Lista de exercícios de ordenação

1 – O problema de ordenação consiste em organizar ou reorganizar elementos de uma coleção, como um vetor ou lista, em uma ordem específica, geralmente tendo como objetivo, ordem crescente ou decrescente, posições e elementos são comparados entre si para alcançar a ordenação desejada.

2 – Ordenação in-loco ordena uma lista sem criar outa inteiramente nova, requerendo um espaço adicional 0(1) constante para ordenação.

3 – Ordenação estável mantém a ordem relativa dos elementos que possuem chaves (valores) iguais.

4 - [(4, 'A'), (3, 'B'), (3, 'C'), (2, 'D')]

Pode resultar em - [(2, 'D'), (3, 'C'), (3, 'B'), (4, 'A')]

Porém (3, 'C') não vem antes de (3, 'B') em uma ordem crescente, isto acontece pois o Selection Sort pode inverter posições de elementos iguais.

5 - Os algoritmos de ordenação por inserção, seleção e heap são todos baseados em comparações, mas diferem em sua forma de organizar os elementos. O insertion sort constrói a lista ordenada gradualmente, inserindo cada novo elemento na posição correta, simples, eficiente para listas pequenas e estável, embora tenha custo O(n²). O selection sort, por sua vez, seleciona repetidamente o menor elemento do restante do vetor e o coloca na posição correta, sendo não estável e também O(n²). Já o heap sort é uma versão otimizada do selection sort, pois usa uma estrutura de dados chamada heap para encontrar o maior (ou menor) elemento de forma mais eficiente, alcançando complexidade O(n log n), mas também não é estável.

6 –

1. [3 4 9 2 5 8 2 1] | [7 4 6 2 9 8 5 1] - divisão

[3 4 9 2] [5 8 2 1] | [7 4 6 2] [9 8 5 1] – divisão novamente

[3][4][9][2][5][8][2][1][7][4][6][2][9][8][5][1] – última divisão até resultar

[1 1 2 2 2 3 4 4 5 5 6 7 8 8 9 9] – elementos intercalados

1. [ ] [1] [3 4 9 2 5 8 2 7 4 6 2 9 8 5] pivô = 1, maiores que 1 vão a direita

[1 1 2 2 2 3 4 4 5 5 6 7 8 8 9 9] pivô = último da parte direita = 5, particiona recursivamente até alcançar o resultado.

1. [9 8 9 4 5 8 2 3 7 4 6 2 2 1 5 1] constrói o heap

[1 8 9 4 5 8 2 3 7 4 6 2 2 1 5 9] troca o maior (9) com o último

[1 1 2 2 2 3 4 4 5 5 6 7 8 8 9 9] após repetir o processo de heapify, temos

7 - Verificação por índices:

* i = 0 → 23; filhos em 1,2 → 17, 14 → 23 ≥ 17, 14
* i = 1 → 17; filhos em 3,4 → 6, 13 → 17 ≥ 6, 13
* i = 2 → 14; filhos em 5,6 → 10, 1 → 14 ≥ 10, 1
* i = 3 → 6; filhos em 7,8 → 5, **7** → 6 ≥ 7 ? Violação (7 > 6)
* i = 4 → 13; filho em 9 → 12 → 13 ≥ 12

Como há uma violação no nó i = 3 (valor 6) porque um de seus filhos tem valor 7 maior que ele, o arranjo NÃO representa um heap máximo.

8 - Arranjo inicial [5, 1, 7, 4, 9, 1, 3, 4, 8, 6]

Passo 1 – Heapify dos últimos nós não-folha

Último nó não-folha: índice floor(n/2)-1 = floor(10/2)-1 = 4

i = 4 (valor 9)

* Filhos: 2*4+1=9 → 6, 2*4+2=10 → fora do vetor
* 9 ≤ 6? → troca 9 e 6

[5, 1, 7, 4, 6, 1, 3, 4, 8, 9]

i = 3 (valor 4)

* Filhos: 7 → 4, 8 → 8
* 4 ≤ 4 e 4 ≤ 8 → sem mudança

i = 2 (valor 7)

* Filhos: 5 → 1, 6 → 3
* 7 > 1 → troca 7 e 1

Arranjo intermediário [5, 1, 1, 4, 6, 7, 3, 4, 8, 9]

i = 1 (valor 1)

* Filhos: 3 → 4, 4 → 6
* 1 ≤ 4 e 6 → sem mudança

i = 0 (valor 5)

* Filhos: 1 → 1, 2 → 1
* 5 > 1 → troca com filho menor → 1 no índice 1

[1, 5, 1, 4, 6, 7, 3, 4, 8, 9]

Heapify recursivo no índice 1 (valor 5)

* Filhos: 3 → 4, 4 → 6
* 5 > 4 → troca com 4 (índice 3)

[1, 4, 1, 5, 6, 7, 3, 4, 8, 9]

Heapify recursivo no índice 3 (valor 5)

* Filhos: 7 → 4, 8 → 8
* 5 > 4 → troca com 4 (índice 7)

[1, 4, 1, 4, 6, 7, 3, 5, 8, 9]

Heapify recursivo no índice 7 → sem filhos → ok

Resultado final – Heap mínimo

[1, 4, 1, 4, 6, 7, 3, 5, 8, 9]

9 - [(4, 'A'), (1, 'B'), (3, 'C'), (4, 'D')]

Arranjo inicial: [(4,'A'), (1,'B'), (3,'C'), (4,'D')]

Após Heap Sort: [(1,'B'), (3,'C'), (4,'D'), (4,'A')]

Ordem relativa de elementos iguais (4) não preservada → Heap Sort não é estável.

10 - Os algoritmos de ordenação por intercalação (Merge Sort) e por particionamento (Quick Sort) estão relacionados por ambos utilizarem a estratégia "dividir para conquistar", ou seja, dividem o problema de ordenar um vetor em subproblemas menores, resolvem cada subproblema e combinam os resultados. No Merge Sort, a divisão é feita separando o vetor em metades e a ordenação é obtida através da intercalação(merge) das partes já ordenadas. No Quick Sort, a divisão é baseada na escolha de um pivô, que separa os elementos menores e maiores que ele, e a ordenação é obtida ao ordenar recursivamente essas partições. Assim, ambos compartilham a ideia de dividir o problema em partes menores, mas diferem na forma como combinam ou organizam essas partes para chegar ao resultado final.