Felipe Carrancho da Fonseca Rocha

Matrícula: 202110049611

Trabalho Projeto e Análise de Algoritmo sobre ordenação

Nova Friburgo – 2023

Introdução

Neste trabalho iremos realizar e analisar dois métodos de ordenação de elementos o Quick sort e o Insertion sort. O Quick Sort e o Insertion Sort são algoritmos que adotam abordagens diferentes para ordenar uma lista de elementos. O Quick Sort é reconhecido por sua eficiência em listas extensas, fazendo uso da estratégia "dividir para conquistar". Em contraste, o Insertion Sort é um método mais simples, porém eficaz, especialmente quando aplicado a listas menores ou quase ordenadas. Desta forma iremos analisar e comparar suas complexidades.

Metodologia

Descrevendo o algoritmo

O algoritmo realizado compara o tempo de execução dos algoritmos de ordenação Quick Sort e Insertion Sort para diferentes tamanhos de vetor de entrada(50, 500, 5000 e 50000).

O Quick Sort é um algoritmo de ordenação eficiente que segue a abordagem “dividir e conquistar”. Ele seleciona um elemento "pivô" (pivot) e rearranja os elementos do vetor de modo que todos os elementos menores que o pivô fiquem à sua esquerda e os elementos maiores fiquem à sua direita. Em seguida, o algoritmo é aplicado recursivamente nas duas metades do vetor, ou seja, na metade à esquerda do pivô e na metade à direita do pivô, até que o vetor esteja completamente ordenado.

O Insertion Sort é um algoritmo de ordenação simples que percorre o vetor da esquerda para a direita, inserindo cada elemento em sua posição correta em relação aos elementos anteriores. Ele mantém uma parte ordenada do vetor à esquerda, onde os elementos são inseridos sequencialmente na posição adequada, enquanto a parte restante do vetor à direita contém os elementos não classificados.

O código implementa as funções insertion_sort e quick_sort para realizar a ordenação dos vetores. Em seguida, ele relaciona os tamanhos dos vetores especificados na lista Tamanhos. Para cada tamanho, são criados três tipos de vetor: crescente, decrescente e

aleatório. O tempo de execução de cada algoritmo (Quick Sort e Insertion Sort) é medido para cada tipo de vetor.

Por fim, o código utiliza a biblioteca Matplotlib para plotar gráficos comparativos dos tempos de execução dos algoritmos para os diferentes tamanhos de vetor e tipos de vetor (crescente, decrescente e aleatório). Cada gráfico mostra a comparação entre os tempos do Quick Sort e do Insertion Sort.

Linguagem de Programação usada

Foi utilizada a linguagem de programação python pela existência de certas bibliotecas que facilitam a demonstrar os resultados obtidos como o timeit para mostrar o tempo que demorou para rodar os testes e a biblioteca matplotlib para gerar os gráficos que iremos analisar ainda neste relatório. Tudo isso foi feito usando o Visual Studio Code.

Configuração do Computador Usado Para os Testes

O sistema operacional é um windows 10 e a configuração de hardware do meu notebook é um Nitro 5 modelo AN515-54, com um processador intel core i5 de nona geração, 8GB de memória RAM e uma placa de video da NVIDIA GeForce GTX 1650.

Resultados

O algoritmo todo demorou cerca de 8 minutos para gerar os gráficos. Para ficar mais visível o tempo demorado por cada tamanho foi usado a biblioteca timeit e um print no python para falar quanto tempo em segundos cada vetor teve seu tempo de execução, aqui está os seus resultado na Tabela 1:

Tabela 1: Tempo de execução em segundos em cada tipo de vetor.

Tamanho dos vetores	Tipo do Gráfico	Tempo Quick Sort (segundos)	Tempo Insertion Sort (segundos)
50	Crescente	0.0001600999967195093	7.800001185387373e-0
50	Decrescente	0.0001141999964602291	0.0001283999881707131

50	Aleatório	4.4899992644786835e-05	6.659998325631022e-05
500	Crescente	0.01529469998786226	6.339995888993144e-05
500	Decrescente	0.011218899977393448	0.016671699995640665
500	Aleatório	0.0005475999787449837	0.00689979997696355
5000	Crescente	1.5009609999833629	0.0009414000087417662
5000	Decrescente	1.014320400019642	1.3899665000499226
5000	Aleatório	0.008116799988783896	0.7153436999651603
50000	Crescente	152.50629109999863	0.006997099961154163
50000	Decrescente	100.87385040003574	140.18466960004298
50000	Aleatório	0.1028775000013411	70.56892270001117

Agora vamos para os resultados de cada gráfico.

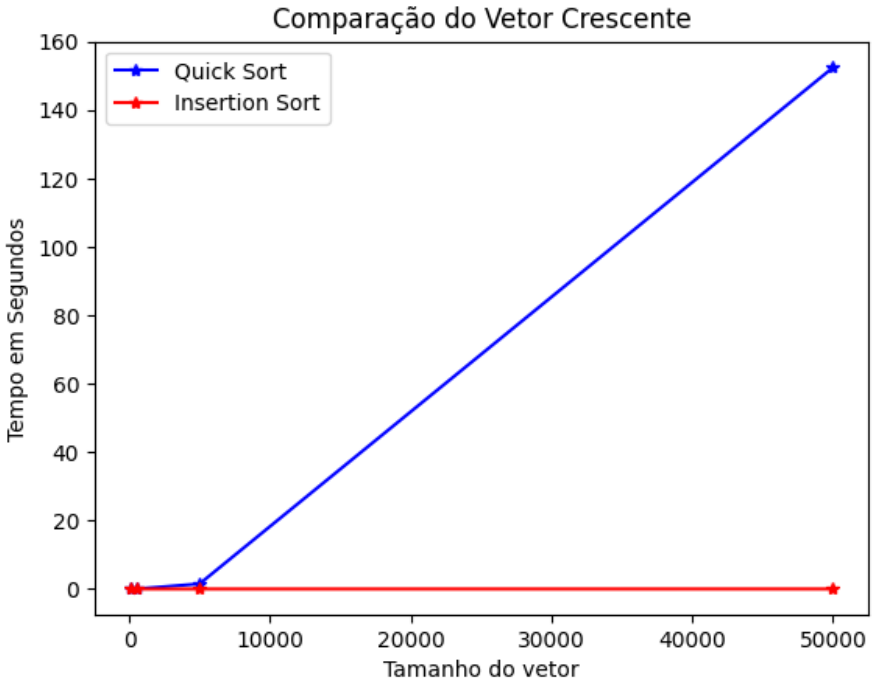
Crescente

Na primeira análise, o Quick Sort apresenta um desempenho não ideal quando aplicado a um vetor ordenado de forma crescente. Isso ocorre devido às chamadas recursivas desnecessárias e à divisão desequilibrada do vetor. Essa abordagem leva a um desempenho menos eficiente em comparação com outros algoritmos de ordenação.

Já na análise do Insertion Sort em um vetor já ordenado de forma crescente, o algoritmo demonstra um desempenho ideal, visto que ele já está em uma ordem desejada. Quando está corretamente ordenado suas condições sempre estarão satisfeitas fazendo com que evite trocas de elementos e possua um tempo de execução de quase milésimos de segundos.

Em resumo, o Quick Sort é menos eficiente em listas ordenadas devido a chamadas recursivas desnecessárias, enquanto o Insertion Sort mostra um bom desempenho evitando chamadas a mais e sem trocas de elementos nesse cenário como visto na Figura 1.

Figura 1:Comparação entre Quick Sort e Insertion Sort em vetor Crescente.

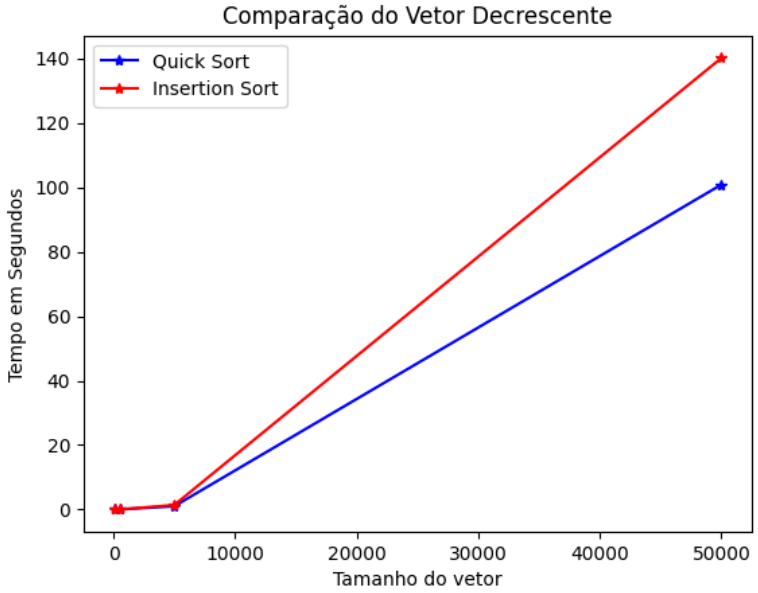


Decrescente

No caso da ordenação do Quick Sort em um vetor decrescente, as partições desequilibradas levam a um desempenho próximo ao pior caso, com complexidade de tempo próxima a O(n^2).

O Insertion Sort também apresenta um desempenho bem ruim neste cenário, com uma quantidade significativa de deslocamentos em cada iteração. Ambos os algoritmos mostram complexidades de tempo semelhantes, mas o Insertion Sort é mais demorado devido às repetições desnecessárias assim ficando atrás do Quick Sort como pode ser observado na Figura 2.

Figura 2:Comparação entre Quick Sort e Insertion Sort em vetor Decrescente.



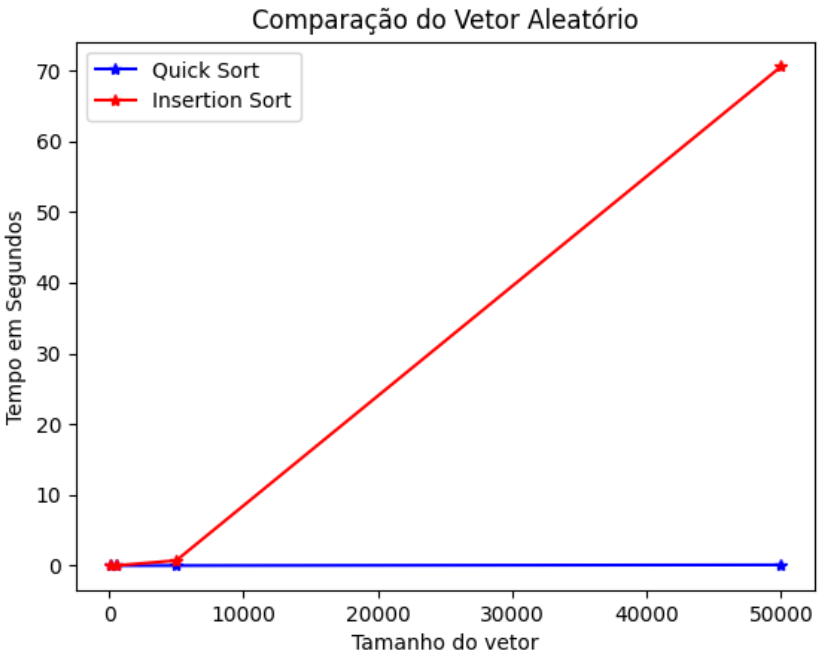
Aleatório

Ao ordenar um vetor aleatório, o Quick Sort demonstra um desempenho eficiente e próximo do melhor caso, devido à sua estratégia de dividir e conquistar e à escolha equilibrada do pivô que contribui para um bom desempenho e sua complexidade média de tempo é O(n log n).

Por outro lado, o Insertion Sort apresenta um desempenho mais lento, com uma complexidade de tempo de O(n^2), devido ao número significativo de movimentos e trocas necessários para inserir e comparar cada elemento .

Na Figura 3 ilustra claramente a diferença de desempenho entre o QuickSort e o Insertion Sort. Enquanto o QuickSort demonstra uma eficiência excepcional, o Insertion Sort se torna mais demorado à medida que lidamos com valores elevados.

Figura 3:Comparação entre Quick Sort e Insertion Sort em vetor aleatório.



Conclusão

Com isso a conclusão que conseguimos observar é de que o algoritmo QuickSort é a melhor opção para ordenação, sendo eficiente em vetores aleatórios e tamanhos elevados e ser ineficiente para vetores decrescentes em termos de tempo de execução. O Quick Sort também apresenta complexidade O(n^2) no pior caso (vetores crescentes) e O(n log n) para vetores aleatórios. Já InsertionSort tem complexidade O(n) no melhor caso, mas O(n^2) no pior caso e é menos eficiente para vetores aleatórios e grandes.

Referências

https://matplotlib.org/stable/api/markers_api.html

https://acervolima.com/timeit-em-python-com-exemplos/