Quick Sort

Funcionamento:

O Quick Sort é um algoritmo de ordenação baseado na técnica de **divisão e conquista**. Ele seleciona um elemento como **pivô** e particiona o vetor em duas partes:

- Elementos menores que o pivô à esquerda;
- Elementos maiores que o pivô à direita.

Depois, aplica recursivamente o mesmo processo às duas subpartes. O desempenho do Quick Sort depende da escolha do pivô.

Notação Big O:

- Melhor caso: O(n log n)
- Caso médio: O(n log n)
- Pior caso: $O(n^2)$

Código em C:

```
c
CopiarEditar
#include <stdio.h>

void quickSort(int vetor[], int inicio, int fim) {
    if (inicio < fim) {
        int pivo = vetor[fim];
        int i = inicio - 1;

        for (int j = inicio; j < fim; j++) {
            if (vetor[j] < pivo) {
                i++;
                int temp = vetor[i];
                vetor[i] = vetor[j];
                vetor[j] = temp;
            }
        }

        int temp = vetor[i + 1];
        vetor[i + 1] = vetor[fim];
        vetor[fim] = temp;

        int indicePivo = i + 1;
```

```
        quickSort(vetor, inicio, indicePivo - 1);
        quickSort(vetor, indicePivo + 1, fim);
    }
}

int main() {
    int vetor[6] = {42, 17, 5, 23, 9, 31};
    quickSort(vetor, 0, 5);

    for (int i = 0; i < 6; i++) {
        printf("%d ", vetor[i]);
    }

    return 0;
}
```

Merge Sort

Funcionamento:

O Merge Sort também utiliza **divisão e conquista**. Ele divide o vetor em duas metades, ordena cada uma recursivamente e depois as **mescla** em ordem crescente. É eficiente e estável, ideal para grandes volumes de dados.

Notação Big O:

- Melhor caso: O(n log n)
- Caso médio: O(n log n)
- Pior caso: O(n log n)

Código em C:

```
c
CopiarEditar
#include <stdio.h>

void merge(int vetor[], int inicio, int meio, int fim) {
    int n1 = meio - inicio + 1;
    int n2 = fim - meio;

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

```
    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];

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

    while (i < n1 && j < n2) {
        if (esquerda[i] <= direita[j]) {
            vetor[k++] = esquerda[i++];
        } else {
            vetor[k++] = direita[j++];
        }
    }

    while (i < n1)
        vetor[k++] = esquerda[i++];

    while (j < n2)
        vetor[k++] = direita[j++];
}

void mergeSort(int vetor[], int inicio, int fim) {
    if (inicio < fim) {
        int meio = inicio + (fim - inicio) / 2;
        mergeSort(vetor, inicio, meio);
        mergeSort(vetor, meio + 1, fim);
        merge(vetor, inicio, meio, fim);
    }
}

int main() {
    int vetor[6] = {34, 7, 19, 2, 46, 13};
    mergeSort(vetor, 0, 5);

    for (int i = 0; i < 6; i++) {
        printf("%d ", vetor[i]);
    }

    return 0;
}
```

Shell Sort

Funcionamento:

O Shell Sort é uma melhoria do Insertion Sort. Ele começa comparando e ordenando elementos que estão distantes entre si. A distância (gap) entre elementos é reduzida a cada iteração até que o vetor esteja praticamente ordenado, e a ordenação final é feita com o Insertion Sort.

Notação Big O:

- Melhor caso: $O(n \log n)$
- Caso médio: $O(n (\log n)^2)$
- Pior caso: $O(n^2)$

Código em C:

```
c
CopiarEditar
#include <stdio.h>

void shellSort(int vetor[], int tamanho) {
    for (int gap = tamanho / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < tamanho; i++) {
            int temp = vetor[i];
            int j = i;
            while (j >= gap && vetor[j - gap] > temp) {
                vetor[j] = vetor[j - gap];
                j -= gap;
            }
            vetor[j] = temp;
        }
    }
}

int main() {
    int vetor[6] = {35, 12, 7, 28, 19, 3};
    shellSort(vetor, 6);

    for (int i = 0; i < 6; i++) {
        printf("%d ", vetor[i]);
    }

    return 0;
```

```
}
```

Heap Sort

Funcionamento:

O Heap Sort transforma o vetor em uma estrutura chamada **heap máximo**, onde o maior valor fica no topo (raiz). O maior elemento (raiz) é trocado com o último, e o heap é ajustado novamente. Esse processo se repete até o vetor estar completamente ordenado.

Notação Big O:

- Melhor caso: O(n log n)
- Caso médio: O(n log n)
- Pior caso: O(n log n)

Código em C:

```
c
CopiarEditar
#include <stdio.h>

void heapify(int vetor[], int tamanho, int i) {
    int maior = i;
    int esquerda = 2 * i + 1;
    int direita = 2 * i + 2;

    if (esquerda < tamanho && vetor[esquerda] > vetor[maior])
        maior = esquerda;

    if (direita < tamanho && vetor[direita] > vetor[maior])
        maior = direita;

    if (maior != i) {
        int temp = vetor[i];
        vetor[i] = vetor[maior];
        vetor[maior] = temp;
        heapify(vetor, tamanho, maior);
    }
}

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

```
    for (int i = tamanho / 2 - 1; i >= 0; i--)
        heapify(vetor, tamanho, i);

    for (int i = tamanho - 1; i > 0; i--) {
        int temp = vetor[0];
        vetor[0] = vetor[i];
        vetor[i] = temp;
        heapify(vetor, i, 0);
    }
}

int main() {
    int vetor[6] = {42, 15, 27, 8, 33, 6};
    heapSort(vetor, 6);

    for (int i = 0; i < 6; i++) {
        printf("%d ", vetor[i]);
    }

    return 0;
}
```