Serviço Público Federal
Ministério da Educação
Fundação Universidade Federal de Mato Grosso do Sul

Implementação Algorítmica

Trabalho Avaliativo 1

Sophya Martins Ribeiro
Soraya Dias Ferreira

INTRODUÇÃO

O presente relatório visa expor resultados e análises dos testes de tempo de execução realizados com os algoritmos de ordenação para a disciplina de Implementação Algorítmica. Os algoritmos usados para o estudo foram: Bubble, Insertion, Merge, Quick, Heap e Counting Sort. Todas as análises e os algoritmos foram desenvolvidos utilizando a linguagem Python com algumas bibliotecas, tais como Time — para leitura dos tempos iniciais e finais de execução —, Matplot — para gerar gráficos baseados nos dados obtidos — e numpy — para melhor visualização do gráfico.

Para realização dos testes, foram utilizados 4 tipos de vetores: vetores completamente aleatórios, vetores reversos (ordenados de maneira decrescente), vetores ordenados (de maneira crescente) e vetores "quase" ordenados (com 10% dos valores fora da ordem e escolhidos aleatoriamente). Portanto, para melhor compreensão, o relatório será dividido pelos tipos de vetores de entrada.

Ademais, em todos os testes realizados é necessário considerar quatro parâmetros principais: INC e FIM — que referem-se consecutivamente à posição inicial e final para ordenação, STP — que denota o passo, ou intervalo, em que serão registrados os tempos de execução e, por fim, RPT — que recebe o número de repetições para ordenar os vetores aleatórios em todos aqueles algoritmos de ordenação citados acima, com o intuito de calcular uma média de tempo de execução para cada um destes.

INSTRUÇÕES PARA EXECUTAR O PROGRAMA

Para executar os programas, é necessário que sua máquina tenha o Python 3 instalado e instalar a biblioteca matplotlib por meio do comando "pip install -U matplotlib", conforme documentação da ferramenta. Tendo isso, basta escolher um ambiente de desenvolvimento integrado de sua preferência e abrir a pasta do

projeto. Há três programas diferentes – (1) um para testar todos os algoritmos juntos, (2) outro para testar apenas os algoritmos de ordenação eficientes, e (3) outro para testar os algoritmos elementares.

(1) Para executar o primeiro, digite em seu terminal “python testar_todos.py” e pressione a tecla enter. O terminal deve exibir um menu, então é preciso digitar um número inteiro com a escolha de teste desejada e depois inserir os valores dos parâmetros. Todos os valores escolhidos devem ser números inteiros positivos, e o parâmetro FIM não pode ser deveras grande, como cinquenta mil, por exemplo, visto que números altos podem gerar o erro de *segmentation fault*, em razão do limite de chamadas recursivas.

(2) Para rodar o teste dos eficientes, digite em seu terminal “python testar_eficientes.py” e pressione enter. O mesmo menu anterior deve aparecer, então basta seguir as instruções, inserindo os valores desejados.

(3) Por fim, para executar o teste dos algoritmos elementares, é preciso digitar no terminal “python testar_elementares.py” e teclar enter. Ele exibirá um menu igual aos demais. Entretanto, é importante lembrar que os algoritmos elementares são lentos, sendo assim, tenha cautela na escolha dos parâmetros, pois valores muito altos levarão muito tempo para gerar um resultado.

VETORES ALEATÓRIOS (VA)

Os vetores aleatórios foram gerados por meio da função "gerar_numeros_aleatorios", que cria uma lista de tamanho n, na qual cada elemento é um número inteiro positivo aleatório, gerado pela função random.randint(1, *n*). Essa abordagem permite gerar valores dentro do intervalo [1, *n*].

O valor superior foi mantido em n, ao invés de n quadrático, porque isso afetaria diretamente o desempenho de algoritmos como o Counting Sort. Esse algoritmo, apesar de ter complexidade O(*n* + *k*) —, em que n é o número de elementos e k é o valor máximo do elemento no vetor —, pode sofrer impacto significativo quando *n* assume valores quadráticos. Isso ocorre porque o Counting Sort precisa de um array auxiliar de tamanho *k*, e se *k* for proporcional a n^2, a memória necessária e o tempo para inicializar esse array crescem rapidamente, tornando o algoritmo menos eficiente.

RESULTADOS OBTIDOS

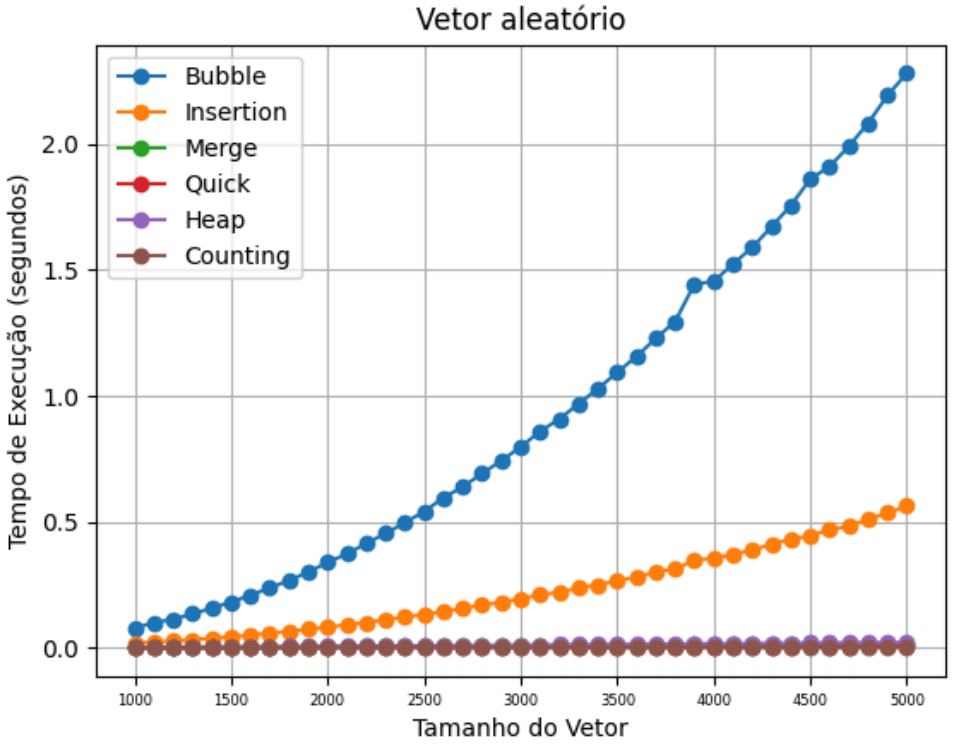
Nesta seção serão apresentados os resultados obtidos para cada parâmetro escolhido, considerando o gráfico de tempo de execução por tamanho do vetor. Os testes de vetor aleatório são divididos em A, B e C, fazendo referência a todos os algoritmos, algoritmos eficientes e algoritmos elementares respectivamente.

Tabela 1 - Parâmetros para realização do teste VA-1 (A e C)

Vetor aleatório - Teste 1 (A e C)				
Parâmetros	INC - Início	FIM - fim	STP - passo	RPT - repetições
Valores	1000	5000	100	8

Fonte: autoras, 2024.

Gráfico 1 - Tempos obtidos no teste VA-1A



Fonte: autoras, 2024.

Para melhor visualização dos tempos de execução de ordenação em algoritmos eficientes, os parâmetros foram ampliados, conforme a tabela 2. Embora

o gráfico de alguns algoritmos tenham se sobreposto, é possível observar que o Bubble Sort e o Insertion Sort são exponenciais, tendo uma diferença de eficiência proeminente com relação aos demais.

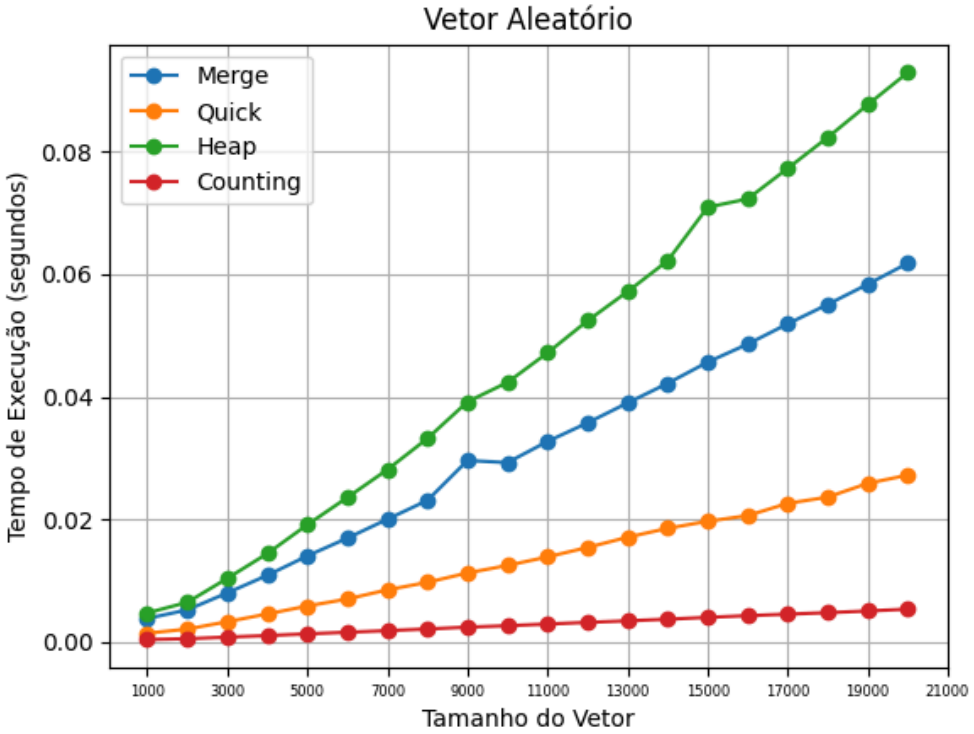
É importante mencionar que, para que o Counting Sort apresentasse tempo de execução linear, foi necessário limitar o valor máximo dentro da lista de números aleatórios. O máximo previsto na descrição do trabalho prático era o quadrado do tamanho do vetor, porém decidimos reduzir esse valor para que pudesse ser igual ao tamanho exatamente. A explicação para tal decisão é que o procedimento depende diretamente do valor máximo, uma vez que ele cria uma lista auxiliar logo no início com tamanho igual a essa quantidade. Se o valor máximo for muito maior que o tamanho do vetor de entrada, o tempo de computação tem chance de ser até exponencial, comportamento verificado ao utilizar o quadrado do comprimento nos testes.

Tabela 2 - Parâmetros para realização do teste VA-1B

Vetor aleatório - Teste 1B				
Parâmetros	INC - Início	FIM - fim	STP - passo	RPT - repetições
Valores	1000	20000	1000	10

Fonte: autoras, 2024.

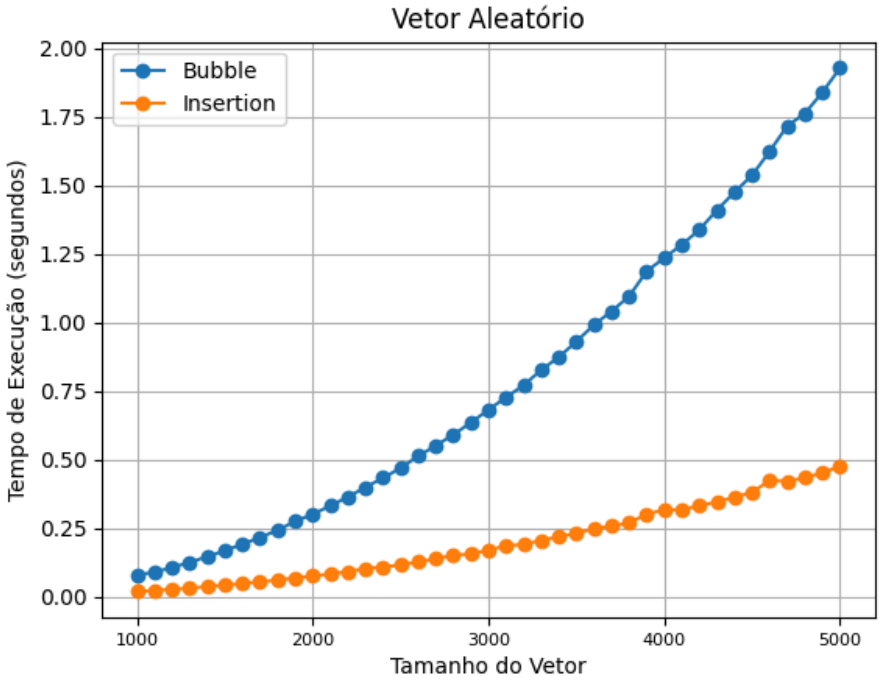
Gráfico 2 - Tempos obtidos no teste VA-1B



Fonte: autoras, 2024.

Nesse caso de teste, foram utilizados apenas vetores com tamanhos grandes. Diante disso, pode-se verificar que o algoritmo Merge Sort apresentou eficiência maior do que o Heap Sort para tais valores, o que ocorreu em razão de o Heap Sort realizar movimentações de dados, demandando várias comparações e, consequentemente, tempo de execução.

Gráfico 3 - Tempos obtidos no teste VA-1C



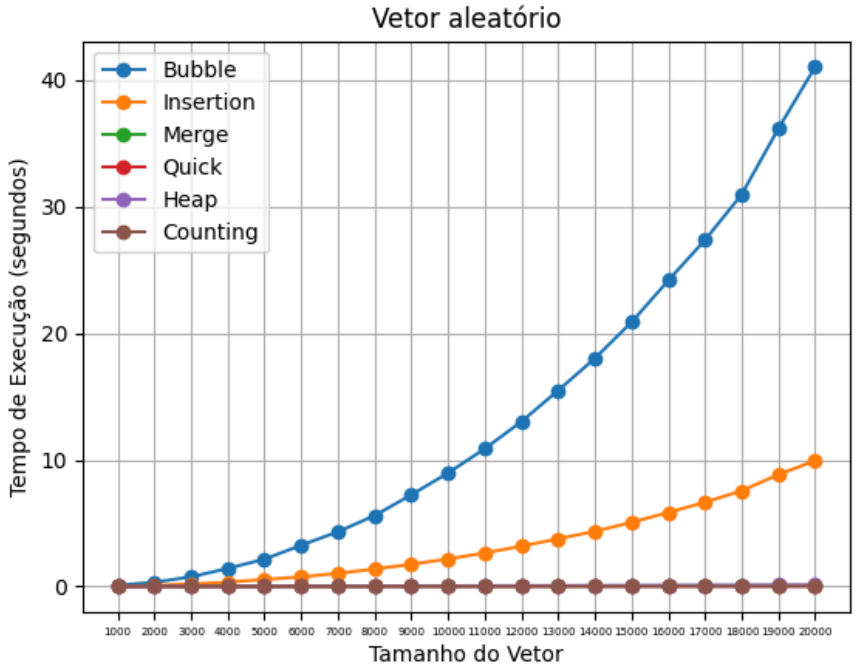
Fonte: autoras, 2024.

Tabela 2 - Parâmetros para realização do teste VA-2

Vetor aleatório - Teste 2				
Parâmetros	INC - Início	FIM - fim	STP - passo	RPT - repetições
Valores	1000	20.000	1000	4

Fonte: autoras, 2024.

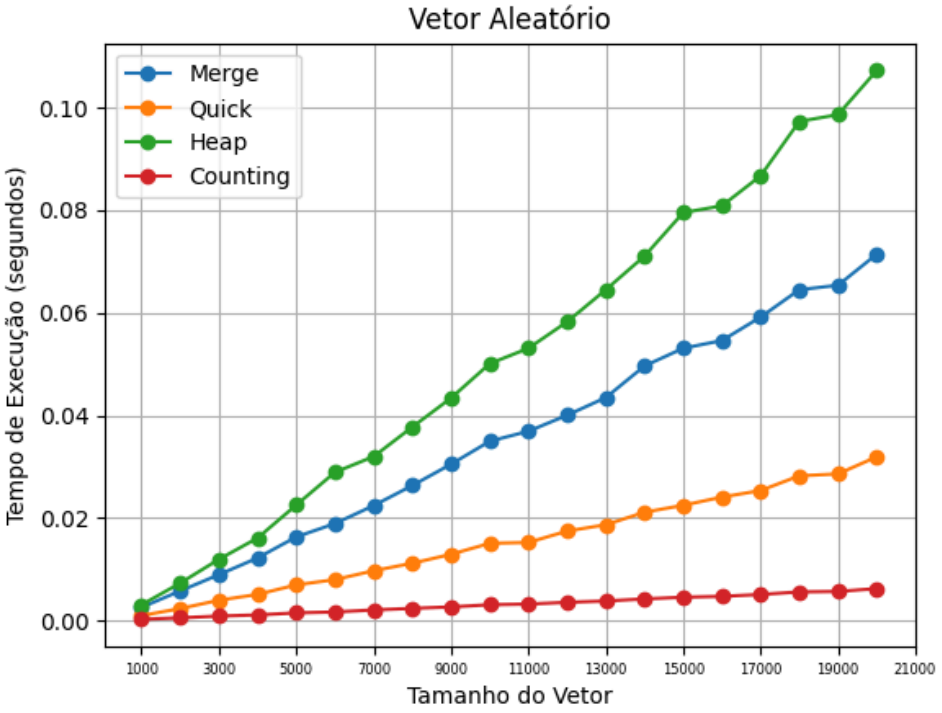
Gráfico 4 - Tempos obtidos no teste VA-2A



Fonte: autoras, 2024.

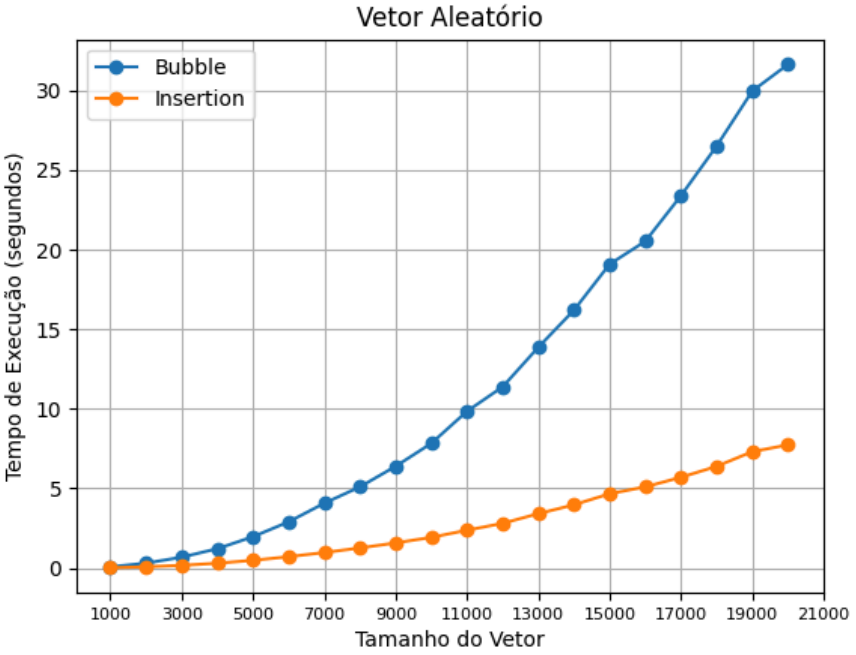
Mesmo com valores maiores para os comprimentos dos vetores, o comportamento dos algoritmos demonstrou estabilidade, seguindo a tendência apresentada no teste anterior, com tamanhos de no máximo cinco mil.

Gráfico 5 - Tempos obtidos no teste VA-2B



Fonte: autoras, 2024.

Gráfico 6 - Tempos obtidos no teste VA-2C



Fonte: autoras, 2024.

VETORES REVERSOS (VR)

Os vetores reversos são aqueles que os elementos estão ordenados de maneira decrescente, o que, na teoria, representa o pior caso para a maioria dos algoritmos de ordenação. Isso ocorre porque os algoritmos precisam realizar o maior número possível de operações para reorganizar todos os elementos na ordem crescente. Cada elemento deve ser movido para a posição oposta no vetor, resultando em um maior número de comparações e trocas, especialmente nos algoritmos elementares, como o Bubble Sort e o Insertion Sort, que têm desempenho significativamente pior em casos como esse.

RESULTADOS OBTIDOS

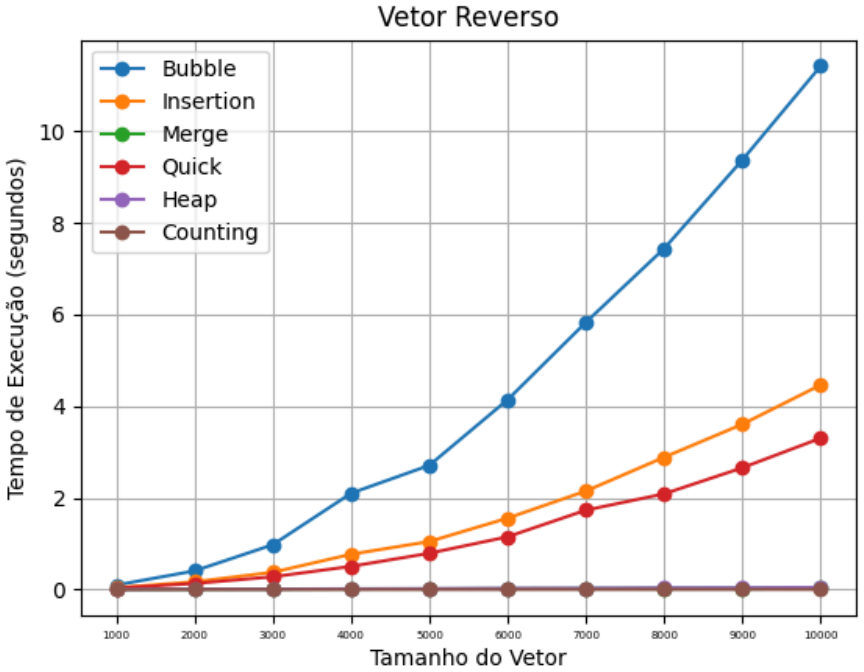
Nesta seção serão apresentados os resultados obtidos para cada parâmetro escolhido, considerando o gráfico de tempo de execução por tamanho do vetor. Os testes de vetor quase reverso são igualmente divididos em A, B e C, fazendo referência aos testes com todos os algoritmos, com algoritmos eficientes e com algoritmos elementares, respectivamente.

Tabela 3 - Parâmetros para realização do teste VR1

Vetor reverso - Teste 1			
Parâmetros	INC	FIM	STP
Valores	1000	10.000	1000

Fonte: autoras, 2024.

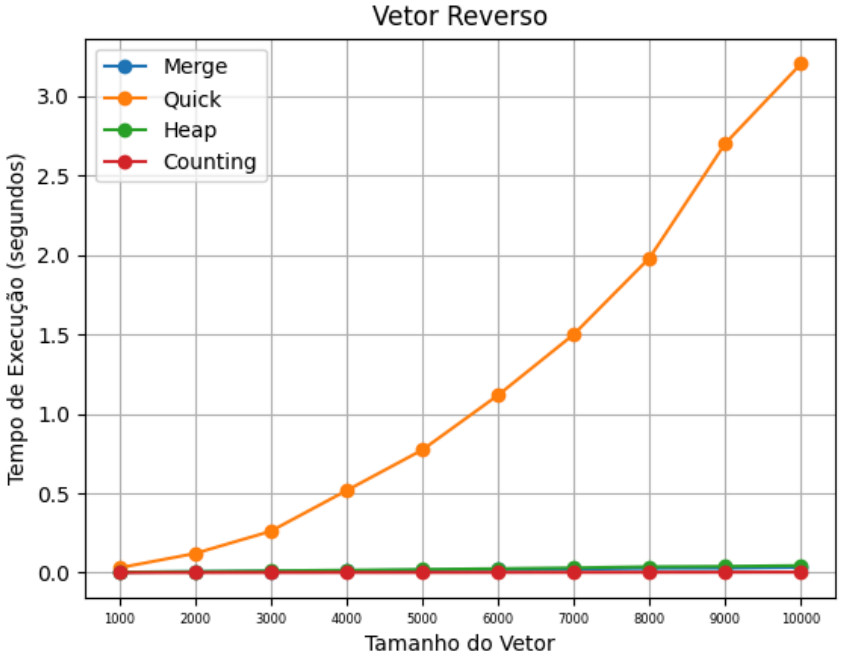
Gráfico 7 - Tempos obtidos no teste VR-1A



Fonte: autoras, 2024.

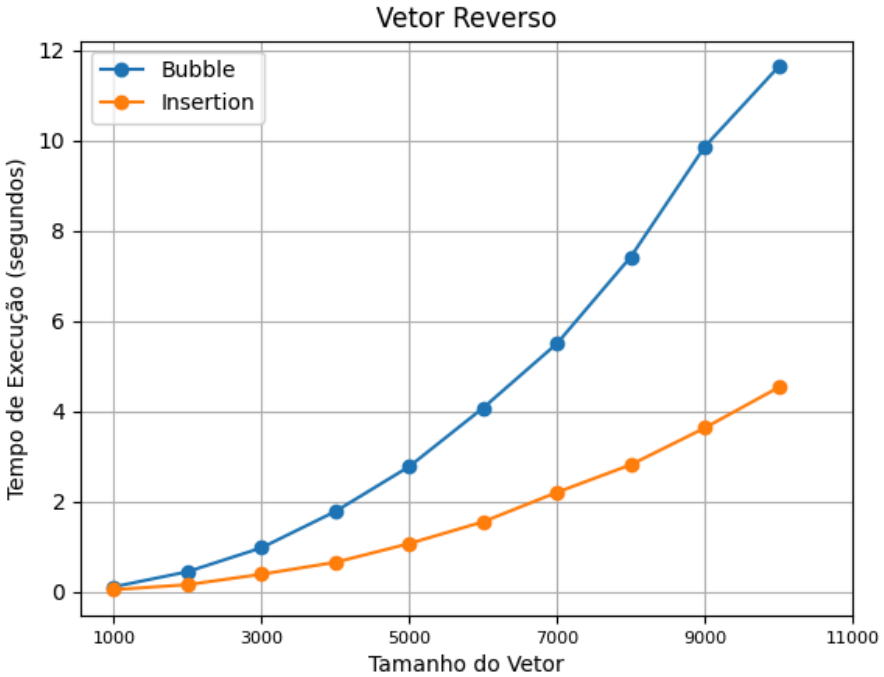
Gráfico 8 - Tempos obtidos no teste VR-1B

O algoritmo Quick Sort exibiu crescimento exponencial em seu tempo de processamento, o que se deve à maneira de escolha do pivô aplicada, dado que ela implica em uma separação não balanceada. Entretanto, esse resultado estava previsto, pois, conforme demonstrações matemáticas, no pior caso, o Quick Sort não é ágil.



Fonte: autoras, 2024.

Gráfico 9 - Tempos obtidos no teste VR-1C



Fonte: autoras, 2024.

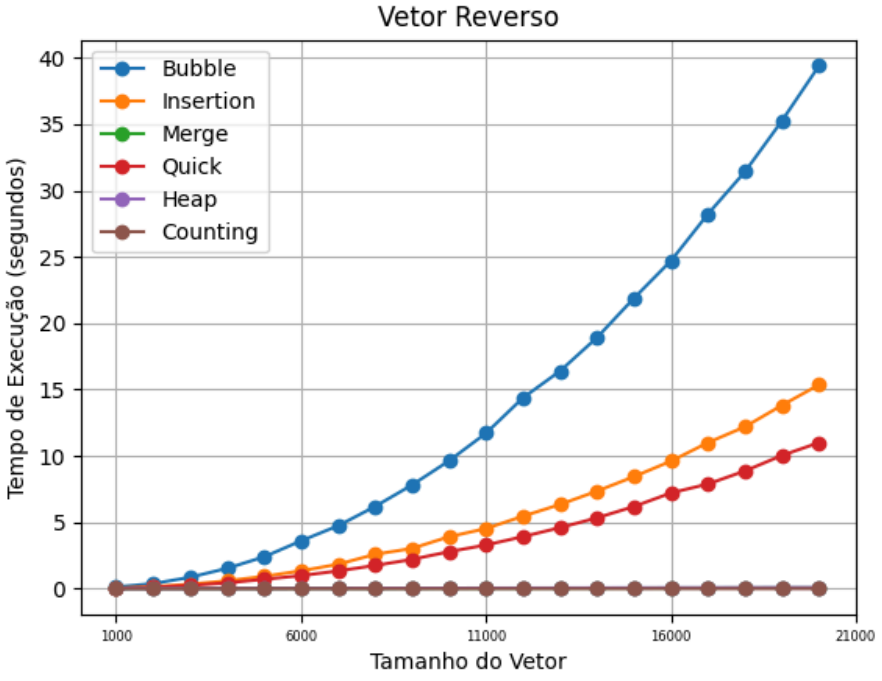
Tabela 4 - Parâmetros para realização do teste VR2

Vetor reverso - Teste 2			
Parâmetros	INC	FIM	STP
Valores	1000	20.000	1000

Fonte: autoras, 2024

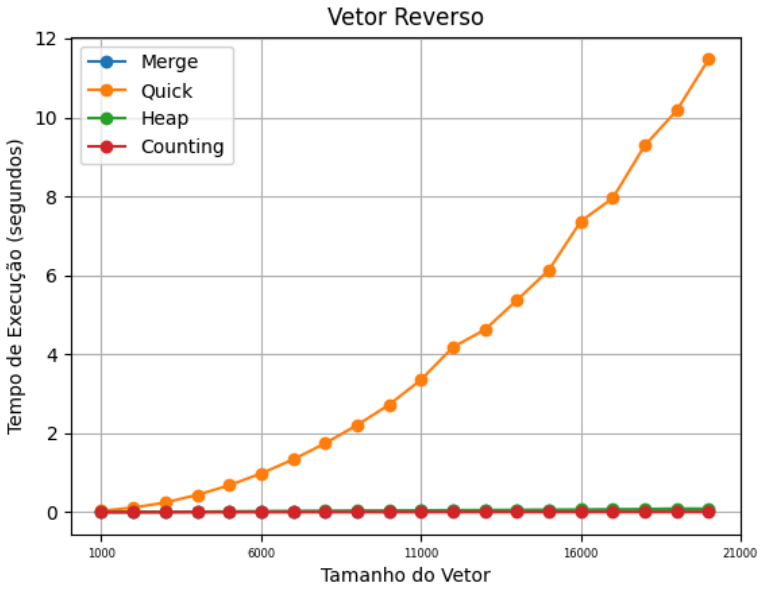
Para testar os todos os algoritmos com vetores ordenados de forma decrescente, foi necessário reduzir o tamanho máximo – parâmetro FIM – para vinte mil, isso porque, ao testar com parâmetro FIM igual a cinquenta mil, houve um erro de segmentação, o que é um evento raro na linguagem Python. Após pesquisa, foi descoberta uma provável causa para tal erro, envolvendo o limite máximo de chamadas recursivas. Nesse sentido, conforme verificado no Python Bug Tracker – uma ferramenta da comunidade Python – ao se manipular o limite de recursão com valores muito maiores que o padrão mil, tal erro pode ser obtido.

Gráfico 10 - Tempos obtidos no teste VR-2A



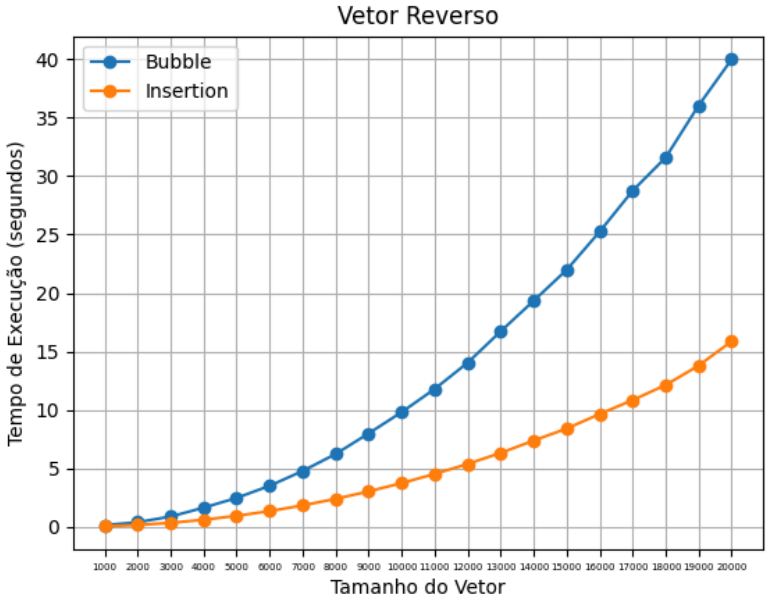
Fonte: autoras, 2024.

Gráfico 11 - Tempos obtidos no teste VR-2B



Fonte: autoras, 2024.

Gráfico 12 - Tempos obtidos no teste VR-2C



Fonte: autoras, 2024.

VETORES ORDENADOS (VO)

Os vetores ordenados são aqueles cujos elementos já estão dispostos em ordem crescente, o que, em teoria, poderia facilitar o processo de ordenação. No entanto, os algoritmos de ordenação não possuem conhecimento de que o vetor já está ordenado. Então, eles seguem todo o procedimento normal de comparação e troca de elementos, como se o vetor estivesse desordenado. Isso faz com que o tempo de execução não seja imediatamente otimizado para todos os algoritmos, principalmente os menos eficientes, como o Bubble Sort, que mesmo com a entrada já ordenada, percorre o vetor repetidamente.

Os vetores foram gerados valores no intervalo de [0, *n*], somando uma unidade a cada posição.

RESULTADOS OBTIDOS

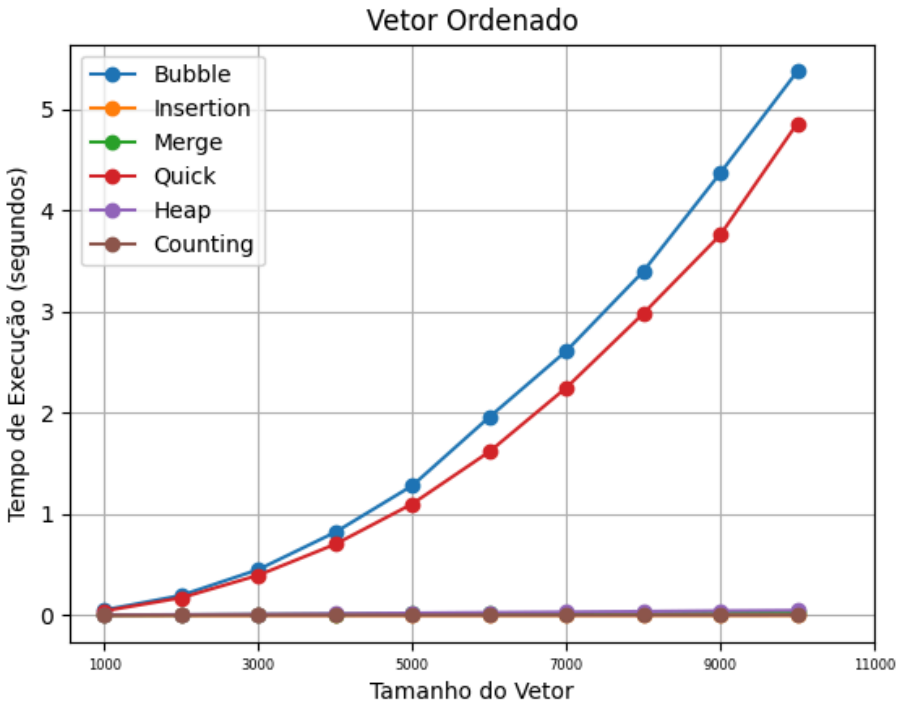
Nesta seção serão apresentados os resultados obtidos para cada parâmetro escolhido, considerando o gráfico de tempo de execução por tamanho do vetor. Os testes de vetor ordenado também são divididos em A, B e C, fazendo referência a todos os algoritmos, algoritmos eficientes e algoritmos elementares, consecutivamente.

Tabela 5 - Parâmetros para realização do teste VO1

Vetor Ordenado- Teste 1			
Parâmetros	INC	FIM	STP
Valores	1000	10.000	1000

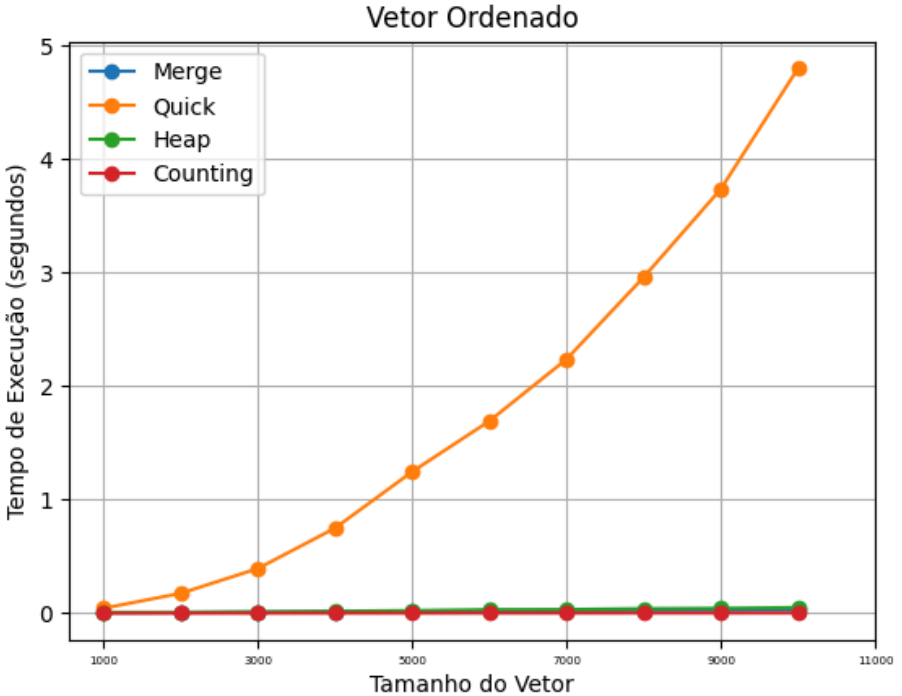
Fonte: autoras, 2024.

Gráfico 13 - Tempos obtidos no teste VO-1A



Fonte: autoras, 2024.

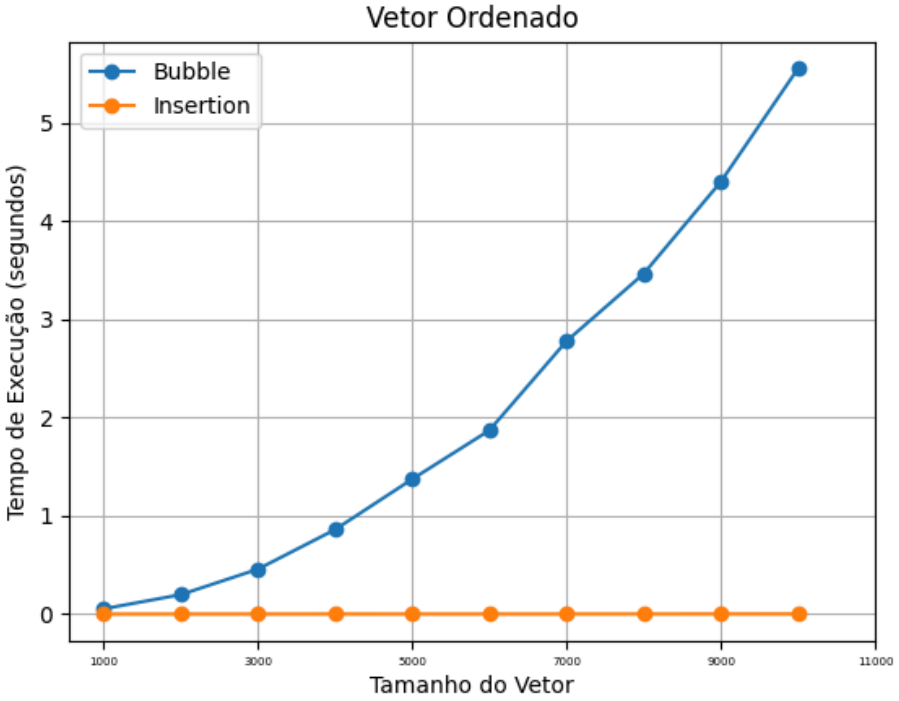
Gráfico 14 - Tempos obtidos no teste VO-1B



Fonte: autoras, 2024.

Nesse resultado, algo que chama atenção é, posto que a lista de inteiros já estivesse ordenada, o algoritmo Quick Sort também demonstrou crescimento exponencial. Mais uma vez, o motivo é o próprio funcionamento do processo e a escolha do pivô.

Gráfico 15 - Tempos obtidos no teste VO-1C



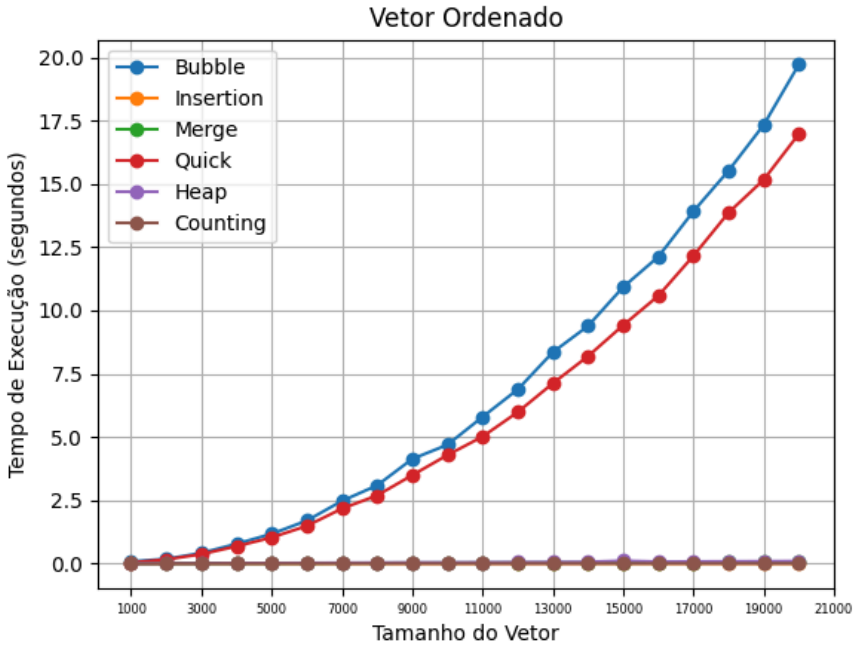
Fonte: autoras, 2024.

Tabela 6 - Parâmetros para realização do teste VO1

Vetor Ordenado - Teste 2			
Parâmetros	INC	FIM	STP
Valores	1000	20.000	1000

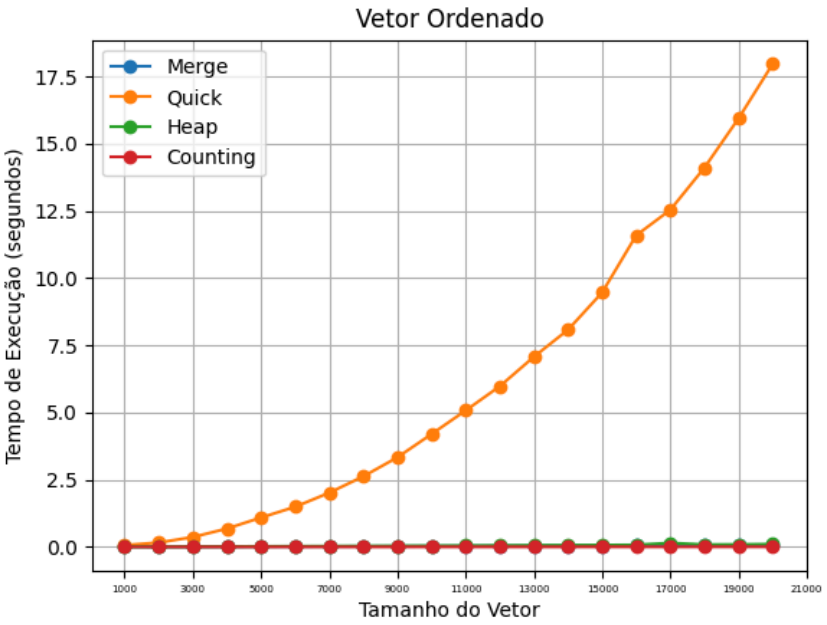
Fonte: autoras, 2024.

Gráfico 16 - Tempos obtidos no teste VO-2A



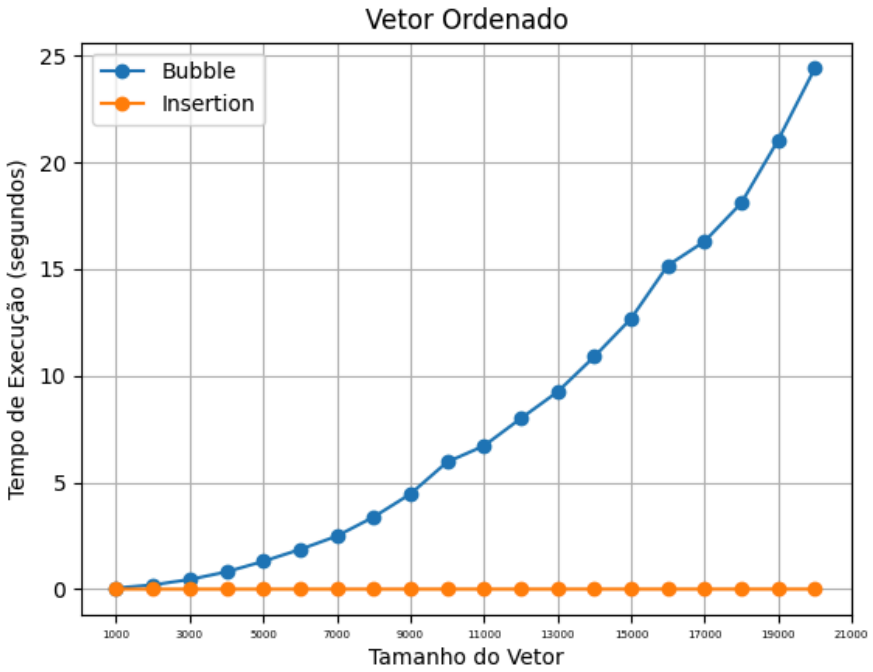
Fonte: autoras, 2024.

Gráfico 17 - Tempos obtidos no teste VO-2B



Fonte: autoras, 2024.

Gráfico 18 - Tempos obtidos no teste VO-2C



Fonte: autoras, 2024.

VETORES QUASE ORDENADOS (VQ)

No caso dos vetores quase ordenados, a ideia principal é gerar uma sequência que esteja majoritariamente em ordem crescente, mas com uma pequena parcela de elementos desordenados. Inicialmente, um vetor de tamanho n é gerado com valores aleatórios, com um princípio similar a função 'gerar_vetor_aleatório, e em seguida, ele é completamente ordenado, utilizando o método *sort()* do python. Após essa ordenação, são escolhidos aleatoriamente 10% dos índices do vetor, e os elementos correspondentes a esses índices são substituídos por novos valores aleatórios.

Essa execução resulta em um vetor no qual 90% dos elementos permanecem ordenados, enquanto os 10% restantes estão fora de ordem, criando assim uma lista quase ordenada que simula uma situação intermediária entre um vetor totalmente desordenado e um vetor completamente ordenado.

RESULTADOS OBTIDOS

Nesta seção serão apresentados os resultados obtidos para cada parâmetro escolhido, considerando o gráfico de tempo de execução por tamanho do vetor. Os

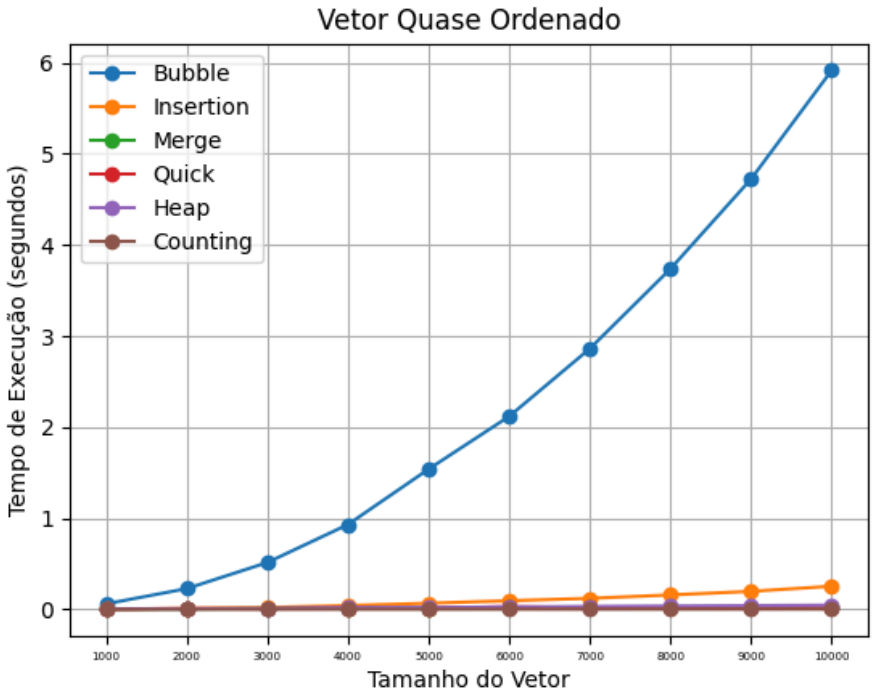
testes de vetor quase ordenado também são divididos em A, B e C, fazendo referência a todos os algoritmos, somente os algoritmos eficientes e somente os algoritmos elementares, consecutivamente.

Tabela 7 - Parâmetros para realização do teste VQ1

Vetor reverso - Teste 1			
Parâmetros	INC	FIM	STP
Valores	1000	10.000	1000

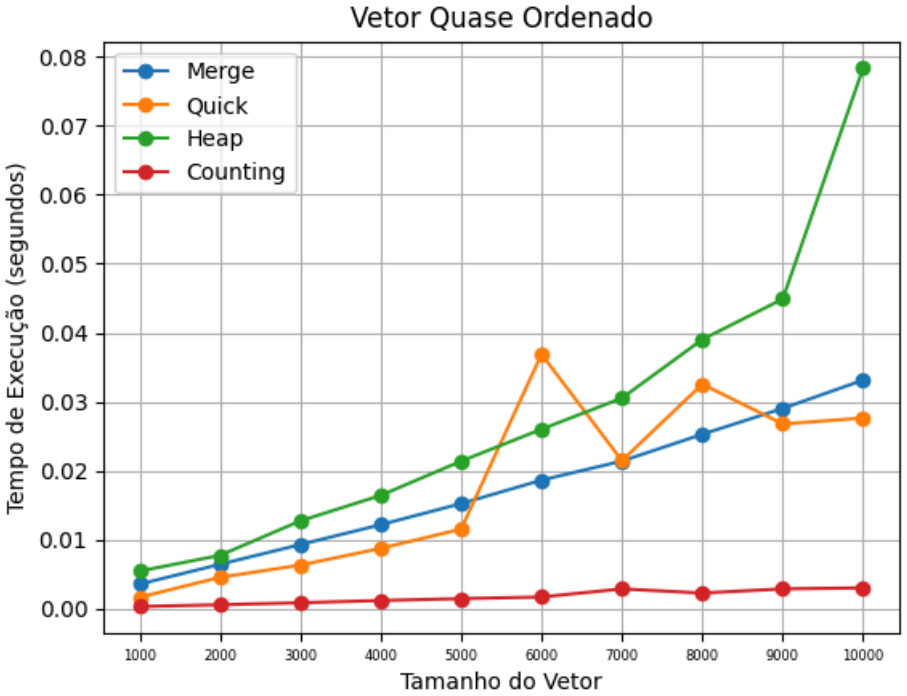
Fonte: autoras, 2024.

Gráfico 19 - Tempos obtidos no teste VQ-1A



Fonte: autoras, 2024.

Gráfico 20 - Tempos obtidos no teste VQ-1B

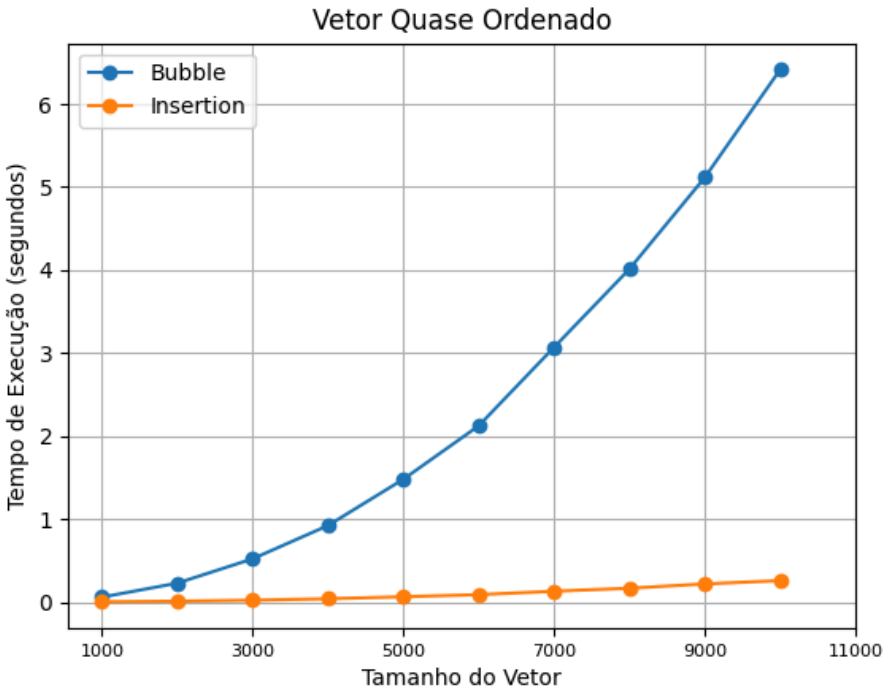


Fonte: autoras, 2024.

No gráfico 20, o algoritmo Quick Sort demonstra um comportamento esdrúxulo. Isso pode ter sucedido em decorrência da forma como o procedimento em si funciona, pois a eficiência dele depende da boa escolha de um pivô, a partir do qual serão feitas as partições e comparações. Em nossa implementação, o pivô é sempre a última posição da lista estática. Sendo assim, se um dos valores embaralhados estiver nas extremidades, e ser um dos maiores ou menores valores, é feita uma separação desbalanceada, e o número de operações necessárias para ordenação pode ser muito maior que na maioria dos casos.

Fora isso, todos os demais algoritmos exibiram o comportamento esperado.

Gráfico 21 - Tempos obtidos no teste VQ-1C



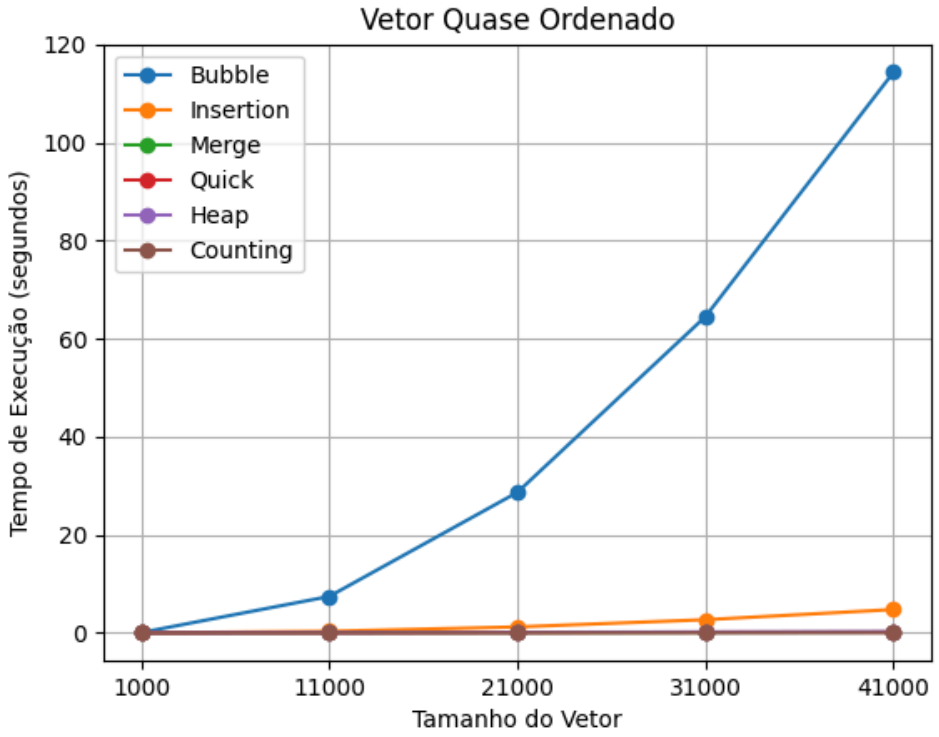
Fonte: autoras, 2024.

Tabela 8 - Parâmetros para realização do teste VQ2

Vetor reverso - Teste 2			
Parâmetros	INC	FIM	STP
Valores	1000	50.000	10000

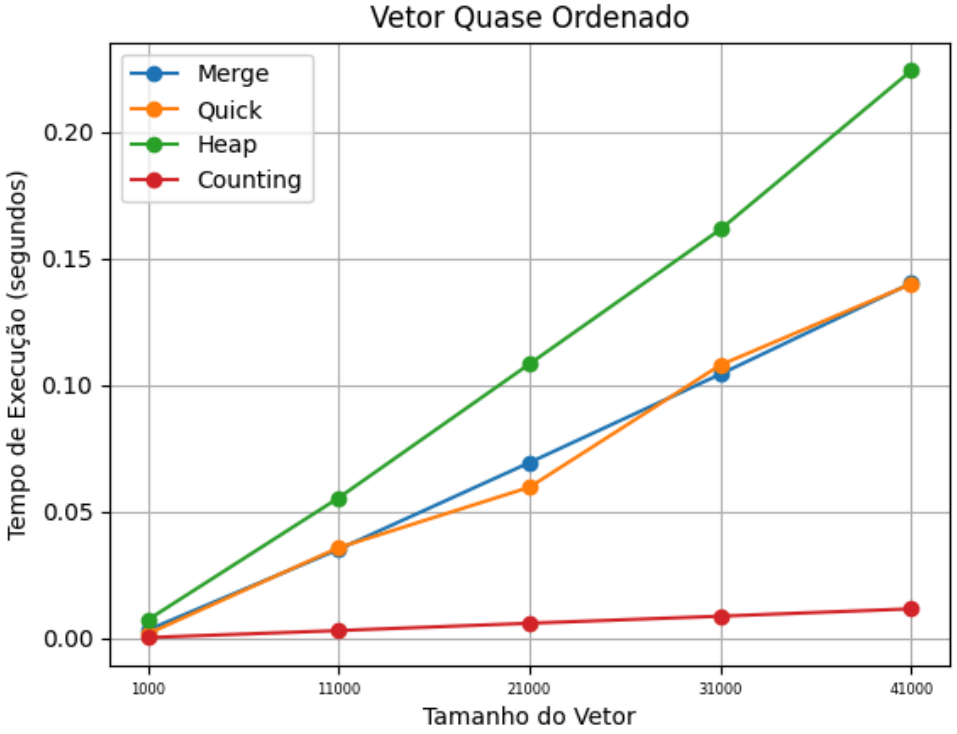
Fonte: autoras, 2024.

Gráfico 22 - Tempos obtidos no teste VQ-2A



Fonte: autoras, 2024.

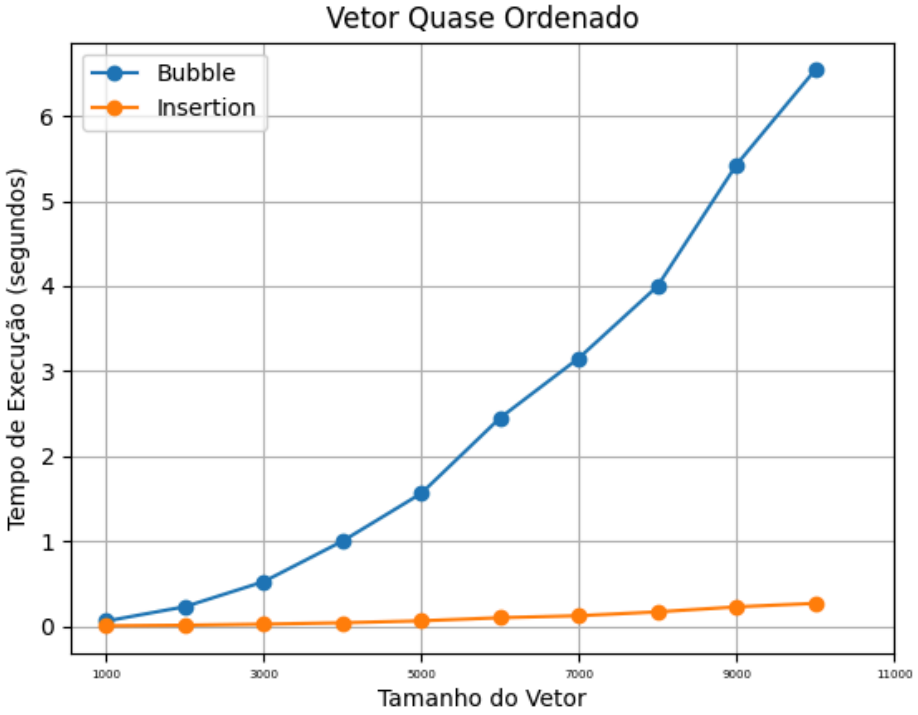
Gráfico 23 - Tempos obtidos no teste VQ-2B



Fonte: autoras, 2024.

No gráfico 23, o crescimento do Quick Sort oscilou menos, porém isso é devido ao menor número de teste realizados e que, por sorte, tiveram uma distribuição dos valores embaralhados muito parecida. Esse resultado não é igual sempre, ou seja, se o programa for testado várias vezes, o formato do gráfico muda de uma para outra, no Quick Sort.

Gráfico 24 - Tempos obtidos no teste VQ-2C



Fonte: autoras, 2024.

CONSIDERAÇÕES FINAIS

Neste relatório, foram analisados os tempos de execução de diversos algoritmos de ordenação aplicados a diferentes tipos de vetores. Os resultados obtidos permitiram uma visão prática de como o tipo de vetor e as características específicas de cada algoritmo influenciam diretamente em seu desempenho. Grande parte dos testes saíram conforme o esperado e o discutido em sala de aula, algumas ressalvas foram descritas na seção dos vetores.

Algoritmos como Bubble Sort e Insertion Sort apresentaram desempenhos piores quando os vetores estavam desordenados ou em ordem reversa, confirmando suas limitações em termos de eficiência. Esses algoritmos não apenas tiveram tempos de execução muito altos, mas também tornaram os testes extremamente lentos. Em um dos cenários testados com vetores aleatórios, utilizando os parâmetros de inc=1000, fim=20000, stp=1000 e rpt=4, o tempo total de execução foi de 52 minutos, mesmo em um processador de alto desempenho. Isso evidencia as deficiências desses algoritmos para grandes volumes de dados, especialmente em comparação com algoritmos mais eficientes, como o counting sort que demonstrou alta eficiência em todos os testes, sem exceção.

REFERÊNCIAS

Analysis of Algorithms. [s.l: s.n.]. Disponível em: http://xenon.stanford.edu/~rashmi/projects/Sorting.pdf Acesso em: 02 set. 2024.

MATPLOTLIB. *User guide*. Disponível em: https://matplotlib.org/stable/users/index. Acesso em: 05 set. 2024.

T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein, Introduction to Algorithms, 3a. ed., MIT Press, 2009

HIGA, Carlos. Slides da Disciplina. Apresentações. Disponível em: https://ava.ufms.br/course/view.php?id=61684. Acesso em 08 set. 2024.