Estrutura de Dados
Trabalho de Pesquisa e Ordenação

Relatório
Ismael dos Santos, Lutercio Jackson

Esse relatório tem como objetivo analisar o desempenho de diferentes tipos de algoritmos de busca e de ordenação com diferentes tamanhos de vetores. Primariamente as especificações da máquina utilizada são as seguintes:

Tablet

Nome	Samsung Galaxy Tab S9 FE
Processador	4x 2.4 GHz + 4x 2.0 GHz
Chipset	Samsung Exynos 1380 (ARM)
GPU	Mali-G68 MP5
RAM	6 GB
Bateria	LiPo 8000 mAh
OS	Android 14 OneUI 6.1

Virtual Machine

Nome	Ubuntu
OS	Ubuntu 22.04.4 LTS aarch64
Kernel	5.15.123-android13-3-28577532
Packages	447 (dpkg)
Terminal	proot
CPU	8
Memória (RAM)	5706 MiB (Reservado)

Todos os testes foram feitos por meio de virtualização na VM listada acima, por isso também foi informado as suas especificações.

Compilador (GNU)

Versão	11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)
Comando utilizado	gcc medirtempo.c -o main && time .\main

Resultados

Para os algoritmos de ordenação, foram feitos 3 testes para cada um dos 5 tamanhos de vetor de inteiros utilizados. Já para os de busca, somente foram feitos os 3 testes com o vetor ordenado de forma crescente.

Busca Sequencial:

A busca sequencial, ou busca linear, é o método mais simples de procurar um elemento em uma lista ou vetor. Ele verifica cada elemento da lista, um por um, até encontrar o elemento desejado ou até ter verificado todos os elementos.

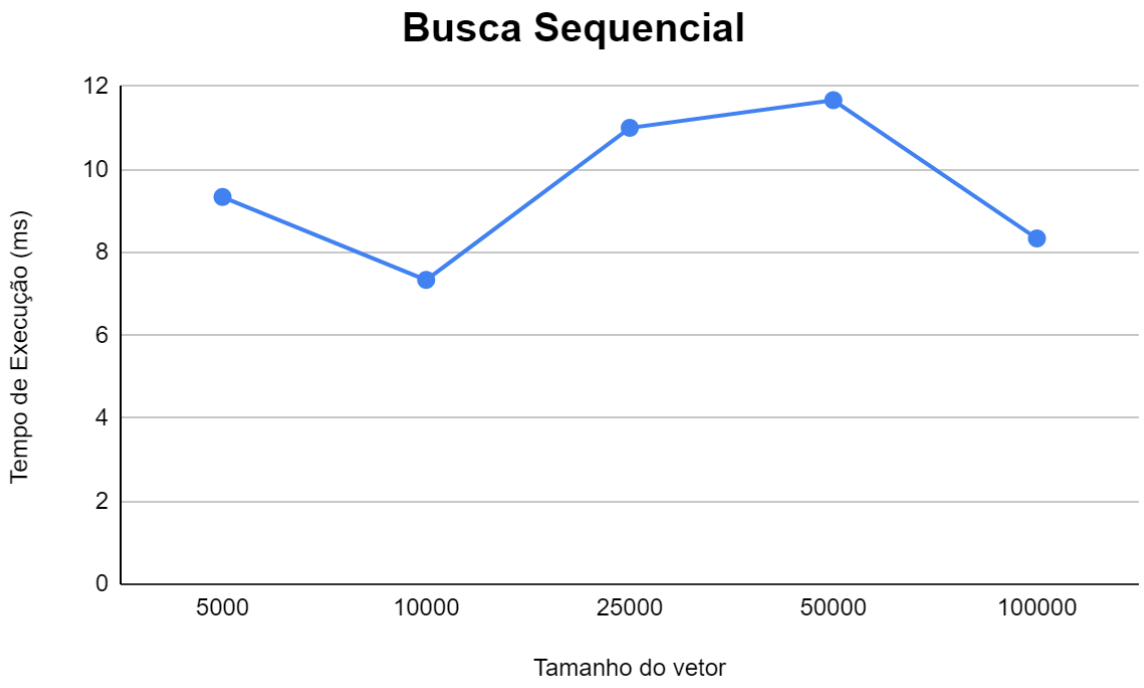

Gráfico 1: Desempenho da Busca Sequencial.

Aqui é visto algo que irá se replicar por parte na Busca binária, a taxa de amostragem do tamanho dos vetores testados parece ser insuficiente para ser notável uma diferença considerável, conforme o número de elementos escala.

Tendo isto em vista, o tempo de execução de todos os tamanhos de vetores (5000, 10000, 25000, 50000 e 100000) é praticamente igual.

Busca Binária:

A busca binária funciona dividindo repetidamente a lista ou vetor pela metade e comparando o elemento alvo com o elemento do meio.

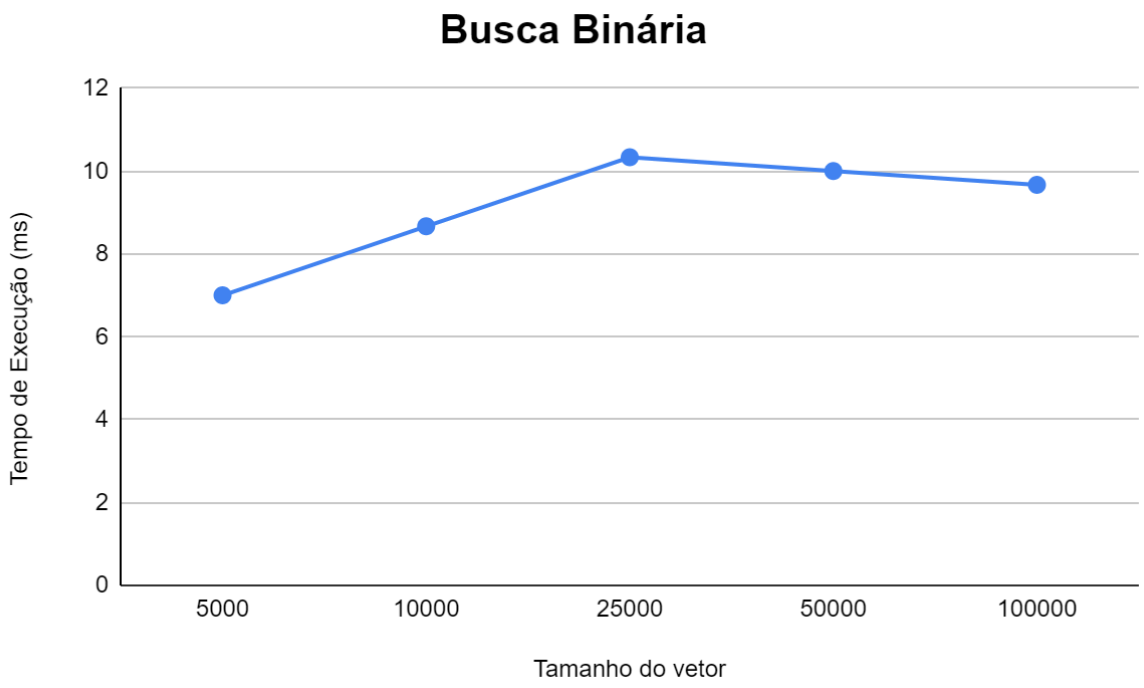

Gráfico 2: Desempenho da Busca Binária.

Como adiantado no método anterior, a amostragem também parece ser um limitante para analisar a diferença entre os tamanhos de vetores, a diferença é visível uma tendência maior a se tornar uma quase constante, perceptível após o ponto 25000, indicando um gráfico mais parecido ao de uma escala logarítmica.

Busca Sequencial X Busca Binária :

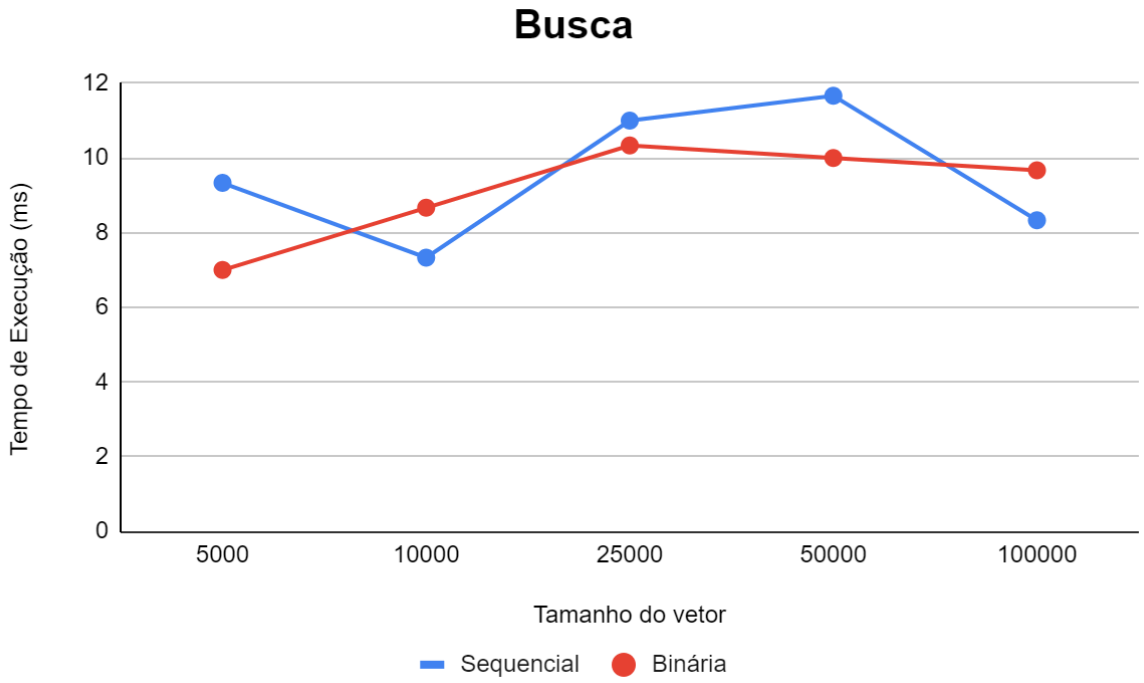

Gráfico 3: Comparativo do tipo de busca utilizado.

No comparativo direto entre os dois métodos de busca utilizados, para o número de elementos testados o desempenho foi próximo, a diferença marcante entre eles é somente a tendência, onde a busca sequência tende a seguir mais linearmente conforme o número de elementos aumenta (exceção da anomalia do ponto 100000), enquanto a busca binária tende a uma escala logarítmica.

Bubble Sort:

O Bubble Sort compara repetidamente pares de elementos adjacentes e os troca se estiverem na ordem errada. Este processo é repetido até que a lista esteja ordenada.

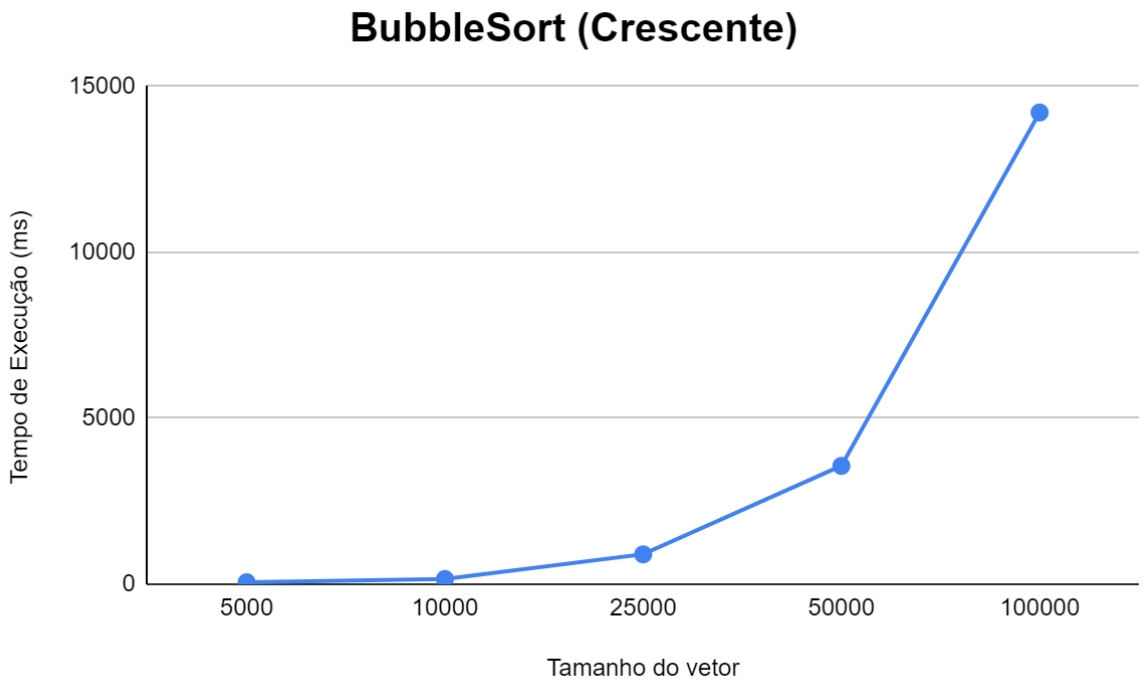

Gráfico 4: Desempenho do Bubble Sort em um vetor crescente.

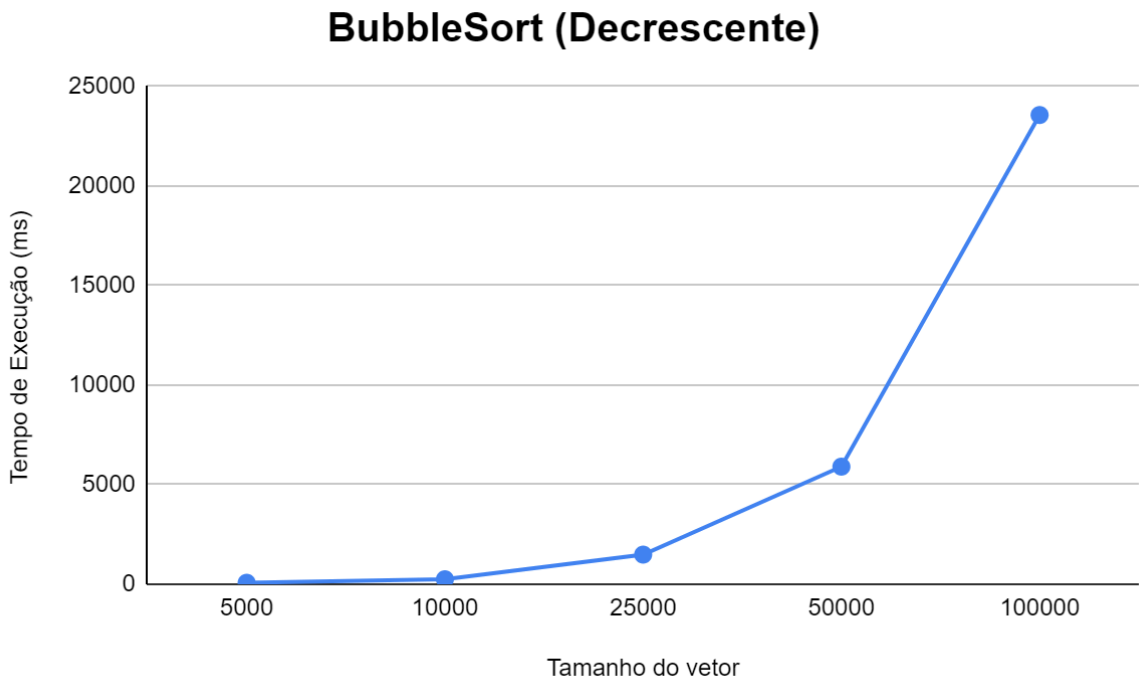

Gráfico 5: Desempenho do Bubble Sort em um vetor decrescente.

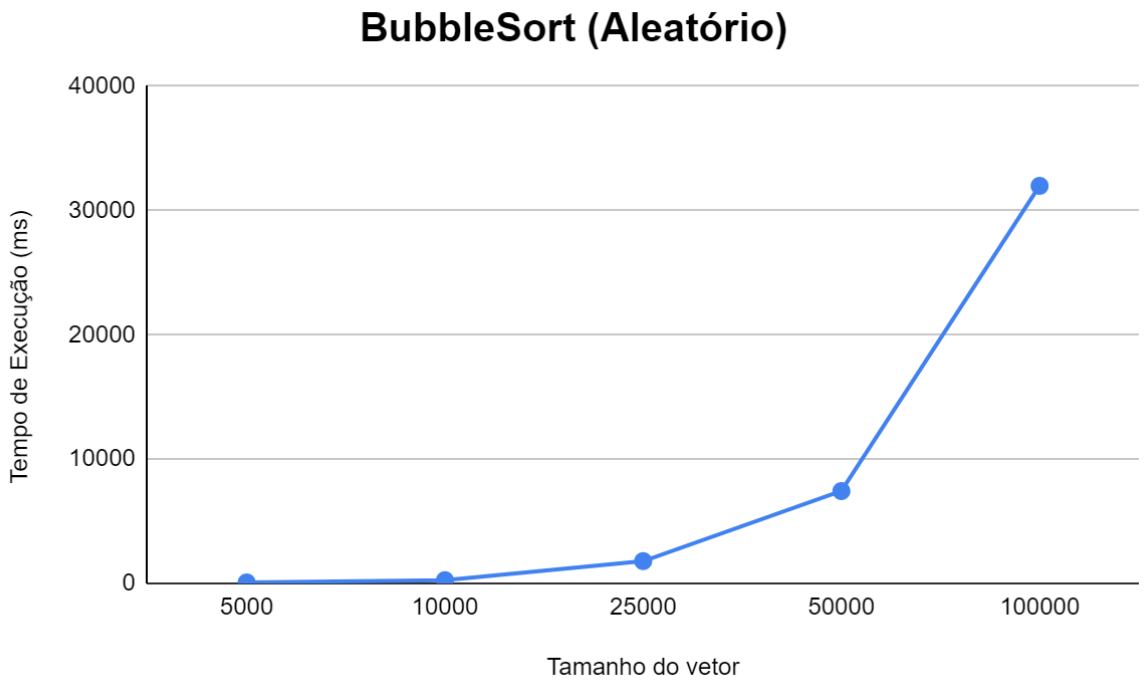

Gráfico 6: Desempenho do Bubble Sort em um vetor aleatório.

Comparando os gráficos aplicando o algoritmo Bubble Sort, é apresentado uma mesma tendência, aumentar quase que de forma exponencial e escalar o tempo de execução conforme o tipo de vetor utilizado, sendo executado de forma mais rápida no vetor crescente e de forma mais demorada no vetor aleatório, para o vetor decrescente, o tempo de execução é um intermediário entre os demais.

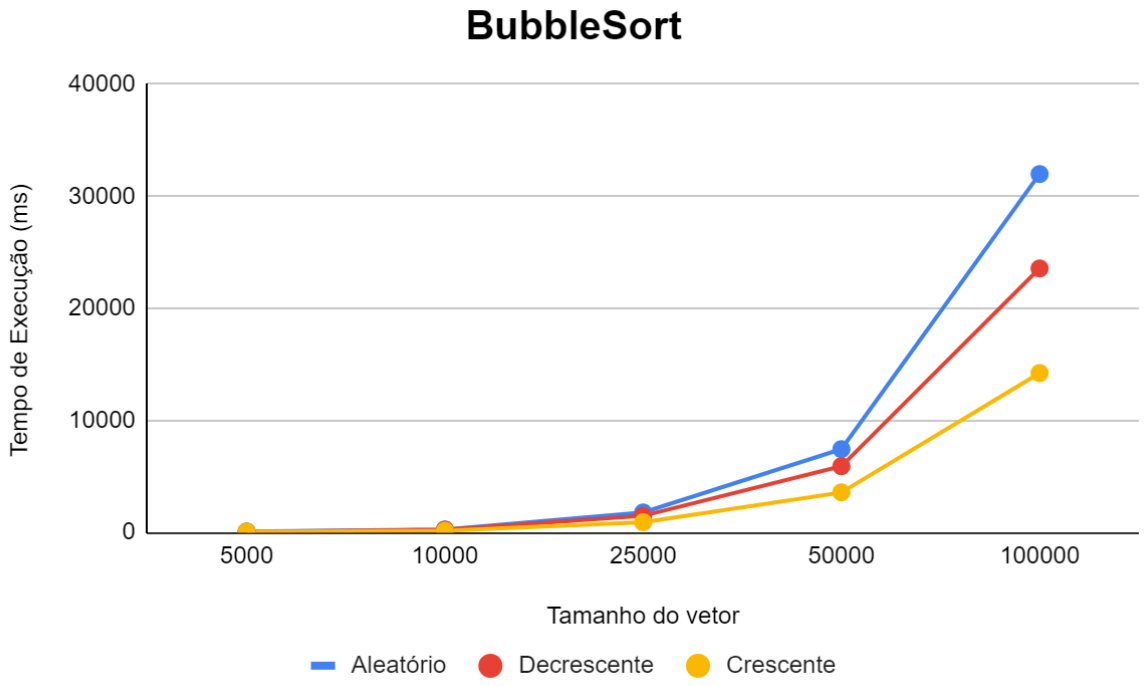

Gráfico 7: Comparativo entre os vetores utilizados com o método Bubble Sort.

Colocando todos os resultados no mesmo gráfico, fica visível que realmente o Bubble Sort tem sempre uma tendência exponencial, ou seja, se for dobrado o tamanho do vetor, o tempo de execução tende a aumentar mais que o dobro. Quanto a diferença entre o tempo de execução entre os tipos de vetores, ela escala dependendo do quanto desorganizado o vetor está.

Selection Sort:

O Selection Sort divide a lista em uma parte ordenada e uma parte não ordenada. Ele repetidamente seleciona o menor elemento da parte não ordenada e o move para o final da parte ordenada.

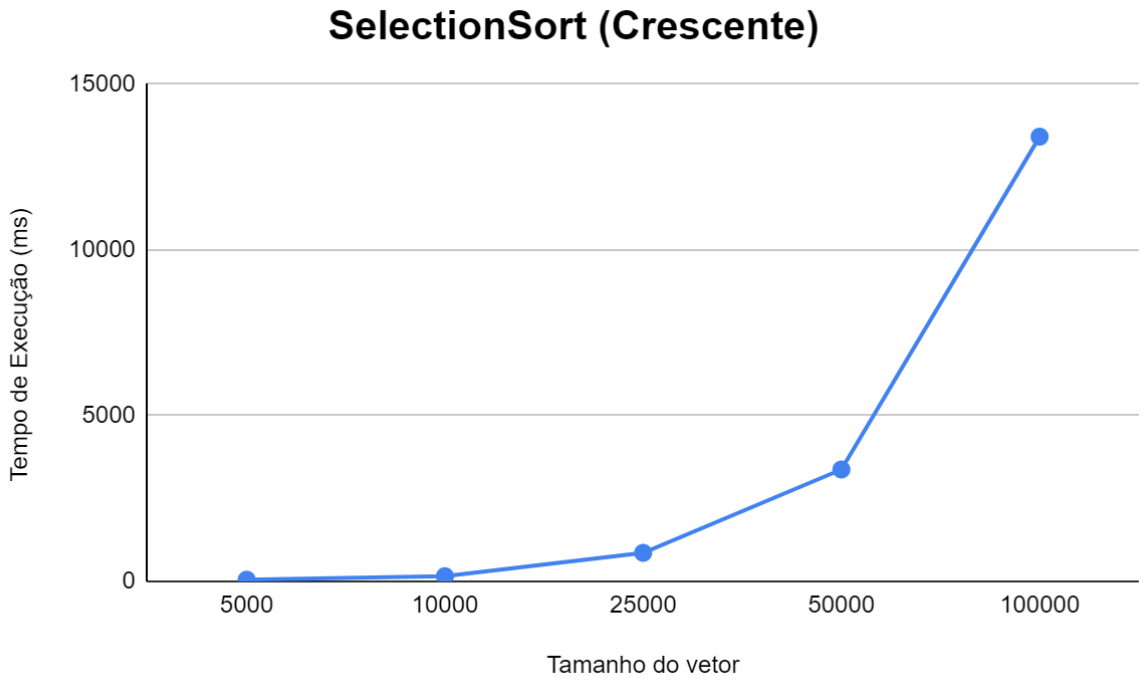

Gráfico 8: Desempenho do Selection Sort em um vetor crescente.

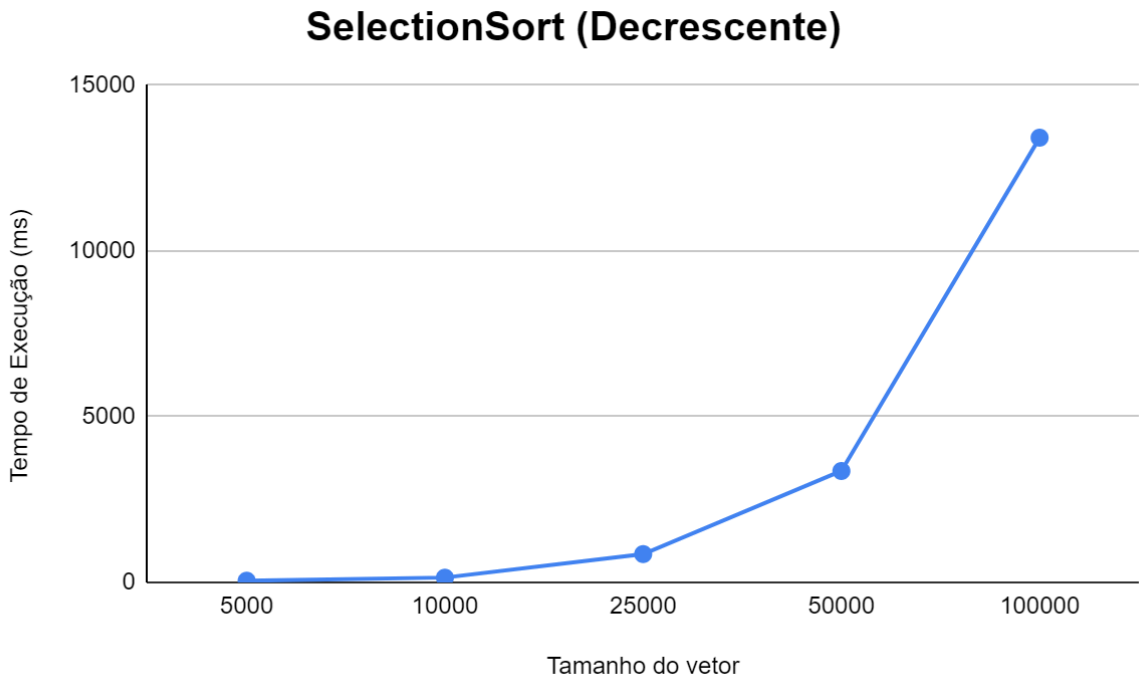

Gráfico 9: Desempenho do Selection Sort em um vetor decrescente.

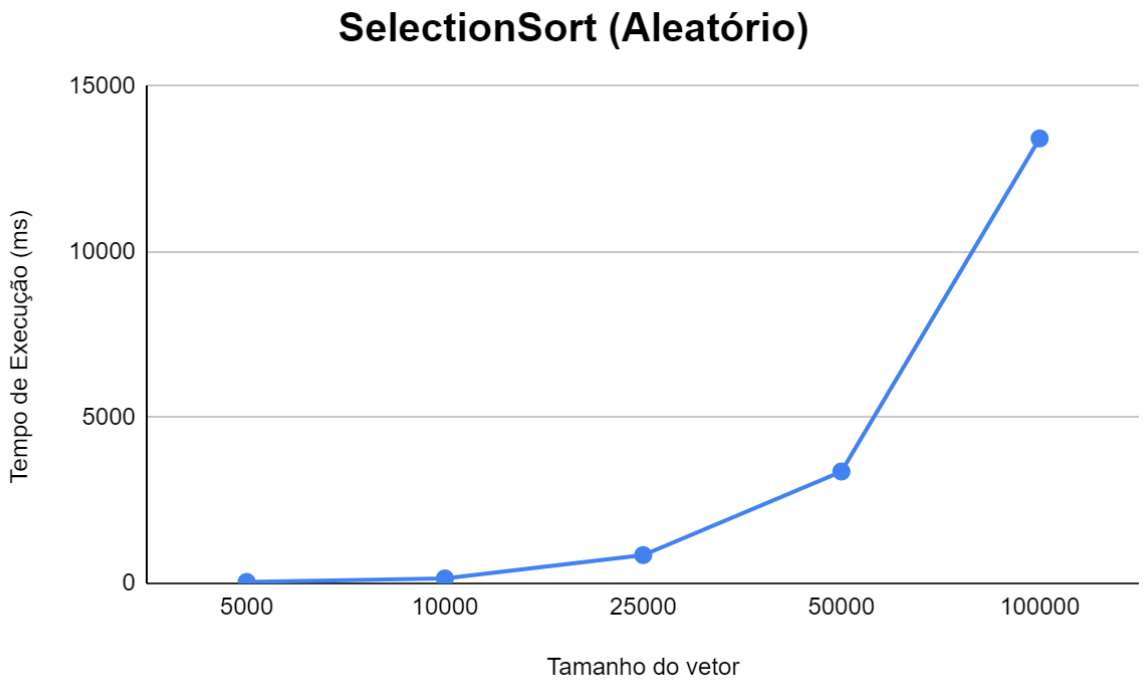

Gráfico 9: Desempenho do Selection Sort em um vetor aleatório.

O algoritmo Selection Sort tem tempo de execução quase idêntico entre todos os tipos de vetores utilizados, deixando o período de execução somente dependente da quantidade de elementos do vetor.

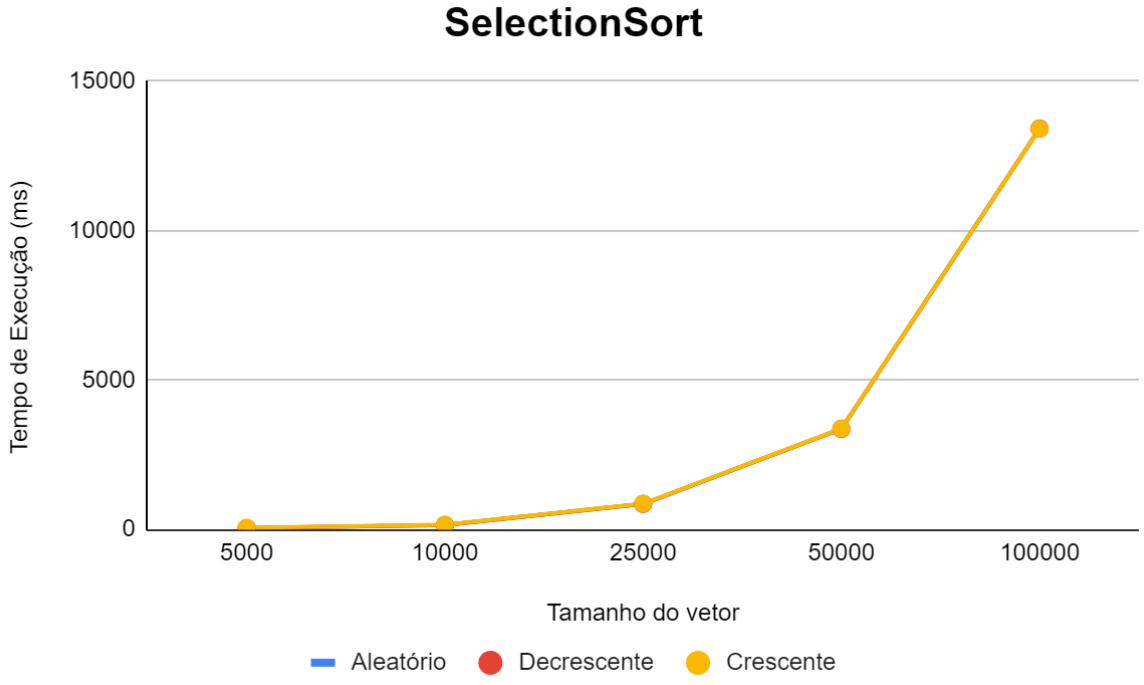

Gráfico 10: Comparativo entre os vetores utilizados com o método Selection Sort.

Ao colocar todos os dados no mesmo gráfico, não é possível ver diferença entre as linhas, pois todos estão sobrepostos perante a escala utilizada. Logo é possível concluir que o tempo de execução do Selection Sort é constante, só depende do número de elementos que serão ordenados.

Quick Sort:

O Quick Sort usa a técnica de divisão e conquista. Ele seleciona um elemento como pivô e particiona a lista ou vetor em duas sublistas: uma com elementos menores que o pivô e outra com elementos maiores. O processo é repetido recursivamente para cada sublista.

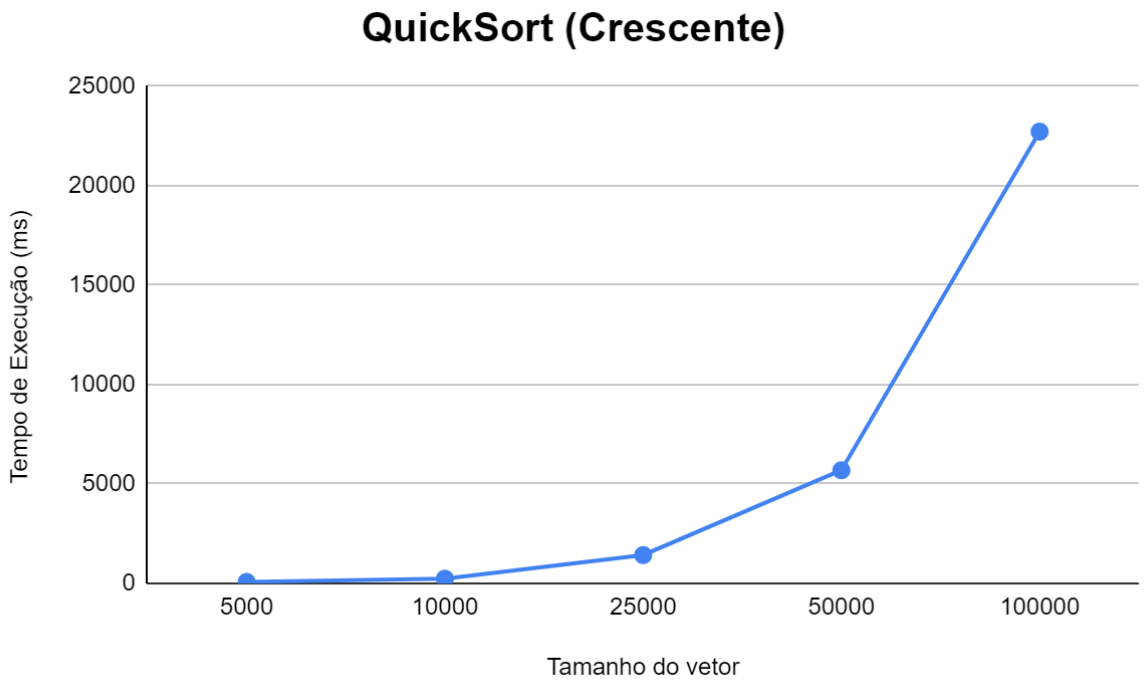

Gráfico 11: Desempenho do Quick Sort em um vetor crescente

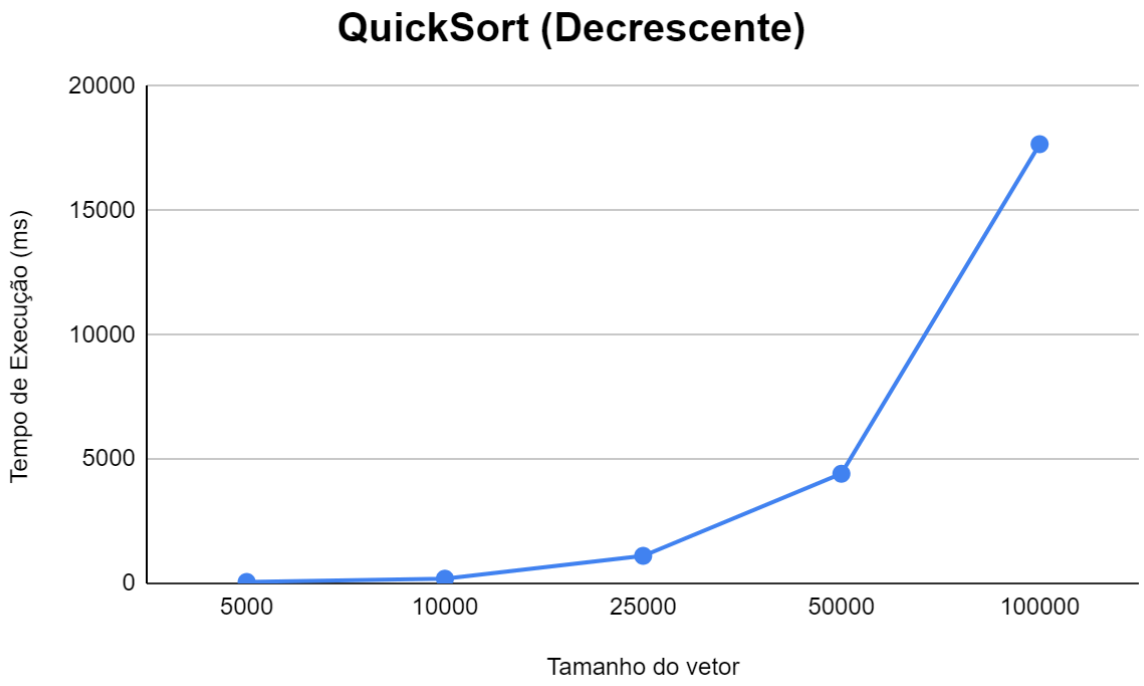

Gráfico 12: Desempenho do Quick Sort em um vetor decrescente.

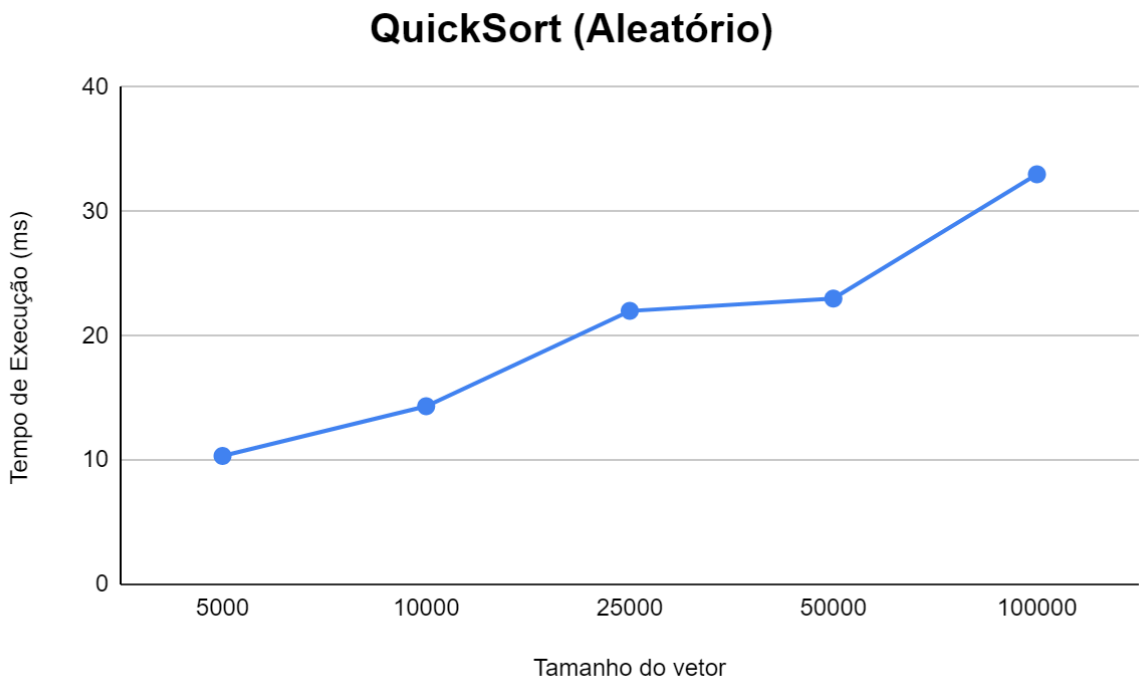

Gráfico 12: Desempenho do Quick Sort em um vetor aleatório.

O método Quick Sort apresenta tendência exponencial tanto para os vetores crescentes quanto decrescentes, com vantagem no tempo de execução menor para o vetor decrescente, já para o aleatório, apresenta tendência linear e de magnitude menor que nos demais.

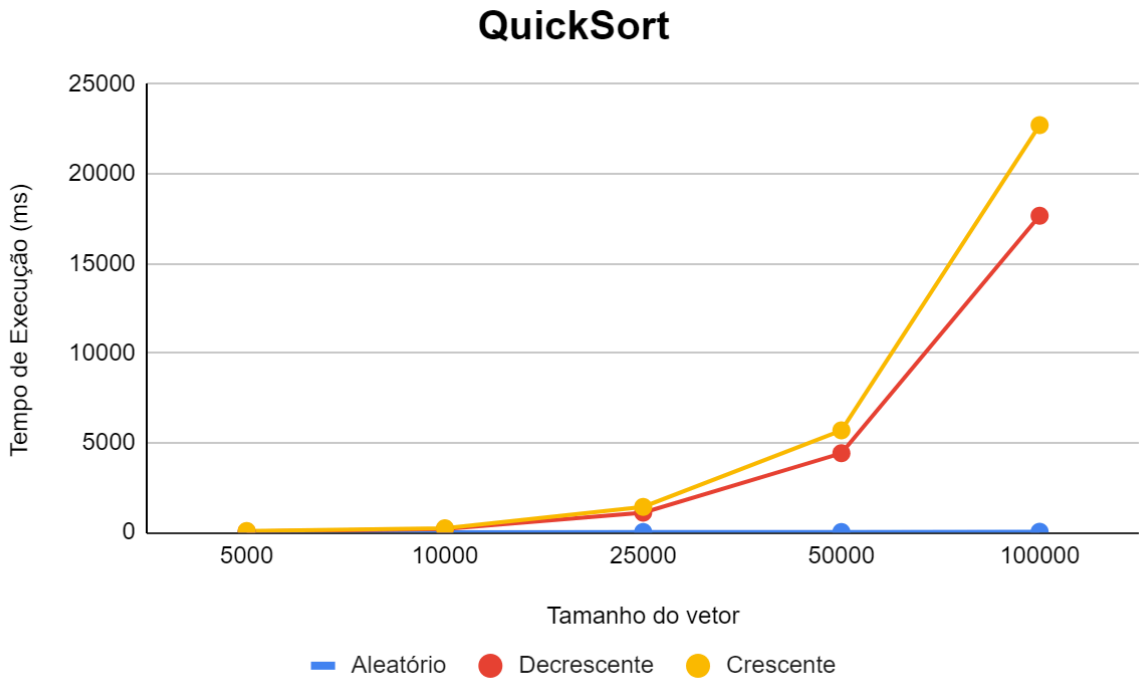

Gráfico 13: Comparativo entre os vetores utilizados com o método Quick Sort.

No gráfico comparativo, é possível observar que o Quick Sort foi mais eficiente ao lidar com vetores aleatórios, apresentando tempos de execução menores em comparação aos vetores crescentes e decrescentes. Isso confirma a natureza eficiente do Quick Sort quando os elementos do vetor estão distribuídos de forma aleatória, permitindo um balanceamento mais adequado das partições.

Conclusão:

Os resultados deste estudo demonstram diferenças significativas entre os algoritmos de busca e ordenação, especialmente em termos de eficiência e comportamento conforme o tamanho e a ordenação inicial dos vetores.

Busca:

- **Busca Sequencial:** A busca sequencial apresentou um desempenho linear, conforme esperado, mostrando-se adequada para listas pequenas ou não ordenadas, mas ineficiente para listas maiores.
- **Busca Binária:** A busca binária, com sua eficiência logarítmica, destacou-se por ser mais rápida em listas ordenadas, especialmente conforme o número de elementos aumentou, embora a amostragem utilizada limitou a visibilidade de diferenças substanciais.

Ordenação:

- **Bubble Sort:** Este algoritmo mostrou-se ineficiente para grandes conjuntos de dados devido à sua complexidade quadrática. A diferença no tempo de execução entre os vetores crescentes, decrescentes e aleatórios foi notável, com o vetor aleatório sendo o mais demorado.
- **Selection Sort:** O Selection Sort, com sua complexidade constante, demonstrou tempos de execução semelhantes para todos os tipos de vetores testados. Embora também tenha uma complexidade quadrática, mostrou-se mais previsível e consistente em comparação ao Bubble Sort.
- **Quick Sort:** O Quick Sort destacou-se por sua eficiência geral, especialmente para vetores aleatórios, onde apresentou uma tendência linear. No entanto, sua tendência exponencial para vetores crescentes e decrescentes indica que a escolha do pivô é crucial para manter sua eficiência.

Em suma, para algoritmos de busca, a escolha entre busca sequencial e binária depende do tamanho e ordenação da lista. Já para algoritmos de ordenação, o Quick Sort mostrou-se a escolha mais eficiente para a maioria dos casos, especialmente para grandes conjuntos de dados aleatórios. O Selection Sort, embora consistente, é superado em eficiência pelo Quick Sort, e o Bubble Sort é menos eficiente e mais adequado para fins didáticos. Portanto, a análise detalhada dos resultados permite concluir que o Quick Sort, apesar de sua sensibilidade ao tipo de vetor, é o mais robusto e eficiente para ordenações de grandes volumes de dados.