MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AGRESTE DE PERNAMBUCO

ALGORITMOS E ESTRUTURA DE DADOS I
MÉTODOS DE ORDENAÇÃO

MARIA EDUARDA DEODATO INTERAMINENSE

GARANHUNS/2022

Sumário

Introdução

Atualmente quando temos alguma dúvida sobre algo recorremos para o navegador na barra de busca, e conseguimos um grande volume de dados em segundos. Já imaginou ter que procurar em uma enciclopédia física, é algo que é bastante trabalhoso e também demanda muito tempo. Com isso, os métodos de ordenação são uma ferramenta fundamental para ordenação dos elementos, podendo ser organizados em ordem decrescente ou crescente e dentre outros, esses algoritmos têm como objetivo facilitar e otimizar o tempo de buscas e pesquisas de dados.

Na computação existem inúmeros tipos de algoritmos de ordenação, que organizam um conjunto de dados e todos possuem técnicas distintas, são conhecidos como métodos de ordenação. Esses métodos se classificam como ordenação interna e externa, a interna os dados são armazenados na memória principal e pode ser acessado imediatamente, enquanto a externa os elementos não são armazenados na memória principal por não haver espaço para eles, e são acessados de forma sequencial.

Dentro da ordenação de métodos internos existem duas categorias: a dos métodos simples e os métodos eficientes. Os métodos simples são ideais para ordenar pequenos vetores. Sua complexidade é $C(n) = O(n^2)$ e fazem $O(n^2)$ comparações . Alguns exemplos de métodos simples são Insertion sort, Selection sort, Bubble Sort e Comb sort. Já os métodos eficientes são mais complexos nos detalhes, fazem menos comparações e são ideais para trabalhar com uma quantidade maior de dados, sua complexidade é C(n) O(n log n). Alguns exemplos de métodos eficientes são Quick sort, Merge sort, Heap Sort, Radix sort, Shell sort e dentre outros.

Os tempos de processamento para o pior e melhor casos são importantes. Geralmente uma ordenação vai ter um bom caso médio, mas um terrível pior caso. A ordenação trabalha menos quando a lista já está ordenada, mas quanto mais a lista estiver desordenada maior é o tempo de ordenação. A determinação do quanto a ordenação demora é baseada no número de comparações e trocas que ela deve executar. Com isso, esse trabalho tem o objetivo de testar alguns métodos de ordenação, analisando seu desempenho e a eficiência de cada um.

Metodologia

O ambiente onde foi realizado o teste foi em um notebook com sistema operacional Windows de 64bits através da Oracle máquina virtual (VM). Sem a utilização de outros programas durante os testes as informações sobre o dispositivo e a máquina virtual se encontram nas figuras abaixo com as seguintes informações na figura 1 e as especificações do computador estão na figura 2.

O código foi desenvolvido totalmente na VM e foi utilizado apenas o editor de texto e o terminal do próprio sistema Linux versão (Debian (64-bit)) para o código dos métodos. Foram geradas listas com quantidades de elementos em ordem aleatória para o teste, os valores das listas de elementos gerados foram de 1.000, 10.000, 50.000, 100.000, 500.000 e 1.000.000 através da classe Generator que foi gerada na linguagem JAVA, pelo IntelliJ.

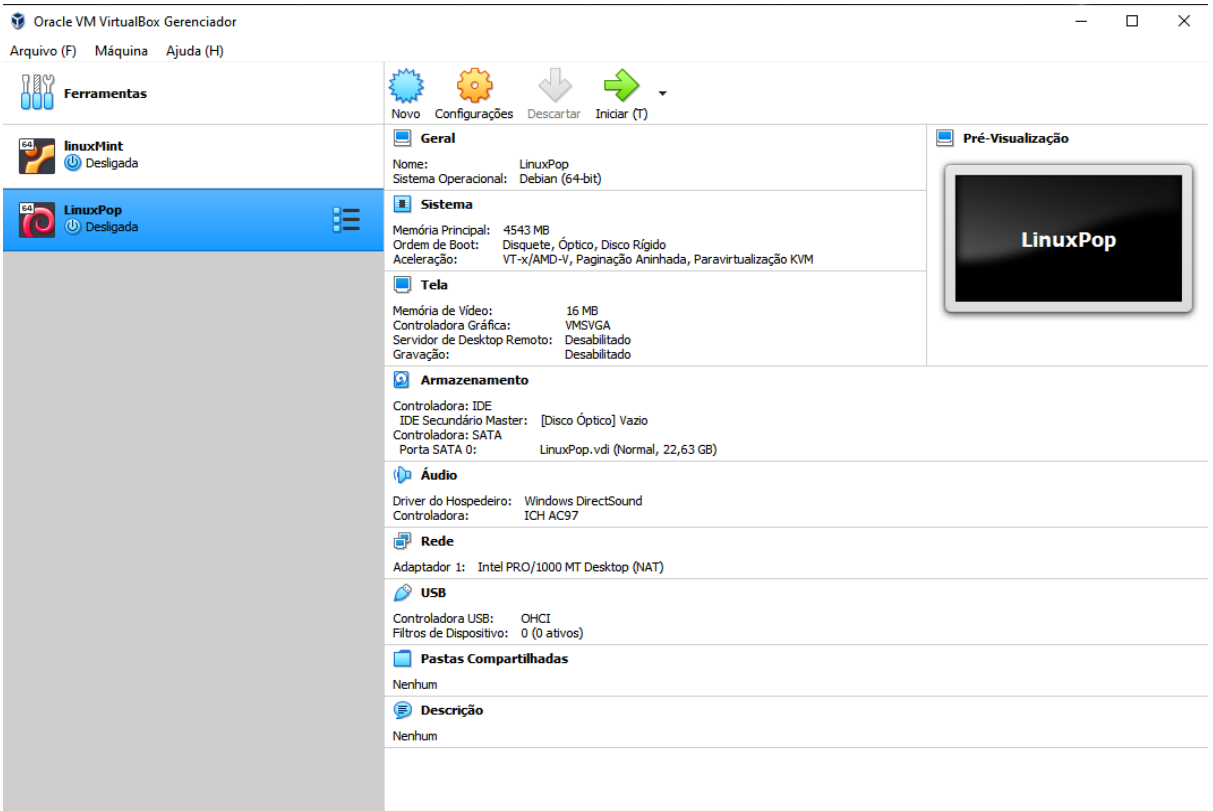

Figura 01 - Especificações da Máquina Virtual

Figura 02 - Especificações do dispositivo

Resultados

Mergesort

É um algoritmo que faz ordenação por intercalação/fusão. A complexidade desse algoritmo de divisão e conquista é O(n log n), sendo mais eficiente que os métodos simples. O processo do merge tem base em dividir o array em dois, cada um com metade dos elementos do array inicial. É esse processo vai sendo reaplicado até que chegue ao seu caso base, que é quando possui apenas um elemento. Com isso a recursão parar. Então os arrays que foram divididos são mesclados em apenas um único array ordenado.

De acordo com os testes, o resultado do desempenho do merge foi o método que ordenou em um menor tempo de execução de todos os métodos testados. Ele é eficiente para listas com grande quantidade de elementos, sua ordenação teve um ótimo rendimento.

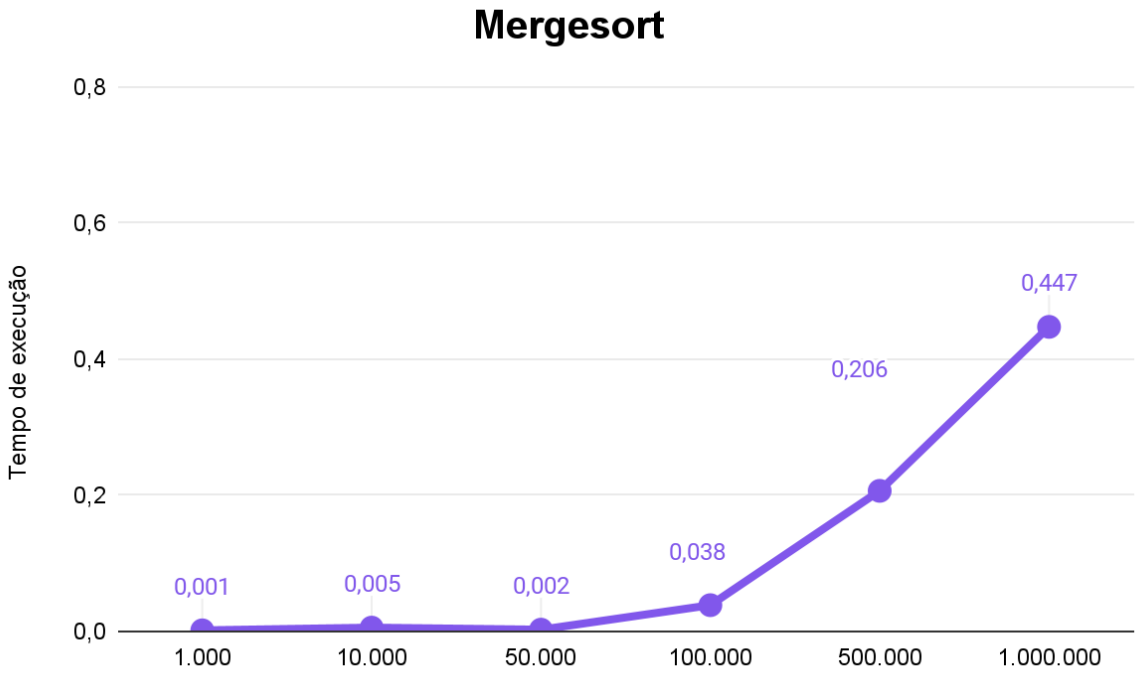

Figura 03 - Gráfico mergesort

Heapsort

Heapsort é um método com um ótimo desempenho e bastante utilizado no gerenciamento de memória nos sistemas operacionais dos computadores por possuir um tempo de execução satisfatória em sequências desordenadas, e por utilizar pouca memória e seu desempenho no pior caso é praticamente igual ao desempenho do caso médio. Isso porque possui complexidade O(n log n). Com isso, torna-se uma ótima possibilidade de utilização.

O método heap teve um desempenho satisfatório, teve pouca diferença em comparação com algoritmo Merge, e grande diferença comparando com os outros métodos. Com isso, esse método é eficiente em casos de ordenação com muitos elementos.

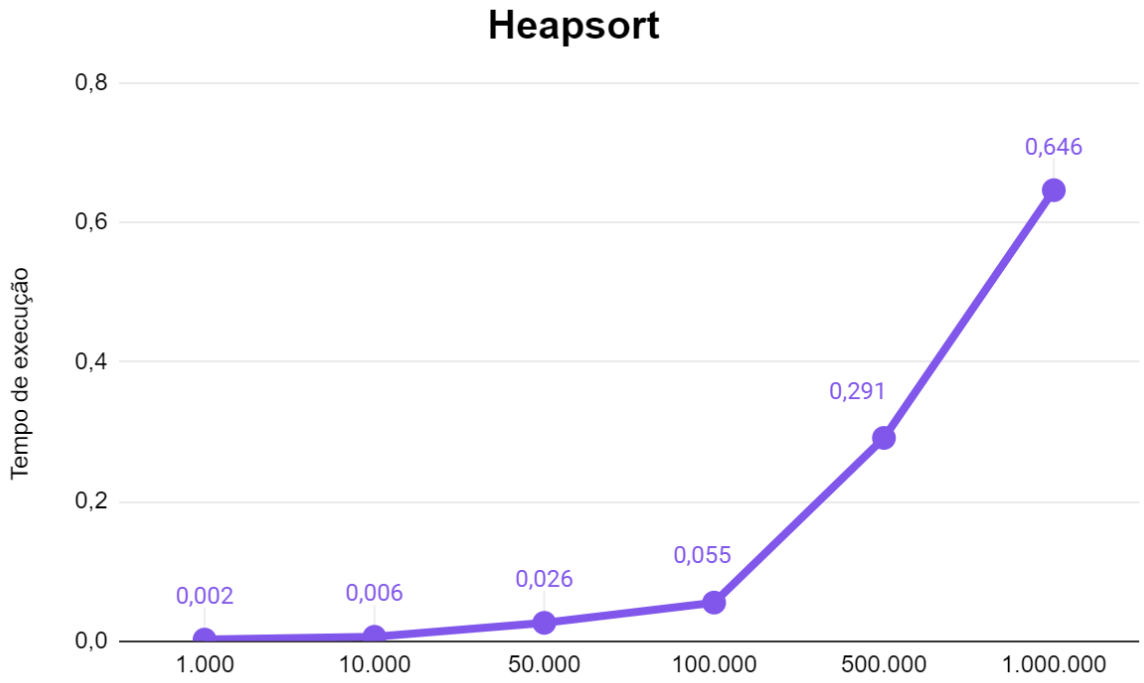

Figura 04 - Gráfico heapsort

Quicksort

Esse método, é considerado a superior de todas as outras ordenações citadas neste relatório, é também considerado o melhor algoritmo para propósito geral atualmente. Esse método ordena os itens por trocas. O quicksort é baseado na ideia de partições. O processo funciona da seguinte maneira escolhe um valor, chamado de “pivô”, e então faz a divisão dos arrays em duas secções, com todos os elementos maiores ou iguais ao valor da partição de um lado valores maiores e do outro menores. Esse processo se repete até que o array esteja ordenado. Mesmo no pior caso o quicksort tem um bom rendimento.

No entanto, há um fator particular problemático do quicksort. Se o valor do pivô, para cada partição, for o maior valor, então esse método se degenera em uma ordenação lenta com um tempo de processamento *n.* Geralmente, isso não acontece.

O resultado desse método foi um surpresa, por ser uns dos métodos mais eficientes pois seu tempo apenas com a lista de 1.000 foi bastante alto em relação à todos os outros exceto o Inserction. O bubble teve um resultado de tempo menor que o Quick. Portanto o teste foi finalizado como pode ser visto na figura 05 testes para o quicksort foram realizados até a lista de 50.000 elementos o qual foi interrompido como mostra na figura 05, o qual demandou aproximadamente 6 minutos enquanto o Merge e o Heap tiveram resultados com o tempo extremamente inferior ao Quick.

Portanto, um segundo teste foi realizado alterando a parte de partição do código, onde foi criada uma função que coloca o pivô em sua devida posição, e faz a partição da lista ele divide a lista em duas e chama recursivamente o quicksort para as duas listas. Com isso, pôde-se analisar na figura 05 o quanto tempo de execução houve uma queda significativa.

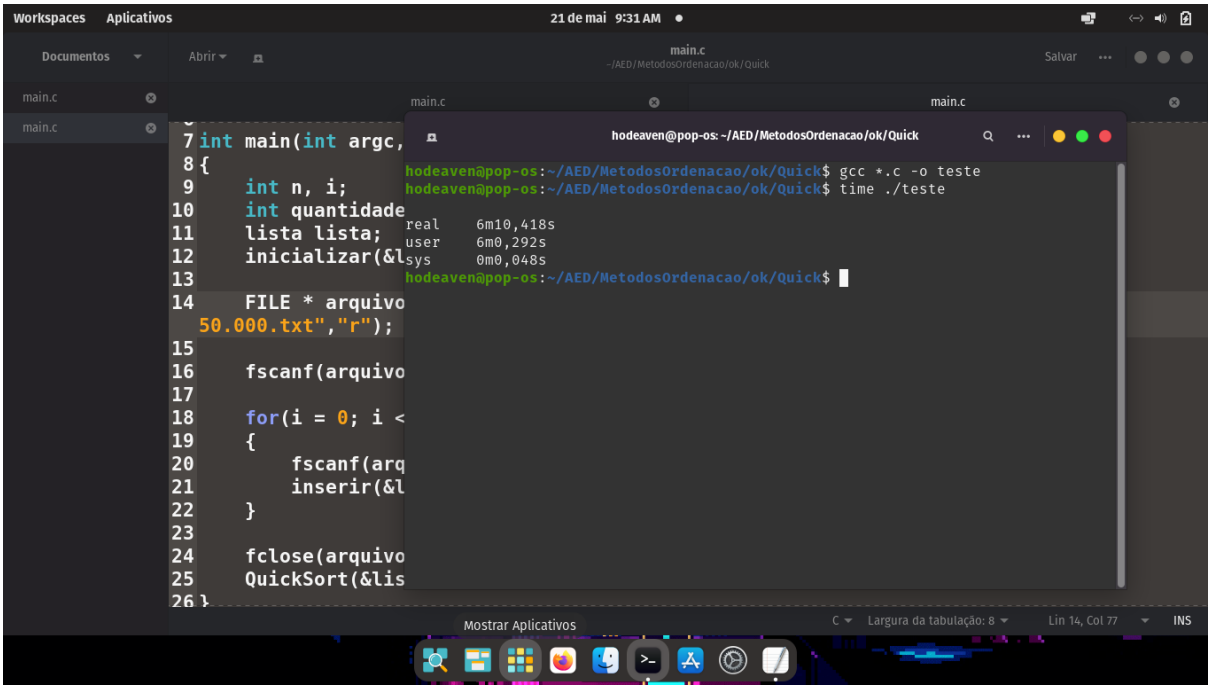

Figura 05 - Encerrando teste quicksort

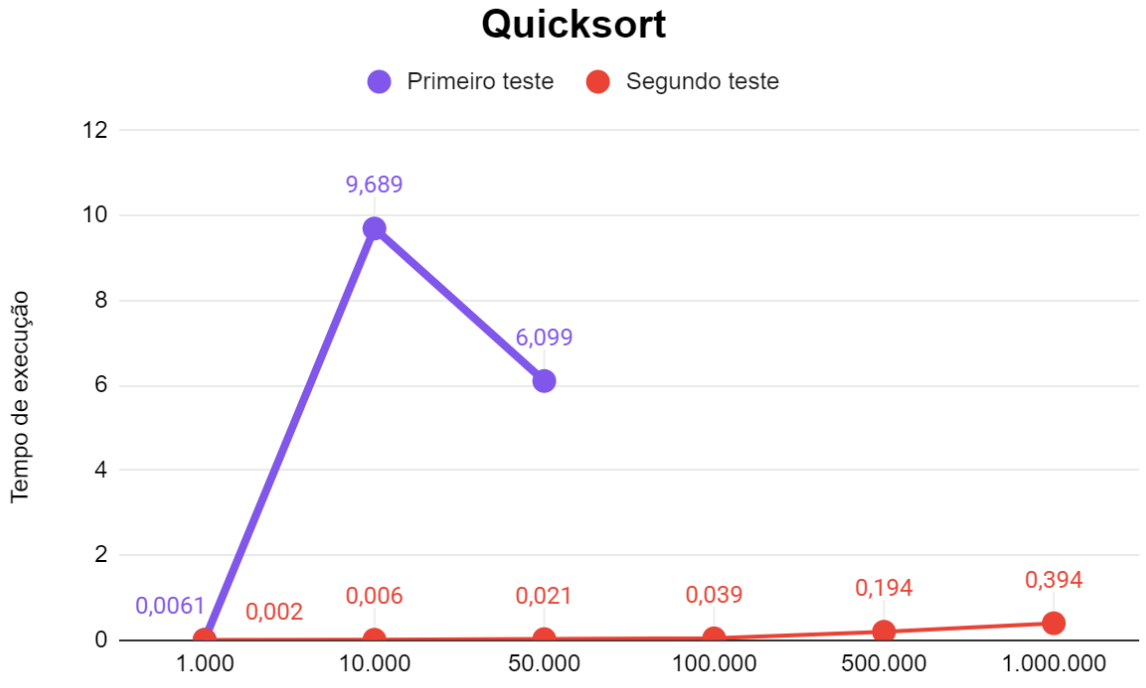

Figura 06 - Gráfico quicksort

Selectionsort

A ordenação por seleção seleciona o elemento de menor valor e troca pelo primeiro elemento. Então, para os n-1 elementos restantes, é encontrado o elemento menor pivô, que troca pelo segundo elemento e assim sucessivamente. As trocas seguem até os dois últimos elementos. Com isso, esse método de ordenação é muito lento para um número grande de elementos comparando com o bubble sort o número de trocas, no caso médio, é muito menor para a ordenação por seleção. Contudo, existem ordenações melhores.

Com o teste realizado nota-se que o tempo de ordenação a partir de 500.000 elementos o tempo de execução tem um alto custo, então não é interessante utilizar esse método para listas acima de 500.000 itens.

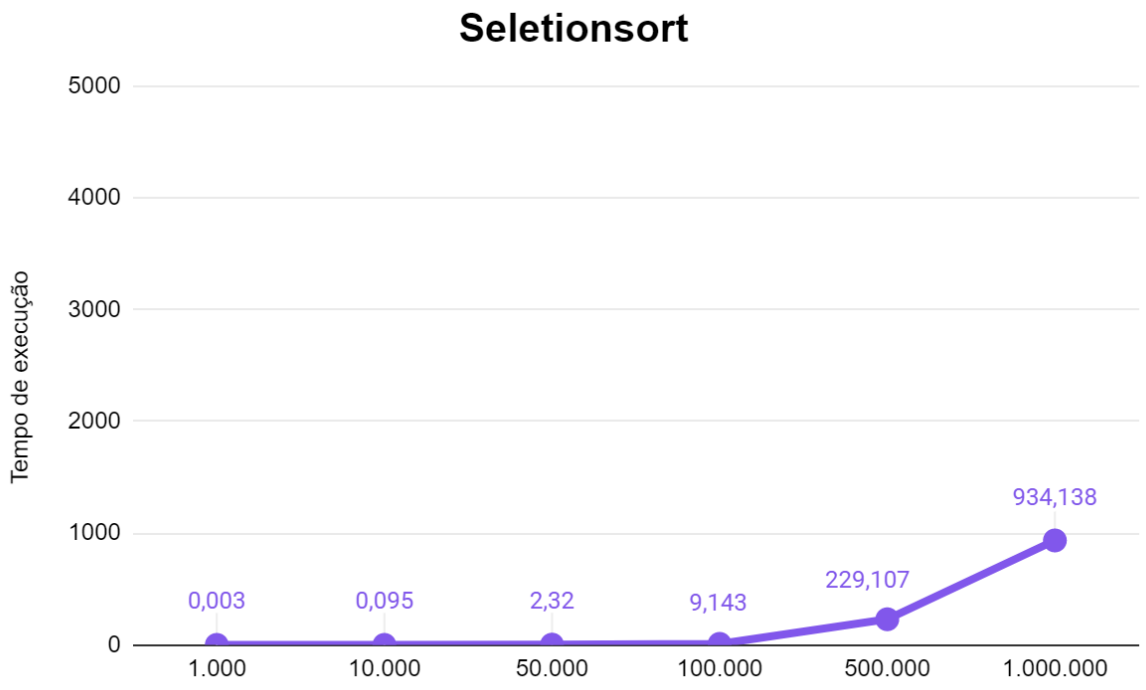

Figura 07 - Gráfico Selectionsort

Insertionsort

A ordenação por inserção é um dos algoritmos simples, de início, esse método ordena os dos primeiros números da posição do array. E logo após, insere o terceiro elemento na sua posição ordenada comparando com os dois primeiros membros. Então, insere o quarto elemento na lista dos três elementos, e assim o processo segue até que todos elementos estejam organizados.

Diferente do bubble sort e selection sort, as comparações ocorrem durante a ordenação por inserção, entretanto depende do estado inicial da lista se está ordenada o número de comparações será n-1. Se estiver fora de ordem o número de comparações será de ½(n^2+n). A equação para cada caso é diferente para o melhor, médio e pior.

O inserction desde a primeira lista de 1.000 elementos demandou muito tempo, ultrapassando de 6min o qual foi encerrado, outro teste foi realizado novamente com a mesma lista de 1.000 elementos e o tempo se estendeu além de 15min, sendo assim o programa e os testes foram interrompidos pelo que foi tempo gasto apenas com a lista de 1.000 elementos.

Entretanto, com o alto tempo de execução do insertionsort, o código foi alterado, especificamente em uma função auxiliar do insertionsort, o “inserir ordem”, onde passou a adicionar o elemento ao final da lista por ele ser maior do que o último elemento da lista, evitando comparações. Com isso, podemos observar na figura 08 quanto tempo de execução reduziu.

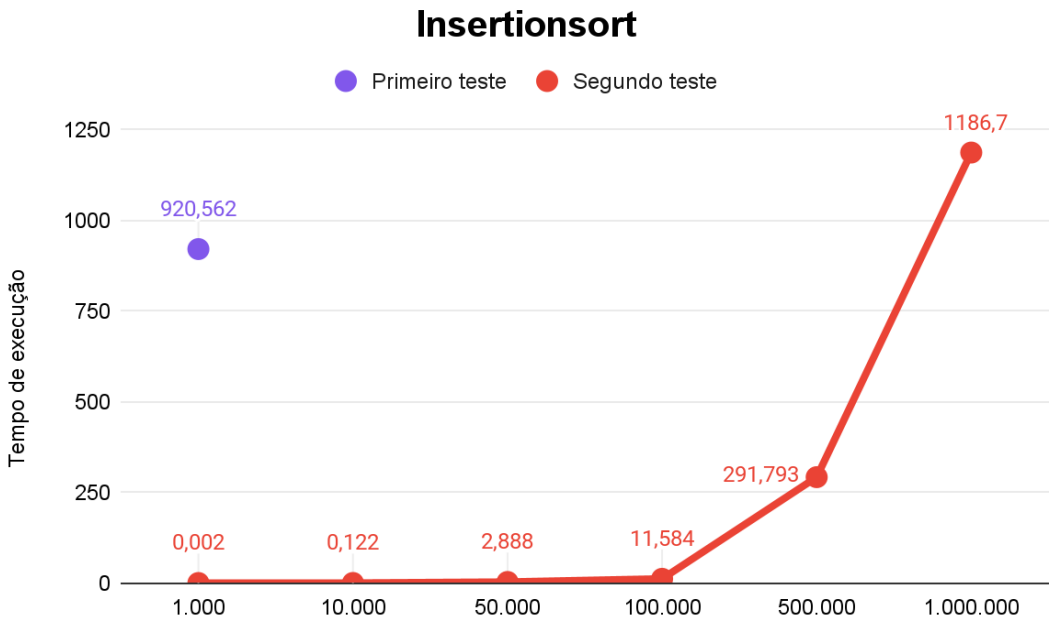

Figura 08 - Gráfico Insertionsort

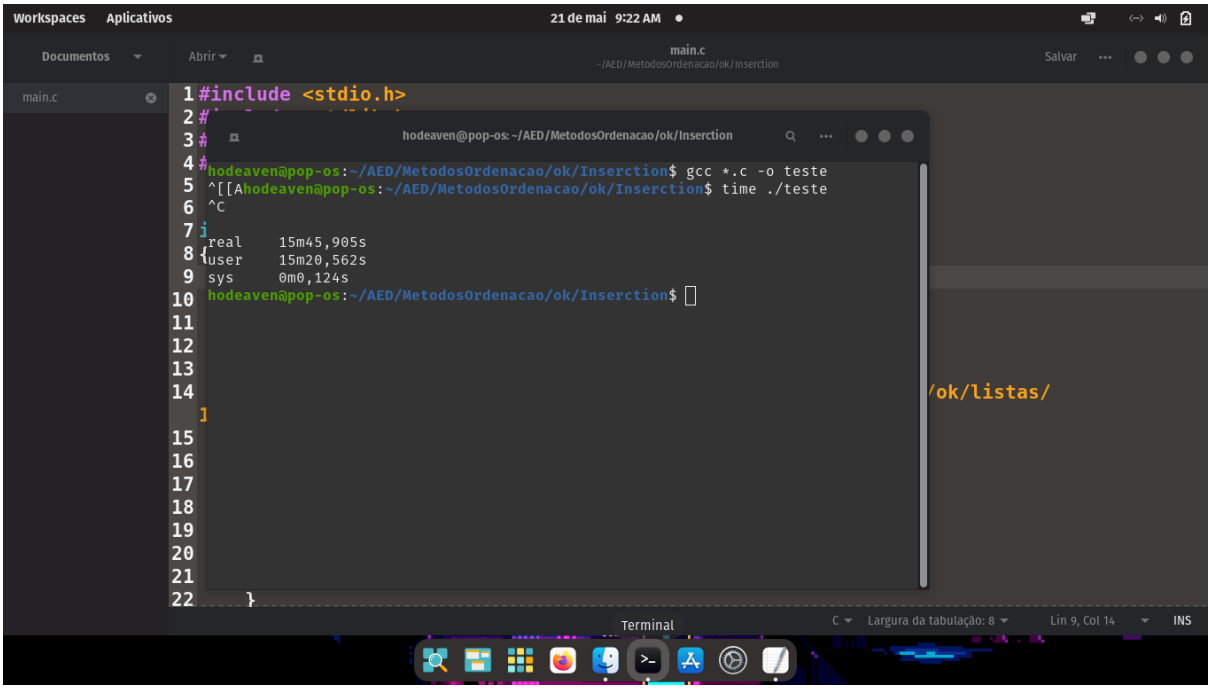

Figura 09 - Encerrando teste Insertionsort

Bubblesort

Bubble sort conhecida como ordenação bolha, é uma ordenação por trocas. Ela faz inúmeras comparações, e dependendo da comparação dos elementos faz a troca. Essa é uma ordenação simples. Pois seu tempo de execução é O(n²). Esse tipo de algoritmo é ineficiente quando o número há uma elevada quantidade de elementos, por isso o tempo de execução está diretamente ligado ao número de comparações e trocas

Com os resultados dos testes, observa-se que o desempenho do tempo de execução do bubble foi o que demandou mais tempo, exceto pelo Quick e Inserction. Com o aumento de itens na lista a partir de 500.000 elementos , houve um grande custo computacional seu tempo de ordenação levou em média de 18 minutos, e mais de 1h para ordenar os dados de 1.000.000. Com isso, esse tipo de método é ineficiente para listas com números altos.

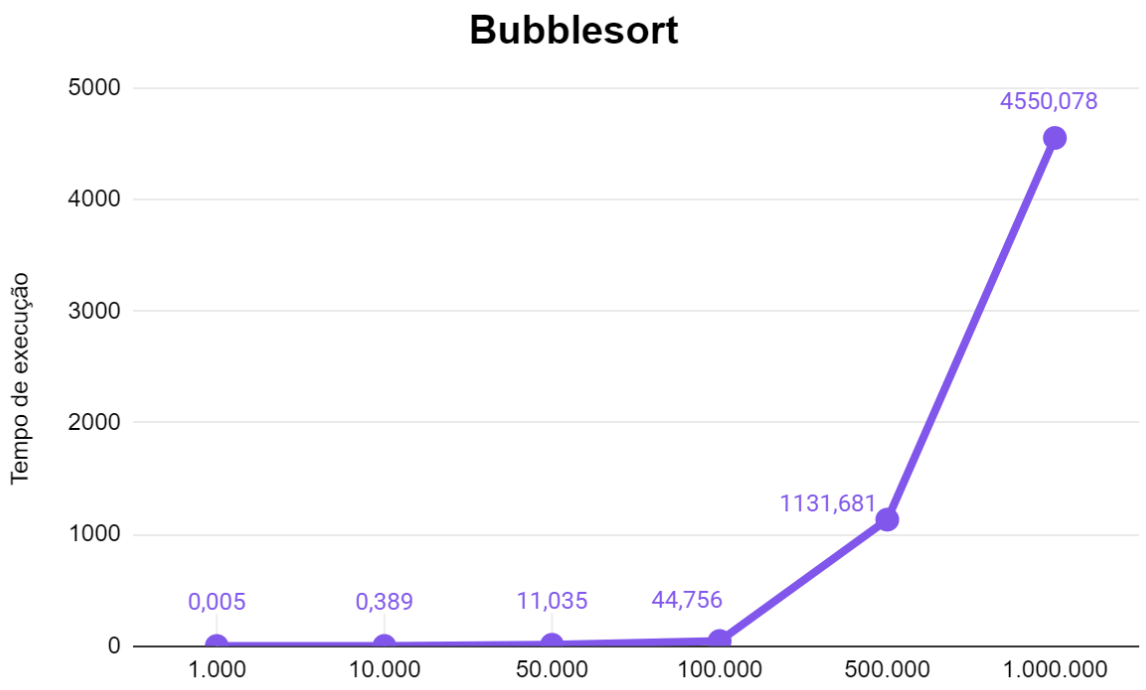

Figura 10 - Gráfico Bubblesort

Considerações finais

Com as análises dos dados dos testes podemos ver o desempenho de cada método, sendo assim alguns conseguem ter um bom desempenho na maioria dos casos, enquanto outros não. Não sendo satisfatório para tais casos. Esta análise é necessária para escolher o algoritmo mais adequado para resolver um dado do problema que tenha o menor custo computacional.

Foi formidável fazer os testes e ver os resultados e analisar todos os dados coletados. Principalmente sobre a problemática do Quicksort que foi algo frustrante no momento, mas após foi interessante, por motivos que gerou a curiosidade sobre o caso particular, e em que tive oportunidade de compartilhar com os colegas de turma que gerou discussões de possibilidades de novos testes, e teorias do quicksort não ter ordenado em um tempo de execução menor, apesar do pouco tempo foi uma conversa bastante interessante.

Diante de todos resultados, foi realizada uma nova análise considerando os testes que tiveram resultados diferentes do esperado, como o quicksort e o insertionsort. Portanto, algumas partes do código foram alteradas que mudaram o resultado anterior, onde podemos observar na figura 11 o quanto o desempenho do insertion teve uma redução de tempo gasto na ordenação, após as alterações o insertionsort se destacou, e se mostrou mais eficiente do que os outros métodos simples: seletionsort e o bubblesort.

Do mesmo modo, o quicksort sobressaiu com o melhor tempo de execução entre os métodos complexos testados, com 24,4% do tempo total de execução. De início os três métodos são equivalentes, no entanto, com o crescimento de elementos o quicksort apresentou um menor tempo de execução mesmo com o aumento de elementos e demonstrou seu desempenho eficiente para listas com uma quantidade maior de elementos.

Com isso, foram gerados novos conhecimentos com os erros cometidos e como eles afetaram o tempo de execução e desempenho dos métodos, como pode ser observado no gráfico do desempenho geral de todos os métodos na figura 15 e o tempo de execução na figura 16. Além disso, também adquiri a experiência de fazer uma análise de dados e entender mais sobre o conceito de ciência de dados, também foi possível aplicar meus conhecimentos e me desenvolver em utilizar outras plataformas para a formação deste relatório, como as ferramentas: Google Analytics, Google sheets e estatística.

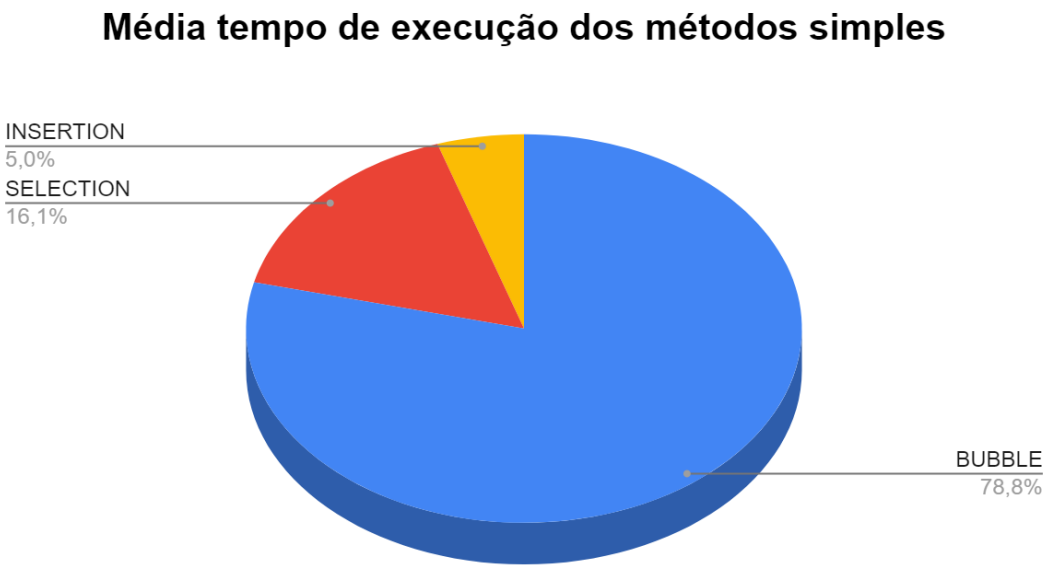

Figura 11 - Gráfico média tempo de execução métodos simples

Figura 12 - Gráfico média tempo de execução métodos complexos

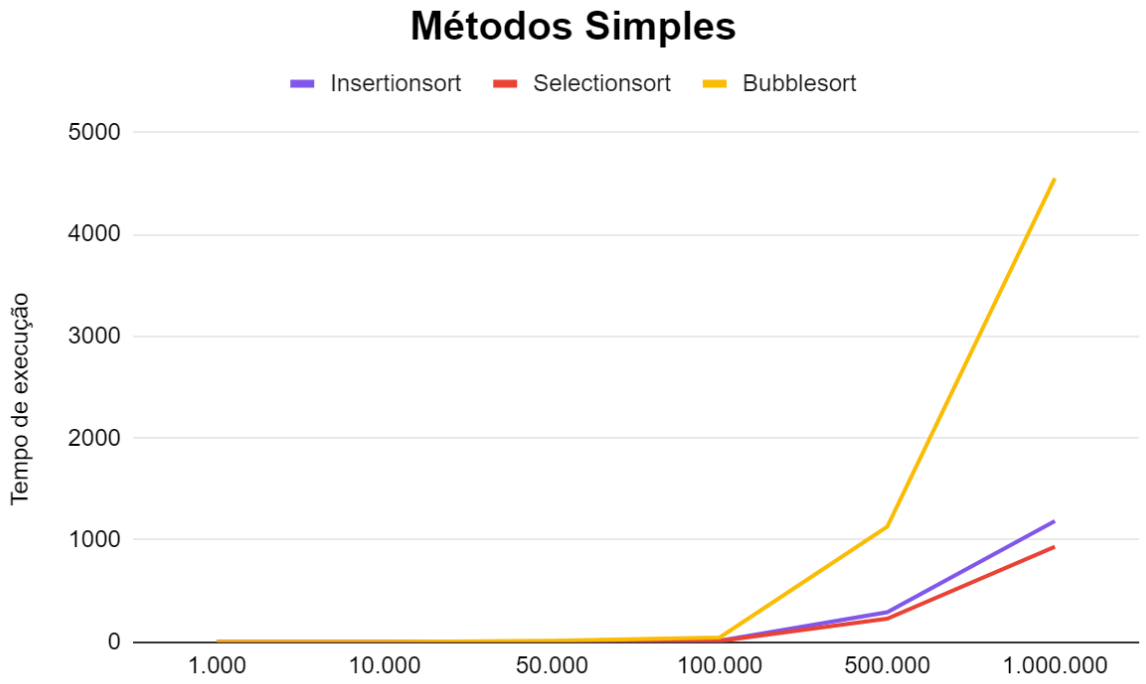

Figura 13 - Gráfico métodos simples

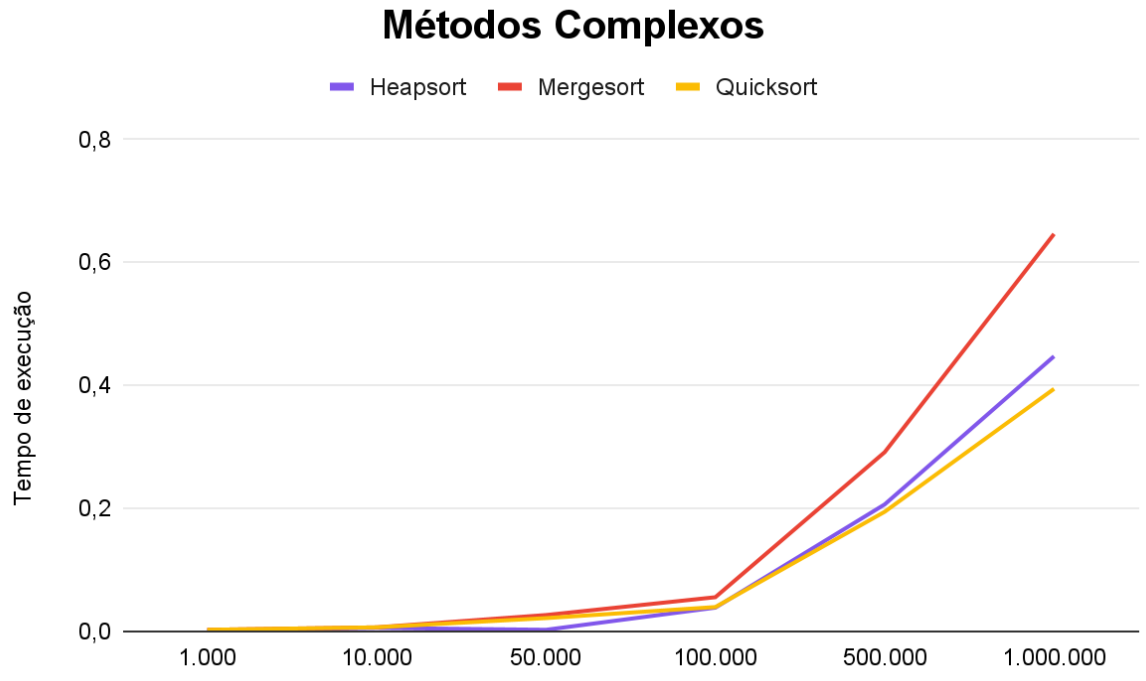

Figura 14 - Gráfico métodos complexos

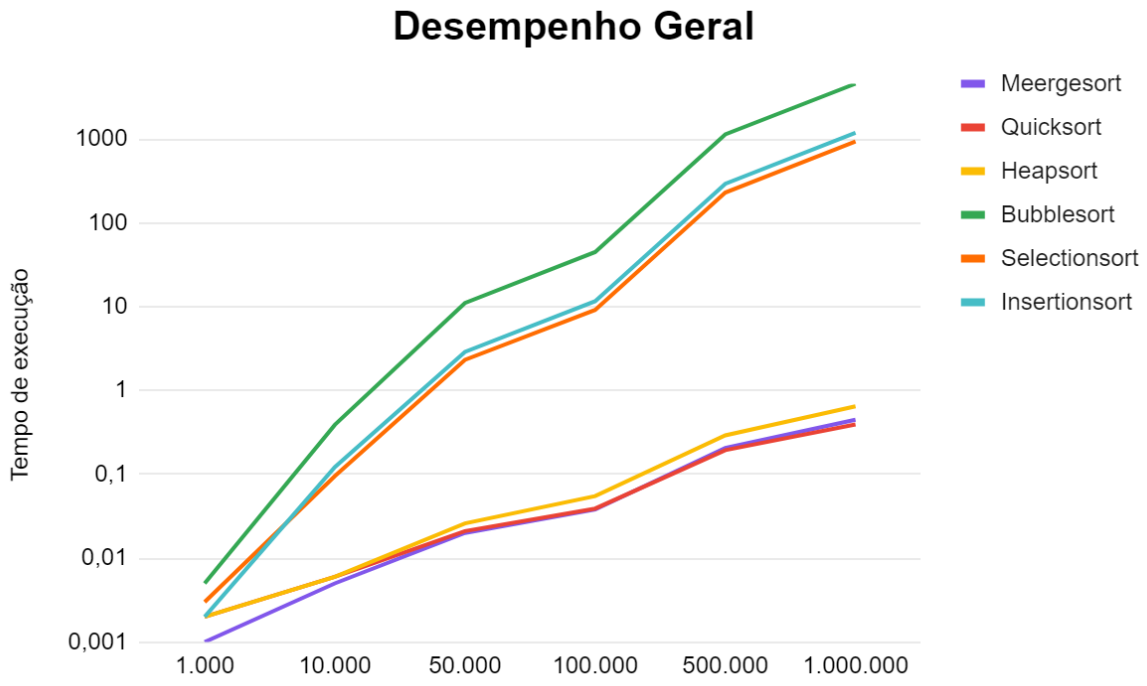

Figura 15 - Gráfico desempenho geral

n° elementos	Merge	Quick	Heap	Bubble	Selection	Inserction
1.000	0,001	0,002	0,002	0,005	0,003	0,002
10.000	0,005	0,006	0,006	0,389	0,095	0,122
50.000	0,002	0,021	0,026	11,035	2,32	2,888
100.000	0,038	0,039	0,055	44,756	9,143	11,584
500.000	0,206	0,194	0,291	1131,681	229,107	291,793
1.000.000	0,447	0,394	0,646	4550,078	934,138	1.186,70

Figura 16 - tabela tempo de execução dos métodos geral