ATIVIDADES PARA ESTUDO

NOME: JULIA TEIXEIRA BARBOSA **RA:** 006797

1. **Explique o princípio de funcionamento do Bubble Sort.**
   O princípio do Bubble Sort é fazer a comparação sempre em pares, de forma consecutiva, com o objetivo de remanejar os maiores elementos para o final da lista.

2. **Qual é a complexidade de tempo do Bubble Sort no pior caso, melhor caso e caso médio?**
   n → tamanho da lista
   Pior caso: n^2 (quando a lista está inversamente ordenada)
   Melhor caso: n (quando a lista está ordenada)
   Caso médio: n^2 (quando a lista está distribuída aleatoriamente)

3. **Por que o Bubble Sort não é eficiente para grandes conjuntos de dados?**
   Porque ele não possui um mecanismo para evitar comparações desnecessárias.

4. **Descreva como o Selection Sort funciona.**
   O princípio do Selection Sort é a seleção do elemento mínimo ou máximo. Consiste em dividir a lista em duas partes (ordenada e desordenada), selecionando sempre o menor ou maior elemento da parte desordenada, inserindo-o na posição correta.

5. **Qual é a complexidade de tempo do Selection Sort no pior caso, melhor caso e caso médio?**
   n → tamanho da lista
   Pior caso: n^2 (quando a lista está inversamente ordenada)
   Melhor caso: n^2 (quando a lista está ordenada)
   Caso médio: n^2 (quando a lista está distribuída aleatoriamente)

6. **O Selection Sort é estável? Explique.**
   Não é estável. Não há garantia de que a ordem relativa dos elementos com chaves iguais será preservada. Durante as trocas repetidas para colocar o menor elemento na posição correta, a ordem relativa entre elementos com chaves iguais pode ser alterada. Portanto, o Selection Sort não é considerado um algoritmo de ordenação estável.

7. **Explique o princípio de funcionamento do Insertion Sort.**
   O princípio do Insertion Sort é a criação de uma sublista ordenada, onde serão inseridos novos elementos, realizando a comparação dois a dois, até a ordenação desta sublista.

8. **Qual é a complexidade de tempo do Insertion Sort no pior caso, melhor caso e caso médio?**
   n → tamanho da lista
   Pior caso: n^2 (quando a lista está inversamente ordenada)
   Melhor caso: n (quando a lista está ordenada)
   Caso médio: n^2 (quando a lista está distribuída aleatoriamente)

9. **Como o Insertion Sort se comporta em listas quase ordenadas?**
   A cada iteração do Insertion Sort, quando ele encontra um elemento que precisa ser movido para a posição correta, ele só precisa percorrer os elementos anteriores na lista ordenada até encontrar o lugar certo para inserir o elemento atual. Como a lista está quase ordenada, esse processo de encontrar a posição correta geralmente é rápido, resultando em um desempenho muito bom.

10. **Descreva como o Merge Sort divide e combina os elementos da lista.**
    Ele divide a lista não ordenada em sublistas recursivamente até que cada sublista contenha apenas um elemento. Depois, combina essas sublistas ordenadamente, fundindo-as em uma lista ordenada final.

11. **Qual é a complexidade de tempo do Merge Sort no pior caso, melhor caso e caso médio?**
    n → tamanho da lista
    Pior caso: n log n (quando a lista está totalmente desordenada)
    Melhor caso: n log n (quando a lista está ordenada)
    Caso médio: n log n (semelhante ao pior caso)

12. **O Merge Sort é um algoritmo in-place? Explique.**
    O Merge Sort não é considerado in-place. Isso ocorre porque, durante a etapa de combinação (merge), o Merge Sort cria novos arrays temporários para armazenar as sublistas mescladas antes de copiá-las de volta para a lista original.

13. **Explique o princípio de funcionamento do Quick Sort.**
    O princípio do Quick Sort é dividir o problema em problemas menores, resolver esses problemas menores e, em seguida, combinar as soluções dos problemas menores em uma solução para o problema original. Essa abordagem de divisão e conquista é fundamental para a eficiência e velocidade do algoritmo.

14. **Como o Quick Sort escolhe o pivô e por que isso é importante?**
    A escolha do pivô no Quick Sort é fundamental para o desempenho do algoritmo. Ele divide a lista em duas partes durante a ordenação. Estratégias comuns incluem escolha fixa (primeiro, último, meio), aleatória, mediana de três e amostragem aleatória. Uma escolha eficaz minimiza o risco de sub-listas desbalanceadas, resultando em menor tempo de execução.

15. **Qual é a complexidade de tempo do Quick Sort no pior caso, melhor caso e caso médio?**
    n → tamanho da lista
    Pior caso: n^2 (quando o pivô escolhido resulta em uma partição desbalanceada)
    Melhor caso: n log n (quando o pivô escolhido resulta em uma partição balanceada)
    Caso médio: n log n (semelhante ao melhor caso)

16. **O Quick Sort é um algoritmo estável? Explique.**
    Não, o Quick Sort não é um algoritmo estável. Um algoritmo de ordenação é considerado estável se preserva a ordem relativa de elementos com chaves iguais durante o processo de ordenação. No Quick Sort, a troca de elementos ocorre livremente durante o particionamento da lista, o que pode resultar na alteração da ordem relativa de elementos com chaves iguais.

17. **Em que cenários cada algoritmo de ordenação é mais adequado?**
    - Bubble Sort:
        - Quando o conjunto de dados é pequeno ou quase ordenado.
        - Em situações onde a complexidade não é uma preocupação e a simplicidade é preferida.
    - Selection Sort:
        - Situações onde o espaço de memória é limitado, pois o Selection Sort têm uma complexidade de memória constante.
        - Em cenários onde a troca de elementos é mais cara do que as comparações, já que o Selection Sort minimiza o número de trocas.
    - Insertion Sort:
        - Quando o conjunto de dados é pequeno ou quase ordenado, pois o Insertion Sort têm um desempenho eficiente nessas situações.

- Merge Sort:
    - Para ordenar grandes conjuntos de dados, pois possui uma complexidade de tempo garantida $O(n \log n)$ em todos os casos.
    - Quando a estabilidade é importante, pois o Merge Sort é um algoritmo de ordenação estável.
- Quick Sort:
    - Para ordenar grandes conjuntos de dados, especialmente quando o desempenho é uma prioridade, pois geralmente tem um desempenho rápido em média.
    - Em cenários onde a aleatoriedade dos dados é alta, pois o Quick Sort é menos suscetível ao pior caso em comparação com outros algoritmos de ordenação.

18. Defina o que significa um algoritmo de ordenação ser estável.

Um algoritmo de ordenação é considerado estável se preserva a ordem relativa dos elementos com chaves iguais durante o processo de ordenação. Em outras palavras, se dois elementos tiverem chaves iguais, a ordem relativa entre eles no array original é mantida no array ordenado.

19. Quais dos algoritmos mencionados são estáveis?

Apenas o Merge Sort e o Insertion Sort são algoritmos estáveis.

- Merge Sort: É estável porque mantém a ordem relativa de elementos iguais durante a fase de intercalação.
- Insertion Sort: Também é estável, pois apenas move elementos para a posição correta se necessário, sem alterar a ordem dos elementos iguais.

20. Defina o que significa um algoritmo de ordenação ser in-place.

Um algoritmo de ordenação é considerado "in-place" se ele ordena os elementos da lista diretamente na própria estrutura de armazenamento, sem a necessidade de alocação de memória adicional para armazenar cópias dos elementos. Isso significa que o algoritmo reorganiza os elementos dentro da mesma área de memória que a lista original ocupa, sem criar uma cópia separada.

21. **Quais dos algoritmos mencionados são in-place?**
    - Selection Sort: É um algoritmo in-place. Ele opera na lista original e não requer memória adicional além de algumas variáveis de índice.
    - Bubble Sort: Também é in-place. Assim como o Selection Sort, ele opera diretamente na lista original e não requer memória adicional significativa.
    - Insertion Sort: É in-place. Ele opera na lista original, movendo os elementos para a posição correta à medida que percorre a lista.

22. **Quais são algumas otimizações comuns que podem ser aplicadas ao Bubble Sort, Selection Sort e Insertion Sort para melhorar sua eficiência?**

    Bubble Sort:

    1. Verificação de trocas:
        - Durante cada passagem pelo array, acompanhe se houve trocas. Se não houver trocas em uma passagem completa, isso significa que o array já está ordenado, e o algoritmo pode ser interrompido.
    2. Limitar a verificação:
        - Após cada passagem pelo array, o maior elemento já estará na posição correta. Portanto, nas próximas passagens, não é necessário verificar as últimas posições que já estão ordenadas.

    Selection Sort:

    1. Minimizar as trocas:
        - Embora tenha uma complexidade de tempo invariável independentemente da ordem dos elementos, pode-se otimizar minimizando o número de trocas. Isso pode ser feito verificando se o elemento atual já é o menor antes de fazer a troca.

    Insertion Sort:

    1. Busca binária para a posição de inserção:
        - Em vez de percorrer linearmente a lista ordenada para encontrar a posição de inserção de um elemento, podemos usar busca binária para localizar a posição de inserção de forma mais eficiente.

    2. Deslocamento de elementos:
        - Em vez de trocar repetidamente os elementos para frente até a posição correta, podemos realizar deslocamentos dos elementos maiores do que o elemento a ser inserido,

abrindo espaço para inserir o novo elemento de forma mais eficiente.

23. **Explique o conceito de "Divisão e Conquista" e como ele é aplicado nos algoritmos Merge Sort e Quick Sort.**
"Divisão e Conquista" é um paradigma de resolução de problemas que envolve a divisão de um problema em subproblemas menores, resolução dos subproblemas de forma independente e, em seguida, combinando suas soluções para resolver o problema original.

Merge Sort:
- Divisão: Divide recursivamente a lista original pela metade até que cada sublista contenha apenas um elemento.
- Conquista: Ordena as sublistas menores de forma independente.
- Combinação: Combina as sublistas ordenadas em uma única lista ordenada, mesclando-as em pares.

Quick Sort:
- Divisão: Escolhe um elemento pivot e divide a lista em duas partições, uma contendo elementos menores que o pivot e outra contendo elementos maiores que o pivot.
- Conquista: Recursivamente ordena as partições menores.
- Combinação: Não há uma etapa de combinação explícita no Quick Sort. A ordenação é realizada durante a fase de conquista, enquanto as partições são ordenadas independentemente. No entanto, como o Quick Sort é in-place, não há necessidade de uma etapa de combinação separada.