Trabalho Projeto e análise de algoritimos

Matheus Marendino

Junho 2023

1 Introdução

Este trabalho tem como objetivo testar e comparar dois algoritmos de ordenação amplamente utilizados: Quicksort e Insertionsort. Esses algoritmos são amplamente estudados na área de projeto e análise de algoritmos, sendo essenciais para a resolução de problemas que envolvem a ordenação de elementos em uma sequência. Nesse teste usaremos vetores de tamanho: 50, 500, 5000 e 50000

2 Metodologia

2.1 Quick-Sort

A ideia principal do Quick-Sort é selecionar um elemento do array, chamado de "pivô", e particionar o array em dois subarrays: um contendo elementos menores que o pivô e outro contendo elementos maiores que o pivô. Em seguida, o Quick-Sort é aplicado recursivamente a esses subarrays até que todo o array esteja ordenado. A complexidade de tempo do Quick-Sort é em média O(n log n), onde n é o número de elementos no array a ser ordenado. No entanto, em casos raros, quando a escolha do pivô não é otimizada, o Quick-Sort pode ter uma complexidade de tempo de pior caso de $O(n^2)$. Essa situação ocorre quando o pivô escolhido divide o array em partes muito desequilibradas, resultando em muitas iterações.

2.2 Insertion-Sort

O Insertion Sort é um algoritmo de ordenação simples que funciona da seguinte maneira: ele percorre um array a partir do segundo elemento e, para cada elemento, o insere na posição correta entre os elementos anteriores que já estão ordenados. A ideia principal do Insertion Sort é manter uma porção do array já ordenada enquanto insere cada novo elemento na posição correta. A cada iteração, o algoritmo compara o elemento atual com os elementos anteriores na porção ordenada, deslocando-os para a direita, se necessário, para abrir espaço para a inserção do elemento atual.

2.3 Linguagem de programação utilizada

Nesse trabalho eu utilizei python 3, no Visual Studio Code. Para rodar foi necessário implentar a biblioteca matplotlib.pyplot, e o tempo de execução foi em média 6 minutos nos testes feitos.

2.4 Configuração do computador

Processador: Intel(R) Core(TM) i3-10100F CPU @ 3.60GHz Ram instalada: 24,0 GB Placa de video: Nvidia GTX 1660 6GB

3 Resultados

3.1 Gráfico 1- Vetores de ordem crescente

Na figura 1, percebemos que o quick-sort(azul no gráfico), em comparação ao insertion-sort (Laranja), é muito mais demorado, mas isso se aplica a casos de vetores em ordem crescente.

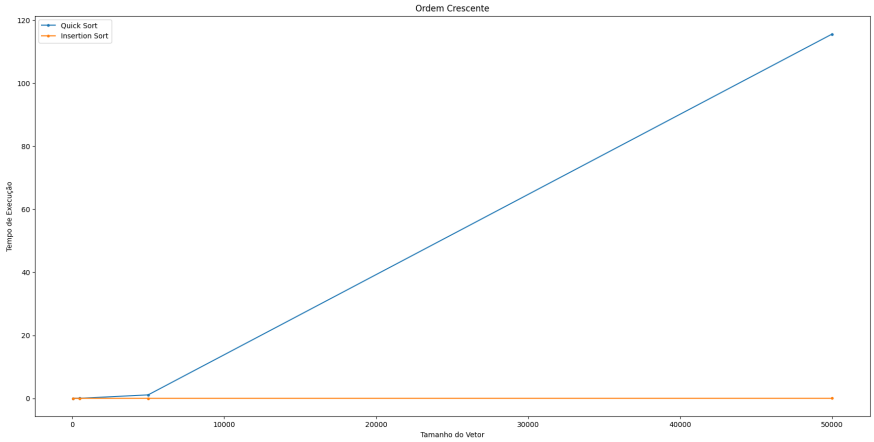

Figura 1

O Quick-Sort pode ter um desempenho ruim em casos específicos, como um vetor ordenado de forma crescente. Isso ocorre porque o QuickSort, nessas circunstâncias, dividirá o vetor em subvetores com tamanho desequilibrado, onde um subvetor é muito maior que o outro. Por outro lado o insertion-sort é um algoritmo de ordenação simples e eficiente para pequenos conjuntos de dados ou quando o vetor já está quase ordenado. Ele percorre o vetor da esquerda para a direita, e como o vetor está ordenado na ordem crescente, acaba tendo um tempo de execução e uma eficiência muito superior ao Quick-sort.

3.2 Gráfico 2- Vetores de ordem decrescente

Na figura 2, percebemos que o Insertion Sort(Laranja) pode ser menos eficiente do que o QuickSort(Azul) em algoritmos ordenados em ordem decrescente devido às diferenças nas abordagens de cada algoritmo.

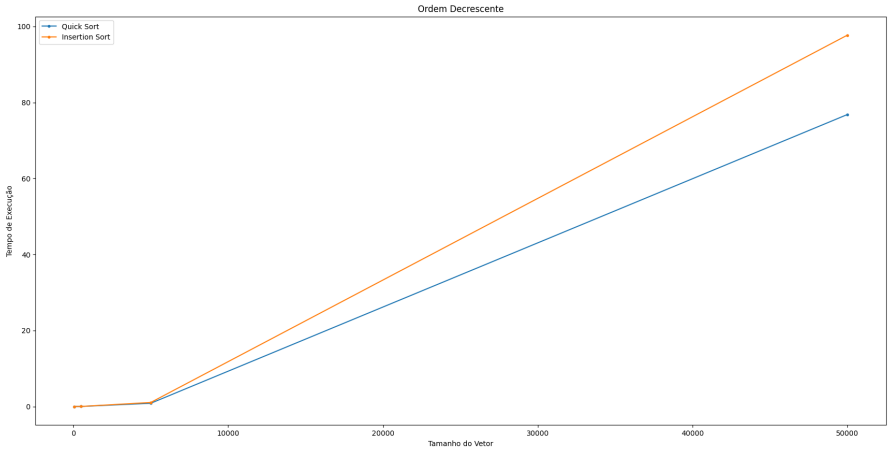

Figura 2

Isso ocorre, pois, o Quick-sort é um algoritimo de ordenação baseado em divisão e conquista. Ele pega um elemento pivô da matriz e reorganiza os elementos de forma que todos os elementos menores que o pivô fiquem à esquerda e todos os elementos maiores que o pivô fiquem à direita. Em seguida, aplica o quick-sort recursivamente a ambas as metades do vetor. Esse processo de dividir e conquistar continua até que o vetor esteja completamente ordenado.O pivô é escolhido de forma inteligente para otimizar o desempenho do algoritmo. Já o insertion-sort, ele percorre o vetor da esquerda para direita, e insere cada elemento em sua posição correta, na parte ordenada do vetor. No caso de um vetor ordenado de forma decrescente, cada novo elemento a ser inserido estará sempre no início do vetor, o que requer deslocar todos os elementos maiores uma posição à direita para abrir espaço para o novo elemento, isso resulta em um número muito grande de movimentação dos elementos, oque faz com que o tempo de execução e a eficiência do algoritimo Insertion-sort seja inferior ao Quick-sort. Por isso em casos de vetores de ordem decrescente é preferível se utilizar os algoritimos de ordenação "Quick-sort".

3.3 Gráfico 3 - Vetores ordenados aleatoriamente

Na figura 3, fica nítido que em casos de vetores aleatorios, o Quick-sort(Azul) é um algoritimo muito mais eficiente do que o Insertion-sort(Laranja), isso ocorre devido as abordagens de cada algoritimo nesse caso em específico. Enquanto o Quick-sort demorou apenas milésimo de segundo, o Insertion pode durar até

minutos. Como foi citado anteriormente, nas outras abordagens, o Quick-sort

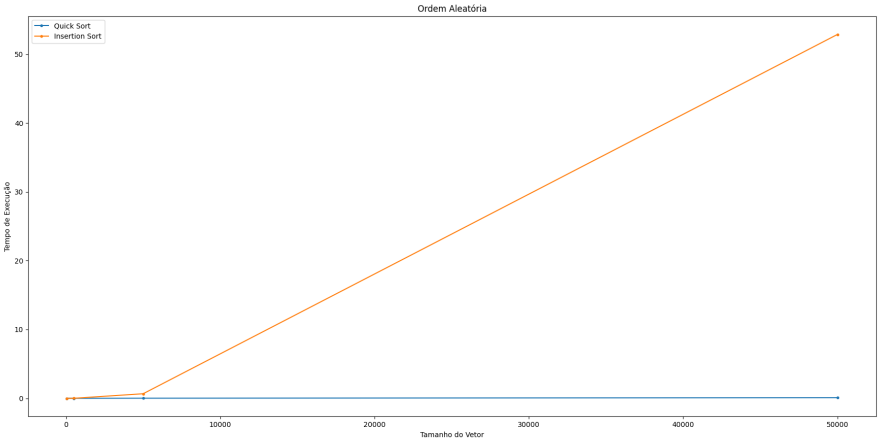

Figura 3

trabalha utilizando a divisão e conquista, onde utiliza um elemento do vetor como um pivô, de forma que elementos menores ficarão a esquerda e os maiores a direita, dessa forma, o algoritmo é aplicado recursivamente nas duas metades do vetor. Essa abordagem permite que o Quicksort divida o vetor em partes menores de forma eficiente, reduzindo o tempo necessário para ordenar o vetor completo. O pivô escolhido pode ser selecionado de várias maneiras, mas uma escolha eficiente é selecionar um pivô aleatório ou um pivô que esteja próximo do valor mediano do vetor. Isso garante que o vetor seja dividido aproximadamente em duas partes iguais, o que ajuda a evitar casos extremos onde uma das partes é significativamente maior do que a outra. Além disso, o Quick-sort também pode utilizar técnicas de otimização, como por exemplo o QuickSelect, para selecionar o pivô de forma mais eficiente. Por outro lado temos o Insertion-sort, que teve um tempo de execução muito superior, O algoritimo é eficiente para pequenos tamanhos de vetor ou quando o vetor já está quase ordenado, mas seu desempenho é afetado negativamente à medida que o tamanho do vetor aumenta, e a ordem do vetor é aleatória. Em vetores aleatórios, o Quicksort geralmente supera o Insertion Sort porque a estratégia de divisão e conquista do Quicksort permite dividir rapidamente o vetor em partes menores, enquanto o Insertion Sort precisa percorrer o vetor repetidamente para inserir cada elemento em sua posição correta.

4 Conclusão

Nesse trabalho é possível analisar que não existe um "algoritimo melhor", apenas casos em que um irá funcionar melhor que o outro e executar em menos tempo. No caso dos algoritimos testados, podemos perceber que para vetores

ordenados de forma crescente o Insertion-Sort tem um tempo de execução muito inferior ao Quick-Sort. E para casos de ordem decrescente ou ordem aleatória é preferível se utilizar o algoritimo de ordenação Quick-sort. Insertion Sort possui uma complexidade de tempo de $O(n^2)$, independentemente da distribuição dos elementos no vetor, por isso tem uma vantagem em relação ao Quicksort para vetores pequenos ou quase ordenados, pois seu desempenho é melhor em casos onde o vetor já está quase ordenado. Já o Quicksort é geralmente preferido em termos de desempenho em casos médios, onde a distribuição dos elementos é aleatória ou decrescente, devido à sua complexidade média de tempo de O(n log n). A escolha do algoritmo adequado depende das características do vetor e dos requisitos específicos do problema. Em alguns casos, pode ser necessário considerar outros algoritmos de ordenação que não foram análisados.