

Nome: Lucas Tabosa Guedes

## ESPECIFICAÇÕES

Modelo do processador e velocidade: AMD Ryzen 7 5700X 8-Core Processor    3.60GHz

RAM instalada : 16GB

Sistema operacional: Windows 10 Pro

## QUESTÃO 1

Optou-se por empregar os métodos de seleção e merge sort para ordenar listas contendo números inteiros aleatórios, variando no intervalo de 1 a 1 milhão. Cabe ressaltar que o tempo de geração dessas listas de números inteiros aleatórios foi também considerado no momento de avaliar o desempenho de ambos os métodos.

A seguir está uma explicação de como cada um dos métodos de ordenação funciona, além de falarem sobre a soma ordem de complexidade no pior e melhor caso.

Método de Seleção:

O método de seleção funciona selecionando repetidamente o menor elemento da lista não ordenada e movendo-o para o início da lista ordenada. Esse processo continua até que toda a lista esteja ordenada.

Melhor Caso: O melhor caso ocorre quando a lista já está ordenada, resultando em uma complexidade de tempo de  $O(n^2)$ .

Pior Caso: O pior caso ocorre quando a lista está em ordem reversa ou desordenada, ainda resultando em uma complexidade de tempo de  $O(n^2)$ .

Merge Sort:

O Merge Sort é um algoritmo de ordenação baseado na técnica de "dividir para conquistar". Ele divide a lista em duas metades, ordena cada metade separadamente e, em seguida, combina as duas metades ordenadas para obter a lista final ordenada.

Melhor Caso: O melhor caso é quando a lista está dividida em duas metades de tamanho igual e já está parcialmente ordenada. Nesse caso, a complexidade de tempo é  $O(n \log n)$ .

Pior Caso: O pior caso ocorre quando a lista está completamente desordenada, e a complexidade de tempo também é  $O(n \log n)$ .

A seguir está o resultado com tempo total de execução e tempo médio de execução desses algoritmos de ordenação em listas de 62.500, 125.000, 250.000 e 375.000 posições.

	Tamanho	Tempo Total(ms)	Tempo Medio(ms)
Merge Sort	62500	454	9
	125000	741	14
	250000	1600	32
	375000	2378	47
	Tamanho	Tempo Total(ms)	Tempo Medio(ms)
Seleção	62500	41902	838
	125000	167401	3348
	250000	666292	13325
	375000	1500229	30004

Ao analisar os dados da planilha, é possível observar evidências claras da complexidade do algoritmo de seleção, que é de  $O(n^2)$ , e do algoritmo Merge Sort, que possui uma complexidade de  $O(n \log n)$ . Isso pode ser constatado ao comparar o tempo total de execução do algoritmo de seleção em um array com 62.500 posições e outro com 375.000 posições.

No caso do algoritmo de seleção, notamos que o tempo de execução aumenta aproximadamente 35,8 vezes quando a quantidade de elementos é aumentada em apenas 6 vezes. Esse comportamento é característico de uma complexidade quadrática ( $n^2$ ), onde pequenos aumentos no tamanho da entrada resultam em grandes aumentos no tempo de execução.

Da mesma forma, ao analisar o algoritmo Merge Sort, observamos que o tempo total de execução aumenta cerca de 5,2 vezes ao aumentar o tamanho do array em 6 vezes. Esse padrão é consistente com a complexidade  $O(n \log n)$ , onde o tempo de execução cresce de forma mais moderada à medida que o tamanho da entrada aumenta.

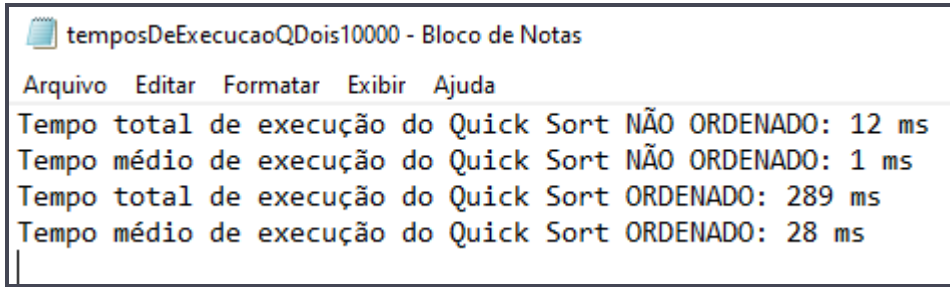
## QUESTÃO 2

Quicksort:

O quicksort é um algoritmo de ordenação eficiente. Funciona selecionando um elemento, chamado de 'pivô', da lista e particionando os elementos em duas sublistas: aqueles menores que o pivô e aqueles maiores. Em seguida, ele repete o processo nas sublistas. Isso continua até que toda a lista esteja ordenada

**Melhor Caso:** O melhor caso ocorre quando o pivô é escolhido de forma que ele divide a lista em duas partes quase iguais, ou seja, as sublistas geradas têm tamanhos aproximadamente iguais. Nesse cenário, o quicksort é altamente eficiente e tem uma ordem de complexidade de tempo de  $O(n \log n)$ , onde 'n' é o número de elementos na lista.

**Pior Caso:** O pior caso ocorre quando o pivô é escolhido de forma que ele divide a lista em duas sublistas desequilibradas, uma com a maioria dos elementos e outra com apenas alguns. Isso pode acontecer, por exemplo, quando a lista já está ordenada ou quase ordenada. Nesse cenário, o quicksort pode se tornar ineficiente, com uma ordem de complexidade de tempo de  $O(n^2)$ .



```
temposDeExecucaoQDois10000 - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
Tempo total de execução do Quick Sort NÃO ORDENADO: 12 ms
Tempo médio de execução do Quick Sort NÃO ORDENADO: 1 ms
Tempo total de execução do Quick Sort ORDENADO: 289 ms
Tempo médio de execução do Quick Sort ORDENADO: 28 ms
```

Os resultados do Quicksort em arrays ordenados e não ordenados refletem suas características fundamentais. Para arrays não ordenados, o Quicksort é eficiente, com um tempo médio de 1 ms, alinhando-se ao seu melhor caso  $O(n \log n)$ . Porém, em arrays ordenados, seu desempenho piora, com uma média de 28 ms e um tempo total de 289 ms, devido ao pior caso  $O(n^2)$ . Esses resultados destacam a importância de escolher algoritmos de ordenação adequados para diferentes situações, considerando as características dos dados a serem ordenados para otimizar o desempenho.