# Projeto e Análise de Algoritmos

## Trabalho

**Gustavo Dias de Oliveira - 202010078511**

## Introdução

A ordenação de elementos é uma tarefa essencial na computação, e neste trabalho, vamos comparar dois métodos de ordenação: Bubble Sort e Quick Sort. O Bubble Sort é simples, mas pode ser lento para grandes conjuntos de dados. O Quick Sort, por outro lado, é mais sofisticado, rápido e baseado no conceito de "dividir para conquistar". Analisaremos suas complexidades, vantagens, desvantagens e desempenho.

## Metodologia

*Algoritmo Bubble Sort*

O algoritmo utiliza dois loops while aninhados. O loop externo controla o número de iterações necessárias para garantir que todos os elementos sejam comparados e ordenados corretamente. O loop interno percorre o vetor, comparando pares de elementos adjacentes e realizando trocas quando necessário. A variável "troca" é inicializada como 1 para entrar no primeiro loop e iniciar o processo de ordenação. A cada iteração do loop externo, "troca" é definida como 0 para indicar que não houve nenhuma troca na iteração atual. Se uma troca ocorrer no loop interno, o valor de "troca" é atualizado para 1, indicando que ainda existem elementos fora de ordem e que o loop externo precisa continuar. Dentro do loop interno, os elementos adjacentes são comparados. Se o elemento atual for maior que o próximo elemento, ocorre uma troca entre eles. A variável auxiliar "aux" é usada para realizar a troca dos elementos. Após concluir o loop externo e realizar todas as iterações necessárias, o vetor é retornado, agora ordenado em ordem crescente.

*Algoritmo Quick Sort*

A função "particao" recebe como entrada um vetor, um índice de início e um índice de fim. Ela seleciona um elemento chamado de "pivot" no vetor, com base no valor do elemento no índice de início. Em seguida, define um índice superior (inicialmente igual ao índice de fim) e um índice inferior (inicialmente igual ao índice de início). O algoritmo, então, entra em um loop while que continua enquanto o índice inferior for menor que o índice superior. Dentro desse loop, há dois loops while: o primeiro incrementa o índice inferior enquanto o elemento no índice inferior for menor ou igual ao "pivot"; o segundo decrementa o índice superior enquanto o elemento no índice superior for maior que o "pivot". Se o índice inferior ainda for menor que o índice superior após os loops while, ocorre uma troca entre os elementos nos índices inferior e superior.Após o loop while, o "pivot" é colocado na posição correta no vetor, definida pelo índice superior. Ou seja, o "pivot" é colocado em uma posição onde todos os elementos à sua esquerda são menores que ele, e todos os elementos à sua direita são maiores que ele. A função retorna o índice superior, que representa a posição correta do "pivot" no vetor. A função "quickSort" implementa o algoritmo de ordenação Quick Sort usando uma pilha para armazenar os intervalos de índices a serem processados. Inicialmente, a pilha é preenchida com o intervalo completo do vetor (índice 0 até o

último índice). Enquanto houver elementos na pilha, o algoritmo retira o intervalo do topo da pilha. Se o intervalo for válido (o índice de início é menor que o índice de fim), é chamada a função "particao" para encontrar o índice de pivot correto. Em seguida, são adicionados na pilha dois novos intervalos: um do índice de início até o índice do pivot - 1 (para processar os elementos menores que o pivot), e outro do índice do pivot + 1 até o índice de fim (para processar os elementos maiores que o pivot). Esse processo continua até que a pilha esteja vazia. Ao final, o vetor é retornado, agora ordenado em ordem crescente pelo algoritmo Quick Sort.

A linguagem de programação utilizada foi o Python em um Jupyter Notebook.

Os testes foram executados no Visual Studio Code, para isso é necessário fazer as seguintes configurações:

1. Instalar a extensão do Jupyter no Visual Studio Code
2. baixar a biblioteca mathplotlib "pip install matplotlib"

Com apenas isso, o codigo deve rodar tranquilamente, sem nenhum erro!

A configuração de Hardware do computador em que foi realizado os testes, no caso, o processador, é um Ryzen 7 5700x, com essa configuração o codigo foi executado em um tempo de 11 minutos aproximadamente.

## Resultados

Na figura 1 conseguimos observar que o Bubble Sort foi muito mais rapido que o Quick Sort, mas isso porque o vetor se econtra ordenado crescentemente.

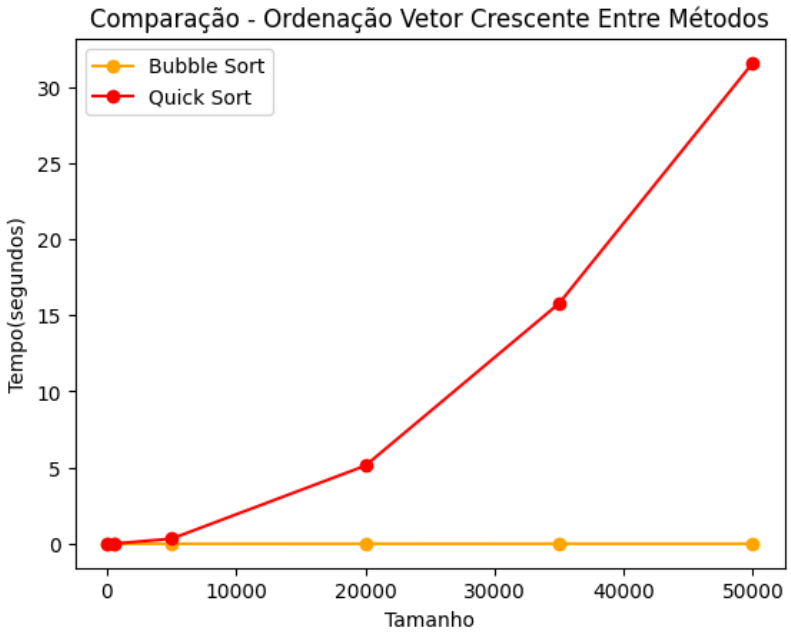


*Figura 1 - Comparaçao com vetor crescente*

O bubble sort no melhor caso, quando o vetor já está ordenado, o algoritmo percorre o vetor uma única vez (O(n)) para confirmar que não há trocas necessárias. Isso ocorre porque nenhum par de elementos adjacentes precisa ser trocado, pois o vetor já está em ordem. Com isso ele termina imediatamente após uma única passagem pelo vetor, tornando-se muito eficiente no melhor caso.

No melhor caso do quick sort, quando o vetor já está ordenado, ele seleciona o primeiro elemento como pivô. No entanto, como o vetor já está ordenado, a partição resultante divide o vetor em uma sublista vazia e outra sublista com todos os elementos restantes. Isso significa que o algoritmo precisa ser aplicado em todo o vetor novamente para ordenar a sublista não vazia. Essa recursão adicional torna o quick sort menos eficiente no melhor caso em comparação com o bubble sort.

Na figura 2 o quick sort é mais rápido que o bubble sort devido às características de cada algoritmo.

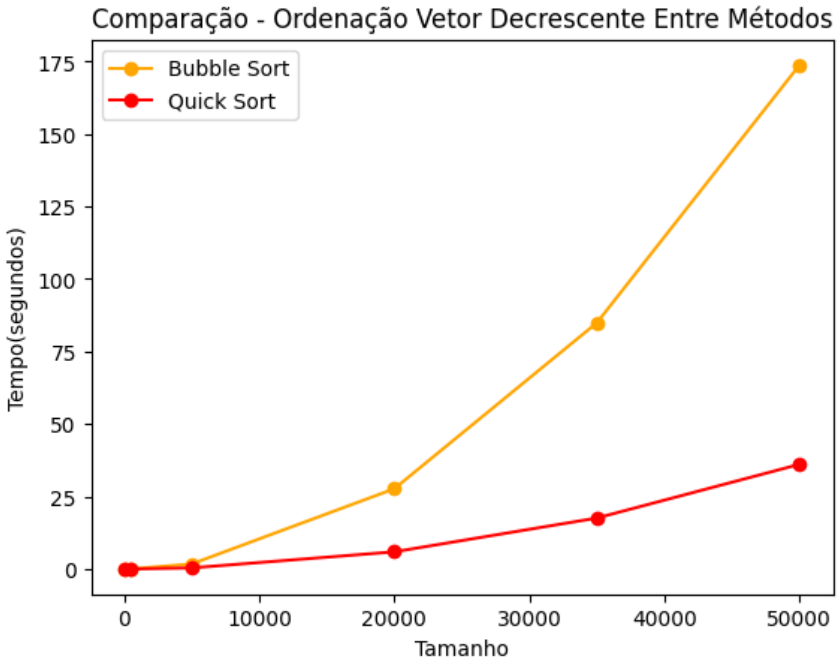


*Figura 2 - Comparaçao com vetor decrescente*

No caso do bubble sort, quando o vetor está em ordem decrescente, o algoritmo precisa percorrer o vetor várias vezes, trocando repetidamente os elementos adjacentes para movê-los para suas posições corretas. Isso resulta em um grande número de comparações e trocas.

Por outro lado, o quick sort utiliza uma estratégia de divisão e conquista, selecionando um pivô e rearranjando os elementos do vetor em torno dele. No caso de um vetor em ordem decrescente, o quick sort tem a vantagem de escolher o pivô de

maneira eficiente, o que leva a uma divisão mais equilibrada do vetor e reduz o número de comparações e trocas necessárias.

Na figura 3, percebe-se que o quick sort é muito mais rapido que o bubble sort, isso tambem devido às características de cada algoritmo.

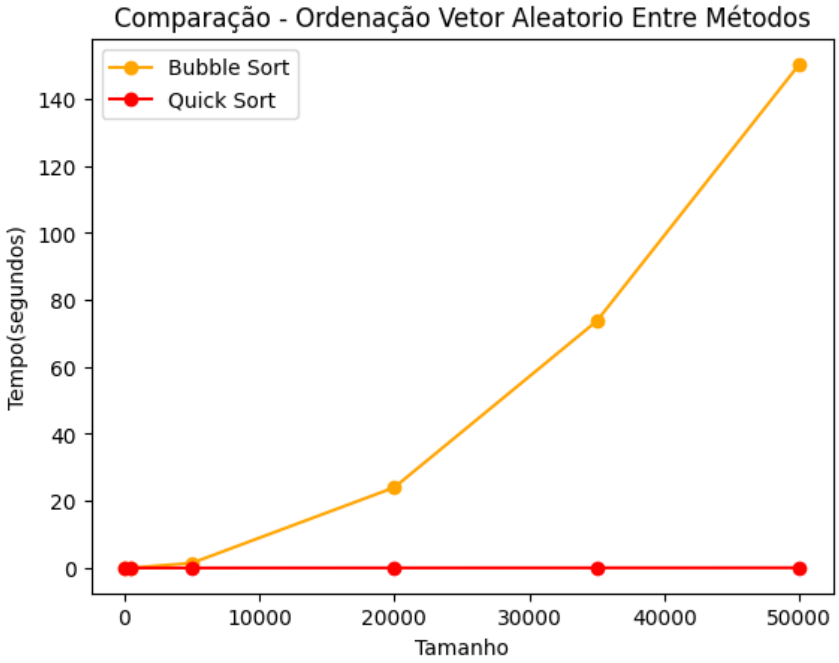


*Figura 3 - Comparaçao com vetor aleatório*

No bubble sort, quando o vetor está em ordem aleatória, o algoritmo precisa percorrer o vetor várias vezes, comparando e trocando elementos adjacentes até que o vetor esteja completamente ordenado. Isso requer um número considerável de comparações e trocas, tornando o bubble sort relativamente lento quando confrontado com um vetor em ordem aleatória.

Já o quick sort utiliza a estratégia de divisão e conquista, escolhendo um pivô e rearranjando os elementos do vetor em torno dele. No caso de um vetor em ordem aleatória, o quick sort tem a vantagem de dividir o vetor de forma equilibrada e rápida. O pivô escolhido aleatoriamente tende a dividir o vetor em sublistas de tamanhos aproximadamente iguais, o que facilita o processo de ordenação.

O Bubble Sort tem uma complexidade de tempo de $O(n^2)$, mas quando o vetor está ordenado é de tempo $O(n)$, enquanto o Quick Sort tem uma complexidade de tempo $O(n \log n)$, mas quando o vetor está ordenado de forma crescente ou decrescente é $O(n^2)$.

## Conclusão

Com isso, podemos concluir que o desempenho do Bubble Sort e do Quick Sort depende das características do vetor de entrada. No melhor caso, quando o vetor já está ordenado, o Bubble Sort é mais eficiente, pois realiza apenas uma passagem pelo vetor. No entanto, o Quick Sort se destaca quando o vetor está em ordem decrescente ou aleatória, devido à sua estratégia de divisão e conquista, que reduz o número de comparações e trocas necessárias. O Bubble Sort possui uma complexidade de tempo de $O(n^2)$, mas se beneficia do melhor caso de tempo $O(n)$, enquanto o Quick Sort possui uma complexidade de tempo de $O(n \log n)$, mas pode se tornar menos eficiente no melhor caso com tempo $O(n^2)$ quando o vetor está ordenado.