Lista de exercícios 4

Beatriz Gerogiannis Mendonça – 202200495

Questão 1:

A notação "Big O" é uma notação utilizada na análise de algoritmos e complexidade computacional para medir a eficiência ou o desempenho de um algoritmo conforme o tamanho dos dados de entrada aumenta.

Alguns exemplos comuns incluem:

O(1): Complexidade constante, ou seja, não depende do tamanho de entrada.

O(n): Complexidade linear, ou seja, cresce linearmente em relação ao tamanho de entrada.

Questão 2:

A pesquisa sequencial é mais simples de implementar e pode ser usada em qualquer lista, mas seu tempo de busca cresce linearmente com o tamanho da lista. Por outro lado, a pesquisa binária é altamente eficiente, mas exige que a lista esteja ordenada e divide a lista pela metade a cada passo, o que resulta em uma busca mais rápida, especialmente para listas grandes.

Questão 3:

A complexidade computacional analisa a eficiência de algoritmos e o uso de recursos à medida que o tamanho dos dados de entrada aumenta.

Complexidade eficiente: crescem de maneira razoável à medida que o tamanho da entrada aumenta. Por exemplo, O(log n) e O(n) são consideradas eficientes, pois o aumento no tempo de execução ou operações é relativamente controlado em comparação com o crescimento da entrada. Por exemplo, o Merge Sort (O(n log n)).

Complexidade ineficiente: crescem rapidamente com o aumento do tamanho da entrada. Por exemplo, O(n^2), O(2^n) e O(n!) são consideradas não eficientes, pois podem levar a tempos de execução impraticáveis para entradas grandes. Por exemplo, o Seletion Sort (O(n^2)).

Questão 4:

Selection Sort:

Funciona selecionando repetidamente o menor ou maior elemento da lista não ordenada e movendo-o para o início ou final da lista ordenada.

Inicialmente, a lista é dividida em duas partes: a parte ordenada e a parte não ordenada.

Em cada iteração, o algoritmo encontra o menor elemento na parte não ordenada e o troca com o primeiro elemento da parte não ordenada, expandindo assim a parte ordenada.

Esse processo é repetido até que toda a lista esteja ordenada.

Insertion Sort:

Constrói a lista ordenada um elemento de cada vez, inserindo cada novo elemento na posição correta em relação aos elementos já ordenados.

A lista é dividida em duas partes: a parte ordenada e a parte não ordenada.

O algoritmo itera pela parte não ordenada e, em cada iteração, seleciona um elemento e o insere na posição correta na parte ordenada, empurrando os elementos maiores para a direita.

Isso é repetido até que todos os elementos estejam na parte ordenada.

Bubble Sort:

Compara pares de elementos adjacentes e os troca se estiverem na ordem errada, fazendo com que os maiores ou menores elementos "subam" para o final da lista em cada iteração.

O processo de comparação e troca é repetido para cada elemento na lista, percorrendo a lista várias vezes.

À medida que as iterações continuam, os elementos maiores (ou menores) gradualmente se movem para o final da lista, até que toda a lista esteja ordenada.

Merge Sort:

Algoritmo de ordenação recursivo que divide a lista em metades menores, ordena essas metades e, em seguida, combina as sublistas ordenadas para obter a lista final ordenada.

A lista é dividida pela metade repetidamente até que cada sublista contenha apenas um elemento.

As sublistas ordenadas são então mescladas (combinadas) de forma ordenada, comparando os elementos das sublistas e movendo-os para a lista final ordenada.

Quick Sort:

Algoritmo de ordenação dividir-para-conquistar que escolhe um elemento pivô da lista, particiona a lista em subconjuntos menores, elementos menores que o pivô e elementos maiores que o pivô, e, em seguida, recursivamente ordena os subconjuntos.

O pivô é escolhido, por exemplo, como o último elemento da lista.

Os elementos são rearranjados de forma que todos os elementos menores que o pivô estejam à esquerda e todos os elementos maiores estejam à direita.

O processo é repetido para os subconjuntos menores até que toda a lista esteja ordenada.

Questão 5:

Selection Sort:

Complexidade de Tempo:

Melhor Caso: O(n^2)

Caso Médio: O(n^2)

Pior Caso: O(n^2)

Complexidade de Espaço: O(1) (espaço constante)

Insertion Sort:

Complexidade de Tempo:

Melhor Caso: O(n) - quando a lista já está quase ordenada

Caso Médio: O(n^2)

Pior Caso: O(n^2)

Complexidade de Espaço: O(1) (espaço constante)

Bubble Sort:

Complexidade de Tempo:

Melhor Caso: O(n) - quando a lista já está quase ordenada

Caso Médio: O(n^2)

Pior Caso: O(n^2)

Complexidade de Espaço: O(1) (espaço constante)

Merge Sort:

Complexidade de Tempo:

Melhor Caso: O(n log n)

Caso Médio: O(n log n)

Pior Caso: O(n log n)

Complexidade de Espaço: O(n) (espaço linear) - devido à necessidade de espaço adicional para a etapa de mesclagem

Quick Sort:

Complexidade de Tempo:

Melhor Caso: O(n log n) - quando o pivô divide a lista aproximadamente pela metade

Caso Médio: O(n log n)

Pior Caso: O(n^2) - quando o pivô é sempre o maior ou menor elemento

Complexidade de Espaço: O(log n) - para a pilha de recursão no pior caso, devido às chamadas recursivas