Pedro Lovatto

1. Identificar qual algoritmo de ordenação apresentado em cada SLIDE e justificar.

Algoritmo 1: É o Insertion Sort, pois a ideia dele é iniciar no segundo elemento e então comparar com os elementos anteriores, aí caso o valor anterior seja menor, os valores são trocados, e assim segue, na próxima "rodada" o valor atual será o terceiro elemento da lista e vai comparar com os anteriores, caso esse valor atual seja menor, ele irá trocar de lugar. No caso do slide se seguiu isso, o laço de repetição começa no valor "2" e ele é comparado com o valor "3", como 2 é menor que 3 o valor 2 troca de lugar com o 3.

Algoritmo 2: É o Selection Sort, pois no slide mostra ele inicialmente atribuindo a variável "menor" para o primeiro elemento da lista e depois compara com o resto de elementos da lista para achar o menor, quando acha, esse menor valor troca de lugar com o primeiro elemento da lista que tinha servido como valor inicial para a variável "menor", na próxima repetição, ele irá a fazer a mesma coisa, só que agora comparando os valores entre o índice 1 e o resto da lista, ou seja, a variável "menor" agora terá como valor inicial o valor do elemento que está na posição i=1 e irá comparar o resto da lista, se houver um valor menor, eles trocam de posição.

Algoritmo 3: É o Bubble Sort, pois no slide mostra que a comparação é sempre feita da seguinte maneira, o valor atual (i) com o seu próximo valor (i+1), caso o próximo valor seja menor ele troca de posição com o valor de i. No slide a comparação inicial seria 6 e 5, 5 é menor então troca de lugar com o 6, na próxima repetição a comparação seria entre 6 e 3 e assim segue.

Algoritmo 4: É o Bubble Sort mais uma vez, mas agora uma variação usando Do-While e a variável "continua" que irá receber o valor de i caso caia na condição do valor na posição de i for maior que o valor do próximo elemento, quando essa variável "continua" receber um valor diferente de 0 o laço irá parar.

Algoritmo 5: É o Merge Sort, cuja ideia é "dividir para conquistar" que é o que o slide apresenta, no início ele divide entre duas metades (usando recursividade) e depois começa a dividir em partes menores ainda e depois começa a juntar os elementos e já os ordenando na posição correta.

Algoritmo 6: É o Quick Sort, pois logo no primeiro slide já fala sobre “pivô” que é justamente a ideia por trás desse algoritmo, a escolha de um valor inicial para ser um pivô e então ele particiona a lista de um jeito em que os menores valores fique a esquerda do pivô e os maiores valores fiquem a direita do pivô, e então começa a ordenação.

2. Descreva/pesquise o que é notação assintótica? O que é Big O, Pior caso, Caso médio, Melhor caso?

Notação assintótica é uma linguagem que nos permite analisar o tempo de execução de um algoritmo através da identificação de seu comportamento com o crescimento da entrada oferecida. Big-O, também escrita como O, é uma Notação Assintótica para o pior caso. Para o melhor caso é o Big-Omega, também descrito como Ω. Para o caso médio é Theta, também descrito como (Θ).

Fonte: https://learnxinyminutes.com/docs/pt-br/asymptotic-notation-pt/

3. Identificar qual o melhor e o pior caso (pesquisar/encontrar qual o melhor e o pior caso de cada algoritmo estudado)

Insertion Sort:

Melhor caso: O(n), se a lista já estiver classificada, onde n é o número de elementos na lista.
Pior caso: O(n 2), se a lista estiver em ordem reversa.

Selection Sort:

Melhor caso: Ocorre quando a lista já está ordenada

Pior caso: O(n 2), caso a lista esteja em ordem reversa.

Bubble Sort:

Melhor caso: O melhor caso ocorre quando a lista já está ordenada, e o Bubble sort apenas percorre a lista uma vez para verificar isso

Pior caso: O pior caso ocorre quando a lista está ordenada na ordem inversa, exigindo que o algoritmo realize várias trocas e percorra a lista várias vezes.

Merge Sort:

Melhor caso: Quando a lista já está ordenada ou quase ordenada. A complexidade é O(n*log n).

Pior caso: Quando a lista está em ordem inversa. A complexidade também é O(n*log n).

Quick Sort:

Melhor caso: Quando o pivô escolhido em cada etapa divide o array em metades aproximadamente iguais. Complexidade: O(N log N),

Pior caso: Quando o pivô escolhido é o primeiro ou o último elemento, ocasionando em partições desbalanceadas. Complexidade: $O(n^2)$

Shell Sort:

Melhor caso: Quando a lista já está ordenada.

Pior caso: Quando a lista está em ordem inversa. Complexidade $O(n^2)$

Heap Sort:

Melhor caso: No melhor caso, o Heap Sort ainda precisa construir o heap inicial e, em seguida, reorganizar o heap a cada remoção do maior elemento. Complexidade: O(n log n)

Pior caso: No pior caso, também precisa construir o heap e garantir a propriedade do heap para cada nó, assim como no melhor caso. Isso significa que, independentemente da entrada, ele precisa garantir a manutenção do heap, o que leva a um comportamento consistente em termos de complexidade. Complexidade: O(n log n)

Counting Sort:

Melhor caso: O melhor caso ocorre quando os elementos da entrada estão distribuídos de maneira uniforme e dentro de um intervalo pequeno. Complexidade: O(n + k)

Pior caso: No pior caso, o intervalo de valores dos elementos é muito grande, ou seja, o valor máximo (k) é significativamente maior que o número de elementos (n). Mesmo que o número de elementos seja pequeno, o algoritmo ainda precisa contar todas as possíveis ocorrências até o valor k, o que torna o algoritmo menos eficiente para grandes intervalos. Complexidade: O(n + k)

Merge Sort Externo

Melhor caso: ocorre quando os dados estão parcialmente ordenados ou quando a ordenação pode ser feita com um número mínimo de operações de leitura e escrita no disco. Complexidade: O(n log n)

Pior caso:
O número de elementos é muito grande e a memória disponível é muito pequena em relação ao tamanho total dos dados, forçando muitas operações de leitura e gravação no disco ou quando as leituras e gravações no disco são ineficientes, e muitos blocos pequenos precisam ser intercalados, aumentando significativamente o número de passagens. Complexidade: O(n log n)

Fonte:

- https://www.geeksforgeeks.org/insertion-sort-algorithm/
- https://joaoarthurbm.github.io/eda/posts/selection-sort/
- http://desenvolvendosoftware.com.br/algoritmos/ordenacao/selection-sort.html
- https://elemarjr.com/clube-de-estudos/artigos/o-que-e-e-como-funciona-o-bubblesort/
- https://www.geeksforgeeks.org/merge-sort/
- https://www.geeksforgeeks.org/quick-sort-algorithm/
- https://www.geeksforgeeks.org/shell-sort/
- Chatgpt (Pro heap Sort, Counting Sort e Merge Sort externo, não tava conseguindo encontrar informações em sites.)

4. Qual dos algoritmos estudados está(ão) faltando nestes slides?

Está faltando o Heap Sort, o Shell Sort, o Counting Sort e o Merge Sort (no caso, o Merge Sort Externo)