Análise de desempenho de dois algoritmos de ordenação: MergeSort e Super MergeSort

Eduarda Elger, Ellen Bonafin, Heloisa Apª Alves

Centro de Ciências Exatas e Tecnológicas – Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 801 – 85.814-110 – Cascavel – PR – Brazil

{eduarda.elger@unioeste.br, ellen.marques1@unioeste.br, heloisa.alves@unioeste.br}

Abstract. *This article examines the MergeSort and SuperMerge Sort sorting algorithms concerning the chronological time spent during execution, using various test sets. We analyze the performance and behavior of these methods in diverse scenarios, providing cruciais insights for the selection and optimization of sorting algorithms in practical contexts. The results offer a deep understanding of the efficiencies and limitations of these algorithms, contributing to well-informed decision-making in the choice of sorting algorithms for real-world applications.*

Resumo. *Este artigo examina os algoritmos de ordenação MergeSort e SuperMergeSort em relação ao tempo cronológico gasto durante a execução, utilizando conjuntos de testes variados. Analisamos o desempenho e comportamento desses métodos em cenários diversificados, oferecendo insights cruciais para a escolha e otimização de algoritmos de ordenação em contextos práticos. Os resultados fornecem uma compreensão aprofundada das eficiências e limitações desses algoritmos, contribuindo para a tomada de decisões informadas na seleção de algoritmos de ordenação em aplicações do mundo real.*

1. Introdução

Proposto por John von Neumann em 1945 o MergeSort é um algoritmo clássico de ordenação baseado no princípio de dividir e conquistar. Ele divide um subarray não ordenado em subarrays de menor tamanho até que cada subarray contenha somente um elemento garantindo estabilidade e eficiência inerente. Com complexidade de tempo O(n log n), o MergeSort é especialmente adequado para lidar com conjuntos de dados extensos. Este algoritmo destaca-se por sua simplicidade e desempenho consistente em diversas situações de entrada (Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., 2009). Ele serve como a base para várias técnicas eficazes de ordenação, permanecendo um tema essencial no ensino e na pesquisa em ciência da computação.

Para melhor visualização e compreensão desse tema, será analisado nesse artigo o comportamento de duas estratégias distintas de ordenação MergeSort. Primeiramente, será aplicado o MergeSort clássico, onde o problema é dividido ao meio a cada iteração, resultando em duas novas instâncias do problema. Além disso, será explorado o Super Merge, uma variação do MergeSort, na qual o problema é dividido em quatro subpartes, e para cada uma delas é realizada uma nova chamada recursiva. Este estudo visa avaliar

o desempenho dessas duas abordagens diante de conjuntos de testes com características diversas. Durante a execução dos testes, o critério de análise será o tempo cronológico gasto para a execução dos métodos, proporcionando uma compreensão aprofundada de seu comportamento em diferentes cenários.

O trabalho foi subdividido em: implementação (seção 2) contendo informações sobre a linguagem utilizada para a implementação, o ambiente de teste e informações sobre os dos dois algoritmos de ordenação. Na seção 3 são apresentados a métrica utilizada para a avaliação das implementações propostas e os resultados; e na seção 4 as considerações finais.

2. Implementação

Para a modelagem e testes dos algoritmos, foi utilizada a linguagem de programação C em conjunto com o ambiente de desenvolvimento online Repl.it que utiliza o sistema operacional Ubuntu na versão 20.04.2 LTS.

2.1 MergeSort Clássico

A implementação do MergeSort baseia-se no princípio de dividir para conquistar, ou seja, ele pega o vetor de entrada e o divide em n partes, neste trabalho o MergeSort está dividindo em 2 partes. Esta divisão é feita de forma recursiva através da função “MergeSort”.

Para realizar a leitura dos vetores de testes fornecidos é utilizado a estrutura de array, ou seja, é realizada de forma dinâmica a alocação, esta parte é feita através da função “MergeSortForFile”, após isto é chamada a função para realizar de fato a leitura dos arquivos, sendo ela a “ReadFileToArray”, ela será a responsável por armazenar os valores no vetor que será ordenado. Ao fim deste processo então será chamada a função “MergeSort”, nela será feita de forma recursiva a divisão em duas partes do vetor de entrada e a ordenação dos subvetores criados. Por fim, a função “Merge” pega os subvetores e os mescla de forma ordenada em apena um vetor, sendo ele o vetor final ordenado.

O processo de divisão e conquista é uma característica fundamental do MergeSort e é responsável pela sua eficiência. Dividir a lista em duas partes permite a ordenação independente das metades, o que simplifica o processo e torna mais fácil combinar as partes ordenadas.

2.2 Super MergeSort

O Super MergeSort é uma variação do MergeSort clássico apresentado na subseção anterior. Sua principal diferença é que a divisão do problema agora é feita em quatro, ou seja, o vetor de entrada é dividido em quatro subvetores, e o processo de ordenação é realizado em etapas. As etapas seguintes são realizadas da mesma forma que fora explicada para o MergeSort tradicional.

Essa versão do MergeSort com quatro partes pode ser útil em cenários específicos, onde a divisão em quatro partes permite uma exploração mais eficiente da arquitetura de hardware ou acelera o processo de ordenação.

3. Testes

Para a realização dos testes foi feita a leitura dos vetores de entrada. Essa leitura é feita através de arquivos que foram fornecidos via google drive, neles teremos quatro arquivos diferentes em cada tipo de teste que serão realizados, sendo eles: vetores aleatórios, decrescentes, crescentes e parcialmente ordenados.

A fim de avaliar o impacto no tempo de processamento dos algoritmos de ordenação, cada um dos tipos de arquivos recebeu testes com vetores de tamanho: 100, 200, 500, 1000, 2000, 5000, 7500, 10000, 15000, 30000, 50000, 75000, 100000, 200000, 500000, 750000, 1000000, 1250000, 1500000 e 2000000.

Os resultados são apresentados em segundos por meio da função *clock_ start_time = clock()*. A seguinte função retorna um valor de precisão dupla igual ao número de segundos desde o valor inicial do relógio de tempo real do sistema operacional.

3.1 MergeSort

Considerando os tempos de processamento de cada entrada será apresentado através de quadros cada um dos resultados obtidos.

No Quatro 1 temos o tempo de processamento do MergeSort com entradas aleatórias, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Merge Sort - Aleatório	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
100	0.000604
200	0.000651
500	0.000771
1000	0.000834
2000	0.001085
5000	0.001994
7500	0.002649
10000	0.005497
15000	0.005167
30000	0.012191
50000	0.017743

75000	0.024819
100000	0.037190
200000	0.069964
500000	0.166036
750000	0.283815
1000000	0.362182
1250000	0.455103
1500000	0.573232
2000000	0.756152

Quadro 1 – Resultados MergeSort com entradas aleatórias.

No Quatro 2 temos o tempo de processamento do MergeSort com entradas em ordem decrescente, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Decrescente	
Tamanho do Conjunto de Dados	Tempo de Execução (segundos)
100	0.000587
200	0.000547
500	0.000581
1000	0.000856
2000	0.001015
5000	0.001663
7500	0.002421
10000	0.003210
15000	0.003907
30000	0.008768
50000	0.012155
75000	0.017785
100000	0.024898
200000	0.049320
500000	0.125045
750000	0.179848

1000000	0.242127
1250000	0.313310
1500000	0.389145
2000000	0.534382

Quadro 2 – Resultados MergeSort com entradas em ordem decrescente.

No Quatro 3 temos o tempo de processamento do MergeSort com entradas em ordenadas, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Ordenado	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
100	0.000496
200	0.000564
500	0.000766
1000	0.001031
2000	0.001368
5000	0.001744
7500	0.002517
10000	0.003037
15000	0.004346
30000	0.007088
50000	0.011741
75000	0.018083
100000	0.026538
200000	0.051672
500000	0.133097
750000	0.196458
1000000	0.266830
1250000	0.336555
1500000	0.412882
2000000	0.565649

Quadro 3 – Resultados MergeSort com entradas ordenadas.

No Quatro 4 temos o tempo de processamento do MergeSort com entradas parcialmente ordenadas, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Parcialmente Ordenado	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
100	0.000590
200	0.000573
500	0.000679
1000	0.000740
2000	0.000952
5000	0.001653
7500	0.002463
10000	0.003267
15000	0.005256
30000	0.006827
50000	0.011956
75000	0.017947
100000	0.025494
200000	0.069388
500000	0.154353
750000	0.204447
1000000	0.278041
1250000	0.336954
1500000	0.432105
2000000	0.546807

Quadro 4 – Resultados MergeSort com entradas parcialmente ordenadas.

Já, no Quatro 5 temos o tempo de processamento do MergeSort com todas as entradas juntas, a coluna da esquerda representa qual o tipo de entrada foi testado enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Tipo de Dados	Tempo de Execução (segundos)

Aleatório	2.846085
Decrescente	2.052572
Ordenado	1.878793
Parcialmente Ordenado	1.936301

Quadro 5 – Resultados MergeSort em segundos testando todas as entradas.

3.2 Super MergeSort

Considerando os tempos de processamento de cada entrada será apresentado através de quadros cada um dos resultados obtidos.

No Quatro 6 temos o tempo de processamento do Super MergeSort com entradas aleatórias, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Super MergeSort - Aleatório	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
Todos	2.001487
100	0.000615
200	0.000559
500	0.000650
1000	0.000884
2000	0.001253
5000	0.001581
7500	0.002278
10000	0.002591
15000	0.004213
30000	0.007380
50000	0.011475
75000	0.018765
100000	0.022752
200000	0.046153
500000	0.132645
750000	0.174108
1000000	0.251580
1250000	0.334342

1500000	0.383796
2000000	0.513777

Quadro 6 – Resultados Super MergeSort com entradas aleatórias.

No Quatro7 temos o tempo de processamento do Super MergeSort com entradas em ordem decrescente, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Decrescente	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
Todos	1.445430
100	0.000544
200	0.000564
500	0.000574
1000	0.000675
2000	0.000852
5000	0.001323
7500	0.001836
10000	0.002271
15000	0.002877
30000	0.006168
50000	0.010361
75000	0.013222
100000	0.017078
200000	0.030753
500000	0.088409
750000	0.131297
1000000	0.180714
1250000	0.245444
1500000	0.279681
2000000	0.377656

Quadro 7 – Resultados Super MergeSort com entradas em ordem decrescente.

No Quatro 8 temos o tempo de processamento do Super MergeSort com entradas em ordenadas, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Ordenado	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
Todos	1.364498
100	0.000707
200	0.000534
500	0.000539
1000	0.000653
2000	0.000871
5000	0.001421
7500	0.001993
10000	0.002161
15000	0.003020
30000	0.005677
50000	0.011058
75000	0.013735
100000	0.015599
200000	0.030432
500000	0.084397
750000	0.129860
1000000	0.188242
1250000	0.234357
1500000	0.270343
2000000	0.374394

Quadro 8 – Resultados Super MergeSort com entradas em ordenadas.

No Quatro 9 temos o tempo de processamento do Super MergeSort com entradas parcialmente ordenadas, a coluna da esquerda representa o tamanho das entradas enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Parcialmente Ordenado	
Tamanho de Entrada	Tempo Médio de Execução (segundos)
Todos	1.435099
100	0.000468
200	0.000598
500	0.000597
1000	0.000703
2000	0.000780
5000	0.001297
7500	0.001993
10000	0.002029
15000	0.002876
30000	0.005230
50000	0.007799
75000	0.017870
100000	0.017343
200000	0.030993
500000	0.080502
750000	0.131424
1000000	0.169332
1250000	0.236124
1500000	0.261035
2000000	0.354017

Quadro 9 – Resultados Super MergeSort com entradas em parcialmente ordenadas.

Já, no Quatro 10 temos o tempo de processamento do MergeSort com todas as entradas juntas, a coluna da esquerda representa qual o tipo de entrada foi testado enquanto a coluna da direita é o tempo médio de execução das entradas em segundos.

Arquivos de Teste	Tempo de Execução (segundos)
Aleatório	2.001487
Decrescente	1.445430

Ordenado	1.364498
Parcialmente Ordenado	1.435099

Quadro 10 – Resultados Super MergeSort em segundos testando todas as entradas.

3.3 Comparação entre MergeSort Super MergeSort

Ao executar as implementações observou-se que o Super MergeSort obteve um resultado um melhor que o MergeSort tradicional, uma explicação para isso é o fato de que o algoritmo MergeSort utiliza apenas duas subdivisões enquanto o Super MergeSort realiza a ordenação em quatro subpartes, o que acaba agilizando o processo, permitindo assim uma exploração mais eficiente da arquitetura de hardware.

O Gráfico 1 demonstra um comparativo de desempenho em segundos, dos dois algoritmos executando todas as quatro entradas: aleatórias, decrescentes, ordenadas e parcialmente ordenadas. O eixo horizontal representa as entradas processadas enquanto o eixo vertical representa o tempo em segundos de execução das entradas. A cor azul representa o algoritmo MergeSort e a cor vermelha o Super MergeSort. Ao analisar o Gráfico 1 podemos ver que de fato o Super MergeSort tem um desempenho melhor comparado a sua versão tradicional.

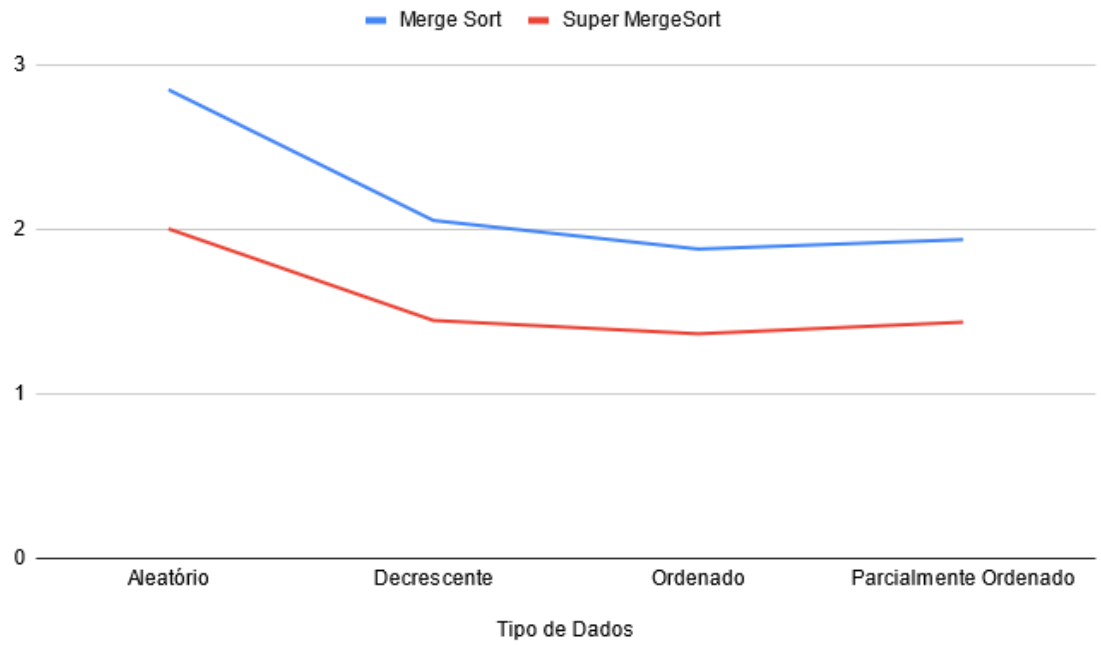

Gráfico 1 – Comparativo entre MergeSort e Super MergeSort testando todas as entradas.

Abaixo será apresentado os gráficos de comparação de cada tipo de entrada entre os dois algoritmos, os gráficos foram feitos utilizando a linguagem de programação Python na versão 3.12.0. Cabe salientar que serão dois gráficos para um determinado teste, um com escala e outro sem, isso se dá por causa da sobreposição das informações

no eixo X persistir. Para resolver esse problema foi rotacionado os rótulos do eixo X para melhorar a legibilidade.

O Gráfico 2, apresenta o comparativo das entradas aleatórias, no eixo x temos o tamanho dos arquivos enquanto o eixo y apresenta a escala do tempo em segundos.

Ao comparar o gráfico, podemos observar que o MergeSort é mais rápido que o MergeSort para todos os tamanhos de vetor de entrada. A diferença de desempenho entre os dois algoritmos é maior para vetores de entrada menores. Para vetores de entrada de até 10000 elementos, o MergeSort é cerca de 2 vezes mais rápido que o Merge Sort. A partir de 20000 elementos, a diferença de desempenho se reduz para cerca de 1,5 vezes.

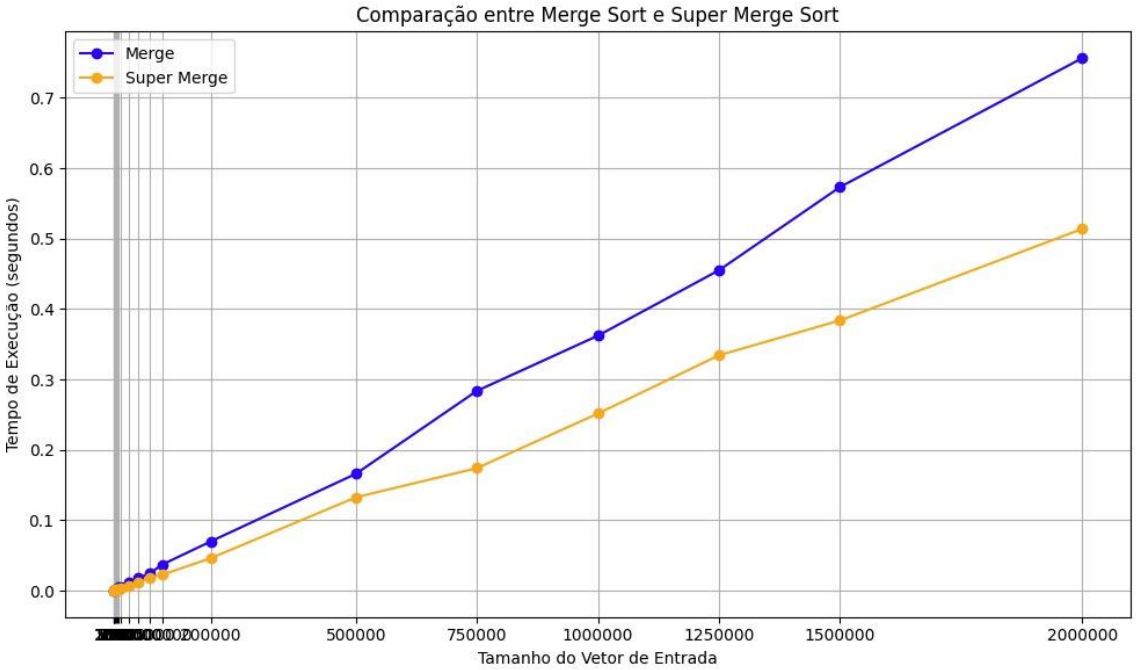

Gráfico 2 – Comparativo entre MergeSort e Super MergeSort testando entradas aleatórias.

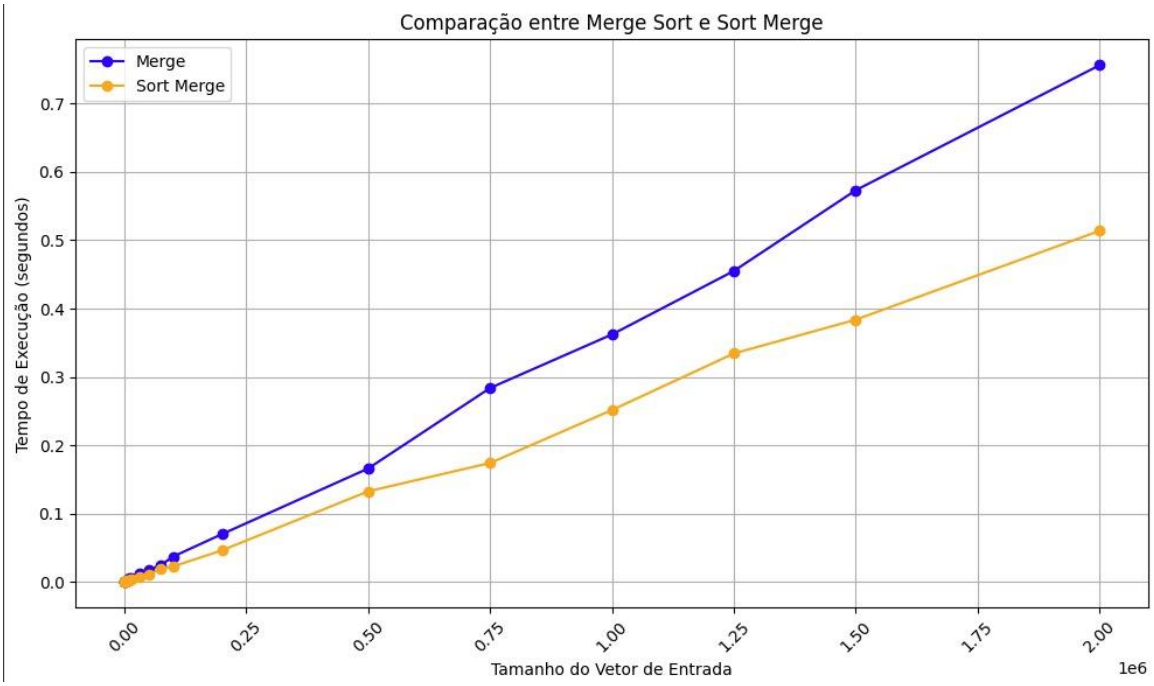

Gráfico 3 – Comparativo entre MergeSort e Super MergeSort testando entradas aleatórias (gráfico com alteração na escala).

O Gráfico 4, apresenta o comparativo das entradas ordenadas, no eixo x temos o tamanho dos arquivos enquanto o eixo y apresenta a escala do tempo em segundos.

Ao comparar o gráfico, podemos observar que o Super MergeSort ainda é mais rápido que o MergeSort para vetores de entrada maiores. No entanto, a diferença de desempenho é menor do que antes. Para vetores de entrada de até 5000 elementos, o MergeSort é mais rápido do que o Super MergeSort. A partir de 10000 elementos, o Super MergeSort é mais rápido.

A diferença de desempenho entre os dois algoritmos é menor do que antes porque os valores modificados para o MergeSort são menores. Isso significa que o MergeSort é mais eficiente para vetores de entrada menores.

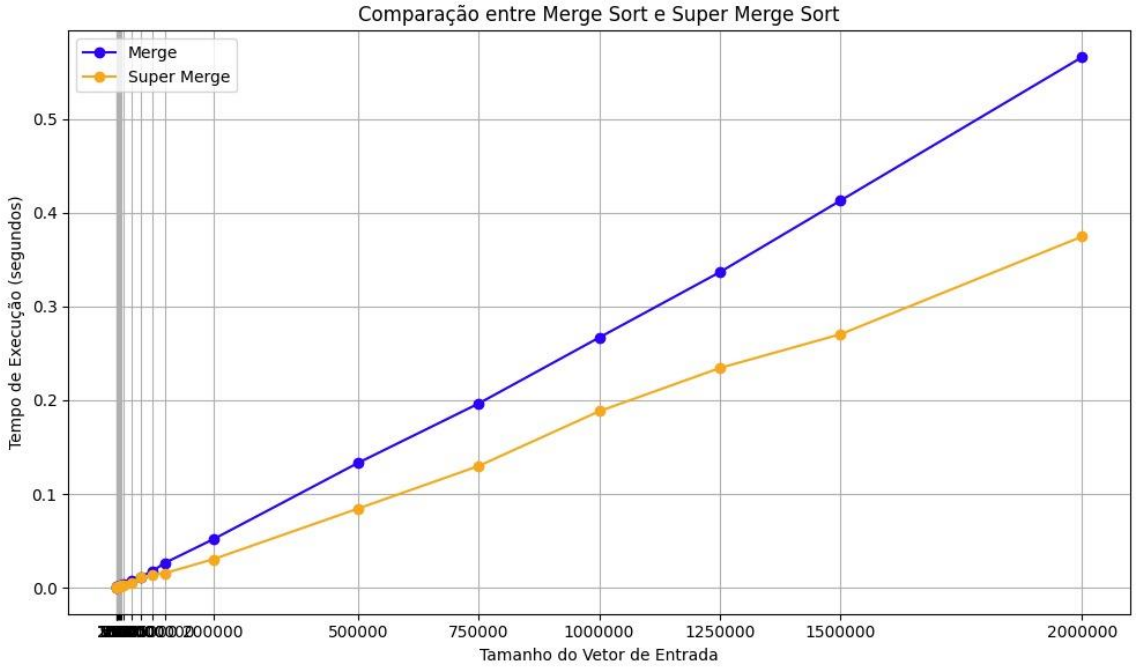

Gráfico 4 – Comparativo entre MergeSort e Super MergeSort testando entradas ordenadas.

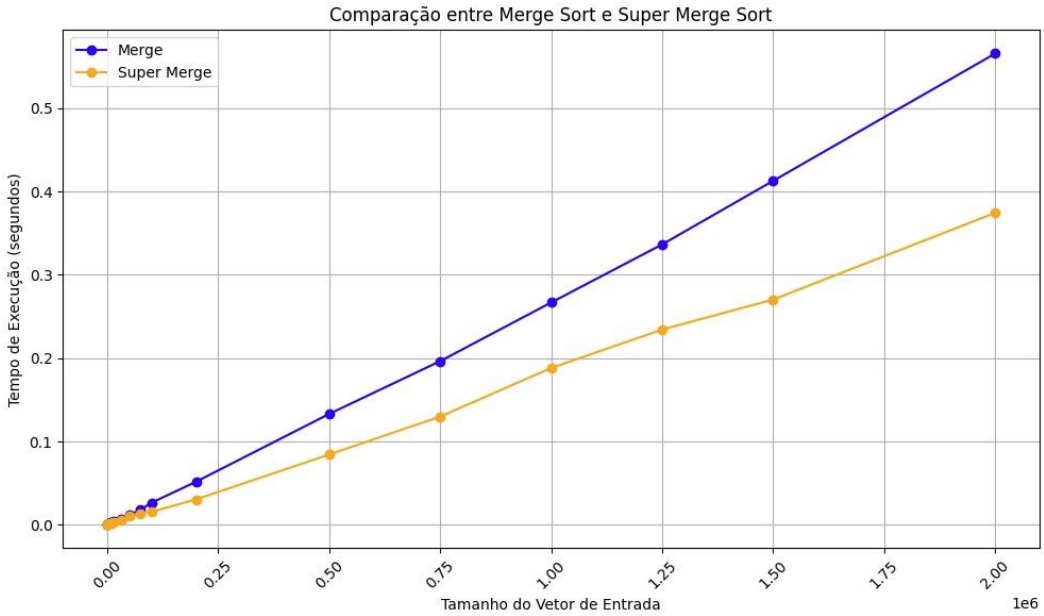

Gráfico 5 – Comparativo entre MergeSort e Super MergeSort testando entradas ordenadas (gráfico com alteração de escala).

O Gráfico 6, apresenta o comparativo das entradas parcialmente ordenadas, no eixo x temos o tamanho dos arquivos enquanto o eixo y apresenta a escala do tempo em segundos.

Podemos observar que o Super MergeSort ainda é mais rápido que o MergeSort para vetores de entrada maiores. A diferença de desempenho é menor do que antes. Para vetores de entrada de até 2000 elementos, o MergeSort é mais rápido do que o Super MergeSort. A partir de 5000 elementos, o Super MergeSort é mais rápido.

A diferença de desempenho entre os dois algoritmos é menor do que antes porque os valores modificados para o MergeSort são menores. Isso significa que o Merge Sort é mais eficiente para vetores de entrada menores.

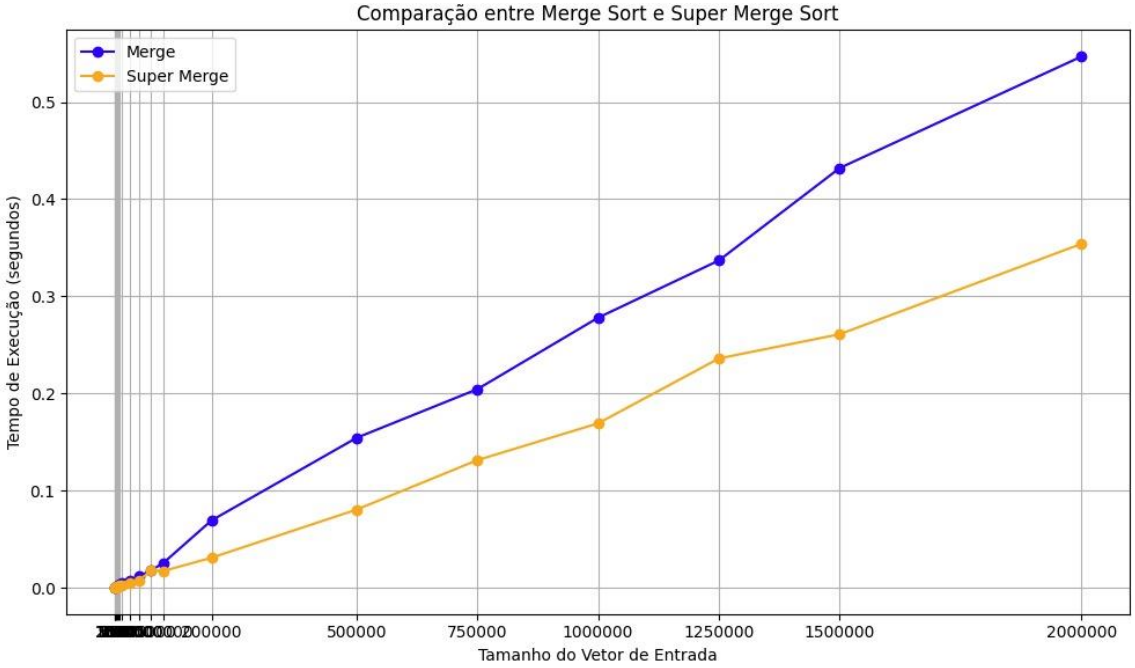

Gráfico 6– Comparativo entre MergeSort e Super MergeSort testando entradas parcialmente ordenadas.

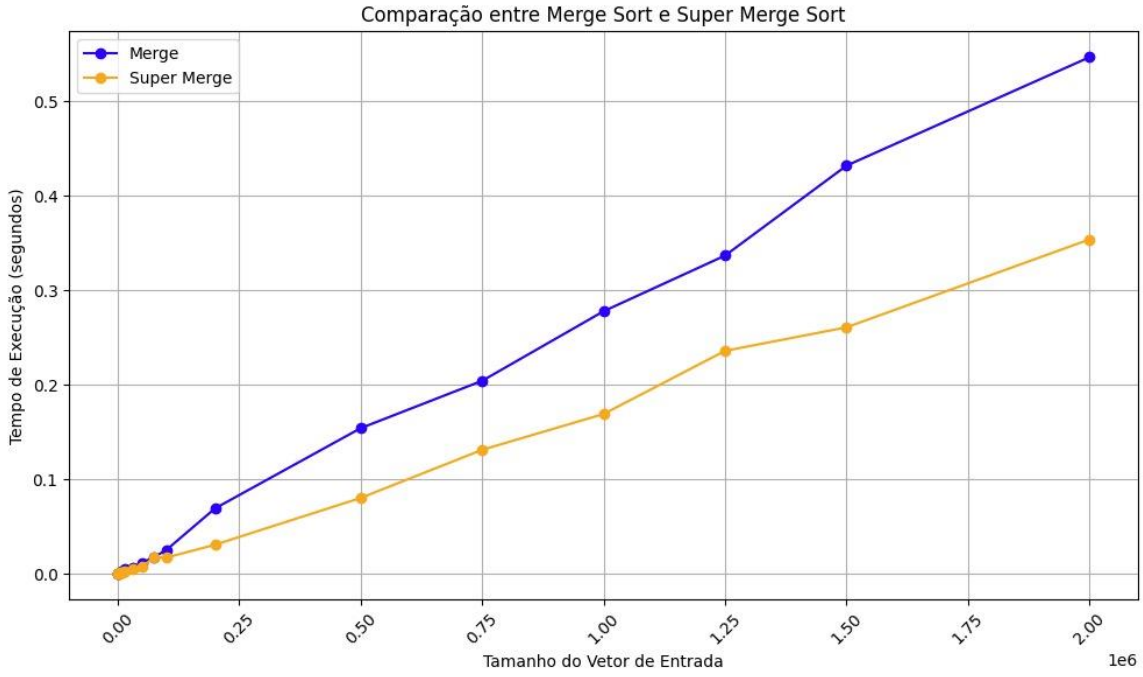

Gráfico 7– Comparativo entre MergeSort e Super MergeSort testando entradas parcialmente ordenadas (gráfico com escala alterada).

O Gráfico 8, apresenta o comparativo das entradas decrescentes, no eixo x temos o tamanho dos arquivos enquanto o eixo y apresenta a escala do tempo em segundos.

Ao comparar o gráfico, podemos observar que o Super MergeSort ainda é mais rápido que o MergeSort para vetores de entrada maiores. No entanto, a diferença de desempenho é menor do que antes. Para vetores de entrada de até 10000 elementos, o MergeSort é mais rápido do que o Super MergeSort. A partir de 20000 elementos, o Super MergeSort é mais rápido.

A diferença de desempenho entre os dois algoritmos é menor do que antes porque os valores modificados para o MergeSort são menores. Isso significa que o MergeSort é mais eficiente para vetores de entrada menores.

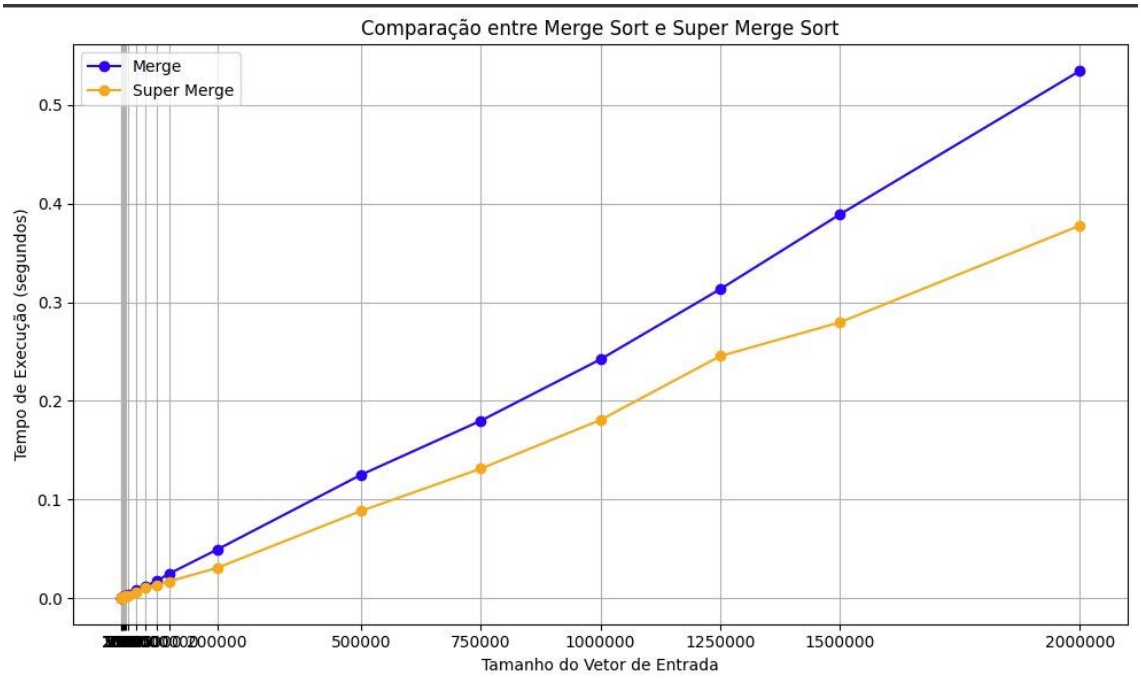

Gráfico 8 – Comparativo entre MergeSort e Super MergeSort testando entradas decrescentes.

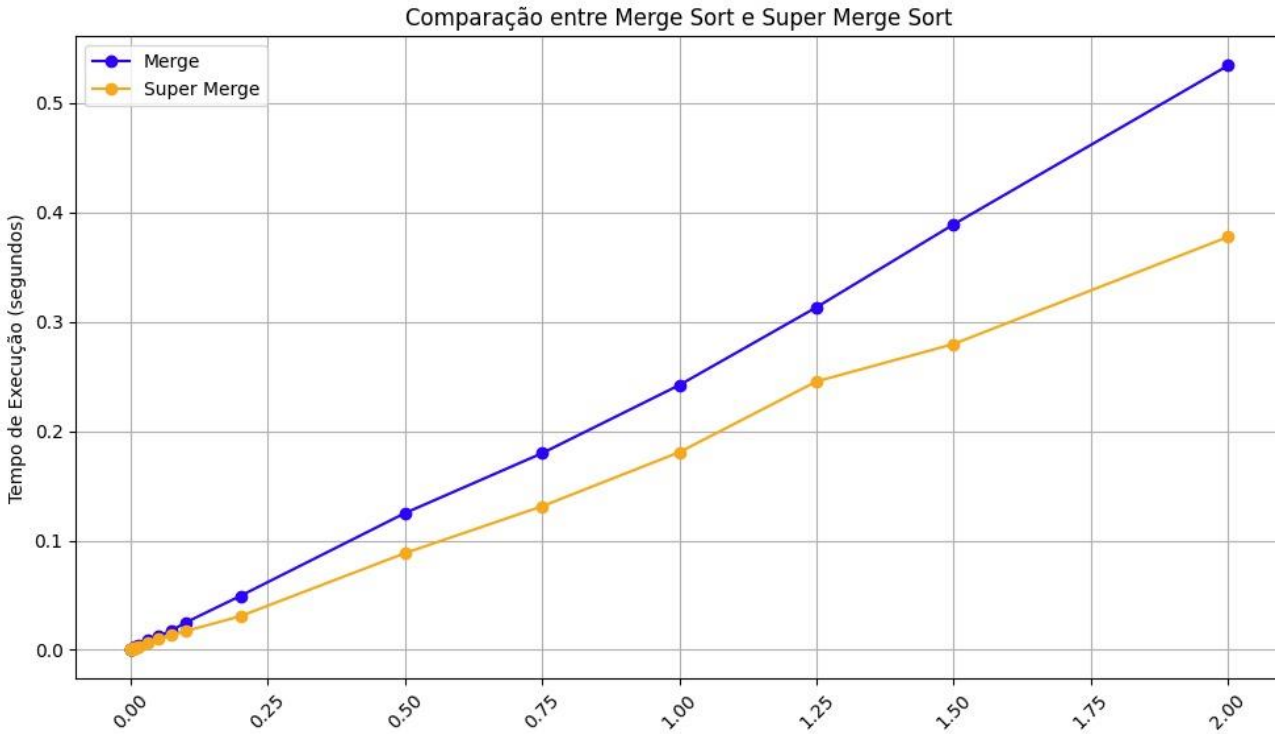

Gráfico 9 – Comparativo entre MergeSort e Super MergeSort testando entradas decrescentes (gráfico com alteração na escala).

4. Conclusão

Através dos resultados obtidos nos testes de desempenho podemos concluir que a aplicação do Super MergeSort obteve um resultado melhor que o MergeSort, esse fator se dá devido a sua divisão em quatro partes, onde facilitam o processo de ordenação e age quase paralelamente.

Em geral, o MergeSort com duas divisões é mais comum e amplamente utilizado. Ele é eficiente e adequado para a maioria dos casos de ordenação, apresentando uma complexidade de tempo de O(n log n), onde "n" é o tamanho da lista a ser ordenada.

Por outro lado, o MergeSort com quatro divisões pode ser útil em cenários muito específicos, onde a otimização de desempenho é uma prioridade e a divisão em quatro partes se ajusta melhor ao hardware ou à arquitetura do sistema. Isso pode ser relevante em casos de ordenação paralela, onde você deseja aproveitar ao máximo os núcleos do processador ou a capacidade de paralelismo disponível.

5. Referências

CORMEN, Thomas et al. Advanced Algorithms-CS 6/76101. 2009.