IMPACTO DAS RESTRIÇÕES DE HARDWARE NA EFICIÊNCIA DE ALGORITMOS DE ORDENAÇÃO: UM ESTUDO COMPARATIVO

Jonas Lima Cavalcante[1]; Rubens Fernandes Nunes[2]; Emanuel Ferreira Coutinho[3]

Resumo
A ordenação eficiente de dados tem desempenhado um papel vital em muitas aplicações, desde pesquisas rápidas até a otimização do desempenho de algoritmos mais complexos. No entanto, com o constante crescimento dos conjuntos de dados e informações disponíveis na Internet, a seleção do algoritmo de ordenação torna-se uma tarefa crucial, dada a diversidade de opções disponíveis. Assim, a comparação de algoritmos em diferentes cenários ajuda a identificar suas vantagens e limitações, contribuindo para uma escolha mais precisa em cada contexto. O objetivo deste trabalho é, portanto, realizar uma análise comparativa da eficiência dos algoritmos de ordenação *Merge Sort* e *Quick Sort*, a fim de identificar qual é o mais adequado para diferentes cenários de uso. A metodologia consiste em experiências utilizando duas máquinas virtuais e uma máquina física, onde são avaliados o consumo de memória e o tempo de execução. Os resultados mostram que o *Merge Sort*, apesar de consumir mais memória em todos os cenários avaliados, tem melhor desempenho em termos de tempo médio de execução nos casos com mais elementos, enquanto o *Quick Sort* mantém sua vantagem consistente em termos de consumo médio de memória nos três cenários avaliados.
Palavras-chave: Algoritmos de ordenação; eficiência; desempenho.

Abstract
Efficient data sorting has played a vital role in many applications, from quick searches to optimizing the performance of more complex algorithms. However, with the constant growth of data sets and information available on the Internet, the selection of the sorting algorithm becomes a crucial task, given the diversity of options available. Thus, comparing algorithms in different scenarios helps to identify their advantages and limitations, contributing to a more precise choice in each context. The aim of this work is therefore to carry out a comparative analysis of the efficiency of the Merge Sort and Quick Sort algorithms, in order to identify which is the most suitable for different usage scenarios. The methodology consists of experiments using two virtual machines and one physical machine, where memory consumption and execution time are evaluated. The results show that Merge Sort, despite consuming more memory in all the scenarios evaluated, performs better in terms of average execution time in cases with more elements, while Quick Sort maintains its consistent advantage in terms of average memory consumption in the three scenarios evaluated.
keywords: Sorting algorithms; efficiency; performance.

[1] Mestrando em Computação pelo Programa de Pós-graduação em Computação-PCOMP pela Universidade Federal do Ceará – UFC, Campus Quixadá. E-mail: jonasliimac@gmail.com.
[2] Doutor em Ciência da Computação pelo Programa de Pós-Graduação MDCC pela Universidade Federal do Ceará – UFC, Campus Quixadá. E-mail: jonasliimac@gmail.com.
[2] Doutor em Ciência da Computação pelo Programa de Pós-Graduação MDCC pela UFC, professor da Universidade Federal do Ceará -UFC, campus Quixadá. E-mail: rubensfn@gmail.com.
[3] Doutor em Ciência da Computação, professor da Universidade Federal do Ceará -UFC, campus Quixadá. E-mail: emanuel.coutinho@ufc.br.

1 Introdução

Ao passar dos anos, novos algoritmos de ordenação têm surgido. Enquanto alguns são de fácil implementação, existem também aqueles que são considerados mais complexos (Marcellino *et al*., 2021).

De forma simples, a ordenação é um processo que reorganiza uma lista de elementos na ordem correta (AL-KHARABSHEH et al., 2013). Esse rearranjo ocorre de acordo com algum critério específico (ADHIKARI, 2007), sendo essencial a utilização de algoritmos encarregados de organizar determinada estrutura (YANG et al., 2011).

Em tal contexto, a seleção do algoritmo ideal pode ser uma tarefa desafiadora, especialmente quando se considera a complexidade de diferentes algoritmos e as características específicas de cada conjunto de dados. Consequentemente, isso pode levar a gargalos de desempenho e a um uso ineficiente de recursos (DIKA and PREVALLA, 2010), afetando negativamente a experiência do usuário, caso o algoritmo seja escolhido de forma equivocada.

É preciso destacar que a eficiência dos algoritmos de ordenação pode ser significativamente afetada pelas restrições de *hardware*, o que pode levar a diferentes resultados em termos de tempo de execução e consumo de memória. Portanto, é importante avaliar o desempenho desses algoritmos em diferentes cenários de *hardware* para identificar qual é o mais adequado para cada situação.

Diante dessa problemática, tem-se como objetivo para este estudo realizar uma análise comparativa de eficiência de dois algoritmos, nomeados pela literatura de *Merge Sort* e *Quick Sort*, a fim de identificar qual o mais adequado para diferentes cenários de uso, com foco principal na análise do impacto das reduções de recursos em simulações de ambientes controlados.

A motivação principal para a realização de estudos comparativos de algoritmos de ordenação está na crescente quantidade de informações disponíveis na internet. Isso tem ocorrido devido ao crescimento do rasto digital, implicando na criação cada vez maior de dados (ZERANSKI AND SANCAK, 2021; RAFAEL, 2022).

Ademais, este estudo está estruturado da seguinte forma. Na Seção 2, são examinados trabalhos anteriores relacionados à pesquisa. A Seção 3 apresenta o referencial teórico, abordando conceitos fundamentais e teóricos relacionados aos algoritmos de ordenação. A Seção 4 concentra-se na execução prática, com três subseções que descrevem as métricas, os ambientes de experimento e as ferramentas utilizadas. Em seguida, na Seção 5, são analisados

e discutidos os resultados. Ao fim, a Seção 6 conclui o estudo, resumindo as principais descobertas e oferecendo sugestões de trabalhos futuros.

2 Trabalhos relacionados

Esta seção apresenta alguns trabalhos relacionados que buscaram analisar a eficiência de algoritmos de ordenação, com foco comparativo na memória ou no tempo de execução, identificando as semelhanças e diferenças relacionadas a este estudo.

No trabalho de Dika e Prevalla (2010), a eficiência do algoritmo *Merge Sort* foi avaliada em *desktops* com diferentes capacidades de processamento, considerando entradas de 10.000 a 100.000 dados desordenados em três cenários distintos. Os resultados indicaram que o desempenho do *Merge Sort* depende principalmente do processador, e não da memória. No entanto, o estudo foi limitado a um único algoritmo e a máquinas físicas com *hardware* ultrapassado.

Folador et al. (2014) usaram um *software* comparador com interface gráfica para analisar algoritmos de ordenação, sem restrições quanto ao ambiente de desenvolvimento ou à linguagem de programação. O *software* permitiu aos usuários comparar algoritmos como *Bubble Sort, Merge Sort, Quick Sort e Shell Sort*, mas não forneceu acesso ao código ou à visualização prévia dos resultados. Embora tenham considerado o consumo de memória, os resultados não abordaram o desempenho em termos de memória, mas apenas do tempo.

Marcellino *et al*. (2021), focando no tempo de execução, exploraram cinco algoritmos de ordenação. Inicialmente, com uma escala de entrada pequena, todos os algoritmos apresentaram baixo custo de tempo, algo que já se esperava. Contudo, na medida em que o tamanho da entrada de dados foi aumentando, o *Merge Sort* e o *Quick Sort* se destacaram, demonstrando uma vantagem significativa em termos de rapidez de processamento em comparação com os outros algoritmos. Isso influenciou a escolha desses dois algoritmos para uma análise mais aprofundada, considerando seus desempenhos notáveis ao lidar com bases de dados maiores.

Uma das semelhanças, portanto, é que tanto este estudo quanto os trabalhos relacionados abordam a análise da eficiência de algoritmos de ordenação, com foco na comparação de métricas de desempenho, como tempo de execução e uso de memória. Além disso, ambos utilizam conjuntos de dados para avaliar o desempenho dos algoritmos em diferentes cenários.

Entretanto, o estudo comparativo apresentado tem algumas diferenças em relação aos trabalhos relacionados: uma abordagem mais abrangente, com a análise de diferentes métricas de desempenho, como tempo de execução e uso de memória; utilização de conjuntos de dados

maiores; e diferentes configurações de *hardware* para avaliar o desempenho dos algoritmos em diferentes cenários, com o uso de máquinas virtuais com restrições de *hardware*.

Portanto, a abordagem mais abrangente e detalhada do estudo em relação aos trabalhos relacionados justifica a sua realização e contribui para o avanço do conhecimento na área de algoritmos de ordenação.

3 Fundamentação Teórica

Esta seção apresenta brevemente algumas noções e terminologias básicas sobre complexidade de algoritmos de ordenação de dados e as principais diferenças entre os dois algoritmos inicialmente escolhidos.

Em termos simples, a complexidade de um algoritmo está relacionada ao seu desempenho e eficiência em relação ao tamanho dos dados de entrada. Embora o tamanho da entrada influencie diretamente no tempo necessário para produzir o resultado (SUBANDIJO, 2011), quanto maior a eficiência do algoritmo, menor será o impacto do tamanho dos dados nesse tempo gasto. Portanto, a complexidade de um algoritmo é melhor compreendida ao analisar sua eficiência em diferentes cenários, e não apenas pelo tamanho absoluto dos dados.

O tempo de execução do algoritmo pode ser classificado em três grandes grupos, conforme tabela 1, nomeadamente pior caso, caso médio e melhor caso (GOLDREICH, 2006). O consumo da memória também é levado em consideração, assim como o tempo. De maneira geral, quanto maior a entrada, que no caso é fornecida através de uma lista de dados, mais tempo e memória são necessários.

Tabela 1 - Complexidades de tempo para os algoritmos de ordenação

ALGORITMO	PIOR CASO	CASO MÉDIO	MELHOR CASO
Merge Sort	*O (n log n)*	*O (n log n)*	*O (n log n)*
Quick Sort	*O (n²)*	*O (n log n)*	*O (n log n)*

Fonte: autoria própria (2023).

3.1 Merge Sort

O *Merge Sort* é um algoritmo eficiente para ordenar grandes conjuntos de dados. Seu processo envolve dividir a lista de dados em duas metades, ordenar cada metade separadamente de forma recursiva e, em seguida, combinar as metades ordenadas para criar uma lista ordenada (BHARGAVA, 2018; DE AZEVEDO, 2016). Isso reflete na criação de uma árvore de recursão com $log_2(n)$ níveis, onde cada nível da árvore representa uma divisão adicional do conjunto de ‘n’ dados.

A recursão continua até que cada subdivisão contenha apenas um elemento, considerado o caso base. Devido a essa propriedade logarítmica, considerando que a quantidade de trabalho necessário em cada nível é proporcional ao tamanho dos dados, sua complexidade de tempo é de *O(n log n)*. Uma vantagem da divisão dos dados é permitir que o algoritmo seja paralelizado para aproveitar múltiplos processadores ou *threads* (CENTURION, 1998).

No entanto, é um algoritmo que requer memória extra para armazenar subconjuntos durante a ordenação (NAKASHIMA, 2017), o que pode ser uma desvantagem em comparação com outros algoritmos. A complexidade de espaço no *Merge Sort* é *O(n)*, pois em cada nível é necessário armazenar temporariamente as duas metades da lista sendo ordenada (GEEKSFORGEEKS, 2023). Mas depois que a combinação é concluída para um determinado nível, os espaços de armazenamento temporários podem ser liberados (AFONSO, 2009).

3.2 Quick Sort

O *Quick Sort* é outro algoritmo de ordenação que também utiliza a abordagem de divisão e conquista. Seu processo consiste em escolher um elemento pivô, particionar a lista em torno do pivô, e ordenar recursivamente as duas partições. O particionamento é uma operação linear que consiste em posicionar corretamente o pivô na lista, colocando todos os elementos menores a sua esquerda e todos os maiores a sua direita (POLLMAN, 2009; TOSCANI AND VELOSO, 2001).

O algoritmo *Quick Sort* se destaca em cenários em que as partições são equilibradas, ou seja, quando o pivô escolhido divide a lista aproximadamente em metades iguais (FOLADOR, et al., 2014). Nesses casos, a complexidade total é *O(n log n)*. No pior caso, o pivô escolhido pode ser sempre o menor ou o maior elemento da lista, resultando em partições desequilibradas. Isso leva a uma complexidade de tempo quadrática de $O(n^2)$.

Diferentemente do *Merge Sort*, o *Quick Sort* é um algoritmo *in-place*, o que significa que ele não requer espaço adicional para armazenar cópias temporárias da lista (RAJPUT et al., 2012). O espaço adicional é necessário apenas para a pilha de chamadas recursivas, cujo tamanho é determinado pela profundidade máxima da recursão, que no caso geral é de *O(log n)*. Mas no pior caso, quando há muitas chamadas recursivas, o espaço para a pilha recursiva pode ser *O(n)*. O desempenho do *Quick Sort* varia em diferentes cenários da seguinte forma: excelente, para listas aleatórias, grandes conjuntos de dados e aplicações em tempo real; baixo, em dados já ordenados, dados repetidos e listas quase ordenadas.

4 Procedimentos e métodos

Para alcançar os objetivos deste trabalho, a metodologia empregada nesta seção é experimental, adotada por Drumond (2012) e Bentley (2016). Com isso, são avaliadas métricas, como o tempo de execução e o uso de memória, a partir de bases de dados com tamanhos distintos, para diferentes algoritmos, em diferentes cenários de uso.

4.1 Métricas

As métricas adotadas foram o tempo de execução e o consumo de memória. Foram também consideradas entradas de dados de diferentes tamanhos. Para garantir resultados confiáveis, foram realizadas 10 execuções para cada tamanho de entrada de dados (1.000.000, 4.000.000 e 8.000.000 de elementos), a fim de obter a média dos tempos de execução e dos consumos de memória. Essa abordagem de múltiplas execuções permite reduzir a influência de variações aleatórias nos resultados, proporcionando uma visão mais precisa do desempenho dos algoritmos em diferentes cenários. O tempo de execução foi medido em milissegundos (ms) e o consumo de memória foi medido em megabytes (MB).

4.2 Ambientes de Experimento

Nesta subseção, são apresentados os ambientes de experimento utilizados, compreendendo três máquinas com configurações distintas. O Cenário 1 serve como referência, envolvendo uma máquina com configurações superiores. Sua instalação estabelece um padrão ideal de desempenho, onde o experimento é realizado em uma máquina física (**Tabela 2**), equipada com as seguintes configurações:

Tabela 2 - Cenário 1 de experimento.

COMPONENTE	**ESPECIFICAÇÕES**
Processador	*AMD Ryzen* 5 5500U
Velocidade de *Clock*	2,10 GHz
Threads (Núcleos Lógicos)	12
Memória RAM Disponível	17,8 GB
Sistema Operacional	*Windows* 11 (64 bits)

Fonte: autoria própria (2023).

A máquina conta com um processador de 6ª geração, capaz de lidar com cargas de trabalho complexas e paralelas, assim como seu total de memória RAM (Random Access Memory) disponível permite a execução de processos e algoritmos que demandam uma considerável quantidade de memória.

Logo, esse cenário estabelece um ponto de referência robusto para a avaliação de desempenho dos algoritmos, que é crucial para compreender como eles se comportam em condições ideais. A partir dele, comparações são feitas com máquinas virtuais (Cenários 2 e 3) para analisar o impacto das reduções de recursos nos resultados experimentais.

No Cenário 2, a máquina virtual está configurada para operar em um ambiente controlado, caracterizado por recursos mais limitados, incluindo uma redução significativa na quantidade total de memória RAM. Conforme a **Tabela 3,** o ambiente foi criado através do *Virtual Box*, com a instalação do sistema operacional *Ubuntu*, com a versão mais recente até o momento desta aplicação.

Tabela 3 - Cenário 2 de experimento.

COMPONENTE	ESPECIFICAÇÕES
Processador	*AMD Ryzen* 5 5500U
Velocidade de *Clock*	2,10 GHz
Threads (Núcleos Lógicos)	12
Memória RAM Disponível	4,0 GB
Sistema Operacional	*Ubuntu* 23.10.1 (64 bits)
Software de Virtualização	*VirtualBox* (*Oracle*)

Fonte: autoria própria (2023).

Essa abordagem isola o experimento de interferências de outros processos ou programas em execução no sistema operacional hospedeiro. Além disso, é possível criar diferentes configurações de *hardware* para a máquina virtual, o que permite testar o desempenho dos algoritmos em diferentes cenários. Como é caso do Cenário 3, semelhante ao Cenário 2, que conta com apenas uma *thread* disponível (Tabela 4). Com a redução no número de *threads* disponíveis, espera-se que o desempenho dos algoritmos de ordenação seja afetado, uma vez que a capacidade de processamento da máquina virtual é reduzida.

Tabela 4 - Cenário 3 de experimento.

COMPONENTE	ESPECIFICAÇÕES
Processador	AMD Ryzen 5 5500U
Velocidade de Clock	2,10 GHz
Threads (Núcleos Lógicos)	1
Memória RAM Disponível	4,0 GB
Sistema Operacional	Ubuntu 23.10.1 (64 bits)
Software de Virtualização	VirtualBox (Oracle)

Fonte: autoria própria (2023).

Isso ocorre porque o paralelismo depende da disponibilidade de múltiplos *threads* para executar tarefas simultaneamente. Além de poder levar a um aumento no tempo de execução, podem acontecer alguns casos de levar até mesmo a falhas na execução.

4.3 Ferramentas

Para que os algoritmos sejam avaliados nos cenários, os 3 experimentos são conduzidos utilizando a ferramenta Visual Studio Code, na versão 1.82 (2023), e a linguagem de programação Java. Para permitir a execução do código, é empregada a extensão "Extension Pack for Java". Essa padronização do ambiente de desenvolvimento e da linguagem são requisitos importantes para a realização do experimento, a fim de garantir uma plataforma estável para a análise e implementação dos algoritmos em questão.

5 Apresentação dos resultados

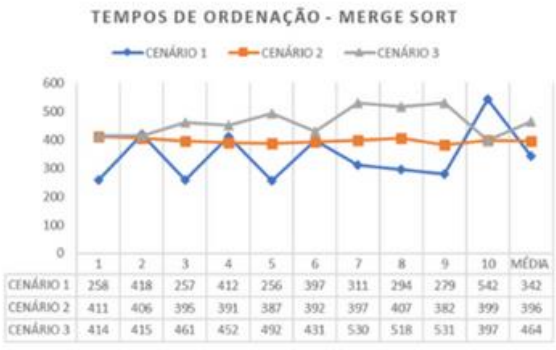
Nesta seção são apresentados os resultados comparativos dos algoritmos de ordenação *Merge Sort* e *Quick Sort*. em tabelas e gráficos, a partir dos diferentes conjuntos de dados, mostrando o tempo de execução e o consumo de memória para cada cenário. Em seguida, os resultados obtidos são analisados e interpretados para cada algoritmo, nos três cenários, destacando e discutindo o impacto das restrições de *hardware*.

5.1 Ordenação – 1.000.000 de dados

A análise comparativa entre os algoritmos de ordenação Merge Sort e Quick Sort revelou diferenças significativas nas médias relacionadas ao desempenho dessas abordagens nos três diferentes cenários, conforme mostrado na Figura 1.

Figura 1 - Tempo médio para ordenação de 1.000.000 elementos

	1	2	3	4	5	6	7	8	9	10	MÉDIA
CENÁRIO 1	216	195	180	174	269	184	186	186	182	179	195
CENÁRIO 2	213	226	192	168	192	199	251	256	248	245	219
CENÁRIO 3	223	252	304	350	352	324	369	337	356	345	321



	1	2	3	4	5	6	7	8	9	10	MÉDIA
CENÁRIO 1	258	418	257	412	256	397	311	294	279	542	342
CENÁRIO 2	411	406	395	391	387	392	397	407	382	399	396
CENÁRIO 3	414	415	461	452	492	431	530	518	531	397	464

Fonte: autoria própria (2023).

Observando os resultados para conjuntos de 1.000.000 de elementos, é evidente que o *Quick Sort* superou o *Merge Sort* em termos de tempo de execução nos três cenários propostos. Em média, o *Quick Sort* apresentou tempos significativamente menores do que o *Merge Sort*, indicando uma eficiência temporal superior para conjuntos de dados dessa magnitude. Esse desempenho mais ágil do *Quick S*ort pode ser atribuído à sua natureza de ordenação *in-place* e

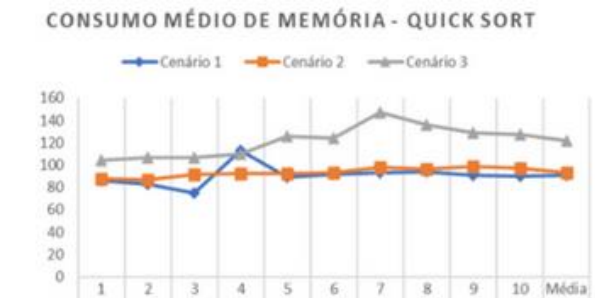
à menor sobrecarga associada ao gerenciamento de memória, o que se torna mais proeminente em conjuntos de dados menores.

No que diz respeito ao consumo médio de memória, de acordo com a **Figura 2**, o *Quick Sort* também se destacou nos três cenários. Esse comportamento pode ser explicado também pela sua característica *in-place*, resultando em uma menor quantidade de memória adicional necessária para realizar as operações de ordenação.

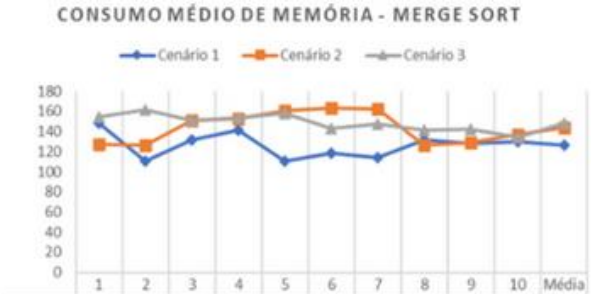
É importante observar que, embora o *Quick S*ort tenha demonstrado um desempenho superior nos cenários de 1.000.000 de elementos, as características específicas de cada algoritmo devem ser levadas em conta ao escolher o mais adequado para uma determinada aplicação.

O *Quick Sort* é conhecido por sua eficiência em conjuntos de dados menores, enquanto o *Merge Sort*, apesar de um desempenho ligeiramente inferior nesses cenários, pode se mostrar mais estável e eficaz em cenários de ordenação de grandes conjuntos de dados.

Figura 2 - Consumo médio de memória para ordenação de 1.000.000 elementos



	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	85,91	82,97	75,07	112,91	89,52	91,59	92,91	93,77	90,53	89,69	90,48
Cenário 2	87,93	86,49	91,57	92,64	92,03	93,47	97,67	96,55	99,01	97,11	93,44
Cenário 3	104,45	106,48	106,33	109,47	125,45	123,96	146,78	135,87	129,06	127,22	121,5



	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	148,45	110,49	132,12	141,47	110,35	118,56	113,68	131,82	127,93	129,7	126,45
Cenário 2	127,31	126,36	151,35	152,62	160,81	163,61	162,28	126,84	129,44	137,01	143,76
Cenário 3	154,28	162,01	151,07	153,92	158,12	142,94	147,77	141,81	142,06	134,29	148,82

Fonte: autoria própria (2023).

5.2 Ordenação – 4.000.000 de dados

Contrariamente ao desempenho observado nos conjuntos de 1.000.000 de elementos, o *Merge Sort* demonstrou uma melhoria notável em termos de tempo de execução, superando o *Quick Sort* em média nos três cenários propostos (Figura 3).

Figura 3 - Tempo médio para ordenação de 4.000.000 elementos

	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	768	738	781	722	731	749	734	775	752	741	749,1
Cenário 2	863	776	862	758	762	869	771	776	838	725	800
Cenário 3	992	906	941	943	951	945	952	962	1021	1004	961,7

	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	591	674	643	649	703	701	649	599	681	675	656,5
Cenário 2	646	639	721	762	783	733	701	707	711	699	710,2
Cenário 3	824	825	741	782	782	771	748	808	761	793	783,5

Fonte: autoria própria (2023).

Entretanto, ao considerar o consumo médio de memória (**Figura 4**), o *Quick Sort* manteve sua vantagem consistente nos três cenários. O consumo de memória mais elevado do *Merge Sort* pode ser atribuído ao uso de memória adicional devido ao armazenamento temporário necessário das duas metades da lista sendo ordenada em cada nível da recursão.

Figura 4 - Consumo médio de memória para ordenação de 4.000.000 elementos

	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	178,51	175,48	211,59	169,83	171,73	215,82	224,11	172,52	231,61	178,19	192,94
Cenário 2	240,11	313,29	210,12	229,07	251,01	218,02	261,91	215,95	282,15	241,06	246,27
Cenário 3	321,11	322,91	331,04	309,01	361,2	357,92	361,89	348,73	359,38	363,09	343,63

	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	292,85	250,81	281,28	255,01	354,22	326,52	409,5	421,87	412,49	359,33	336,39
Cenário 2	327,31	316,29	301,52	312,47	331,11	333,18	379,28	421,72	316,71	417,86	345,75
Cenário 3	424,1	422,91	451,94	459,12	452,31	451,92	463,89	401,73	442,38	467,19	443,75

Fonte: autoria própria (2023).

Comparando os tempos de execução mostrados nas Figuras 1 e 3, pode-se ressaltar a natureza estável e escalável do *Merge Sort*, tornando-o mais adequado para conjuntos de dados mais extensos. O *Merge Sort*, apesar de consumir mais memória, revelou-se uma escolha mais eficaz na medida em que o tamanho do conjunto de dados aumentou, proporcionando tempos de execução mais vantajosos em comparação com o *Quick Sort.*

Esta observação destaca a importância de considerar não apenas o desempenho em cenários específicos, mas também a escalabilidade e a capacidade de adaptação dos algoritmos às mudanças no tamanho dos conjuntos de dados.

5.3 Ordenação – 8.000.000 de dados

Os resultados para conjuntos de 8.000.000 de elementos revelam uma continuidade na tendência observada nos conjuntos menores, com o *Merge Sort* consumindo mais memória em

comparação com o *Quick Sort*, enquanto apresenta tempos de execução significativamente inferiores em todos os cenários.

A média de tempo de execução para o *Merge Sort* foi notavelmente melhor em todos os cenários, registrando 1292,9 milissegundos no cenário 1, 1491,4 no cenário 2 e 1816,5 no cenário 3. Por outro lado, o *Quick Sort* apresentou médias mais altas, com 2441,5, 2618,3 e 3085,4 milissegundos para os cenários 1, 2 e 3, respectivamente (Figura 5).

Figura 5 - Tempo médio para ordenação de 8.000.000 elementos

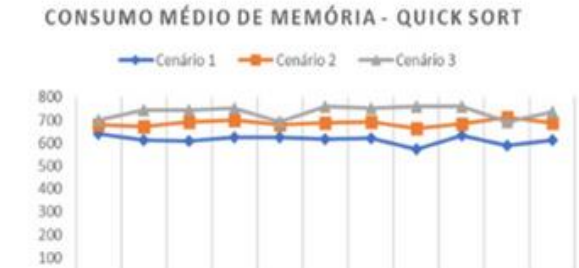
	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	2442	2436	2406	2461	2467	2433	2416	2459	2443	2452	2441,5
Cenário 2	2613	2651	2524	2513	2581	2769	2719	2561	2510	2742	2618,3
Cenário 3	3012	3006	3253	3257	3187	3185	2971	2983	2998	3002	3085,4

	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	1348	1226	1415	1322	1318	1231	1414	1192	1278	1185	1292,9
Cenário 2	1490	1415	1444	1457	1478	1541	1533	1411	1563	1582	1491,4
Cenário 3	1870	1836	1790	1798	1721	1751	1719	1819	1770	2091	1816,5

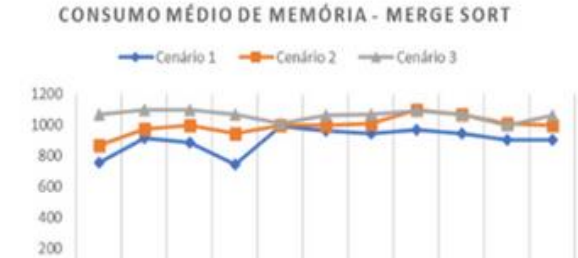
Fonte: autoria própria (2023).

Quanto ao consumo de memória, o *Merge Sort* continuou a ser um problema, com médias de 901,59, 996,86 e 1062,51 megabytes (MB) nos cenários 1, 2 e 3, respectivamente, quando comparado ao *Quick Sort* (Figura 6).

Figura 6 - Consumo médio de memória para ordenação de 8.000.000 elementos.



	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	641,01	611,27	610,51	625,43	622,94	616,1	621,03	572,32	633,38	588,2	614,22
Cenário 2	680,61	673,74	690,32	699,2	681,03	688,29	691,62	665,37	682,31	711,19	686,37
Cenário 3	701,71	742,74	741,84	749,73	691,25	757,75	751,65	758,43	759,32	693,12	734,75



	1	2	3	4	5	6	7	8	9	10	Média
Cenário 1	755,3	913,26	884,76	744,59	994,76	960,67	942,46	968,12	947,34	904,63	901,59
Cenário 2	867,67	976,67	997,67	947,53	999,3	998,65	1007,7	1097,3	1068,7	1007,4	996,86
Cenário 3	1067,6	1096,6	1097,3	1065,6	1007,7	1059,8	1068,8	1094,1	1067,5	1000,2	1062,5

Fonte: autoria própria (2023).

5.4 Análises e Discussões

A análise dos conjuntos de dados de 1.000.000, 4.000.000 e 8.000.000 de elementos, em diferentes cenários, forneceu resultados significativos sobre o impacto das restrições de *hardware* em ambientes controlados, com a execução dos algoritmos *Merge Sort* e o *Quick Sort*.

No Cenário 1, caracterizado por uma máquina física com configurações avançadas, observou-se um padrão ideal de desempenho, em que o *Merge Sort* se destacou com eficiência notável, apresentando tempos de execução mais baixos, enquanto o *Quick Sort*, mesmo com tempos de execução superiores, demonstrou desempenho competitivo.

No Cenário 2, uma máquina virtual com configurações semelhantes, mas com restrições de memória, destacou-se a capacidade do *Merge Sort* de resistir a essas restrições, mantendo tempos de execução inferiores, enquanto o *Quick Sort*, embora tenha experimentado aumentos nos tempos de execução, permaneceu eficiente, evidenciando sua adaptabilidade a ambientes controlados.

Por fim, no Cenário 3, caracterizado por uma máquina virtual com restrição de *threads*, observou-se consistentemente um aumento no consumo de memória e tempo de execução em comparação com os Cenários 1 e 2. A restrição de *hardware*, limitando a execução dos algoritmos a apenas uma *thread*, teve um impacto significativo nos resultados, ressaltando a importância do paralelismo em operações de ordenação.

Dentre as possíveis limitações do estudo, destacam-se a utilização de conjuntos de dados com tamanhos e diversidade limitados, o que pode comprometer a generalização dos resultados para diferentes tipos de dados e tamanhos de entrada. Além disso, a concentração em ambientes experimentais específicos, como máquinas físicas e virtuais com configurações particulares, pode restringir a aplicabilidade dos resultados a outros ambientes de *hardware* e *software*.

A eficiência dos algoritmos também pode ser sensível à implementação específica em termos de linguagem de programação e ambiente de desenvolvimento, introduzindo variações nos resultados que podem não refletir no desempenho dos algoritmos. As métricas utilizadas para avaliar o desempenho, como tempo de execução e consumo de memória, podem não ser completamente abrangentes, não capturando totalmente a eficiência dos algoritmos em cenários diversos.

Essas possíveis limitações ressaltam a importância de interpretar os resultados com cautela e considerar contextos específicos ao aplicar as conclusões do estudo.

6 Conclusão e Trabalhos Futuros

Por meio deste trabalho, buscou-se realizar uma análise comparativa dos algoritmos *Merge Sort* e *Quick Sort*, em três cenários distintos. Para isso, foram utilizados, como ambientes de experimento: uma máquina física, com recursos avançados, sendo 17,8 GB de memória RAM disponíveis e um processador com 12 *threads*, de 6ª geração; e outras duas máquinas

virtuais que foram criadas, através da ferramenta *VirtualBox*, com restrições de memória e redução no número de *threads.*

A escolha dos tamanhos de conjunto de dados foi estrategicamente determinada em relação ao poder de processamento dos algoritmos em questão. O tamanho de 1.000.000 serviu como um ponto de referência para conjuntos de dados consideráveis, enquanto 4.000.000 e 8.000.000 desafiaram ainda mais os algoritmos, explorando seu potencial em lidar eficientemente com volumes altos de dados.

Os resultados, portanto, ao terem sido apresentados, confirmam ser o *Merge Sort* o algoritmo de ordenação mais eficiente em termos de tempo de execução em todos os cenários testados usando conjuntos de dados maiores, enquanto o *Quick Sort* é o mais eficiente em termos de consumo de memória em todos os cenários testados, independente do tamanho da entrada.

Em resumo, no Cenário 1, destacou-se um desempenho ideal, evidenciando a eficiência notável do *Merge So*rt e a competitividade do *Quick So*rt. Já no Cenário 2, as restrições de memória foram superadas pelo *Merge Sort*, enquanto o *Quick Sort* manteve eficiência em ambientes controlados. O Cenário 3, com restrição de *threads*, evidenciou a importância do paralelismo, afetando o desempenho de ambos os algoritmos.

Ou seja, o *Merge Sort* demonstrou uma eficiência notável em conjuntos de dados maiores, apresentando tempos de execução consistentemente melhores. Entretanto, a análise também revelou que, em cenários com restrições de *hardware*, como a limitação de *threads*, ambos os algoritmos foram afetados, indicando a necessidade de considerações especiais ao escolher algoritmos em ambientes com recursos limitados.

Para trabalhos futuros, destaca-se a importância de se explorar outros cenários, linguagens de programação, métricas de desempenho, como o número de comparações e trocas realizadas pelos algoritmos de ordenação, proporcionando uma visão mais completa e refinada das escolhas algorítmicas em diversas aplicações. Para isso, sugere-se investigar o desempenho de algoritmos de ordenação mais recentes e avançados, como o *Tim Sort* e o *Heap Sort*.

7 Agradecimentos

Este trabalho foi desenvolvido durante a disciplina de Análise de Desempenho, no segundo semestre de 2023, pelo Programa de Pós-Graduação em Computação (PCOMP), com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

ADHIKARI, P. **Review On Sorting Algorithms-A comparative study on two sorting algorithms.** Mississippi State, Mississippi, 2007.

AFONSO, J. M. D. **Optimização de estruturas multidimensionais de dados em ambientes OLAP.** 2019. 179 f. Dissertação de Mestrado (Departamento de Ciências e Tecnologia da Informação) - Instituto Universitário de Lisboa. Lisboa, 2019. Disponível em: https://repositorio.iscte-iul.pt/bitstream/10071/3858/4/master_jorge_dias_afonso.pdf. Acesso em: 05 fev. 2025.

AL-KHARABSHEH, K. S. *et al*. Review on sorting algorithms a comparative study**. International Journal of Computer Science and Securit**, v. 7, n. 3, 2013, p. 120-126. Disponível em: https://www.cscjournals.org/manuscript/Journals/IJCSS/Volume7/Issue3/IJCSS-877.pdf. Acesso em: 05 fev. 2025.

BENTLEY, J. **Programming pearls.** Boston**:** Addison-Wesley Professional, 2016.

BHARGAVA, A. Y. **Entendendo Algoritmos: Um guia ilustrado para programadores e outros curiosos.** São Paulo: Novatec Editora, 2018.

CENTURION, A. M. **Análise de desempenho de algoritmos paralelos utilizando plataformas de portabilidade.** 1998. 184 f. Dissertação de Mestrado (Instituto de Ciências Matemáticas e de Computação) Universidade de São Paulo). São Carlos. 1998. Disponível em:https://www.teses.usp.br/teses/disponiveis/55/55134/tde-12032018-162030/pt-br.php. Acesso em: 05 fev. 2025.

DE AZEVEDO, R. C. **Análise dos algoritmos de ordenação**: buscando a eficiência em sistemas de supermercados. 2016.

DIKA, A.; PREVALLA, B. The impact of memory and processor in determining the performance of programs. In: **2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel**. IEEE, p. 000015-000018, 2010. Disponível em:https://ieeexplore.ieee.org/abstract/document/5661956/. Acesso em: 05 fev. 2025.

DRUMOND, P. M. L. D. L. **Uma análise experimental de algoritmos exatos aplicados ao problema da árvore geradora multiobjetivo.** 2012. 84 f. Dissertação de Mestrado (Programa de Pós-Graduação em Sistemas e Computação) - Universidade Federal do Rio Grande do Norte. Natal, 2012. Disponível em: https://repositorio.ufrn.br/bitstream/123456789/18047/1/PatriciaMLLD_DISSERT.pdf. Acesso em: 05 fev. 2025.

FOLADOR, J. P.; NETO, L. N. P.; JORGE, D. C. Aplicativo para análise comparativa do comportamento de algoritmos de ordenação. **Revista Brasileira de Computação Aplicada**, v. 6, n. 2, 2014, p. 76–86. Disponível em: https://seer.upf.br/index.php/rbca/article/view/3792. Acesso em: 05 fev. 2025.

GEEKSFORGEEKS. **Merge sort.** 2023. Disponível em:https://www.geeksforgeeks.org/merge-sort/. Acesso em: 05 fev. 2025.

GOLDREICH, O. **Computational complexity:** a conceptual perspective. Rehovot-Israel: Weizmann Institute of Science, 2006. Disponível em:https://courses.cs.duke.edu/spring07/cps240/books/G/Gbook.pdf. Acesso em: 05 fev. 2025.

MARCELLINO, M. *et al*. Comparative of advanced sorting algorithms (quick sort, heap sort, merge sort, intro sort, radix sort) based on time and memory usage**.** In: **2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)**, v. 1, pages 154–160, Jakarta, Indonesia, 2021. Disponível em: https://ieeexplore.ieee.org/document/9609715/citations#citations. Acesso em: 05 fev. 2025.

NAKASHIMA, E. Y. **Avaliação de atividades de programação submetidas em mooc com emprego de técnicas de visualização.** 2017. 64 f. Trabalho de Conclusão de Curso (Curso de Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Campo Mourão. 2017. Disponível em: https://repositorio.utfpr.edu.br/jspui/handle/1/6016. Acesso em: 05 fev. 2025.

POLLMAN, H. J. S. Probabilistic cost analysis of logic programs**. Ingeniare. Revista chilena de ingeniería**, Santiago do Chile, v. 17, n. 2, p. 195–204, 2009. Disponível em:https://www.scielo.cl/pdf/ingeniare/v17n2/art08.pdf. Acesso em: 05 fev. 2025.

RAJPUT, I. S.; KUMAR, B.; SINGH, T. Performance comparison of sequential quick sort and parallel quick sort algorithms**. International Journal of Computer Applications**, v. 57, n. 9, p. 14–22, 2012. Disponível em: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=16f2590017d1cf27f60d869366ce281eb5e00802. Acesso em: 05 fev. 2025

RAFAEL, A. O. **As implicações econômicas e sociais resultantes da implementação de inteligência artificial pelo estado, em especial pelo setor regulador.** 2022. 48 f. Dissertação de Mestrado (Departamento de Economia Política) - Instituto Universitario de Lisboa. Lisboa. 2022. Disponível em: https://repositorio.iscte-iul.pt/bitstream/10071/27093/1/master_adolfo_oliveira_rafael.pdf. Acesso em: 05 fev. 2025.

SUBANDIJO, S. Efisiensi algoritma dan notasi o-besar**. ComTech,** v**.** 2, n. 2, p. 849–858, 2011. Disponível em: https://journal.binus.ac.id/index.php/comtech/article/view/2835/2230. Acesso em: 05 fev. 2025.

TOSCANI, L. V.; VELOSO, P. A. **Complexidade de algoritmos.** Porto Alegre: Sagra-Luzzatto. Porto Alegre, 2011.

YANG, Y., YU, P., AND GAN, Y. **Experimental study on the five sort algorithms.** In: **2011 Second International Conference on Mechanic Automation and Control Engineering. IEEE**, p. 1314-1317, 2011. Disponível em:https://ieeexplore.ieee.org/abstract/document/5987184. Acesso em: 05 fev. 2025.

ZERANSKI, S.; SANCAK, I. E. Prudential supervisory disclosure (PSD) with supervisory technology (SupTech): lessons from a FinTech crisis. **International Journal of Disclosure and Governance**, v. 18, n. 4, p. 315-335, 2021. Disponível em:https://link.springer.com/article/10.1057/s41310-021-00111-7. Acesso em: 05 fev. 2025.