Lista de Exercícios 4

1 - Defina o significado da notação big-o (Ó grande).
Big-O é uma das notações utilizadas para medir o desempenho de algoritmos conforme o tamanho da entrada aumenta. Ela ajuda a comparar e entender a eficiência de diferentes algoritmos, indicando como o tempo de execução ou uso de recursos cresce em relação ao tamanho da entrada. Alguns exemplos incluem "O(1)" para tempo constante, "O(log n)" para tempo logarítmico e "O(n)" para tempo linear. Essa notação é útil para escolher algoritmos adequados para tarefas específicas

2 - Qual é a diferença entre a pesquisa sequencial e a pesquisa binária?
A pesquisa sequencial percorre todos os elementos de uma coleção até encontrar o desejado, sendo mais eficaz para coleções pequenas. Já a pesquisa binária divide a coleção ordenada pela metade em cada etapa, sendo mais rápida em coleções grandes, mas exige que os elementos estejam ordenados. A pesquisa sequencial tem complexidade O(n), enquanto a pesquisa binária tem complexidade O(log n).

3 - O que a complexidade computacional analisa? Quais são as complexidades eficientes e não eficientes? Explique sua resposta.
A complexidade computacional avalia quanto tempo e recursos um algoritmo requer conforme o tamanho dos dados aumenta. Complexidades eficientes, como O(log n) e O(n), têm crescimento controlado e são adequadas para grandes conjuntos de dados. Complexidades não eficientes, como O(n^2) e O(2^n), têm crescimento rápido e são impraticáveis para tamanhos maiores de dados. Algoritmos com complexidades mais baixas são preferíveis, pois lidam melhor com dados maiores de forma rápida e econômica.

4 - Explique o funcionamento dos seguintes algoritmos de ordenação: selection sort, insertion sort, bubble sort, merge sort e quick sort.
Selection Sort: O Selection Sort percorre a lista em busca do menor elemento e o coloca na primeira posição. Em seguida, ele busca o segundo menor elemento e o coloca na segunda posição, e assim por diante. Ele faz isso selecionando o menor elemento não classificado e trocando-o com o elemento na posição atual.

Insertion Sort: O Insertion Sort funciona da maneira que ordenamos as cartas em nossas mãos. Ele percorre a lista da esquerda para a direita e, para cada elemento, o insere na posição correta entre os elementos já ordenados à esquerda. Basicamente, ele "insere" cada elemento na posição correta no subconjunto ordenado.

Bubble Sort: O Bubble Sort compara elementos adjacentes e os troca se estiverem fora de ordem. Ele continua passando pela lista até que nenhum elemento precise ser trocado, o que significa que a lista está ordenada. O maior elemento "flutua" para o final da lista como uma bolha.

Merge Sort: O Merge Sort é um algoritmo de divisão e conquista. Ele divide a lista em metades menores, ordena cada metade e depois combina essas metades ordenadas para criar a lista final ordenada. A etapa de combinação é fundamental e envolve comparar os elementos de duas listas ordenadas e mesclá-los em ordem.

Quick Sort: O Quick Sort também é um algoritmo de divisão e conquista. Ele escolhe um elemento chamado de "pivot" e reorganiza a lista de forma que os elementos menores que o pivot estejam à esquerda e os elementos maiores à direita. Ele faz isso repetidamente para as sublistas resultantes, dividindo e conquistando até que toda a lista esteja ordenada.

5 - Quais são as complexidades computacionais para os seguintes algoritmos de ordenação: selection sort, insertion sort, bubble sort, merge sort e quick sort.

Selection Sort:
Melhor caso: O(n^2)
Caso médio: O(n^2)
Pior caso: O(n^2)

Insertion Sort:
Melhor caso: O(n)
Caso médio: O(n^2)
Pior caso: O(n^2)

Bubble Sort:
Melhor caso: O(n)
Caso médio: O(n^2)
Pior caso: O(n^2)

Merge Sort:
Melhor caso: O(n log n)
Caso médio: O(n log n)
Pior caso: O(n log n)

Quick Sort:
Melhor caso: O(n log n)
Caso médio: O(n log n)
Pior caso: O(n^2)