|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Curso**: Engenharia de Computação | | **Disciplina**: Estruturas e recuperação de dados B | | |
| **Período**: 04 | **Turma**: 01 | | **Turno**: Integral | **Data**: 15/10/2020 |
| Nome  1 - Guilherme  2 - Lucas Grass Beraldo  3 – Ramon Gazoni Lacerda  4 – Renato Barba dos Santos | | | | RA  19264266  19246925  20062162  19246529 |

1. **INTRODUÇÃO**

Nesse relatório abordaremos 5 algoritmos de ordenação, o Selection Sort, o Insertion Sort, o Merge Sort, o Quick Sort e o Heap Sort. Antes de abordá-los, iremos discutir um pouco do objetivo desses algoritmos e a busca pelos programadores do algoritmo ideal.

Métodos para a organização dos nossos dados ou objetos existem a séculos e os fazemos por um objetivo bem simples, a otimização de processos. Com a criação dos computadores e a possibilidade de a organização ser feita por uma máquina, métodos de ordenação começaram a ser pensados documentados. Em meados de 1951 Betty Holberton já estava trabalhando com algoritmos de ordenaçãoprimitivos e em 1956 o Bubble Sort chega a ser implementado. Com o aumento da tecnologia, estudos e quantidade dados, os algoritmos foram se aperfeiçoando, buscando serem cada vez mais rápidos. Apesar dos programadores e acadêmicos nem sempre conseguirem fazer algoritmos rápidos em todos os quesitos, eles melhoravam alguns aspectos com novas abordagens, como a busca da informação ou a organização para poucos números, ou a organização estável de um grupo de dados. Portanto, os algoritmos de ordenação não devem ser vistos como um melhor que o outro, mas deve-se analisar o problema proposto e identificar qual serve melhor para o problema em questão.

Mesmo assim os programadores continuam num estudo contínuo para desenvolver soluções mais rápidas que as atuais. Essa busca incessante ocorre paralelamente com o constante aumento do volume de dados que tratamos atualmente, fazendo com que busquemos maneiras mais rápidas para fazer com que esse aumento não prejudique os processos. Uma simples otimização de processo pode trazer ganhos consideráveis, seja no campo acadêmico ou empresarial. Nesse contexto tratamos de algumas dessas importantes metodologias de ordenação que foram criadas desde as primeiras implementações para organização de dados em computadores até códigos considerados os mais importantes do século XXI.

1. **RESULTADOS E DISCUSSÃO**

**2.1. SELECTION SORT**

O Selection Sort é um algoritmo com tempo linear, que garante a cada execução a ocupação da posição P = Número da Execução - 1 pelo menor elemento da parte não ordenada do Vetor, a cada execução o algoritmo busca pelo menor elemento da parte não ordenada do vetor, e coloca ele na primeira posição da parte não ordenada ou seja primeiro ele busca o menor elemento e depois troca esse elemento com a posição P = Número da Execução - 1

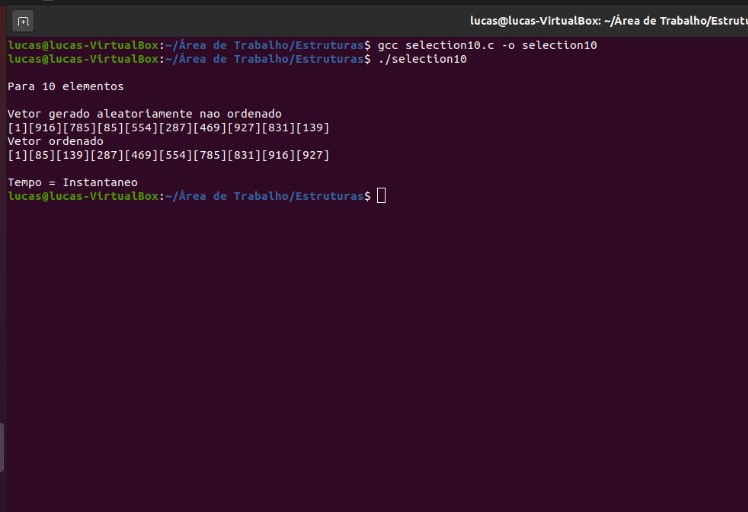
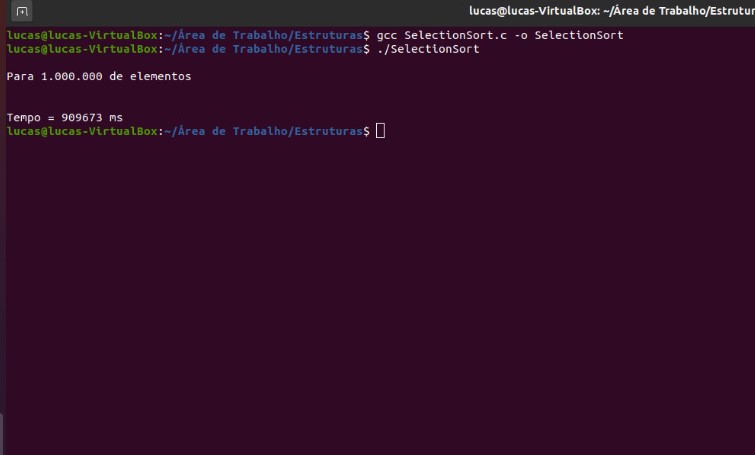


Figura 1. Selection Sort com 10 elementos.

Figura 2. Selection Sort com 1 milhão de elementos.

**2.2. INSERTION SORT**

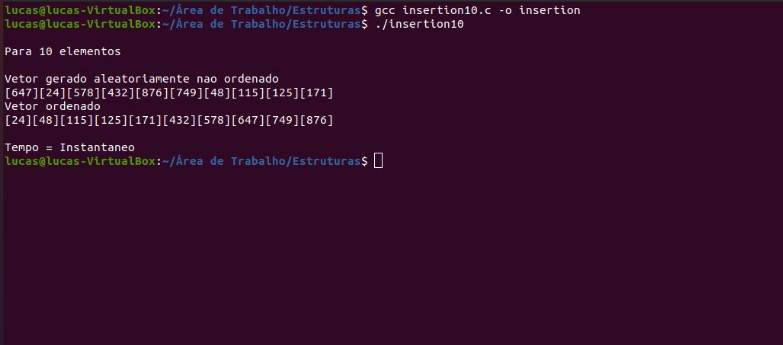
O Insertion Sort é um algoritmo com tempo linear, que garante a cada execução a ocupação da posição P = Número da Execução - 1 pelo menor elemento da parte não ordenada do vetor, sendo assim a cada execução ele compara o elemento da posição i+1 com o elemento da posição i, caso i+1 for < do que i, troca os elementos do vetor de posição e subtrai 1 em cada, caso não for menor, interrompe o processo de comparação e vai para a próxima posição**.**

Figura 3. Insertion Sort com 10 elementos.

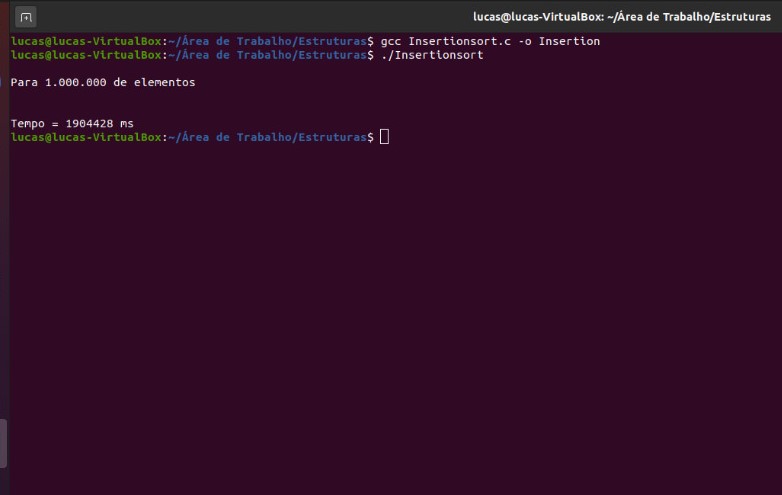


Figura 4. Insertion Sort com 1 milhão de elementos.

**2.3. MERGE SORT**

O Merge Sort é um algoritmo que se baseia em uma estratégia militar de dividir para conquistar **[2]**, com isso, o Vetor é dividido na metade e cada sub grupo é dividido no meio até todos tiverem apenas um elemento. Essa parte do algoritmo é feito a partir de recursão chamando o próprio Merge Sort do início até a metade e da metade + 1 até o fim. Ao fim das chamadas recursivas, cada chamada vai para função merge que faz voltar os sub grupos e ordena eles, ou seja, quando juntar dois sub grupos de um elemento eles são ordenados e forma um sub grupo de dois elementos e sucessivamente até chegar em dois sub grupos apenas, isso se repete até fechar a última recursão.

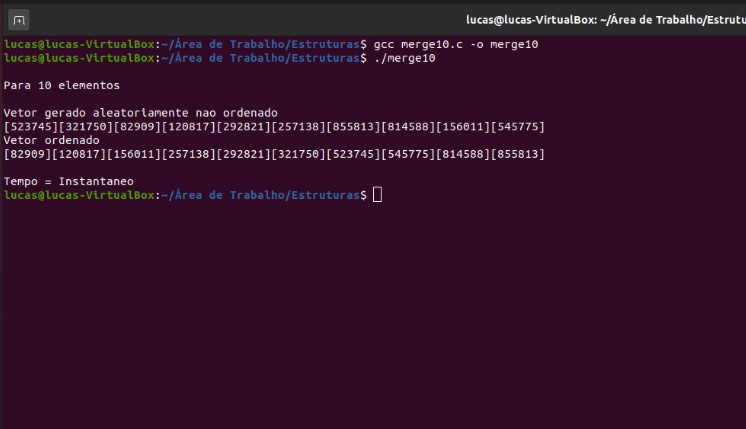
Com as duas metades ordenadas o Vetor passa novamente pela função merge para juntar as últimas metades e ordenar elas. Normalmente, a função merge é feita com dois sub vetores, um com a metade da esquerda e outro com a metade da direita, mas a equipe escolheu utilizar apenas um sub vetor e duas variáveis auxiliares, para simular os dois vetores, e andar com essas duas variáveis de acordo com as condições para o algoritmo decorrer menos tempo e ocupar menos espaço na memória. **[3]**

Figura 5. Merge Sort com 10 elementos.



Figura 6. Merge Sort com 1 milhão de elementos.

**2.4. QUICK SORT**

O Quick Sort foi desenvolvido por Charles Antony Richard Hoare 1960 em uma tentativa de traduzir russo para inglês, o primeiro desenvolvimento do algoritmo foi apenas teórico pois na época não foi possível implementa –lo sendo implementado a primeira vez em 1980 e rendendo ao seu desenvolvedor um Turing Award, prêmio concedido todos os anos para uma pessoa por contribuição a computação. O algoritmo foi amplamente difundido após o estudo e refinamento feito pelo Robert Sedgewick, Professor da universidade de Princeton.

O primeiro passo para garantir uma boa execução do Quick Sort é realizar um Shuffle **[1]**, processo que embaralha de forma aleatória os elementos do vetor, pois diferente do Merge Sort ele não se beneficia de uma pré ordenação dos dados, após esse processo ele utiliza uma estratégia semelhante ao do Merge Sort de conquistar e dividir, ele seleciona um pivo e o utiliza para dividir o vetor em duas partes. O algoritmo separa na esquerda (parte da posição 0 até o pivo) os elementos menores que o pivo e na direita (parte do pivo até a posição final), entrando em uma recursão que continua dividindo o vetor até chegar em um unitário, ou seja um vetor de apenas um elemento, voltando e dividindo novamente os vetores restantes sem considerar a parte já ordenada, garantido a ordenação completa dos dados, é possível fazer muitas melhorias no código de acordo com cada problema acelerando ainda mais sua execução.

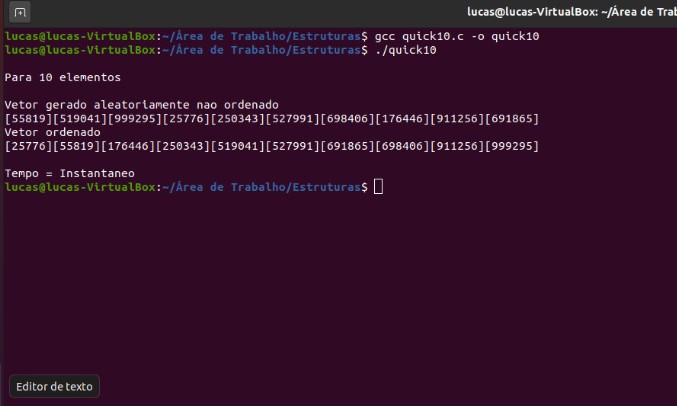
Este algoritmo é muito mais rápido para ordenar vetores grandes do que algoritmos vistos anteriormente, como Selection Sort e Insertion Sort, pois apesar de ter uma complexidade n log n, que seria maior do que uma complexidade linear, por dividir em vários subvetores o n diminui muito reduzindo muito o número de comparações e trocas, realizando um ordenção de 1 milhão de dados instantaneamente. Enquanto o Selection ou o Insertion demorariam de 20 até 40 minutos.

Figura 7. Quick Sort com 10 elementos.

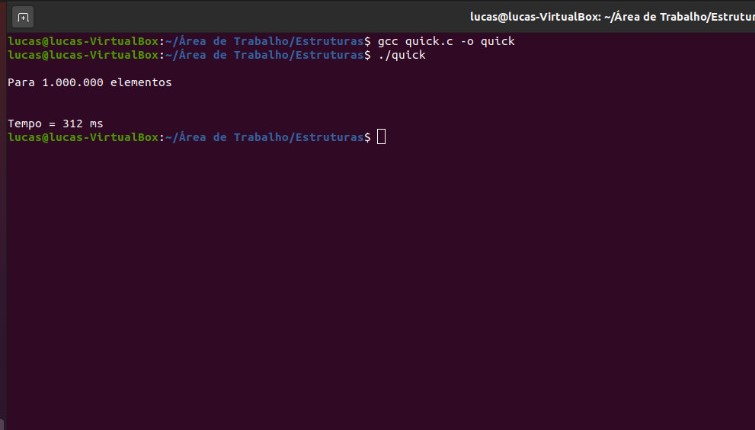


Figura 8. Quick Sort com 1 milhão de elementos.

**2.5. HEAP SORT**

O Heap Sort é um método de ordenação bastante eficiente, no qual se utiliza de uma estrutura de dados chamada heap. Essa estrutura de dados simula uma árvore em um vetor, e isso traz um grande diferencial para a ordenação, poder acessar e navegar pelo vetor como se fosse a estrutura de árvore, porque assim, podemos buscar elementos e varrer a árvore num tempo de aproximadamente logN. Ao incorporar esse comportamento o Heap Sort torna-se capaz de fazer sua ordenação com um número de NlogN comparações e trocas, mesmo no pior caso, tornando-se um dos seus diferenciais. Além disso, o algoritmo em questão utiliza somente a memória alocada pelo vetor, mas não é um método de ordenação estável, é iterativo e de média dificuldade de implementação, comparado com os demais algoritmos de ordenação.

Para que se entenda um pouco mais sobre o Heap Sort o descreveremos um pouco. Iniciamos com vetor desordenado e nosso primeiro passo é implementar uma regra, todas árvores – sendo ela a principal ou uma sub arvore dentro do vetor, devem ter o número do pai maior que dos seus filhos. Para isso percorremos as sub árvores fazendo essa análise e se a regra não estiver sendo cumprida afundamos o pai, ou seja, o trocamos com o maior filho entre os dois, e se esse novo filho tiver outros 2 filhos também verificamos se é cumprida a regra e fazemos isso até chegarmos na folha. Após percorrermos essas sub árvores e chegarmos no pai (ou raiz) e fizermos a análise para ele, teremos todo vetor com essa regra verificada. Como os filhos são menores que o pai temos certeza que a raiz (ou pai central) de toda árvore é o maior número daquele vetor, logo retiramos ele e o imprimimos. Para que o primeiro espaço não fique vazio pegamos o último filho, colocamos ele na primeira posição e o afundamos. Ao afunda-lo novamente podemos garantir que o maior valor estará no pai e por aí vai, até terminar o vetor.

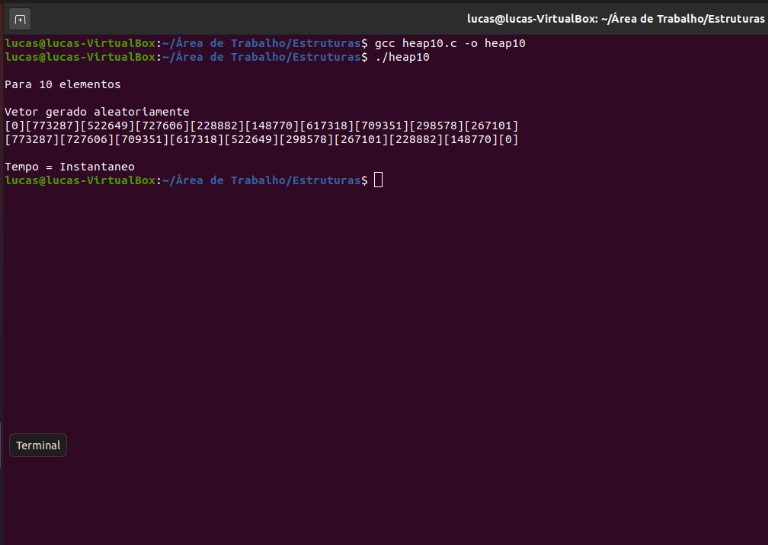
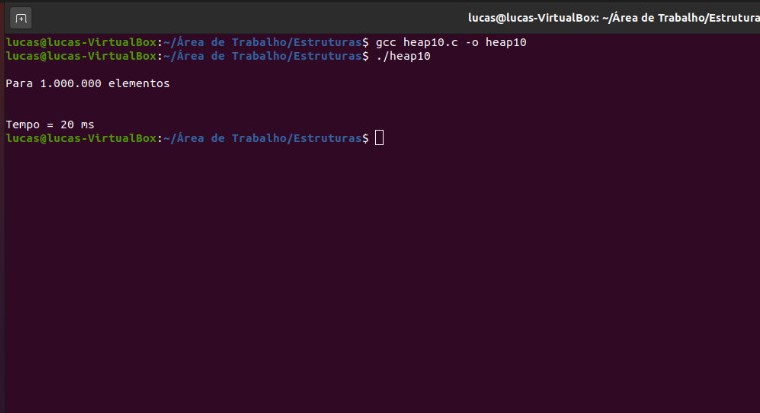


Figura 9. Heap Sort com 10 elementos.

Figura 10. Heap Sort com 1 milhão de elementos.

1. **CONCLUSÃO**

Devido a pandemia não foi possível realizar o projeto presencialmente na universidade, portanto além de reforçar e aprofundar o conhecimento aprendido na prática sobre analise de algoritmos e algoritmos de ordenação, o projeto ajudou no desenvolvimento do aluno acerca de projetos a distância e no aprendizado da individualidade do desenvolvedor, que deve ter uma equipe para desenvolver de forma adequada, mas deve saber se adaptar e buscar conhecimento por si próprio.

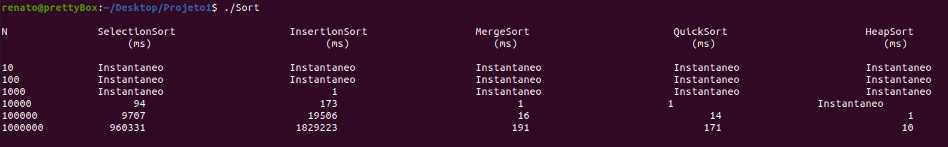
Após o desenvolvimento dos algoritmos pela equipe foram realizados testes afim de comprovar a eficácia e qualidade dos mesmos, que mostraram o resultado esperado pela equipe, sendo assim conseguimos atingir o objetivo esperado e entregar o projeto funcional e sem erros.

Figura 11. Tabela Completa com o Tempo para todo algoritmo e número de elementos (N).

Como evidenciado na tabela a cima, os algoritmos não são iguais e tem seus pontos positivos e negativos, O Selection Sort é linear e usado para vetores pequenos e com muitas trocas, Insertion Sort também é linear e rápido para vetores pequenos ou pré-ordenado, além disso ele é estável, para vetores grandes esses algoritmos são muito demorados, portanto para lidar com vetores maiores do que 100.000 elementos foram desenvolvidos, o Merge e Quick Sort, sendo o Merge um algoritmo Rápido para vetores grande e que te garante Estabilidade e o Quick Sort apesar de não ser estável, ele ocupa menos espaço em memória e pode ser muito otimizado, garantindo assim probabilisticamente o resultado mais rápido. O Heap Sort é um algoritmo que insere os dados em uma arvore de forma já ordenada, aproveitando das regras de inserção para evitar comparações e reduzir o tempo de execução.

1. **BIBLIOGRAFIA**

**[1]** https://www.coursera.org/lecture/algorithms-part1/shuffling-12vcF

**[2]** http://aorta.coop/wp-content/uploads/2017/05/Divide-and-Conquer-1.pdf

**[3]** https://www.geeksforgeeks.org/merge-sort/