Trabalho Prático 2 - Ordenação **Víctor Dias Frota**

1. Introdução:
Este trabalho tem como objetivo programar, testar e analisar quatro algoritmos de ordenação - Quicksort, Mergesort, Selectionsort e Insertionsort- estudados ao longo do semestre. A proposta é avaliar o desempenho desses algoritmos em diferentes cenários.

2. Implementação:
O processo de implementação ocorreu por meio da linguagem C, utilizando o compilador e editor VSCode.

2.1 Estrutura da Struct Aluno: Primeiramente, foi estabelecida a estrutura do objeto que será ordenado, sendo essa uma struct do tipo Aluno, com os seguintes membros: matrícula (int), período (int), nome (string), curso (string) e coeficiente de rendimento (double).

2.2 Manipulação da Struct Aluno: As primeiras funções implementadas foram para a manipulação da struct Aluno, usando a biblioteca <stdlib.h> e <time.h>, uma para criar strings aleatórias usando a função rand() com seed baseada no tempo, que serão usadas nos membros nome e curso, e uma função para criar o vetor de alunos com os tamanhos necessários para os testes, também aleatorizando os demais membros da struct.

2.3 Implementação dos Algoritmos de Ordenação: As próximas funções criadas foram as funções de ordenação, que seriam utilizadas neste trabalho. Todas as quatro já estavam implementadas devido às aulas práticas da disciplina de AEDS 2, sendo necessário apenas adaptá-las para ordenar o vetor de Alunos, uma vez que originalmente ordenam vetores de inteiros. A chave de ordenação escolhida para a struct foi o membro matrícula, já que é um número inteiro e, em uma aplicação real, cada aluno teria um número de matrícula único.

2.4 Funções de Teste: Depois, foram criadas as funções de teste, que combinam a função de criação do vetor de alunos aleatório e a ordenação. Também foram implementados os contadores de desempenho, sendo eles um contador de movimentações de registro, um contador de comparações (para isso foi necessário modificar as funções de ordenação mais uma vez) e um para o tempo de execução (usando a função clock() da biblioteca <time.h>). Repetindo o processo 10 vezes para obtenção de uma média dos valores obtidos.

2.5 Função de Exemplo Interativa: Foi criada também uma função de exemplo que pode ser acessada no terminal e exibe o vetor completo antes e depois da ordenação, podendo-se escolher o tamanho do vetor e o algoritmo de ordenação, para verificar o funcionamento de todos os métodos e contadores com tamanhos de vetores menores.

2.7 Ajustes para Dados Maiores: Foram realizados os primeiros testes com os tamanhos de vetores solicitados (1.000, 10.000, 100.000 e 200.000). Percebi que para os testes de Selectionsort e Insertionsort eram necessários tipos de dados maiores para armazenar o número de movimentos e comparações. Portanto, mudei de 'int' para 'long long'.

2.8 Testes com Vetores Ordenados Crescentemente e Decrescentemente: Após vários debugs, foram criadas as funções para os testes de vetores ordenados crescentemente e

decrescentemente. Sendo esses muito semelhantes aos testes anteriores, porém com a função de criar vetor de alunos modificada, e sem a necessidade da repetição para média de valores.

3. Estudo de Complexidade:

3.1 Quicksort: é baseado na técnica de divisão e conquista. A ideia central é escolher um elemento pivô e particionar o vetor de entrada em dois subvetores, um contendo elementos menores que o pivô e outro com elementos maiores. A complexidade do Quicksort é geralmente expressa como O(n log n) em média, o que o torna eficiente para grandes conjuntos de dados. No entanto, o pior caso é O(n^2), que ocorre quando o pivô é sempre o menor ou o maior elemento. Para evitar o pior caso usamos a mediana de três, e escolhemos o pivô como o elemento mediana entre o menor o maior elemento.

3.2 Mergesort: também utiliza a abordagem de divisão e conquista. Ele divide o vetor em subvetores menores, ordena cada subvetor e, em seguida, combina esses subvetores ordenados para obter o resultado final. O Mergesort tem uma complexidade de tempo de O(n log n) em todos os casos, tornando-o consistente e previsível, independentemente da distribuição inicial dos dados. No entanto, seu principal inconveniente é o uso de espaço adicional para armazenar os subvetores durante o processo de intercalação. Porém esse dado não foi medido nesse experimento.

3.3 Selectionsort: é um algoritmo simples de ordenação por seleção. Ele percorre repetidamente a lista, encontra o menor elemento e o coloca no início. Esse processo é repetido para os elementos restantes. A complexidade de tempo do Selectionsort é O(n^2), tornando-o menos eficiente para grandes conjuntos de dados. Além disso, sua natureza determinística significa que sua complexidade não melhora em média, mesmo quando o vetor já está parcialmente ordenado, porém isso também significa que seu número de movimentações é linear O(n).

4. Insertionsort: é outro algoritmo simples que constrói uma sequência ordenada de elementos um de cada vez. Ele percorre a lista e insere cada elemento na posição correta, comparando-o com os elementos anteriores. A complexidade de tempo do Insertionsort é O(n^2) no pior caso, tornando-o menos eficiente para grandes conjuntos de dados. No entanto, é mais eficiente em termos de espaço em comparação com algoritmos que requerem espaço adicional. A principal vantagem do Insertionsort é sua adaptabilidade, mostrada pelo teste com vetor ordenado em ordem crescente que seria o seu melhor caso, tendo complexidade O(n).

Algoritmo	Quicksort	Mergesort	Selectionsort	Insertionsort
Pior caso	O(n^2)	O(n log n)	O(n^2)	O(n^2)
Melhor caso	O(n log n)	O(n log n)	O(n^2)	O(n)
Caso médio	O(n log n)	O(n log n)	O(n^2)	O(n^2)
Movimentações média	O(n log n)	O(n log n)	O(n)	O(n^2)
Adaptabilidade	sim	não	não	sim

4. Listagem de testes executados:

4.1 Teste 1 – Teste com Vetores Aleatórios

TESTE DE VETORES ALEATÓRIOS				
Tempo de Execução:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	0,0003	0,0003	0,001	0,001
10000	0,0023	0,003	0,112	0,0775
100000	0,0268	0,0382	18,8561	98,714
200000	0,0574	0,0834	116,5522	67,8116
Número de Comparações:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	6286	8705	499500	248959
10000	87241	120456	49995000	24990390
100000	1056727	1536327	4999950000	2497675144
200000	2211262	3272804	19999900000	9998028854
Número de Movimentações:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	2689	18681	999	248959
10000	34825	254072	9999	24990390
100000	446504	3205255	99999	2497675144
200000	964489	6810660	199999	9998028854

Tabela 1 – Teste 1.

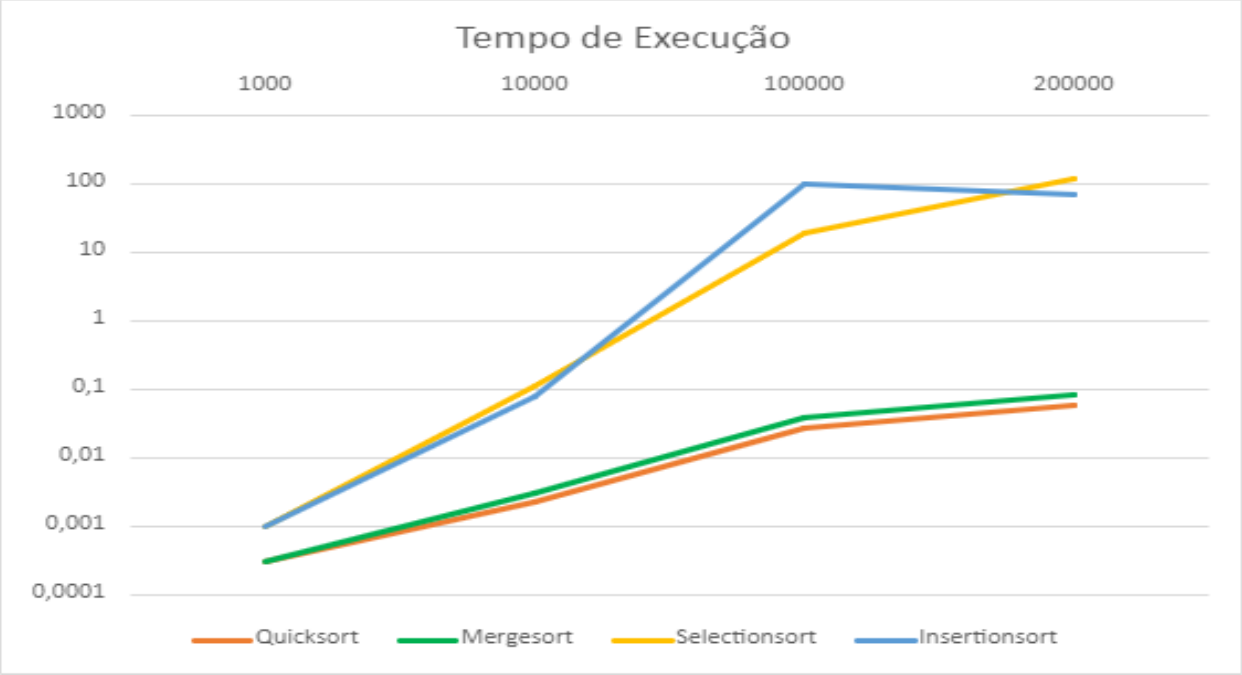

Gráfico 1 – Tempo de execução do teste 1.

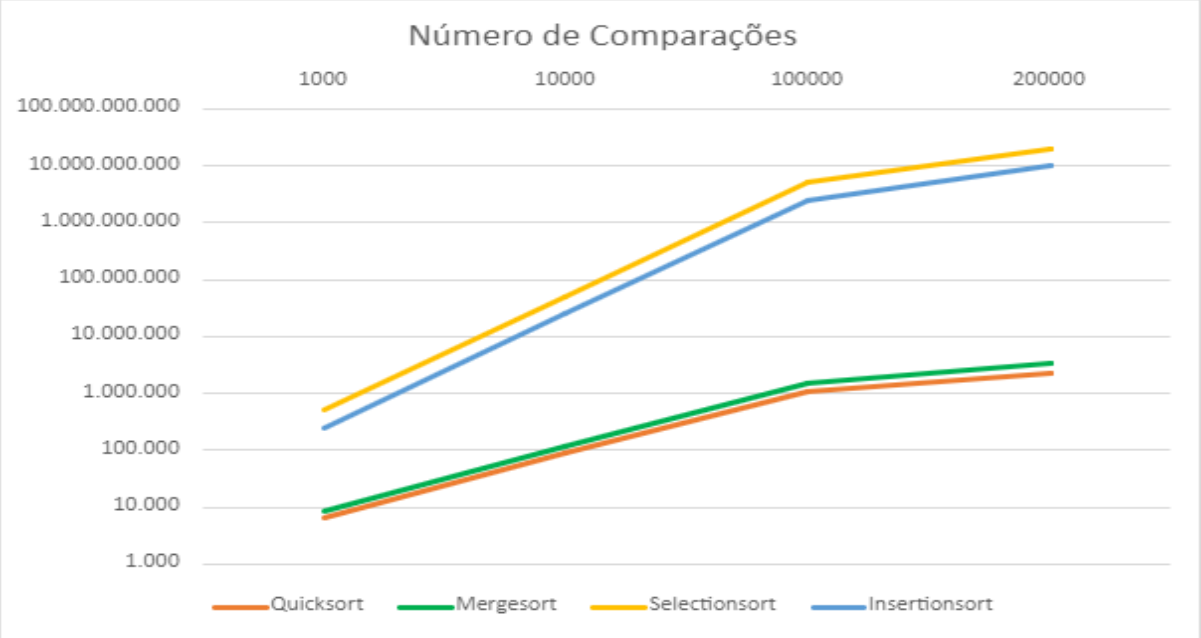

Gráfico 2 - Número de Comparações do teste 1.

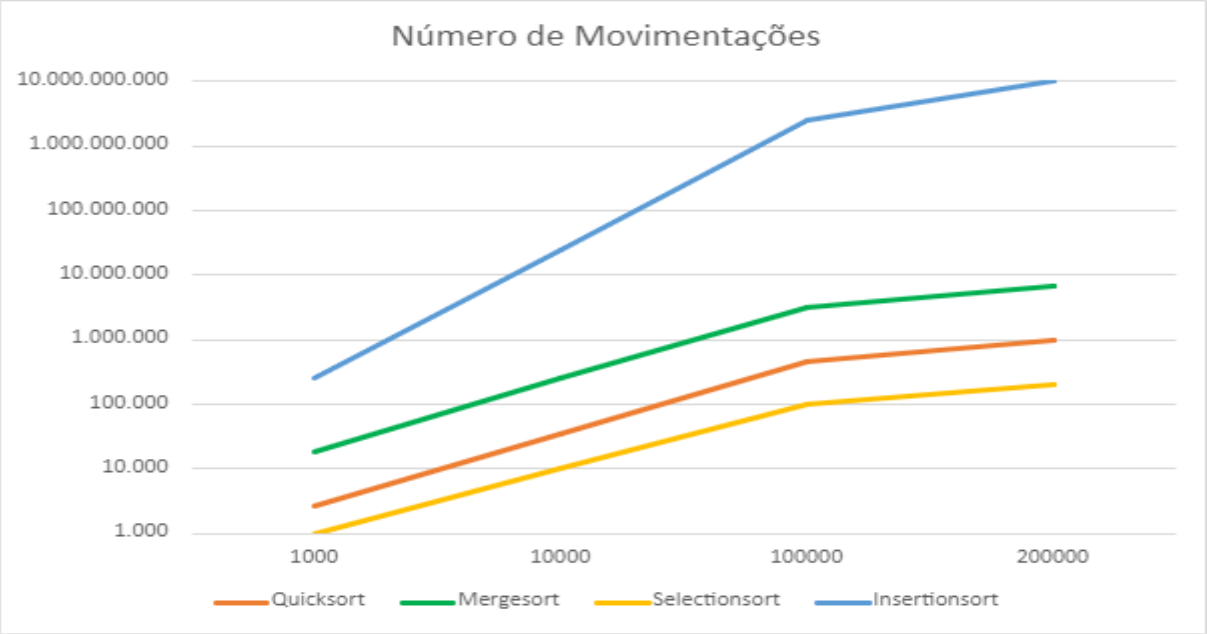

Gráfico 3 - Número de Movimentações do teste 1.

4.2 Teste 2 – Teste com Vetores em Ordem Crescente

TESTE DE VETORES EM ORDEM CRESCENTE				
Tempo de Execução:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	0,001	0,001	0,001	0,001
10000	0,001	0,004	0,12	0,001
100000	0,008	0,035	17,057	0,01
200000	0,017	0,096	133,996	0,001
Número de Comparações:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	7987	4932	499500	999
10000	113631	64608	49995000	9999
100000	1468946	815024	4999950000	99999
200000	3137875	1730048	19999900000	199999
Número de Movimentações:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	511	14908	999	999
10000	5904	198224	9999	9999
100000	65535	2483952	99999	99999
200000	131071	5267904	199999	199999

Tabela 2 – Teste 2

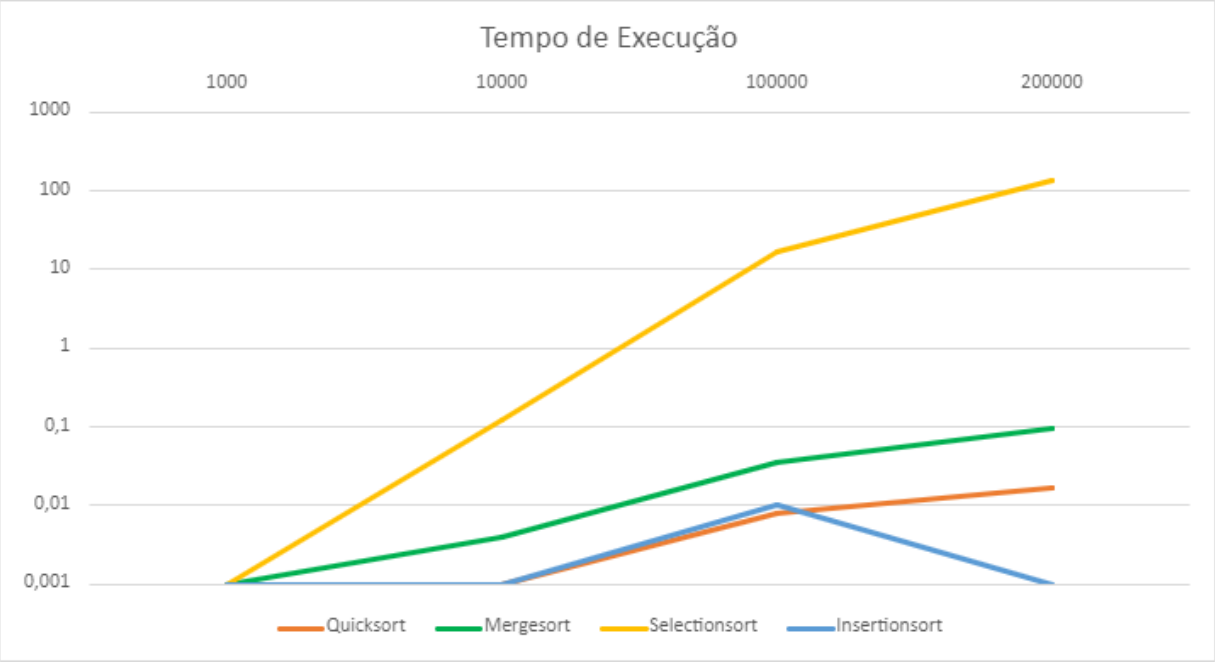

Gráfico 4 – Tempo de Execução do teste 2

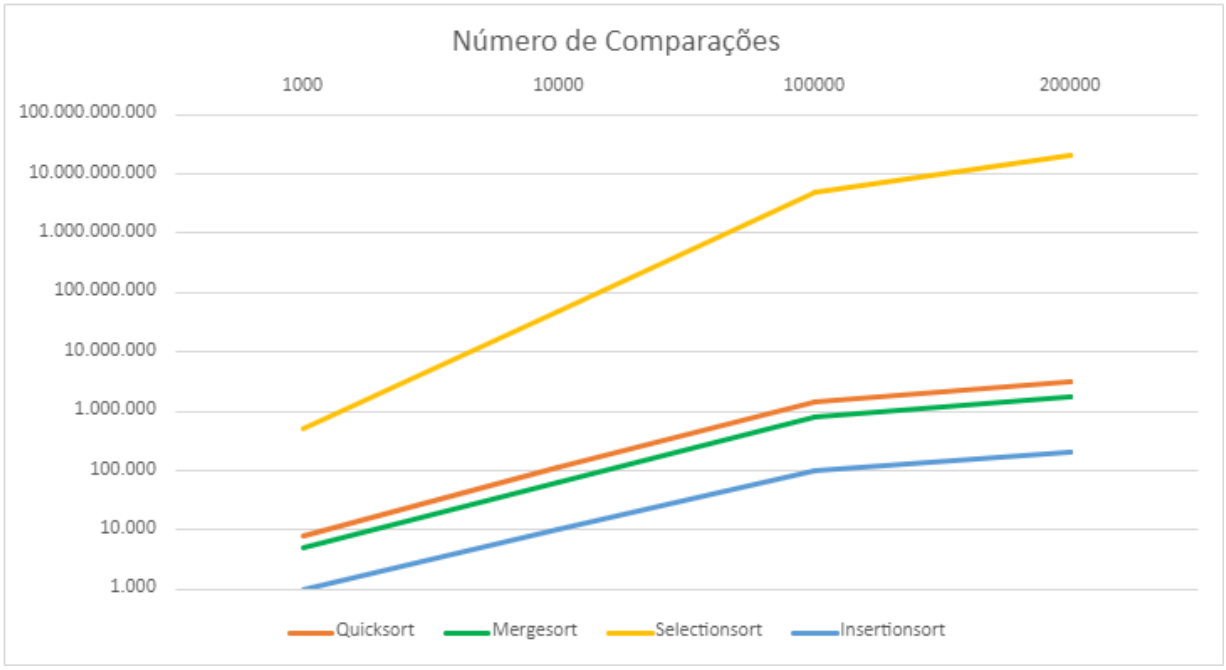

Gráfico 5 - Número de Comparações do teste 2

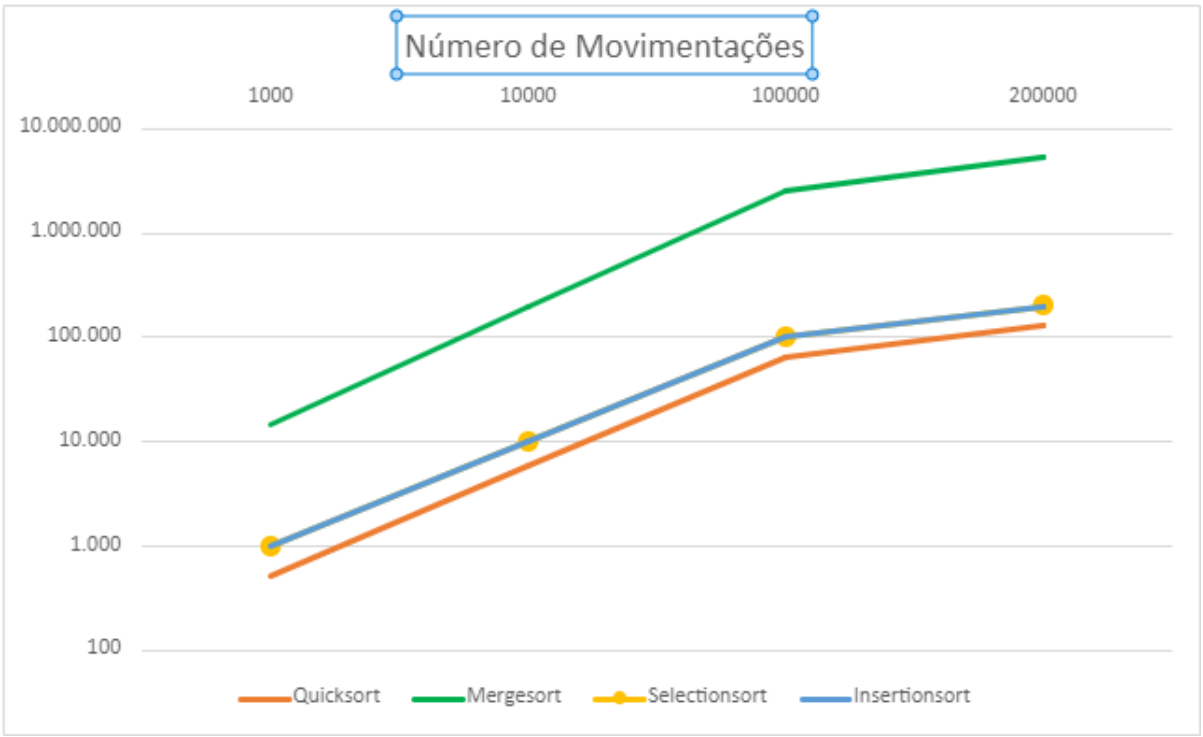

Gráfico 6 - Número de Movimentações do teste 2

4.2 Teste 2 – Teste com Vetores em Ordem Crescente

TESTE DE VETORES EM ORDEM DECRESCENTE				
Tempo de Execução:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	0,001	0,001	0,001	0,001
10000	0,001	0,007	0,13	0,16
100000	0,01	0,045	17,198	23,149
200000	0,021	0,089	138,215	217,519
Número de Comparações:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	6996	5044	499500	500499
10000	103644	69008	49995000	50004999
100000	1368962	853904	4999950000	5000049999
200000	2937892	1807808	19999900000	20000099999
Número de Movimentações:				
Número de Elementos	**Quicksort**	**Mergesort**	**Selectionsort**	**Insertionsort**
1000	1010	15020	999	500499
10000	10904	202624	9999	50004999
100000	115534	2522832	99999	5000049999
200000	231070	5345664	199999	20000099999

Tabela 3 – Teste 3

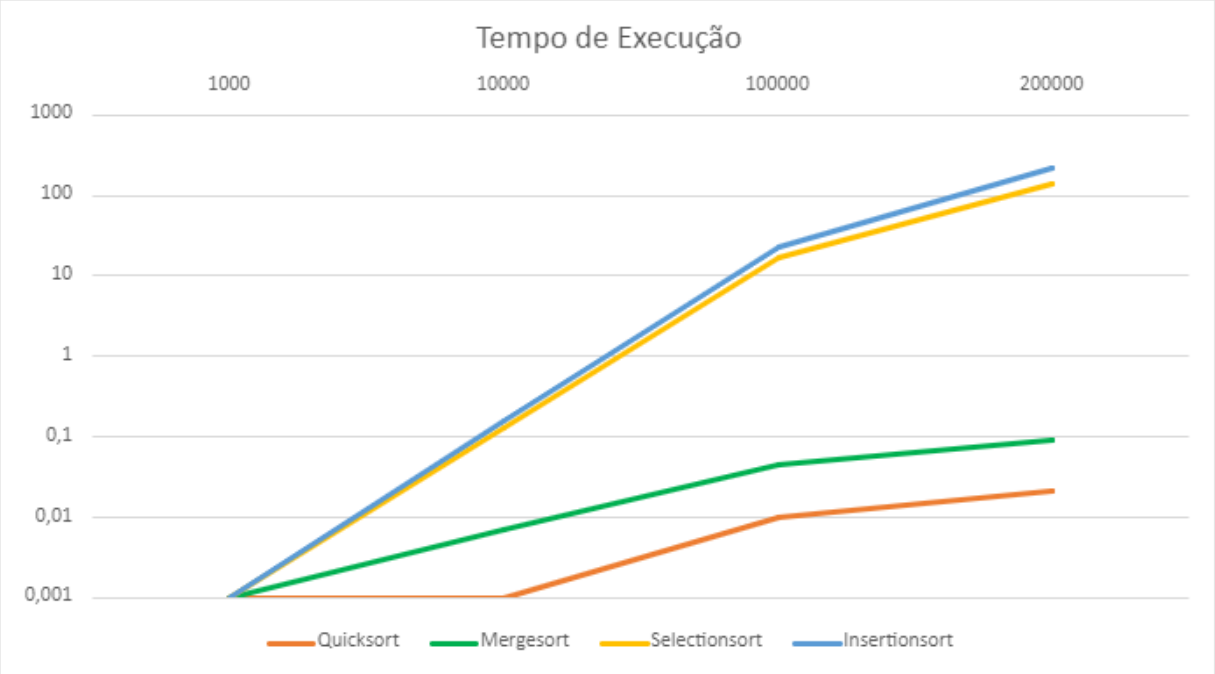

Gráfico 7 – Tempo de Execução do teste 3

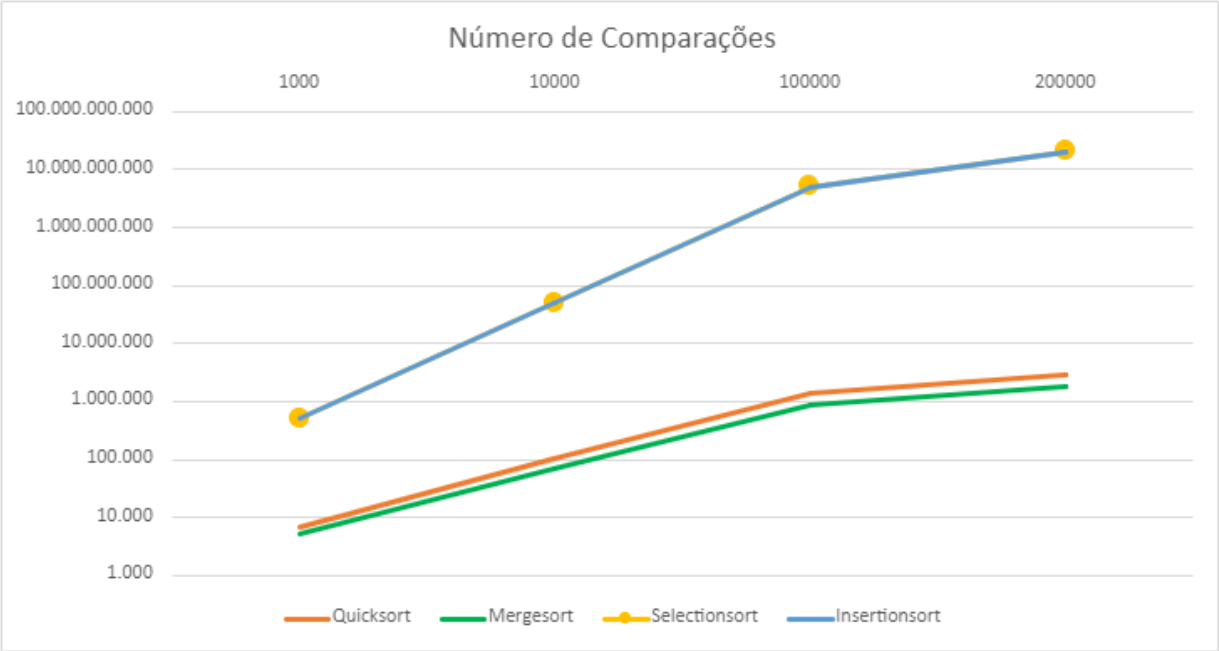

Gráfico 8 - Número de Comparações do teste 3

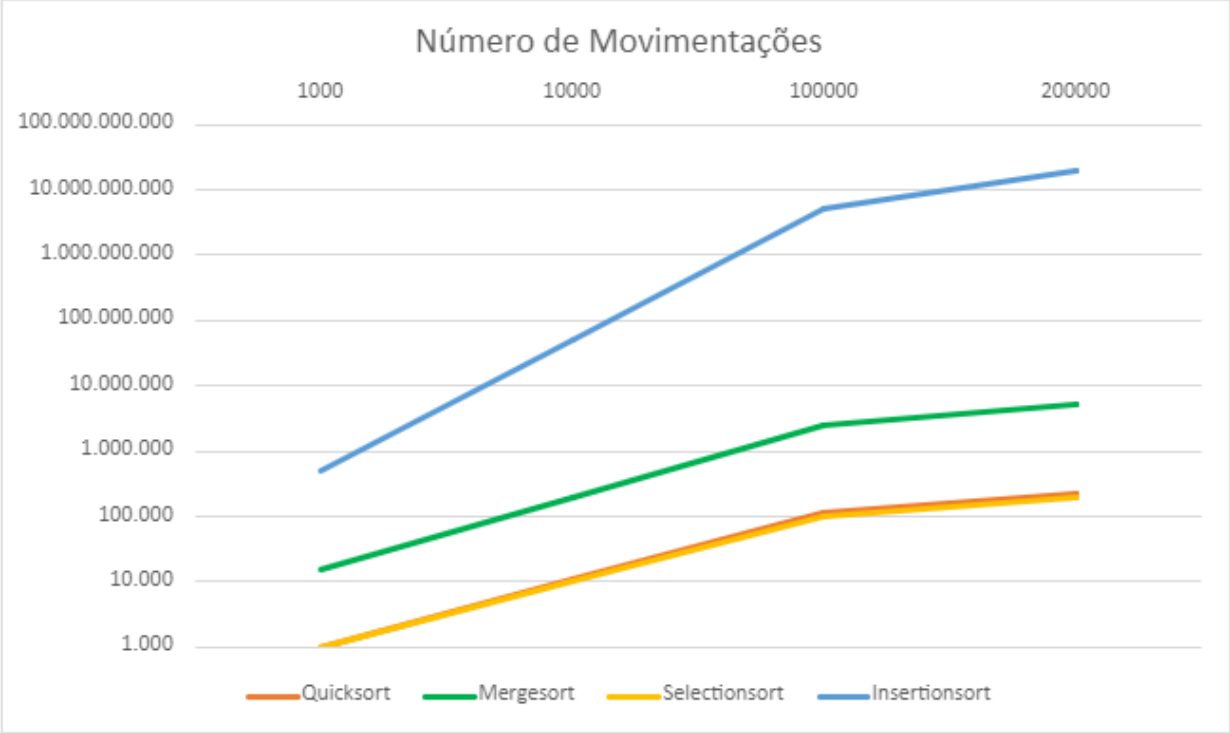

Gráfico 8 - Número de Movimentações do teste 3

5. Conclusão:

Este estudo sobre os algoritmos de ordenação - Quicksort, Mergesort, Selectionsort e Insertionsort - proporcionou uma análise aprofundada de seu desempenho em diversos cenários. A implementação prática em linguagem C, considerando a ordenação de uma struct representando dados de alunos, revelou a importância sobre a eficiência e adaptabilidade de cada algoritmo. A avaliação quantitativa, por meio de contadores de movimentações, comparações e tempo de execução, destacou características distintas, como a capacidade de lidar com conjuntos de dados grandes. Esses conhecimentos são fundamentais para orientar a escolha de cada algoritmos de ordenação em aplicações do mundo real, levando em conta não apenas a teoria de complexidade, mas também considerações práticas e desempenho real.

6. Bibliografia:

Rodrigues, E. (2023). Algoritmos e Estruturas de Dados II, Aulas Teóricas. Google Classroom. URL: https://classroom.google.com/c/NTg5MTc2NDM3NjUy/m/NTg5MTc2NzIwMTkx/details

Pereira, W. (2019). Introdução à Complexidade de Algoritmos. Medium. URL: https://medium.com/nagoya-foundation/introdução-à-complexidade-de-algoritmos-4a9c237e4ecc

Wikimedia Commons. (2022). ASCII-Table. Wikipedia. URL: https://pt.m.wikipedia.org/wiki/Ficheiro:ASCII-Table-wide.svg

Felipe, H. (2018). Quicksort mediana de três. Blog Cyberini. URL: https://www.blogcyberini.com/2018/08/quicksort-mediana-de-tres.html#quicksortm3_c

GeeksforGeeks. (2023). clock() function in C. URL: https://www.geeksforgeeks.org/clock-function-in-c-c/