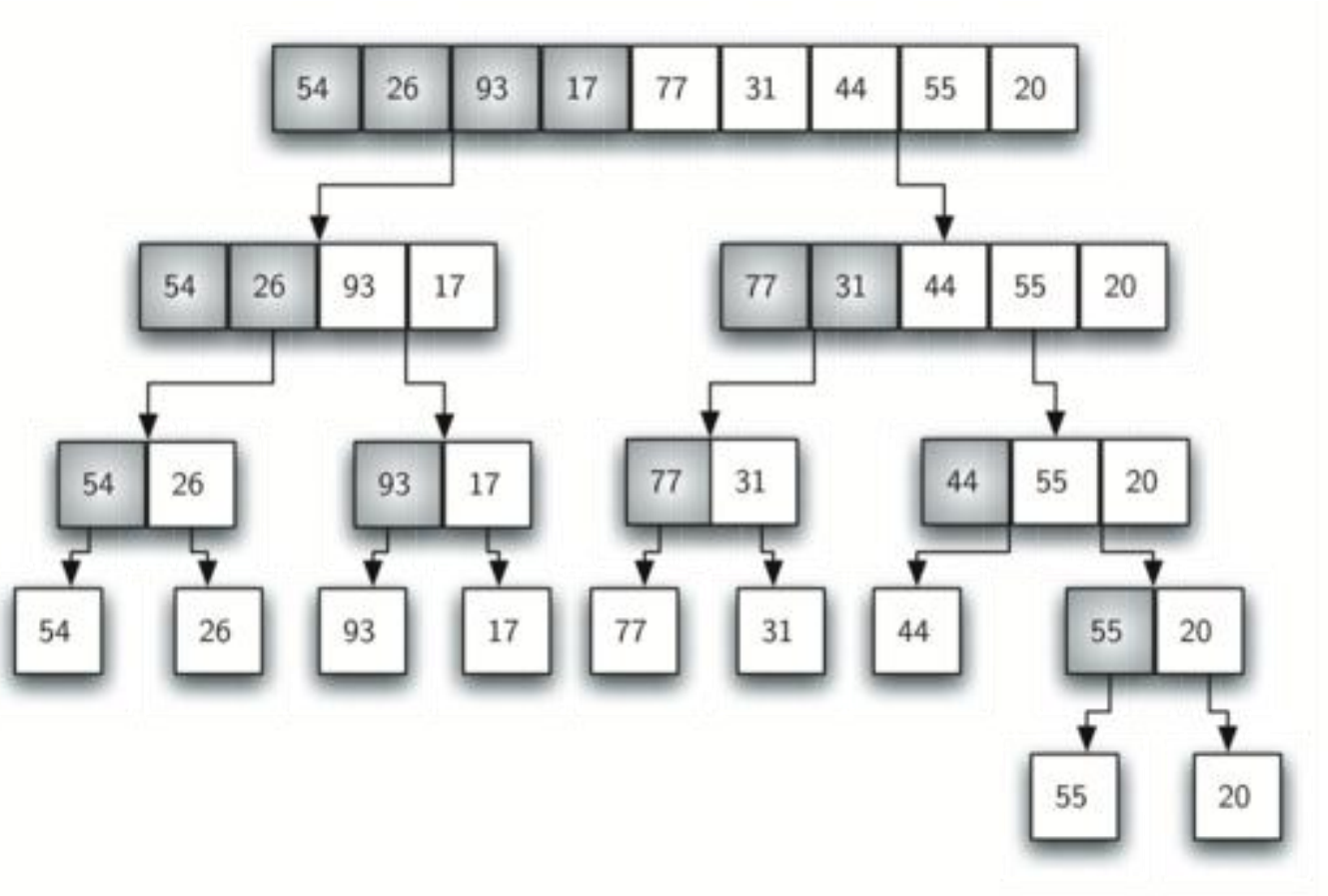
INSTITUTO FEDERAL
Pernambuco

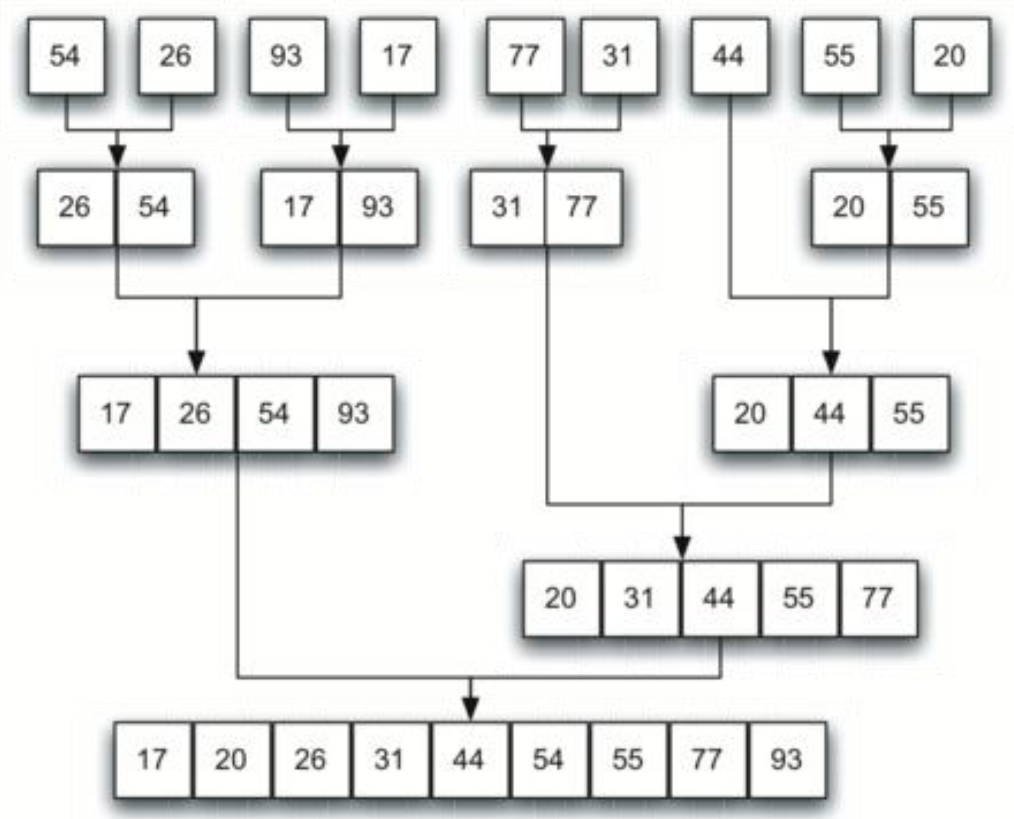


Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OMergeSort.html

Merge sort

INSTITUTO FEDERAL
Pernambuco





Fonte da imagem: https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OMergeSort.html

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

Merge sort

```c
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];

    // Copia os dados para os vetores auxiliares

    int i = 0;
    int j = 0;
    int k = inicio;

    // Intercala os vetores auxiliares
    while (i < n1 && j < n2) {

        if (esquerda[i] <= direita[j]) {
            vetor[k] = esquerda[i];
            i++;
        } else {
            vetor[k] = direita[j];
            j++;
        }

        k++;
    }

    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

INSTITUTO FEDERAL
Pernambuco

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

Merge sort


```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

5	10	8	1	2	7	3

inicio = 0
fim = 6

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

5	10	8	1	2	7	3

inicio = 0
fim = 6

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

5	10	8	1	2	7	3

inicio = 0
fim = 6
meio = 3

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

5	10	8	1	2	7	3

inicio = 0
fim = 6
meio = 3
mergesort({5,10,8,1,2,7,3}, 0, 3)

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

5	10	8	1	2	7	3

inicio = 0
fim = 6
meio = 3
mergesort({5,10,8,1,2,7,3}, 0, 3)

As chamadas recursivas do Merge Sort são executadas em profundidade (depth-first), seguindo a ordem imposta pela pilha de chamadas (call stack).

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

1	5	8	10	2	7	3

inicio = 0
fim = 6
meio = 3
mergesort({5,10,8,1,2,7,3}, 0, 3)
mergesort({1,5,8,10,2,7,3}, 4, 6)

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

1	5	8	10	2	7	3

inicio = 0
fim = 6
meio = 3
mergesort({5,10,8,1,2,7,3}, 0, 3)
mergesort({1,5,8,10,2,7,3}, 4, 6)

No momento em que a segunda chamada recursiva (mergeSort da metade direita) é executada, a metade esquerda daquela divisão já foi totalmente ordenada.

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

1	5	8	10	2	3	7

inicio = 0
fim = 6
meio = 3
mergesort({5,10,8,1,2,7,3}, 0, 3)
mergesort({1,5,8,10,2,7,3}, 4, 6)
merge({1,5,8,10,2,3,7}, 0, 3, 6)

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void mergeSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int meio = (inicio + fim) / 2;

        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);

        merge(vetor, inicio, meio, fim);
    }
}
```

1	5	8	10	2	3	7

inicio = 0
fim = 6
meio = 3
mergesort({5,10,8,1,2,7,3}, 0, 3)
mergesort({1,5,8,10,2,7,3}, 4, 6)
merge({1,5,8,10,2,3,7}, 0, 3, 6)

No momento em que merge() é executada, as metades esquerda e direita já foram completamente processadas e ordenadas pelas chamadas recursivas de mergeSort().

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] = | | | | |

int direita[3] = | | | |

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] = | | | | |

int direita[3] = | | | |

i = 0

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1			

int direita[3] =

i = 0

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] = | 1 | | | |

int direita[3] = | | | |

i = 1

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5		

int direita[3] =

i = 1

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

Continua até preencher esquerda[4]!!!

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
void merge(int vetor[], int inicio, int meio, int fim) {

    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

    int esquerda[n1];
    int direita[n2];

    // Copia os dados para os vetores auxiliares
    for (int i = 0; i < n1; i++)
        esquerda[i] = vetor[inicio + i];

    for (int j = 0; j < n2; j++)
        direita[j] = vetor[meio + 1 + j];
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 0
j = 0
k = 0

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 0
j = 0
k = 0

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 0
j = 0
k = 0

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 0
j = 0
k = 0

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 0
k = 0

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 0
k = 1

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 0
k = 1

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	5	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 0
k = 1

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 0
k = 1

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 1
k = 1

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 1
k = 2

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 1
k = 2

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	8	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 1
k = 2

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 1
k = 2

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 2
k = 2

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 2
k = 3

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 2
k = 3

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	10	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 2
k = 3

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 1
j = 2
k = 3

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 2
k = 3

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 2
k = 4

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 2
k = 4

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	2	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 2
k = 4

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	7	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 2
k = 4

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	7	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 3
k = 4

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	7	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 3
k = 5

Merge sort

```
int i = 0;
int j = 0;
int k = inicio;

// Intercala os vetores auxiliares
while (i < n1 && j < n2) {

    if (esquerda[i] <= direita[j]) {
        vetor[k] = esquerda[i];
        i++;
    } else {
        vetor[k] = direita[j];
        j++;
    }

    k++;
}
```

1	2	3	5	7	3	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 3 ✖
k = 5

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

1	2	3	5	7	8	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 2
j = 3
k = 5

Merge sort

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

1	2	3	5	7	8	7

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 3
j = 3
k = 6

Merge sort

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

1	2	3	5	7	8	10

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 3
j = 3
k = 6

Merge sort

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

1	2	3	5	7	8	10

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 4
j = 3
k = 7

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

1	2	3	5	7	8	10

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 4 ✖
j = 3
k = 7

Merge sort

INSTITUTO FEDERAL
Pernambuco

```
    // Copia os elementos restantes da esquerda
    while (i < n1) {
        vetor[k] = esquerda[i];
        i++;
        k++;
    }

    // Copia os elementos restantes da direita
    while (j < n2) {
        vetor[k] = direita[j];
        j++;
        k++;
    }
}
```

1	2	3	5	7	8	10

inicio = 0
meio = 3
fim = 6

n1 = 4
n2 = 3

int esquerda[4] =

1	5	8	10

int direita[3] =

2	3	7

i = 4
j = 3 ✖
k = 7

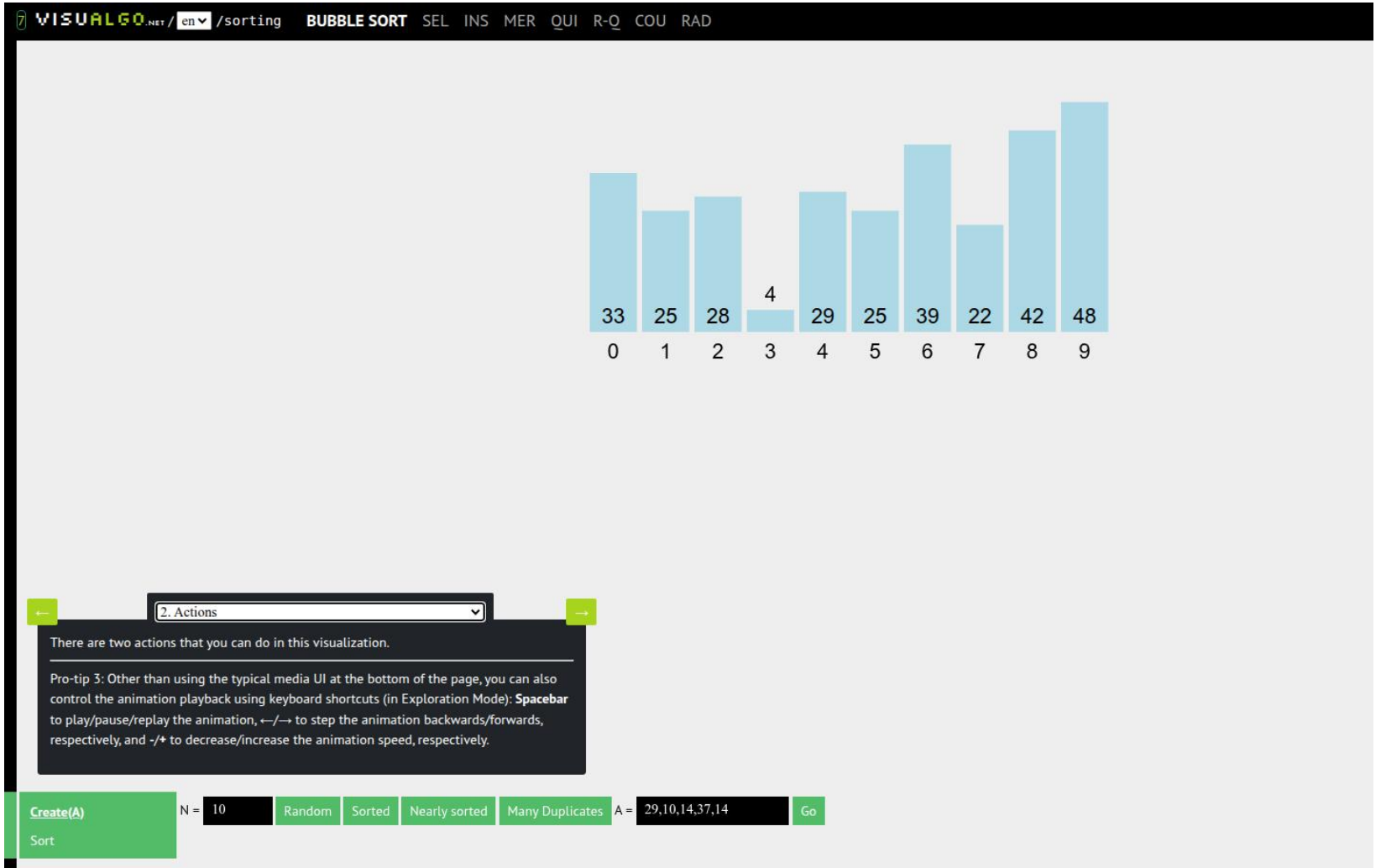
Merge sort

INSTITUTO FEDERAL
Pernambuco

- Vantagens:
  - Alta eficiência: Possui complexidade O(n log n).
  - Desempenho previsível: Seu tempo de execução não depende da disposição inicial dos dados.
  - Adequado para grandes coleções.
  - Eficiente para listas encadeadas.
- Desvantagens:
  - Uso de memória adicional: Requer estruturas auxiliares para realizar a intercalação das sublistas, consumindo O(n) de memória extra.
  - Implementação mais complexa.
  - Sobrecarga da recursão: As chamadas recursivas podem aumentar o consumo de memória.

Animações

- https://visualgo.net/en/sorting



Exercícios

INSTITUTO FEDERAL
Pernambuco

1. Os algoritmos de ordenação estudados (Bubble Sort, Selection Sort, Insertion Sort e Merge Sort) podem ser aplicados não apenas a vetores de inteiros, mas também a vetores de caracteres. Em C, cada caractere é armazenado internamente como um valor numérico correspondente à tabela ASCII, permitindo comparações utilizando os operadores relacionais (<, >, <=, >=).

   a) Modifique os algoritmos Bubble Sort, Selection Sort, Insertion Sort e Merge Sort para receber um vetor de caracteres e ordenar em ordem crescente.

   b) Modifique os algoritmos Bubble Sort, Selection Sort, Insertion Sort e Merge Sort para receber um vetor de caracteres e ordenar em ordem decrescente.

Dúvidas

INSTITUTO FEDERAL
Pernambuco

ALGORITMOS E ESTRUTURA DE DADOS

Curso de Engenharia de Software
Lucas Sampaio Leite