INSTITUTO FEDERAL DE MATO GROSSO DO SUL

CAMPUS AQUIDAUANA

CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

FERNANDA YUKIMI OHNISHI
LUCAS PRADO DA SILVA

TRABALHO DE ESTRUTURA DE DADOS

Aquidauana – MS
2025

Introdução

Organizar dados é essencial na computação. Assim como seria difícil encontrar um nome em uma lista telefônica desordenada, sistemas digitais dependem da ordenação para otimizar buscas, consultas e desempenho.

Esse processo é feito por algoritmos de ordenação, que organizam listas de elementos (números, palavras etc.) em ordem crescente, decrescente ou alfabética. A eficiência desses algoritmos influencia diretamente a velocidade e o uso de recursos dos sistemas.

Este relatório apresenta, de forma objetiva, os principais algoritmos de ordenação — seus funcionamentos, vantagens, desvantagens e complexidades — e compara quais são mais adequados para cada situação.

1. Critérios de Avaliação dos Algoritmos

Para saber se um algoritmo é realmente eficiente, a gente olha principalmente dois pontos:

- Complexidade de Tempo: quanto mais a lista cresce, mais tempo o algoritmo leva pra terminar.
- Complexidade de Espaço: quanta memória extra ele precisa pra funcionar.

Além disso, dá pra analisar o comportamento em três situações:

- Melhor caso: quando os dados já estão quase em ordem.
- Pior caso: quando tá tudo bagunçado.
- Caso médio: o que normalmente acontece.

2. Os principais algoritmos

2.1. Bubble Sort

O Bubble Sort é o mais simples de todos. Ele compara os elementos lado a lado e troca de lugar quando estão fora de ordem. Faz isso várias vezes até a lista ficar certinha.

- **Complexidade:** $O(n^2)$, mas pode ser $O(n)$ se a lista já estiver quase pronta.
- **Vantagens:** Fácil de entender e de programar.
- **Desvantagens:** Super lento com listas grandes.
- **Ideal para:** aprender o básico sobre ordenação.

2.2. Selection Sort

Esse aqui é mais “organizado”. Ele procura o menor elemento da lista e coloca no início. Depois faz o mesmo com o restante, até tudo estar em ordem.

- **Complexidade:** $O(n^2)$, sempre.
- **Vantagens:** Faz poucas trocas, o que ajuda se trocar for algo “caro”.
- **Desvantagens:** Continua sendo lento.
- **Ideal para:** quando você quer economizar nas trocas.

2.3. Insertion Sort

Funciona tipo quando você arruma cartas na mão: pega uma por vez e encaixa no lugar certo entre as que já estão em ordem.

- **Complexidade:** $O(n^2)$, mas pode ser $O(n)$ se a lista já estiver quase certa.
- **Vantagens:** Rápido pra listas pequenas ou quase ordenadas.
- **Desvantagens:** Não funciona bem pra listas grandes.
- **Ideal para:** pequenas listas ou como parte auxiliar em outros algoritmos.

2.4. Merge Sort

Segue o famoso “dividir pra conquistar”. Ele divide a lista no meio várias vezes, até sobrar listas com um item. Depois vai juntando tudo em ordem.

- **Complexidade:** $O(n \log n)$, sempre.
- **Vantagens:** Tem desempenho estável e previsível.
- **Desvantagens:** Usa mais memória.
- **Ideal para:** quando precisa de estabilidade e confiabilidade.

2.5. Quick Sort

Outro que também divide pra conquistar, mas de um jeito diferente. Ele escolhe um pivô e separa os menores pra um lado e os maiores pro outro, repetindo o processo até tudo ficar ordenado.

- **Complexidade:** média de $O(n \log n)$, mas pode chegar a $O(n^2)$ se o pivô for mal escolhido.
- **Vantagens:** Geralmente o mais rápido.
- **Desvantagens:** Pode ficar lento se escolher mal o pivô.
- **Ideal para:** quando a prioridade é velocidade.

2.6. Heap Sort

Usa uma estrutura chamada heap, tipo uma árvore onde o “pai” é sempre maior que os “filhos”. Ele monta o heap, pega o maior elemento, coloca no final da lista e repete até ficar tudo em ordem.

- **Complexidade:** O(n log n).
- **Vantagens:** Rápido e usa pouca memória extra.
- **Desvantagens:** É mais chatinho de implementar.
- **Ideal para:** lidar com grandes quantidades de dados usando pouca memória.

3. Comparação Geral

Algoritmo	**Velocidade Geral**	**Memória Extra**	**Mais Indicado Para...**
Bubble Sort	Muito lento	Não	Ensino e demonstração
Selection Sort	Muito lento	Não	Situações com poucas trocas
Insertion Sort	Lento (rápido se quase ordenado)	Não	Listas pequenas
Merge Sort	Rápido e constante	Sim	Quando é preciso estabilidade
Quick Sort	Muito rápido (geralmente)	Pouca	Processos que exigem alta velocidade
Heap Sort	Rápido e estável	Não	Grandes volumes de dados com pouca memória

Conclusão

No geral, Merge Sort, Quick Sort e Heap Sort são os melhores pra lidar com listas grandes — são rápidos e eficientes.

Já o Bubble, Selection e Insertion Sort servem mais pra entender o conceito de ordenação, já que são simples mas lentos.

Cada um tem seus pontos fortes e fracos. Não existe um “melhor de todos”, e sim o mais adequado pra cada situação — seja priorizando velocidade, memória ou estabilidade.

Saber escolher o certo faz toda a diferença pra deixar os programas mais rápidos e inteligentes.

RELATORIO DE TESTES

MAQUINA DE TESTES

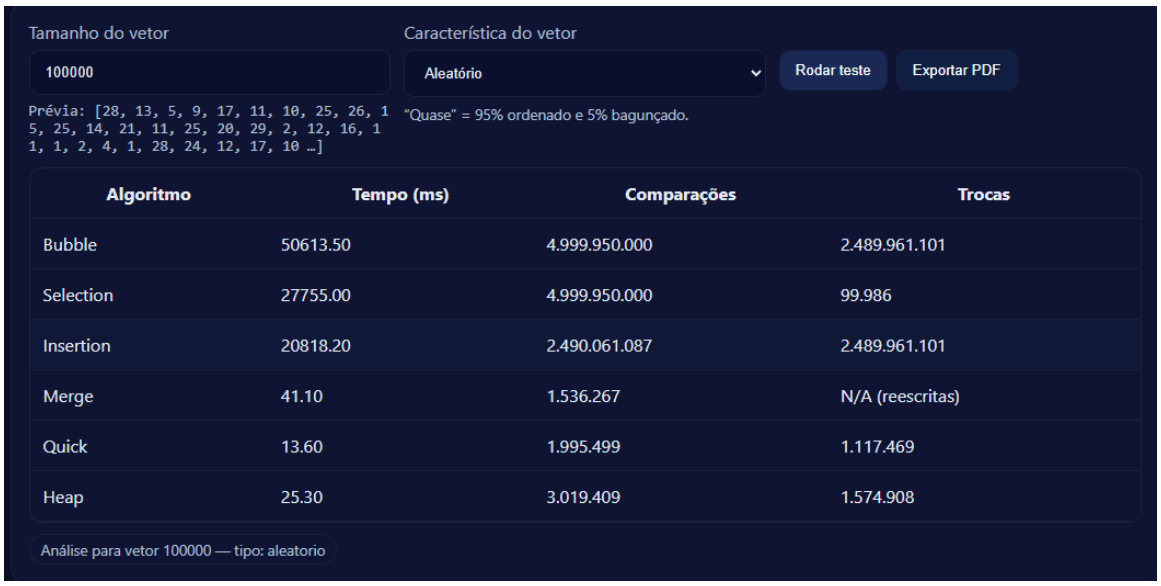
Cpu : RYZEN 5 5600X

MEMORIA RAM 32GB DDR4

SSD : NVME 500GB

PLACA DE VIDEO : GTX 1070TI

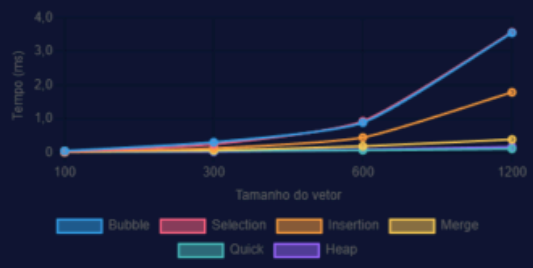
TESTES GERADOS DIRETAMENTE NO NAVEGADOR



📊 Relatório de Testes (5 execuções por ponto)

Para cada algoritmo e tamanho, executamos **5 vezes** no cenário selecionado e usamos a **média** para os gráficos. A tabela abaixo resume *média*, *desvio-padrão*, *mín* e *máx* de **tempo** no maior N testado, além das médias de comparações e trocas.

Tempo (ms) × Tamanho



Comparações × Tamanho

Trocas × Tamanho

Resumo estatístico (N máximo)

Cálculo no maior tamanho (N = 1200), com média de 5 execuções por algoritmo — cenário: aleatorio.

Algoritmo	Tempo — média (ms)	Tempo — desvio (ms)	Tempo — mín (ms)	Tempo — máx (ms)	Comparações (média)
Bubble	3.54	0.05	3.50	3.60	719.400
Selection	3.56	0.05	3.50	3.60	719.400
Insertion	1.78	0.08	1.70	1.90	359.467
Merge	0.38	0.08	0.30	0.50	10.784
Quick	0.10	0.07	0.00	0.20	14.133
Heap	0.16	0.05	0.10	0.20	20.887