1-A notação Big O descreve um limite superior assintótico para o crescimento do tempo de execução.É uma ferramenta fundamental para comparar a eficiência de diferentes algoritmos e descrever o pior caso assintótico.

2-Pesquisa Sequencial:

- Funciona em qualquer lista, seja ela ordenada ou não.
- A complexidade de tempo é O(n), onde "n" é o tamanho da lista.
- É eficaz para listas pequenas ou quando não se conhece a ordem dos elementos.

Pesquisa Binária:

- Requer que a lista esteja ordenada.
- A complexidade de tempo é O(log n), onde "n" é o tamanho da lista.
- É muito eficaz para listas grandes, pois a cada passo a quantidade de elementos possíveis é reduzida pela metade.

3-Analisa o desempenho dos algoritmos em termos de tempo de execução e uso de recursos, como memória.

Complexidade de Tempo: A complexidade de tempo analisa quanto tempo um algoritmo leva para executar em relação ao tamanho da entrada.

Complexidades de tempo eficientes:

- O(1): Tempo constante. O tempo de execução não aumenta com o tamanho da entrada.
- O(log n): Tempo logarítmico. O tempo de execução aumenta de forma muito lenta à medida que o tamanho da entrada aumenta.
- O(n): Tempo linear. O tempo de execução aumenta de forma proporcional ao tamanho da entrada.
- O(n log n): Tempo quasilinear. Comum em algoritmos eficientes para problemas de ordenação e busca.

Complexidades de tempo não eficientes:

- O(n^2), O(n^3), … : Tempos polinomiais. O tempo de execução aumenta de forma significativa à medida que o tamanho da entrada aumenta.
- O(2^n), O(n!): Tempos exponenciais e fatoriais. Geralmente ineficientes para entradas maiores.

Complexidade de Espaço: A complexidade de espaço analisa quanto espaço de memória um algoritmo requer para executar em relação ao tamanho da entrada.

Complexidades de espaço eficientes:

- O(1): Uso constante de memória adicional.
- O(n): Uso linear de memória adicional, onde "n" é o tamanho da entrada.

Complexidades de espaço não eficientes:

- O(n^2), O(n^3), … : Uso polinomial de memória adicional.
- O(2^n), O(n!): Uso exponencial e fatorial de memória adicional.

4-

1. **Selection Sort (Ordenação por Seleção):**
    - Encontra o menor elemento na lista não ordenada.
    - Troca esse elemento com o primeiro elemento da lista não ordenada.
    - Repete o processo para a lista não ordenada restante, excluindo o primeiro elemento.
    - Continua até que todos os elementos estejam ordenados.

2. **Insertion Sort (Ordenação por Inserção):**
    - Inicialmente, a primeira posição é considerada ordenada.
    - Para cada elemento subsequente, move-o para a posição correta na parte ordenada da lista, empurrando os elementos maiores à frente.
    - Repete esse processo até que todos os elementos estejam na posição correta.

3. **Bubble Sort (Ordenação por Trocas):**
    - Compara pares de elementos adjacentes e os troca se estiverem fora de ordem.
    - Repete esse processo para toda a lista várias vezes, até que nenhum elemento precise ser trocado.
    - A cada passagem, o maior (ou menor, dependendo da ordem desejada) elemento "flutua" para a posição correta.

4. **Merge Sort (Ordenação por Fusão):**
    - Divide a lista não ordenada pela metade até que sejam obtidas sublistas de um único elemento.
    - Combina (funde) essas sublistas em ordem crescente, repetindo o processo até que toda a lista esteja ordenada.
    - A etapa de fusão envolve comparar e combinar as sublistas ordenadas para criar uma lista maior ordenada.

5. **Quick Sort (Ordenação Rápida):**
    - Escolhe um elemento pivô da lista.
    - Divide a lista em duas partes: elementos menores que o pivô e elementos maiores que o pivô.
    - Recursivamente, aplica o Quick Sort às duas sublistas.
    - Combina as sublistas ordenadas junto com o pivô para obter a lista final ordenada.

5-

1. Selection Sort:
    - Tempo:
        - Melhor caso: O(n^2)
        - Caso médio: O(n^2)
        - Pior caso: O(n^2)
    - Espaço: O(1)
2. Insertion Sort:

- Tempo:
    - Melhor caso: O(n)
    - Caso médio: O(n^2)
    - Pior caso: O(n^2)
- Espaço: O(1)

3. Bubble Sort:
    - Tempo:
        - Melhor caso: O(n)
        - Caso médio: O(n^2)
        - Pior caso: O(n^2)
    - Espaço: O(1)
4. Merge Sort:
    - Tempo:
        - Melhor caso: O(n log n)
        - Caso médio: O(n log n)
        - Pior caso: O(n log n)
    - Espaço: O(n)
5. Quick Sort:
    - Tempo:
        - Melhor caso: O(n log n)
        - Caso médio: O(n log n)
        - Pior caso: O(n^2)
    - Espaço: O(log n) - no caso médio