Disciplina: Estruturas de Dados I

Aluno(a):_____ Nota: _____

Prof. Tiago Pessoa Ferreira de Lima

Cada questão vale 3,33

1. O **Merge Sort** é um algoritmo de ordenação altamente eficiente, baseado na estratégia "dividir para conquistar". Ele funciona da seguinte maneira:

1. **Dividir:** A lista é dividida ao meio até que cada sub-lista contenha um elemento.
2. **Conquistar (Mesclar):** As sub-listas são mescladas (merge) duas a duas, de forma ordenada, até que reste apenas uma lista, que será a lista final ordenada.

Dada a lista inicial: [7, 4, 1, 9, 3, 5, 2], complete a tabela abaixo, mostrando o estado das listas após cada nível de fusão (merge).

Etapa da Fusão (Merge)	Estado da(s) Lista(s)
Inicial	[7, 4, 1, 9, 3, 5, 2]
Após Divisão Completa	[7] [4] [1] [9] [3] [5] [2]
1º Nível de Fusão	
2º Nível de Fusão	
Fusão Final	

Escreva um pequeno texto respondendo:

- A estratégia "dividir para conquistar" do Merge Sort resulta em uma complexidade de tempo O(nlogn). Explique brevemente por que o termo log n aparece nessa complexidade.
- Qual é a principal desvantagem do Merge Sort em termos de uso de recursos computacionais, especialmente em sistemas com memória limitada?
- Por que implementações de ordenação em algumas bibliotecas padrão (como a de Java para objetos ou a de Python) utilizam variações do Merge Sort, mesmo que o Quick Sort seja frequentemente mais rápido na prática?

2. O **Quick Sort** é um dos algoritmos de ordenação mais rápidos e utilizados, também baseado na estratégia "dividir para conquistar". Seu funcionamento é:

1. **Escolha do Pivô:** Um elemento da lista é escolhido como "pivô".
2. **Particionamento:** A lista é reorganizada de modo que todos os elementos menores que o pivô fiquem à sua esquerda e todos os maiores fiquem à sua direita. Após essa etapa, o pivô está em sua posição final correta.
3. **Recursão:** O processo é repetido recursivamente para as duas sub-listas (à esquerda e à direita do pivô).

Dada a lista inicial: [6, 10, 1, 8, 3, 5, 2] e utilizando sempre o **primeiro elemento** da sub-lista como pivô, complete a tabela abaixo mostrando o estado da lista após cada etapa de particionamento principal.

Partição (Pivô escolhido)	Estado da Lista Após o Particionamento
Inicial	[6, 10, 1, 8, 3, 5, 2]
1ª (pivô = 6)	
2ª (partição da sub-lista adequada)	
3ª (partição da outra sub-lista)	
3ª (partição da sub-lista à direita)	

Escreva um pequeno relatório respondendo:

- Como a estratégia de escolher o primeiro elemento como pivô afeta o desempenho do Quick Sort se a lista de entrada já estiver ordenada ou em ordem inversa?
- O Quick Sort é considerado um algoritmo "in-place". O que isso significa na prática e por que essa característica é uma vantagem importante sobre o Merge Sort?
- Compare a abordagem de trocas (swaps) do Quick Sort com a do Selection Sort. Por que o Quick Sort é muito mais eficiente, mesmo que ambos realizem trocas?

3. Analise os cenários abaixo e responda qual algoritmo seria a escolha mais apropriada para cada um, justificando sua decisão com base nas características da tabela.

Ordenando Listas Encadeadas (Linked Lists) Você precisa ordenar uma grande quantidade de dados armazenados em uma lista encadeada. Nesse tipo de estrutura, o acesso a um elemento no meio da lista é lento (requer percorrer a lista desde o início). Qual algoritmo é estruturalmente mais vantajoso para esta tarefa? Por quê?

Otimização de Bibliotecas Muitas implementações de sort() em linguagens de programação usam uma abordagem híbrida: rodam Quick Sort, mas quando as sub-listas a serem ordenadas se tornam muito pequenas (ex: menos de 15 elementos), elas trocam para o Insertion Sort. Por que essa otimização é eficaz para o Quick Sort?

Desempenho de Hardware Moderno Processadores modernos são mais rápidos quando acessam dados que estão próximos uns dos outros na memória (princípio da "localidade de referência", que otimiza o uso da memória cache). Qual dos dois algoritmos, ao ordenar um array, tende a ter uma melhor localidade de referência e, portanto, um melhor desempenho prático em muitas arquiteturas de hardware? Justifique.

Rascunho