Curso: Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Estrutura de Dados II
Docente: Antonio Angelo de Souza Tartaglia
Discente: Felipe Fernandes Pereira - GU3026965

Projeto Avaliativo I

Comparação do Desempenho de Algoritmos de Ordenação

Guarulhos - SP

2023

1 INTRODUÇÃO

Este trabalho visa o desenvolvimento de uma aplicação, em linguagem de programação C, para o estudo e a comparação do desempenho entre algoritmos de ordenação em diferentes situações.
Este documento apresenta as decisões e os métodos utilizados para a construção do projeto, as dificuldades de implementação e os resultados finais, com gráficos comparativos para cada teste realizado.

2 DESENVOLVIMENTO

2.1 Configurações utilizadas nos testes

Durante o desenvolvimento deste projeto, foram utilizados mais de um computador em diferentes configurações e sistemas operacionais. Para fins de testes finais e obtenção dos dados para as comparações, foram utilizadas as seguintes configurações:

Processador: 11th Gen Intel Core i5 @ 2.70GHz;
Memória: 8,00 GB;
Sistema Operacional: Windows 11 Home.

2.2 Implementação

A construção da aplicação se iniciou com a criação das interfaces e da navegação. Além dos menus para a seleção do algoritmo e para a seleção da quantidade de dados, a aplicação contém uma tela de processamento, que fica disponível enquanto um teste está sendo executado, mostrando assim a medição em tempo real para cada ciclo do teste, e uma tela de resultados, onde todas as medições individuais e a média dos dez ciclos são apresentadas. Ainda na tela de resultados, há a opção de visualizar uma seção do vetor antes e depois de ordenado. Nesta opção, os cinquenta primeiros itens de cada vetor são mostrados, quantidade julgada suficiente para verificar o sucesso da ordenação. Esta opção foi implementada principalmente para fins de conferência durante os testes.

A navegação foi construída de forma que o usuário consiga passar por todos os algoritmos de ordenação, em todas as quantidades sem sair do programa. Em todos os menus, informar a opção 0 resultará no retorno à tela inicial, ou no encerramento da aplicação, caso o usuário já esteja no menu principal. As outras opções são identificadas por números. Ao informar um número inválido em qualquer um dos menus, o programa não executa nenhuma ação.

Por questões de organização, o arquivo main.c contém apenas os procedimentos referentes à navegação principal. Toda a execução dos testes, bem como funções secundárias como geração de itens visuais, funções de verificação e afins, são encontrados na biblioteca principal utilidades.c. Os algoritmos de ordenação foram alocados separadamente na biblioteca sorts.c, e estes são acessados pela função executar() da biblioteca principal, de acordo com o algoritmo escolhido.

2.3 Dificuldades e Problemas Conhecidos

A principal dificuldade no desenvolvimento deste projeto foi a implementação dos diferentes algoritmos em um mesmo contexto. Alterações foram inevitáveis em alguns algoritmos para o correto funcionamento no programa.

Os algoritmos Tim Sort e Radix Sort apresentaram comportamento errôneo e foram, portanto, reescritos, funcionando de maneira favorável posteriormente.

O algoritmo Quick Sort não demonstrou problemas ao processar vetores com 100 mil números aleatórios em nenhum teste, porém o mesmo apresenta erro sempre que testado com vetores ordenados tanto em ordem crescente quanto em ordem decrescente, com essa mesma quantidade. Portanto, a medição nessa faixa para esse algoritmo não foi considerada.

Com as configurações de processamento e memória utilizadas, a maior parte dos algoritmos não executa corretamente quando a quantidade de dados é superior a 100 mil. A opção de quantidade de um milhão resulta no encerramento forçado do programa ou na execução por tempo indeterminado.

3 RESULTADOS

Os testes nas três condições propostas foram executados diversas vezes para fins de verificação da integridade dos resultados, e os valores obtidos são apresentados a seguir.

Comparação com Listas Desorganizadas

Elementos	Tempo de execução (s)								
	Insertion Sort	Selection Sort	Shell Sort	Bubble Sort	Merge Sort	Heap Sort	Quick Sort	Radix Sort	Tim Sort
1 mil	0,000301	0,000709	0	0,000686	0	0	0,000103	0,0001	0,000245
5 mil	0,008909	0,013739	0	0,029759	0,000336	0	0,000451	0,0002	0,001349
10 mil	0,034636	0,052148	0,001241	0,122996	0,001233	0,001564	0,00071	0,000109	0,003657
20 mil	0,138926	0,209387	0,002282	0,656894	0,001983	0,003125	0,001119	0,000975	0,011313
50 mil	0,902623	1,307994	0,00768	4,579688	0,006031	0,007814	0,003899	0,002168	0,027409
100 mil	3,486598	5,313095	0,016333	19,210036	0,012397	0,015624	0,937489	0,004228	0,049608
1 milhão	-	-	0,213443	-	-	0,206356	-	-	-

Tabela 1 - Desempenho dos algoritmos ordenando vetores com elementos aleatórios.

Com base nos valores obtidos, observa-se que para mil elementos o tempo de ordenação se torna próximo de zero para todos os algoritmos, porém as diferenças de desempenho começam a indicar os programas menos eficientes a partir de cinco mil elementos. Utilizando 5 mil ou 10 mil itens, os algoritmos Bubble Sort, Selection Sort e Insertion Sort levam consideravelmente mais tempo para ordená-los do que os demais algoritmos.

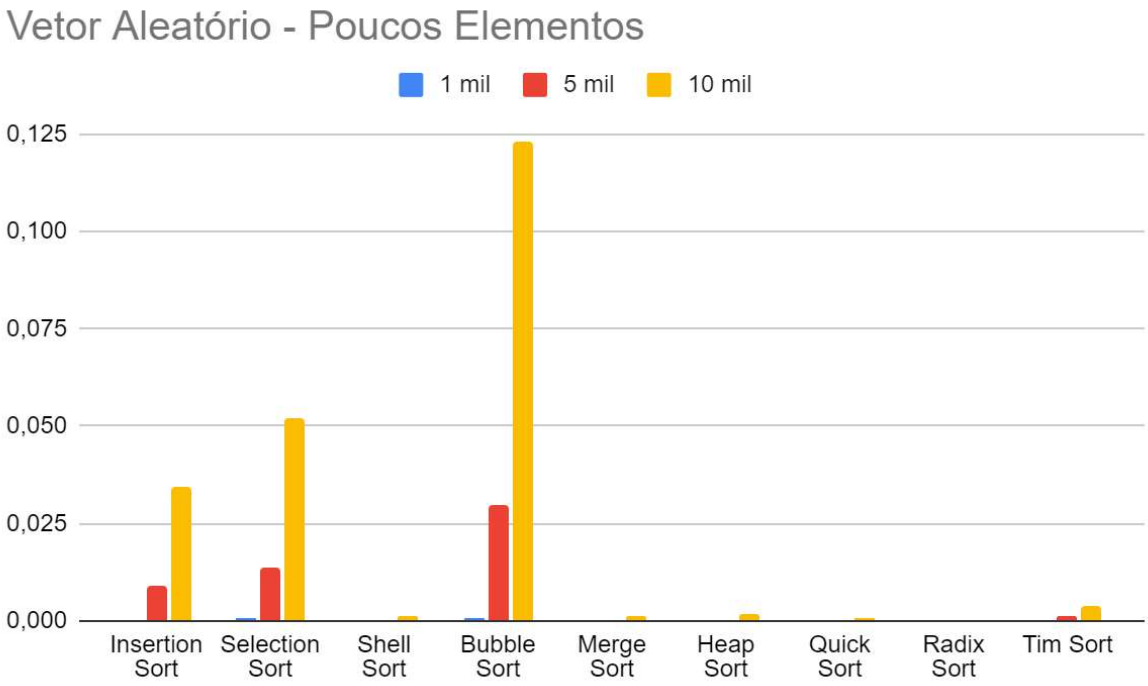

Gráfico 1 - Desempenho dos algoritmos ordenando vetores com elementos aleatórios - 1 mil até 10 mil elementos.

Para vetores maiores, observa-se que o Selection Sort e o Bubble Sort se tornam demasiado ineficientes, levando mais de um segundo para ordenar listas com 50 mil e 100 mil números. Conforme ilustrado no gráfico 2, grande parte dos outros vetores possuem desempenho consideravelmente superior mesmo com quantidades maiores de dados. Ainda de acordo com o gráfico 2, é possível notar que o Radix Sort foi o algoritmo mais eficiente nos testes.

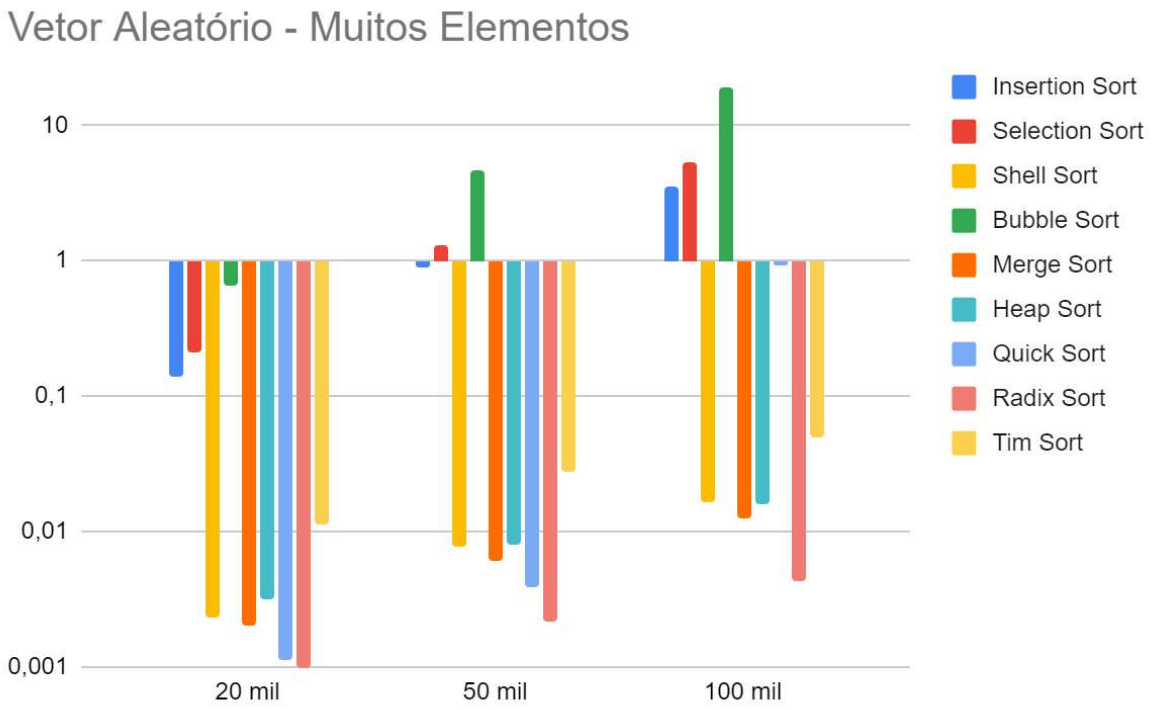

Gráfico 2 - Desempenho dos algoritmos ordenando vetores com elementos aleatórios - 20 mil até 100 mil elementos.

Quando testados com vetores ordenados em ordem crescente, todos os algoritmos desempenharam os testes em tempo próximo à zero para quase todas as quantidades, citando como excessão o Selection Sort, que quando utilizado com 100 mil elementos, leva mais 5 segundos, e o Quick Sort, que apresenta desempenho inferior comparado com a ordenação de um vetor de números aleatórios.

Comparação com Listas Organizadas - Ordem Crescente

Elementos	Tempo de execução (s)								
	Insertion Sort	Selection Sort	Shell Sort	Bubble Sort	Merge Sort	Heap Sort	Quick Sort	Radix Sort	Tim Sort
1 mil	0	0,000992	0	0	0	0	0	0	0
5 mil	0	0,013087	0	0	0	0	0,006541	0	0,001001
10 mil	0	0,052051	0	0	0	0	0,025308	0,000925	0,002001
20 mil	0	0,206734	0	0	0,001514	0	0,088537	0,001331	0,008645
50 mil	0	1,303509	0,001007	0,000506	0,003083	0,015614	0,392427	0,002646	0,022732
100 mil	0,001006	5,224143	0,001992	0,000502	0,00601	0,015625	-	0,005093	0,049608s
1 milhão	-	-	0,023903	-	-	0,140625	-	-	-

Tabela 2 - Desempenho dos algoritmos em comportamento natural.

Ao contrário do observado na comparação com listas desorganizadas, os algoritmos Bubble Sort e Insertion Sort se mostraram mais eficientes do que os demais quando trabalham com listas já ordenadas, enquanto o Selection Sort demonstrou o pior desempenho, levando mais de 1 segundo para percorrer o vetor com 50 mil elementos, e mais de 5 segundos para o vetor de 100 mil elementos.

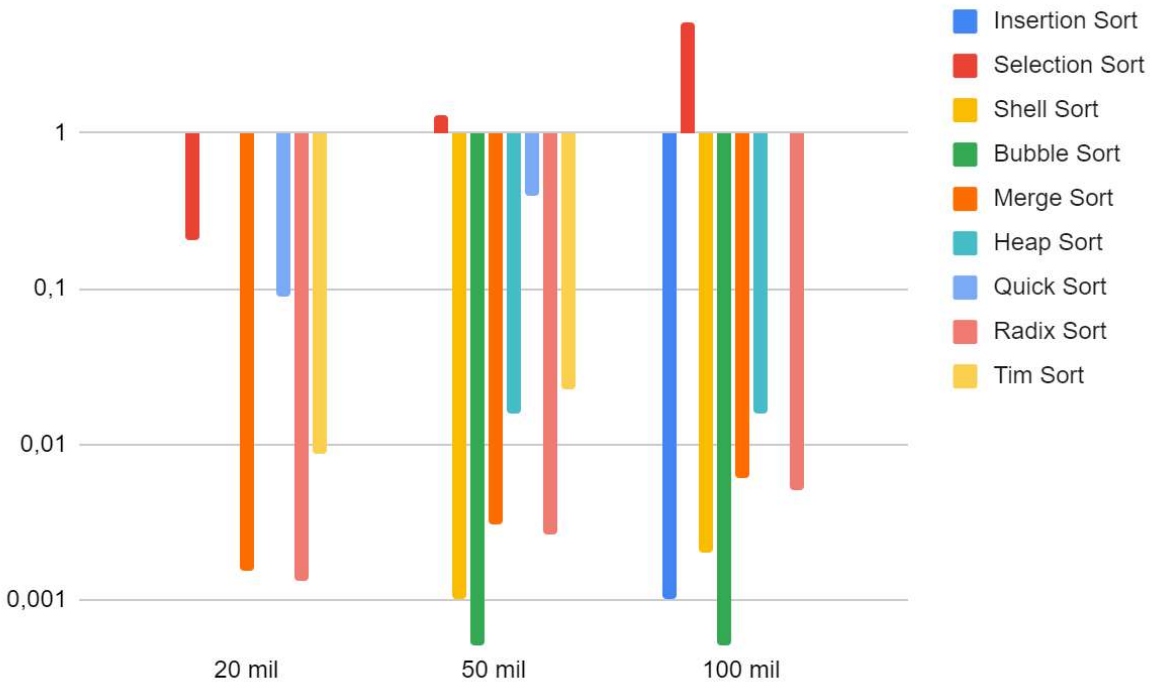

Gráfico 3 - Desempenho dos algoritmos em comportamento natural.

Quando confrontados com vetores com ordenação decrescente, alguns algoritmos obtiveram desempenho superior aos demais testes.

Comparação com Listas Organizadas - Ordem Decrescente

Elementos	Tempo de execução (s)								
	Insertion Sort	Selection Sort	Shell Sort	Bubble Sort	Merge Sort	Heap Sort	Quick Sort	Radix Sort	Tim Sort
1 mil	0,001013	0,000999	0	0	0	0	0,000999	0	0,000931
5 mil	0,018187	0,017663	0	0,036542	0	0	0,00815	0,001	0,015033
10 mil	0,069156	0,068223	0	0,146128	0,001009	0	0,028898	0,001517	0,053159
20 mil	0,277076	0,338826	0,00101	0,584201	0,002012	0	0,097735	0,000929	0,216646
50 mil	1,773195	3,346038	0,00202	3,599981	0,003	0	0,486976	0,002012	1,241655
100 mil	7,006633	10,54402	0,003646	14,418831	0,007015	0,015685	-	0,00466	5,070941
1 milhão	-	-	0,036097	-	-	0,140625	-	-	-

Tabela 3 - Desempenho dos algoritmos em pior caso.

Para vetores com poucos elementos, observa-se uma situação similar à comparação com vetores não ordenados, onde o Insertion Sort, Selection Sort e Bubble Sort apresentam desempenho consideravelmente inferior aos outros algoritmos quando utilizando 5 mil e 10 mil números. No entanto, pode-se notar que o Quick Sort e o Tim Sort também demonstram menor eficiência para listas invertidas.

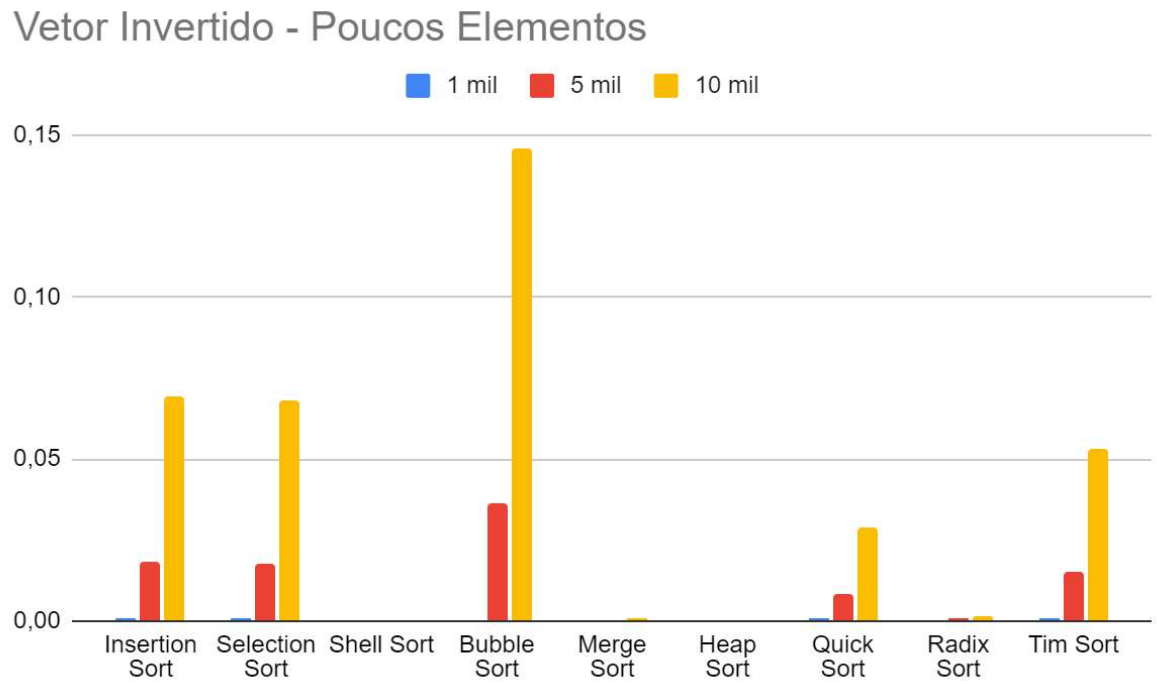

Gráfico 4 - Desempenho dos algoritmos em pior caso - 1 mil até 10 mil elementos.

O mesmo padrão se repete para listas com muitos itens, com exceção do Quick Sort, conforme indicado no gráfico 5. De acordo com o mesmo gráfico, é possível verificar que Shell Sort, Heap Sort, Merge Sort e Radix Sort obtiveram os melhores resultados para o teste de pior caso, e o mesmo pode ser observado para os outros testes.

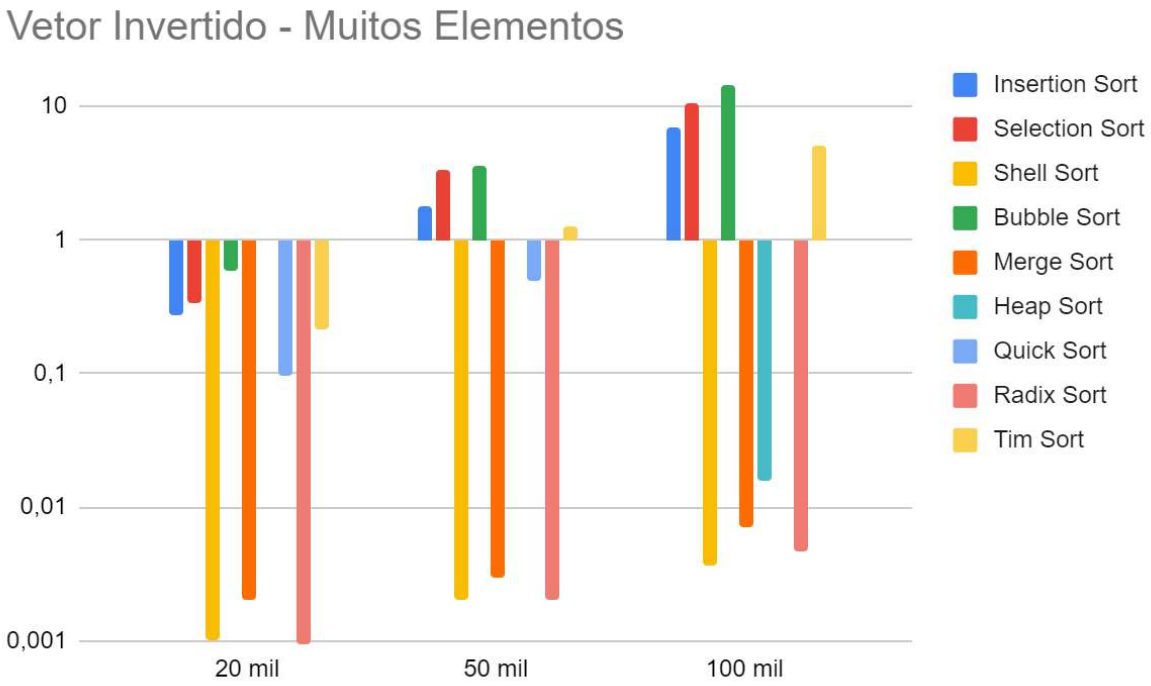

Gráfico 5 - Desempenho dos algoritmos em pior caso - 20 mil até 100 mil elementos.

É digno de menção que, com as limitações das configurações de hardware, apenas Shell Sort e Heap Sort foram capazes de ordenar vetores com um milhão de elementos, e não obstante, o fazem com desempenho melhor do que outros algoritmos fazem com vetores menores.