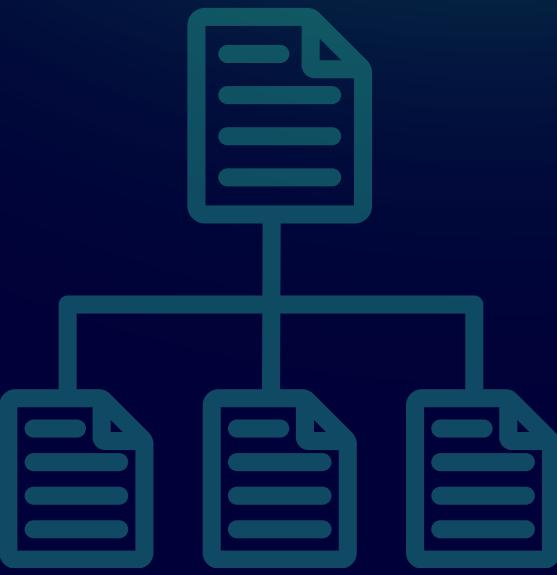


Estrutura de Dados



Método de ordenação

Radix Sort



Introdução ao Radix Sort

1

DEFINIÇÃO

- Criado em 1954;
- Harold Hollerit Seward;
- Surgiu da necessidade de gerenciar os cartões perfurados

Introdução ao Radix Sort

1

DEFINIÇÃO

É um algoritmo de ordenação que organiza os números inteiros e processa-os dígito a dígito, ou seja, individualmente, um por vez. Ele divide os números em suas partes e os ordena com base nessas partes, começando pelo dígito menos significativo, do menor para o maior dígito, ou seja o **LSD** ou Least Significant Digit e o **MSD** ou Most Significant Digit.

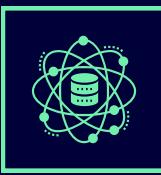
Introdução ao Radix Sort

1

DEFINIÇÃO

Ele usa frequentemente um algoritmo de ordenação estável como o *counting sort* em cada passagem para ordenar os números de acordo com o dígito atual.

Ele é eficiente para grande volumes de dados, principalmente quando o número de dígitos é menor que a quantidade total de números a serem ordenados. E seu tempo de execução é de $O(n.k)$



Considerando esse input:

170

45

75

90

802

24

2

66



A organizando começa com os últimos números:

170

45

75

90

802

24

2

66



Ao selecionar os últimos números, iremos organizar por ordem crescente a partir dos últimos números:

170

45

75

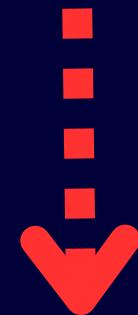
90

802

24

2

66



170

90

802

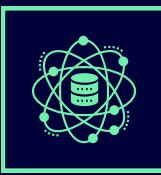
2

24

45

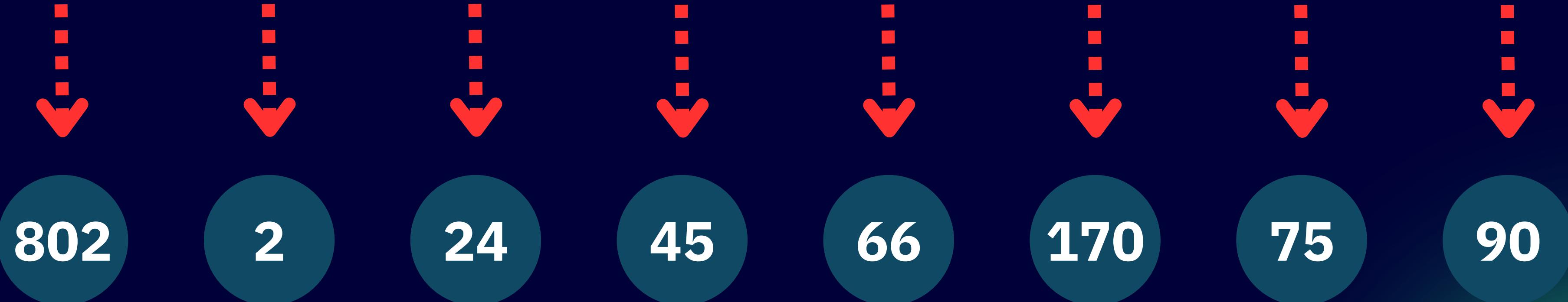
75

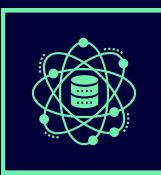
66



Agora, organizando os números do meio:

170 **90** **802** **12** **24** **45** **75** **66**





Por fim, organizando os primeiros números:

802

802

2

2

24

24

45

45

66

66

170

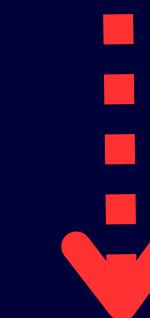
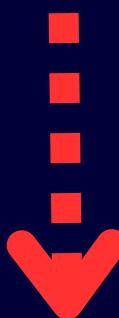
170

75

75

90

90



2

24

45

66

75

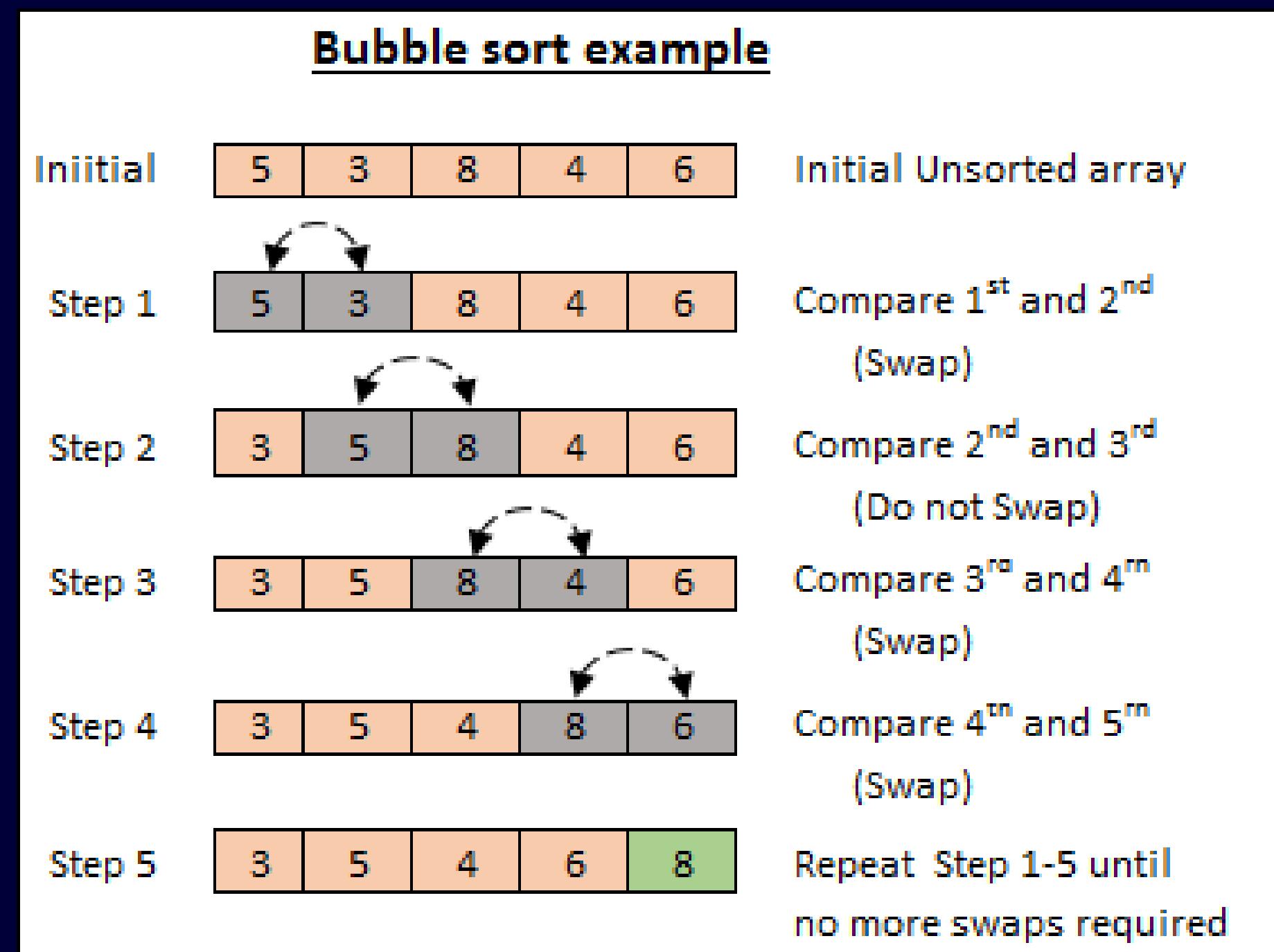
90

170

802

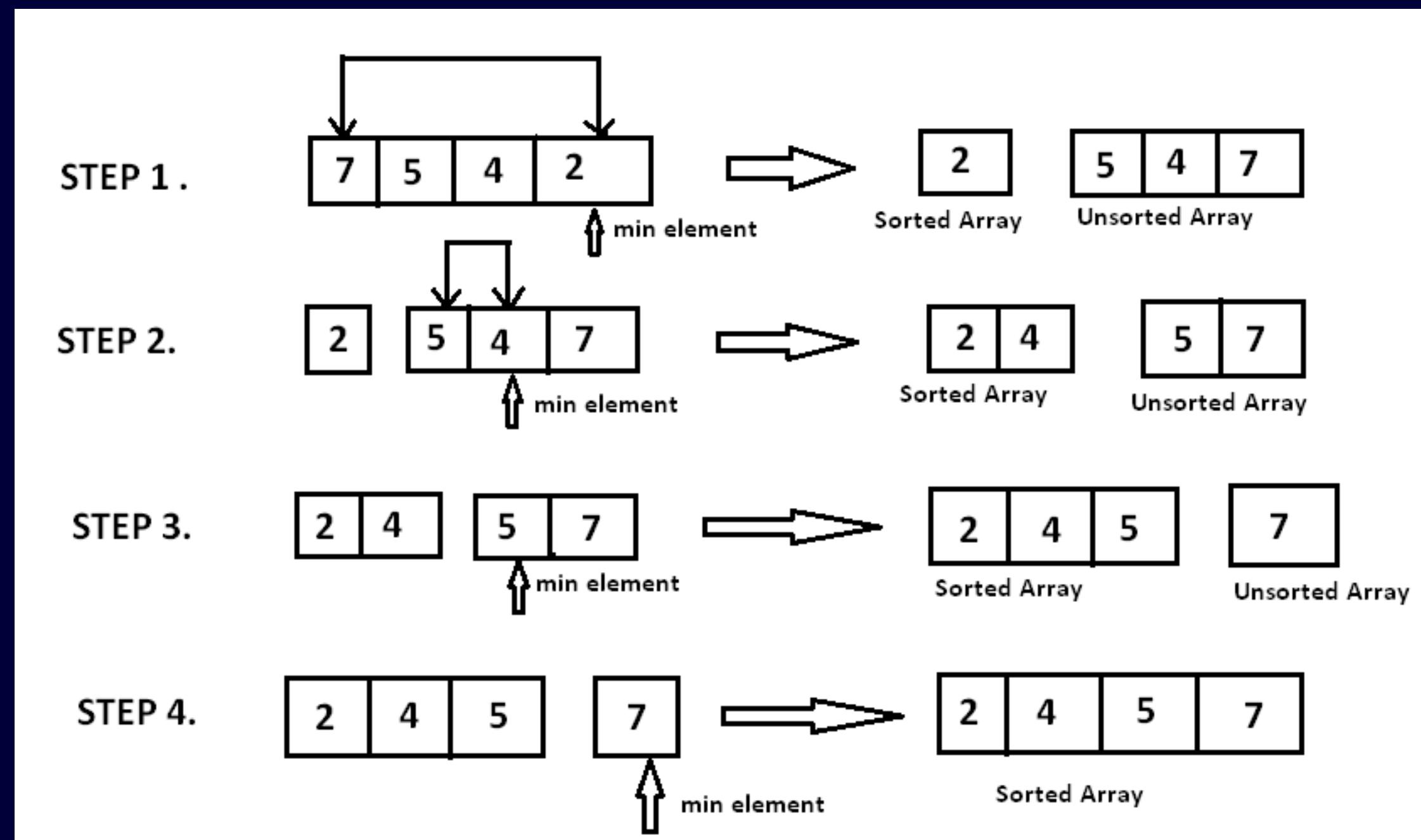
Classificação

Bolha (Bubble Sort) -> Simples, mas não eficiente. Compara elementos adjacentes e os troca se estiverem na ordem errada.



Classificação

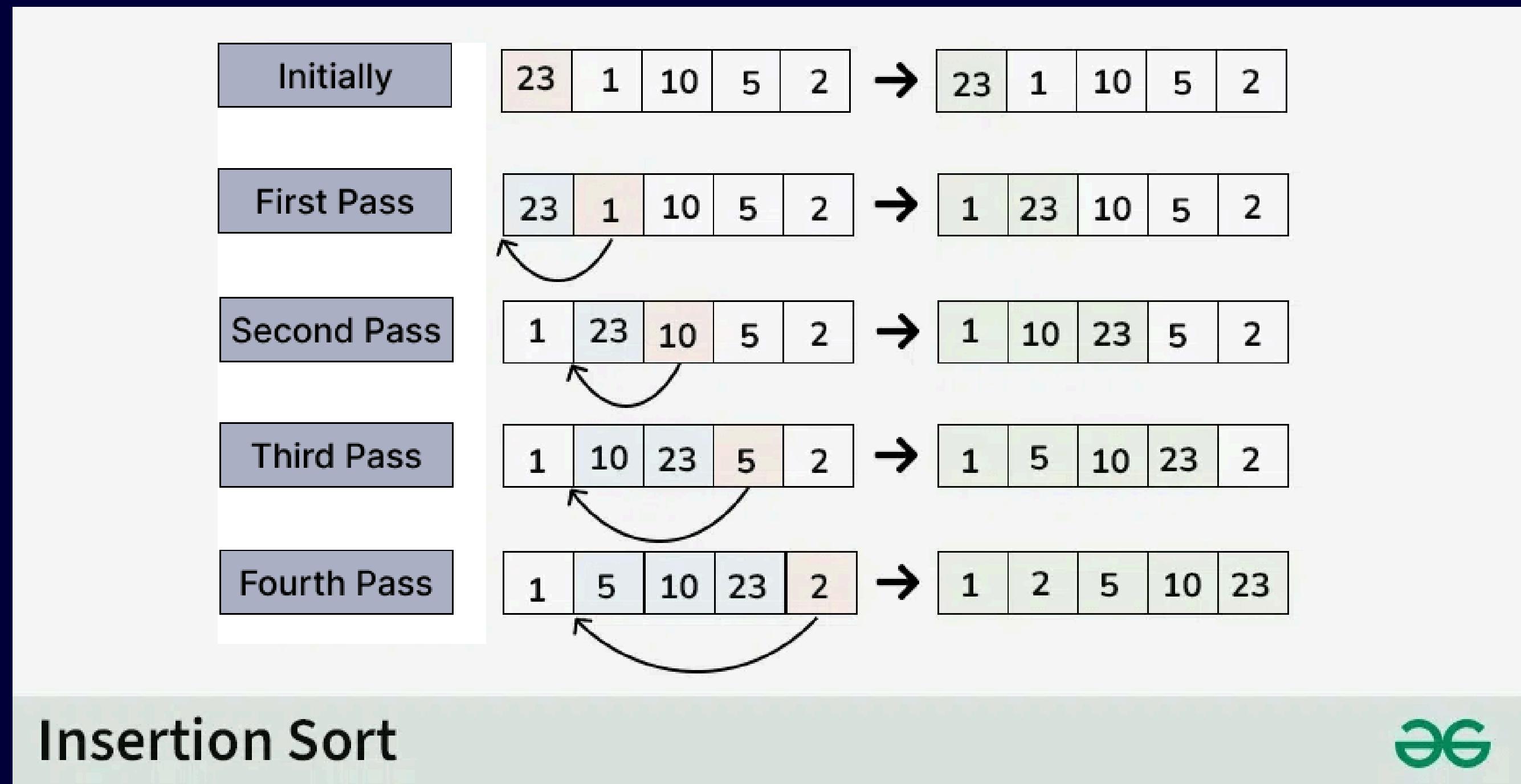
Seleção (Selection Sort) -> Encontra o menor elemento e o coloca na posição correta. Repete o processo para os próximos elementos.



Classificação



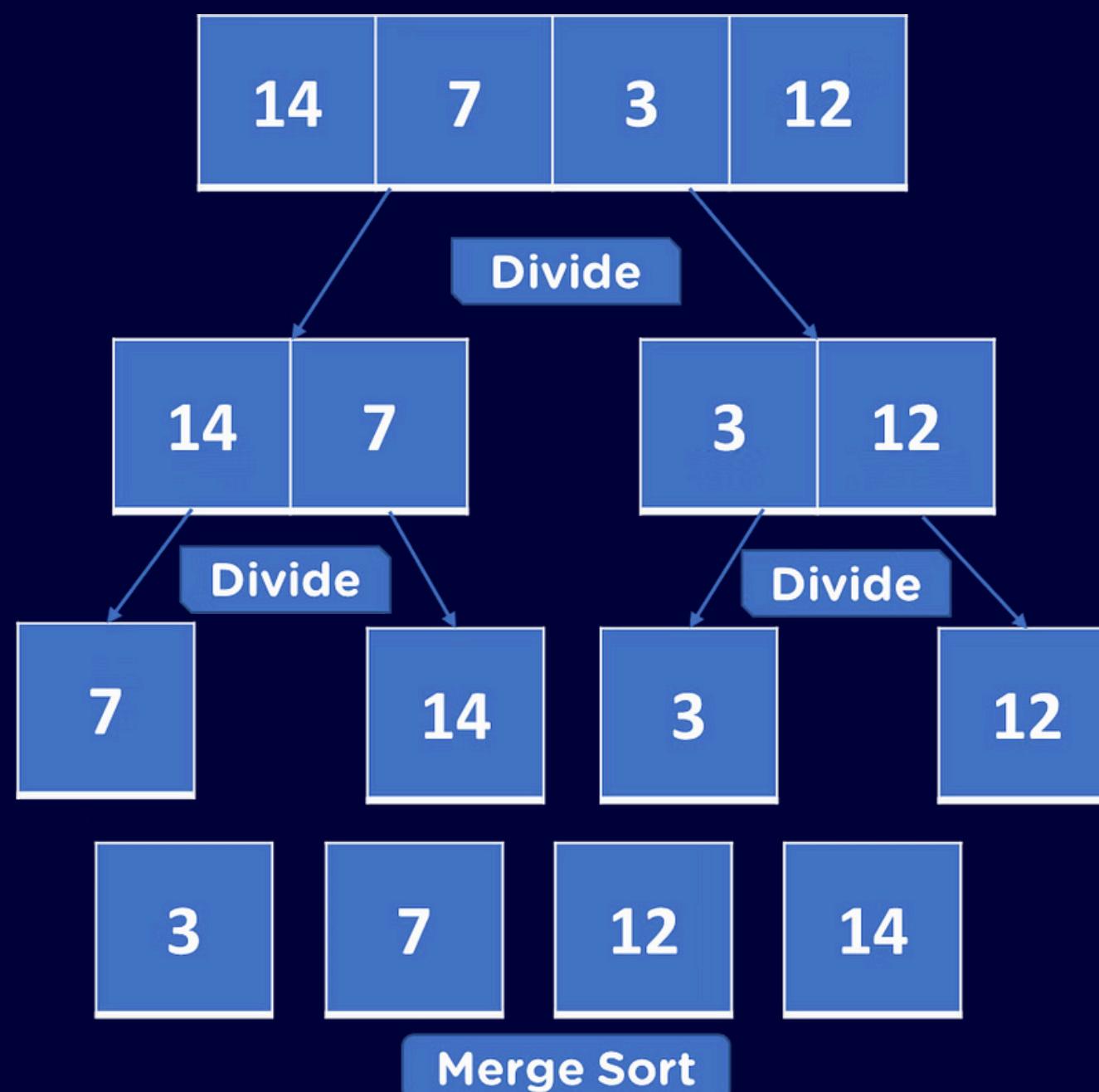
Inserção (Insertion Sort) -> Constrói a lista ordenada inserindo elementos em suas posições corretas uma por uma.



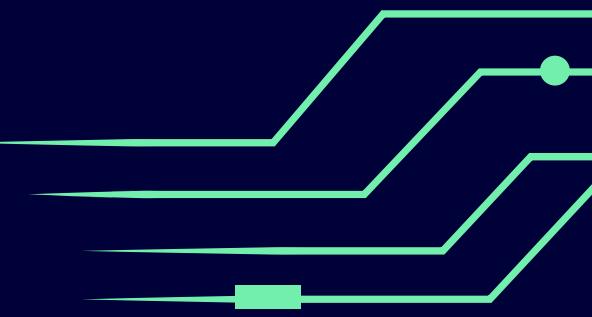


Classificação

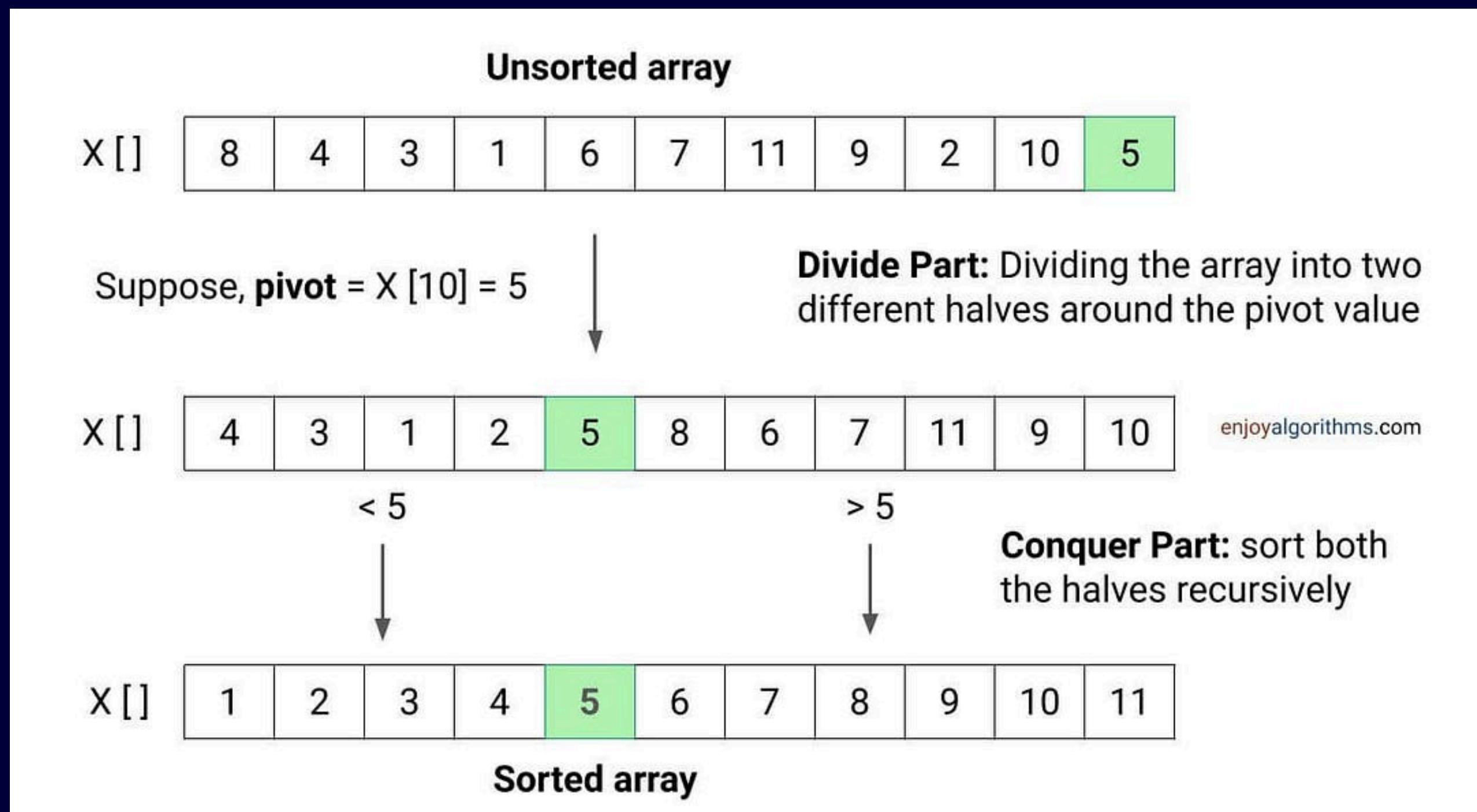
Merge Sort -> Divide a lista em duas , ordena cada metade e depois combina as duas metades ordenadas. Usa a técnica “dividir e conquistar”.



Classificação

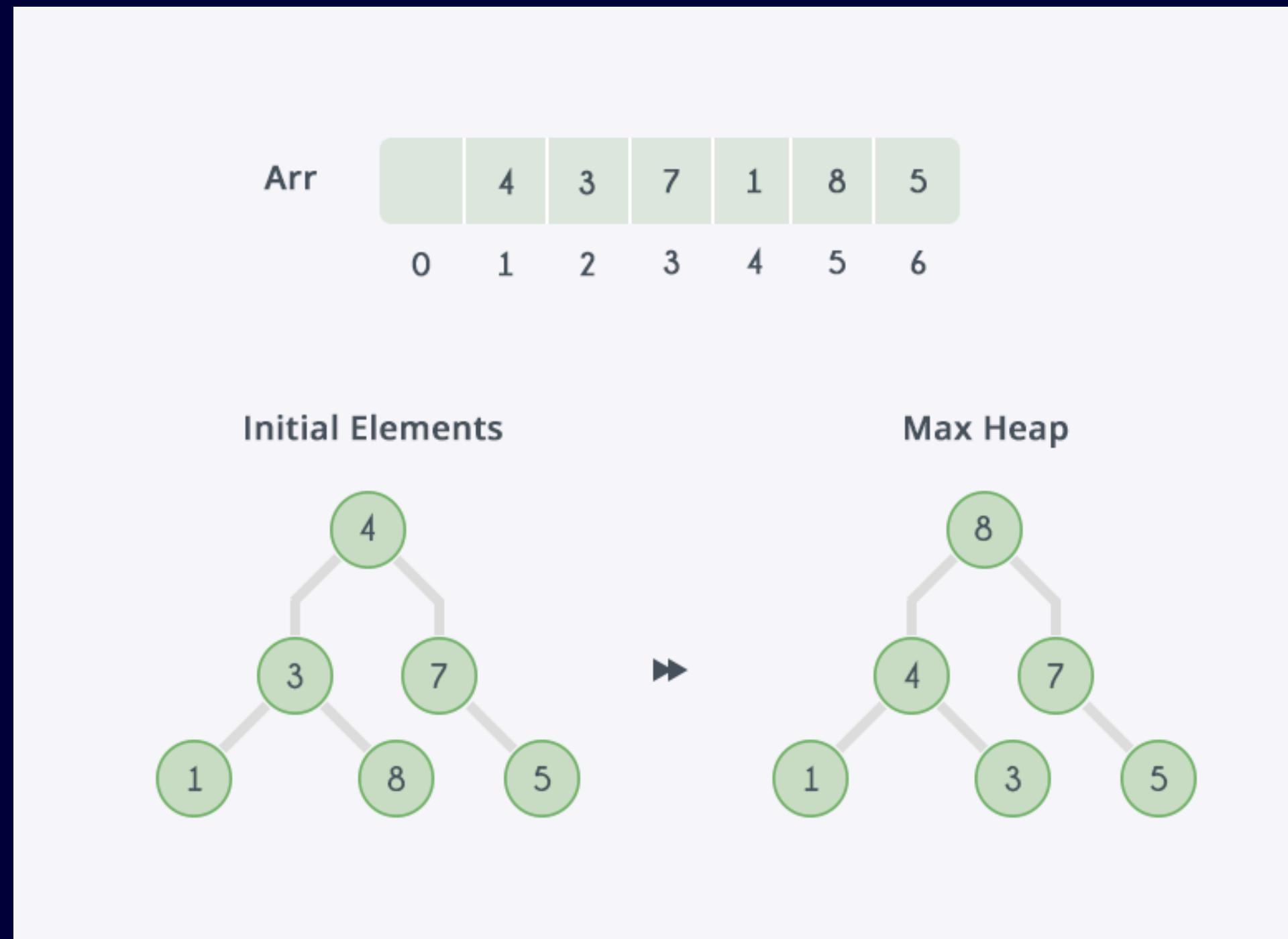


Quick Sort -> Escolhe um “pivô”, reorganiza os elementos em torno do pivô e depois aplica o mesmo processo às sublistas geradas. É rápido na prática.



Classificação

Heap Sort -> Constrói uma árvore de heap e então extrai repetidamente o maior elemento para obter a lista ordenada.

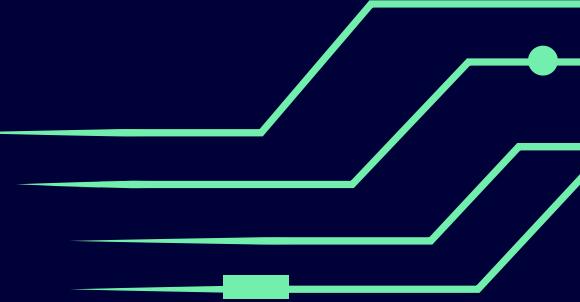


Classificação

Radix Sort -> Ordena os números inteiros digit by digit, começando pelo dígito menos significativo até o mais significativo.



Comparação de desempenho



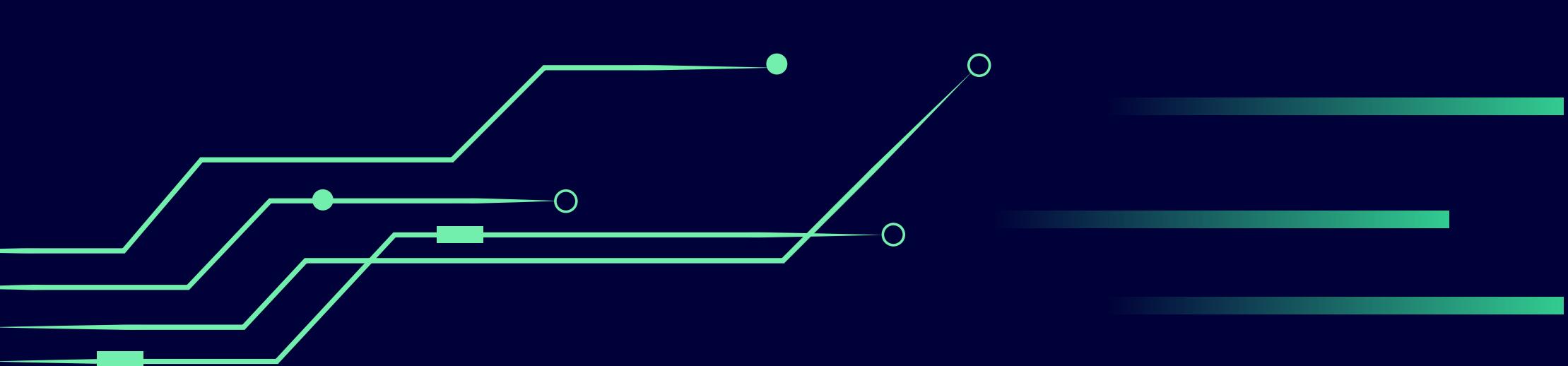
Análise do Big O do Radix Sort

Radix Sort é uma combinação de:

d = número máximo de dígitos nos números (ou caracteres, dependendo do contexto).

n = número de elementos a serem ordenados.

Ele faz uso de um algoritmo de ordenação estável, geralmente Counting Sort, para ordenar os elementos com base nos dígitos.



Comparação de desempenho

Agora, vamos ver o Big O:

Counting Sort tem um custo de $O(n + k)$, onde:

n = É o número de elementos (como dito no slide anterior).

k = É o intervalo dos números que você está ordenando (para Radix Sort, isso seria **10**, já que estamos lidando com dígitos de **0** a **9**).

O Radix Sort executa o Counting Sort d vezes, onde relembrando d é o número máximo de dígitos.

Portanto, o tempo total é:

$$O(d * (n + k)).$$



Comparação de desempenho

Em termos práticos:

Se o número de dígitos **d** é muito pequeno em relação ao número de elementos **n**, o **Radix Sort** pode ser bem eficiente. Já para entradas com muitos dígitos ou números muito grandes, o valor de **d** aumenta, mas ainda assim o crescimento da complexidade tende a ser linear para muitos casos práticos.

Isso faz o **Radix Sort** ter uma performance melhor que **O(n²)**, tornando-se mais eficiente que algoritmos de ordenação quadráticos como o **Bubble Sort** ou **Insertion Sort**.



Implementação Prática

```

#include <stdio.h>
#define TAM 10

void imprimirVet(int vet[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", vet[i]);
    printf("\n");
}

int obterMaximo(int vet[], int n) {
    int maximo = vet[0];
    for (int i = 1; i < n; i++)
        if (vet[i] > maximo)
            maximo = vet[i];
    return maximo;
}

void ordenacaoContagemSimples(int vet[], int n, int exp) {
    int saida[n];
    int i, contagem[TAM] = {0};

    for (i = 0; i < n; i++)
        contagem[(vet[i] / exp) % 10]++;
    for (i = 1; i < TAM; i++)
        contagem[i] += contagem[i - 1];

    for (i = n - 1; i ≥ 0; i--) {
        saida[contagem[(vet[i] / exp) % 10] - 1] = vet[i];
        contagem[(vet[i] / exp) % 10]--;
    }
    for (i = 0; i < n; i++)
        vet[i] = saida[i];
}

void radixSortSimples(int vet[], int n) {
    int maximo = obterMaximo(vet, n);

    for (int exp = 1; maximo / exp > 0; exp *= 10)
        ordenacaoContagemSimples(vet, n, exp);
}

int main() {
    int n;

    printf("Digite o número de elementos: ");
    scanf("%d", &n);

    int vet[n];

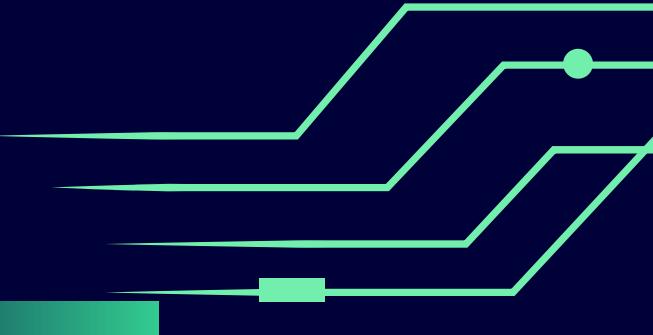
    printf("Digite os elementos do vet:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &vet[i]);
    }

    radixSortSimples(vet, n);
    printf("Vet ordenado:\n");
    imprimirVet(vet, n);

    return 0;
}

```

Questões sobre o Radix Sort



Questão

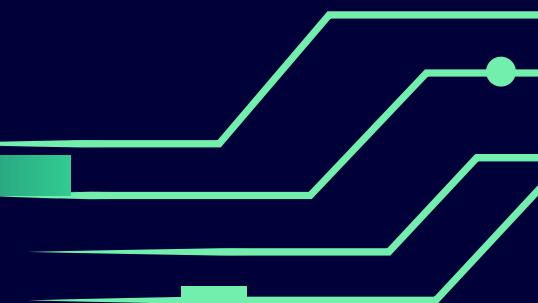
1:

O que torna o Radix Sort um algoritmo de ordenação não comparativo?

O Radix Sort é um algoritmo de ordenação não comparativo porque não realiza comparações diretas entre os elementos durante o processo de ordenação. Em vez disso, ele classifica os elementos com base nos seus dígitos individuais, processando esses dígitos de maneira sequencial, começando pelo dígito menos significativo.

Questão

2:

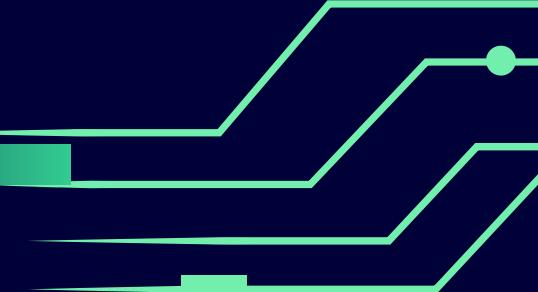


Explique o motivo pelo qual o Radix Sort geralmente é mais eficiente quando combinado com o Counting Sort.

O Counting Sort é frequentemente utilizado como sub-rotina dentro do Radix Sort para ordenar os dígitos individuais de forma eficiente. O Counting Sort tem uma complexidade linear, $O(n + k)$, onde n é o número de elementos a serem ordenados e k é o valor máximo que um dígito pode assumir.

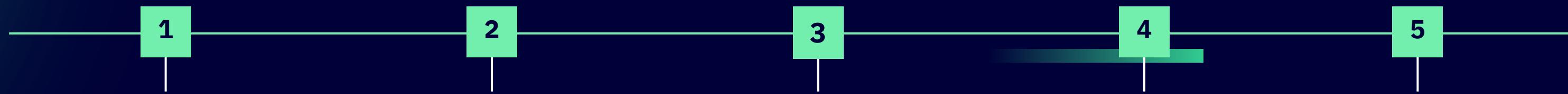
Questão

3:



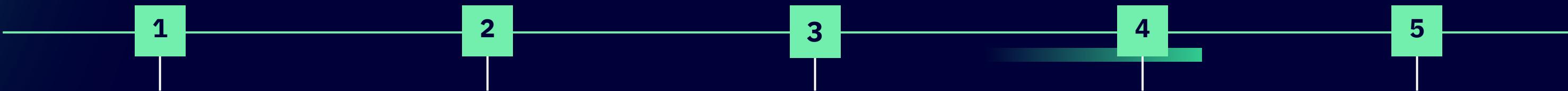
Por que o Radix Sort não é a escolha ideal para ordenar dados com muitos números negativos?

O Radix Sort é ineficiente para ordenar números negativos porque ele foi projetado para operar de maneira eficiente em números inteiros não negativos (positivos). A ordenação por dígitos no Radix Sort assume que todos os números podem ser tratados como inteiros positivos.



Qual das afirmações abaixo descreve corretamente o funcionamento do Radix Sort?

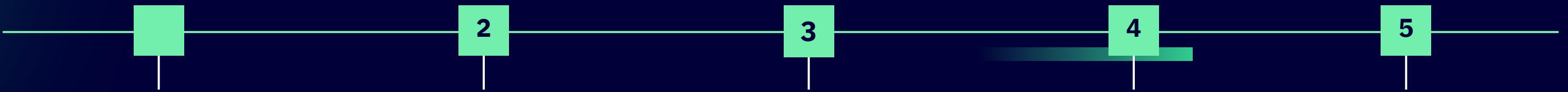
- a) O Radix Sort utiliza o algoritmo de divisão e conquista para ordenar os elementos.
- b) O Radix Sort ordena os elementos comparando diretamente os valores inteiros.
- c) O Radix Sort ordena os elementos processando cada dígito individualmente, da unidade até o dígito mais significativo.
- d) O Radix Sort funciona apenas em números binários.
- e) O Radix Sort é um algoritmo instável.



Qual das afirmações abaixo descreve corretamente o funcionamento do Radix Sort?

- c) O Radix Sort ordena os elementos processando cada dígito individualmente, da unidade até o dígito mais significativo.

O Radix Sort processa os elementos com base em seus dígitos, começando pelo dígito menos significativo (LSD) até o dígito mais significativo (MSD)



Qual é a complexidade de tempo no pior caso do Radix Sort?

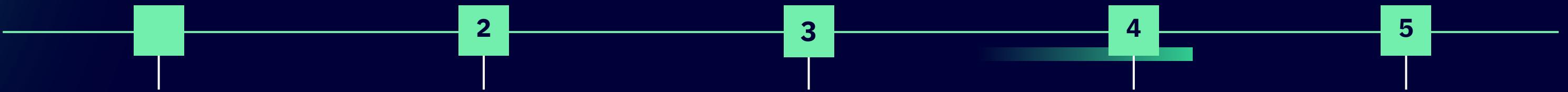
a) $O(n \log n)$

b) $O(n^2)$

c) $O(n \log n + k)$

d) $O(d*(n+k))$, onde d é o número de dígitos e k é o tamanho da base numérica.

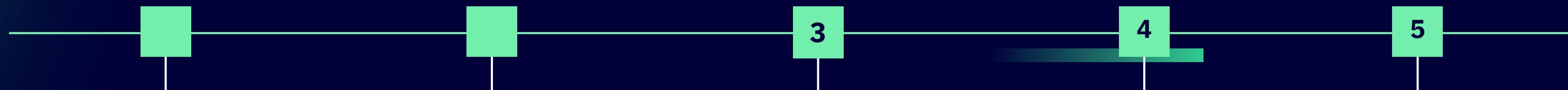
e) $O(n \log k)$



Qual é a complexidade de tempo no pior caso do Radix Sort?

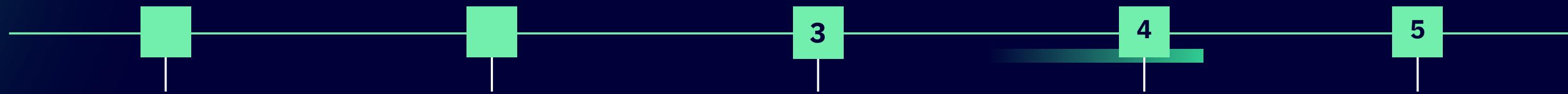
d) **$O(d*(n+k))$, onde d é o número de dígitos e k é o tamanho da base numérica.**

A complexidade de tempo do Radix Sort no pior caso é $O(d*(n+k))$, onde d é o número de dígitos e k é a base numérica.



Qual dos seguintes algoritmos é frequentemente utilizado como Subrotina (um bloco de código que pode ser chamado várias vezes para executar uma tarefa específica dentro de um programa) dentro do Radix Sort para realizar a ordenação dos dígitos?

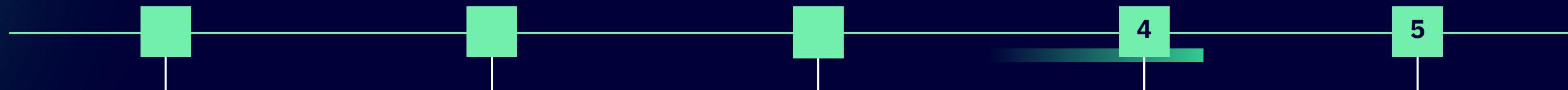
- a) Quick Sort
- b) Bubble Sort
- c) Counting Sort
- d) Merge Sort
- e) Selection Sort



Qual dos seguintes algoritmos é frequentemente utilizado como Sub rotina (um bloco de código que pode ser chamado várias vezes para executar uma tarefa específica dentro de um programa) dentro do Radix Sort para realizar a ordenação dos dígitos?

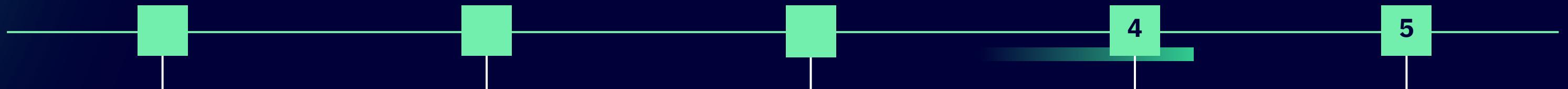
c) Counting Sort

O Counting Sort é frequentemente utilizado como sub rotina dentro do Radix Sort para ordenar os dígitos de forma eficiente



O Radix Sort é adequado para ordenar qual tipo de dados?

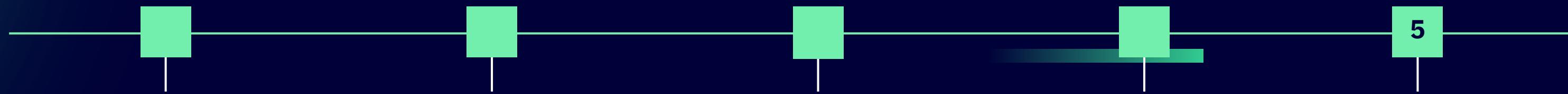
- a) Números inteiros de tamanho fixo e strings de comprimento fixo.
- b) Números em ponto flutuante.
- c) Números negativos com muitos dígitos.
- d) Qualquer conjunto de dados, independentemente do tipo e tamanho.
- e) Dados que já estão parcialmente ordenados.



O Radix Sort é adequado para ordenar qual tipo de dados?

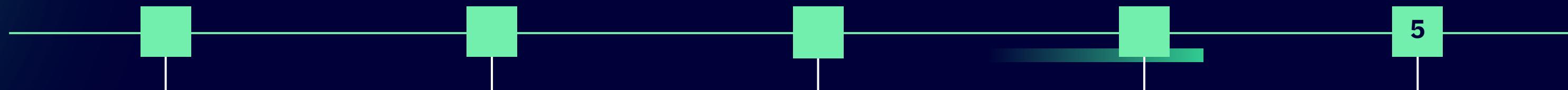
- a) Números inteiros de tamanho fixo e strings de comprimento fixo.

O Radix Sort é mais adequado para ordenar números inteiros de tamanho fixo e strings com comprimento fixo



O que ocorre se os dados a serem ordenados pelo Radix Sort contiverem muitos números negativos?

- a) O algoritmo Radix Sort ainda funcionará da mesma forma sem modificação
- b) O algoritmo Radix Sort será significantivamente mais eficiente com números negativos.
- c) O Radix Sort precisa ser modificado para lidar com números negativos, separando-os dos números positivos
- d) O Radix Sort não funciona com números negativos
- e) O algoritmo ignora os números negativos e ordena apenas os positivos.



O que ocorre se os dados a serem ordenados pelo Radix Sort contiverem muitos números negativos?

c) O Radix Sort precisa ser modificado para lidar com números negativos, separando-os dos números positivos

O Radix Sort precisa de modificações para lidar com números negativos, como separar os números negativos e positivos e ordená-los separadamente antes de combiná-los

