Universidade Federal do Rio Grande do Norte

Instituto Metrópole Digital

IMD0029 - **Estrutura de Dados Básicas**

Prof° João Guilherme

Análise Comparativa de Algoritmos de Ordenação

Lucas Gabriel dos Santos Anizio

Natal/Rn
2025

Introdução

O que são Algoritmos de Ordenação?

Os algoritmos de ordenação são procedimentos utilizados para organizar elementos de uma lista/vetores ou conjuntos de dados em uma ordem específica podendo ser por exemplo crescente, decrescente, numérica ou lexicográfica. São de suma importância, pois estão na base de muitas operações importantes na computação que facilitam a busca e a recuperação de dados, melhorando o desempenho dos algoritmos e organizando todos os dados.

Exemplos de Algoritmos de Ordenação visto em sala:

- Insertion Sort;
- Bubble Sort;
- Sertion Sort;
- Merge Sort;
- Quick Sort.

Fundamentação teórica

Insertion Sort:

O **Insertion Sort** é um algoritmo de ordenação que insere os elementos um a um na posição correta dentro de uma parte já ordenada. Em outras palavras, ele percorre todo o vetor, seleciona cada elemento e o posiciona de forma adequada em relação aos que já foram ordenados. Esse algoritmo é bastante eficiente para listas pequenas ou quase ordenadas, mas é ineficiente em vetores ou listas muito grandes devido à sua complexidade. É especialmente indicado quando a simplicidade e a baixa utilização de memória são prioridades, sendo uma boa opção em sistemas que exigem baixo overhead.

Complexidade:

- Pior caso: $O(n^2)$
- Melhor caso: $O(n)$
- Caso mediano: $O(n^2)$
- Espaço: $O(1)$

Bubble Sort:

O **Bubble Sort Sort** se chama assim, pois os elementos “Flutuam” para o topo ou final da lista/vetor, assim como as bolhas sobem à superfície. Ele funciona comparando pares de elementos adjacentes e trocando suas posições se não estiverem ordenados. O processo vai se repetindo diversas vezes até que o vetor seja completamente ordenado. O Bubble Sort é simples e estável na grande parte das vezes, porém, é lento em em grandes vetores

e ineficiente se for comparado aos outros algoritmos de ordenação, realizando muitas comparações e trocas desnecessárias.

Complexidade:

- Pior caso: O(n²)
- Melhor caso: O(n)
- Caso mediano: O(n²)
- Espaço: O(1)

Selection Sort:

O **Selection Sort** é um algoritmo de ordenação que organiza uma lista selecionando, a cada iteração, pega o menor elemento do vetor e posiciona na posição correta. Por ser um dos métodos mais simples de ordenação, sua eficiência é limitada, tornando-o pouco adequado para grandes volumes de dados.

Complexidade:

- Pior caso: O(n²)
- Melhor caso: O(n²)
- Caso mediano: O(n²)
- Espaço: O(1)

Merge Sort:

O **Merge Sort** utiliza a estratégia de **"Dividir para Conquistar"**, na qual o vetor é dividido em partes menores, cada uma delas é ordenada separadamente e, em seguida, todas são intercaladas na ordem correta para formar o vetor completo e ordenado. Comparado a outros algoritmos de ordenação, o Merge Sort é um dos mais complexos em termos de implementação, porém ele se destaca por ser altamente eficiente em grandes conjuntos de dados.

Complexidade:

- Pior caso: O(n log n)
- Melhor caso: O(n log n)
- Caso mediano: O(n log n)
- Espaço: O(n)

Quick Sort:

O **Quick Sort** é semelhante ao **Merge Sort**, pois também utiliza a estratégia de **"Dividir para Conquistar"**, porém funciona de maneira diferente. Em vez de apenas dividir a lista ao meio, ele seleciona um pivô e reorganiza os elementos ao redor dele, posicionando os menores à esquerda e os maiores à direita. Por essas características, o Quick Sort costuma ser mais eficiente que o Merge Sort na prática, além de funcionar in-place (sem necessidade de memória adicional significativa) e ser amplamente utilizado para grandes volumes de dados.

Complexidade:

- Pior caso: O(n²)
- Melhor caso: O(n log n)
- Caso mediano: O(n log n)
- Espaço: O(1)

Relatório dos testes de código

Tabelas:

Especificações de dados:

- **Tamanhos:** 500, 5000, 10000
- **Tipos:** aleatórios, quase ordenado e inversamente ordenado
- **Método de medição:** O tempo de execução dos algoritmos foi medido utilizando a função chrono da linguagem C++, que registra o tempo entre o início e o fim da execução de cada algoritmo de ordenação.

Tabelas do Tamanho 500:

Tipo Aleatório:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	500	0.0e+000s	124750	64932
Insertion Sort	500	0.0e+000s	65424	64932
Selection Sort	500	0.0e+000s	124750	490
Merge Sort	500	0.0e+000s	3882	1903
Quick Sort	500	0.0e+000s	4739	2752

Tipo Quase Ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	500	0.0e+000s	124750	50
Insertion Sort	500	0.0e+000s	548	50
Selection Sort	500	0.0e+000s	124750	50
Merge Sort	500	0.0e+000s	2297	49
Quick Sort	500	0.0e+000s	113629	114035

Tipo Inversamente Ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	500	0.0e+000s	124750	124750
Insertion Sort	500	0.0e+000s	124750	124750
Selection Sort	500	1.6e-002s	124750	250
Merge Sort	500	0.0e+000s	2216	2216
Quick Sort	500	0.0e+000s	124750	62749

Tabelas do Tamanho 5000:

Tipo Aleatório:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	5000	3.1e-002s	12497500	6165920
Insertion Sort	5000	1.6e-002s	6170907	6165920
Selection Sort	5000	0.0e+000s	12497500	4983
Merge Sort	5000	0.0e+000s	55211	27035
Quick Sort	5000	0.0e+000s	67525	36625

Tipo Quase Ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	5000	1.8e-002s	12497500	466
Insertion Sort	5000	0.0e+000s	5465	466
Selection Sort	5000	1.4e-002s	12497500	464
Merge Sort	5000	5.0e-003s	32202	455
Quick Sort	5000	2.7e-002s	11463113	11467215

Tipo Inversamente Ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	5000	3.8e-002s	12497500	12497500
Insertion Sort	5000	2.0e-002s	12497500	12497500
Selection Sort	5000	1.6e-002s	12497500	2500
Merge Sort	5000	0.0e+000s	29804	29804
Quick Sort	5000	3.2e-002s	12497500	6252499

Tabelas do Tamanho 10000:

Tipo Aleatório:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	10000	1.3e-001s	49995000	25063292
Insertion Sort	10000	6.4e-002s	25073276	25063292
Selection Sort	10000	3.2e-002s	49995000	9989
Merge Sort	10000	0.0e+000s	120461	59174
Quick Sort	10000	0.0e+000s	157761	87530

Tipo Quase Ordenado:

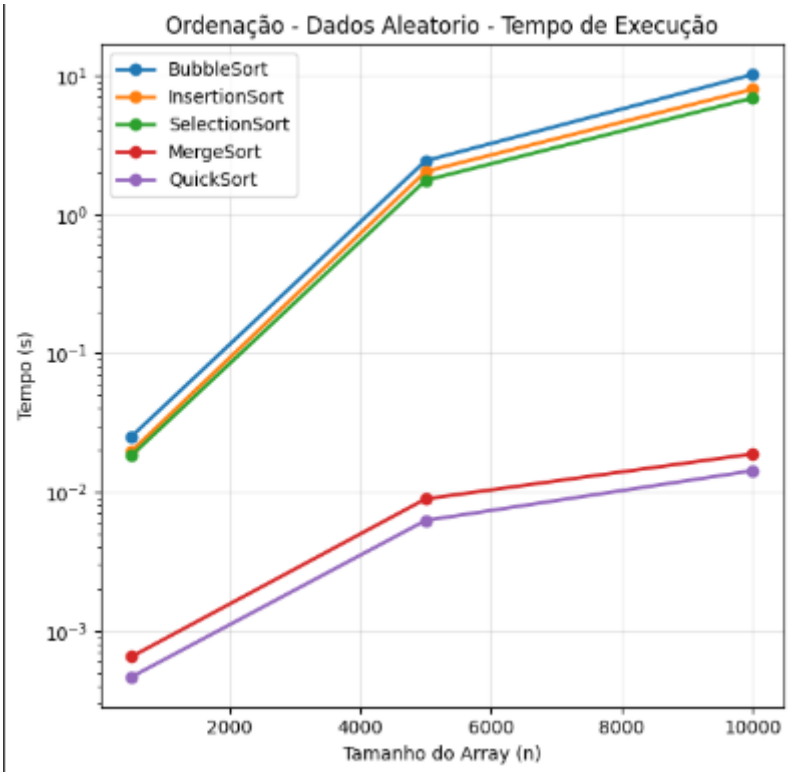
ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	10000	4.0e-002s	49995000	916
Insertion Sort	10000	0.0e+000s	10914	916
Selection Sort	10000	3.9e-002s	49995000	908
Merge Sort	10000	1.6e-002s	69381	876
Quick Sort	10000	1.7e-001s	45937211	45945458

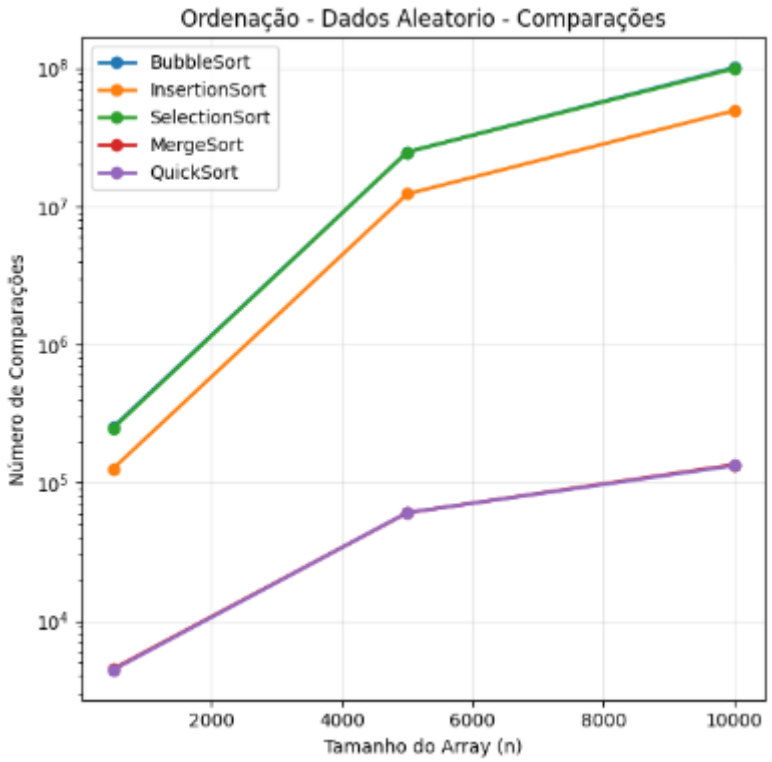
Tipo Inversamente Ordenado:

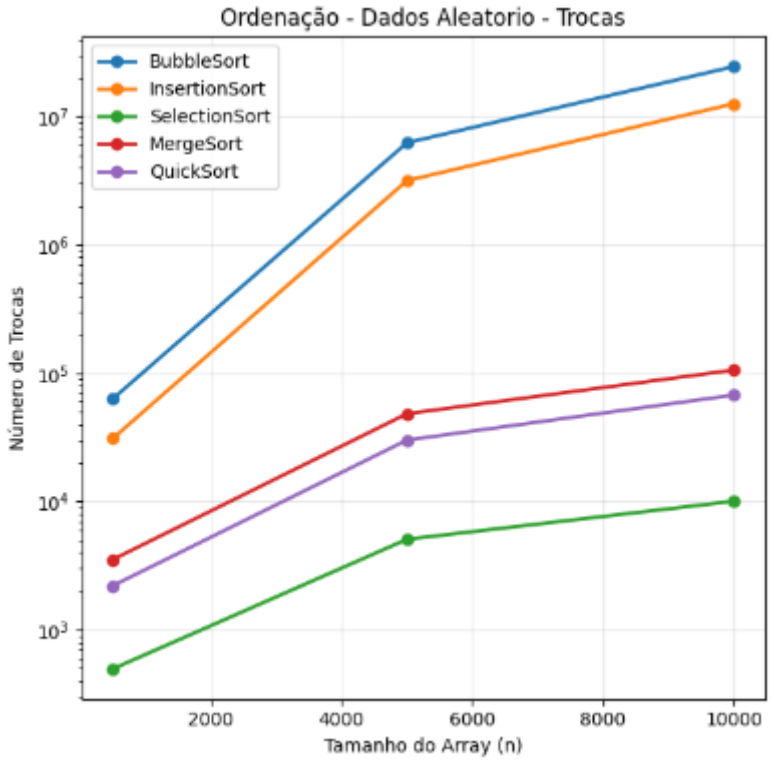
ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	10000	1.2e-001s	49995000	49995000
Insertion Sort	10000	1.2e-001s	49995000	49995000
Selection Sort	10000	3.2e-002s	49995000	5000
Merge Sort	10000	3.0e-003s	64608	64608
Quick Sort	10000	1.0e-001s	49995000	25004999

Gráficos:

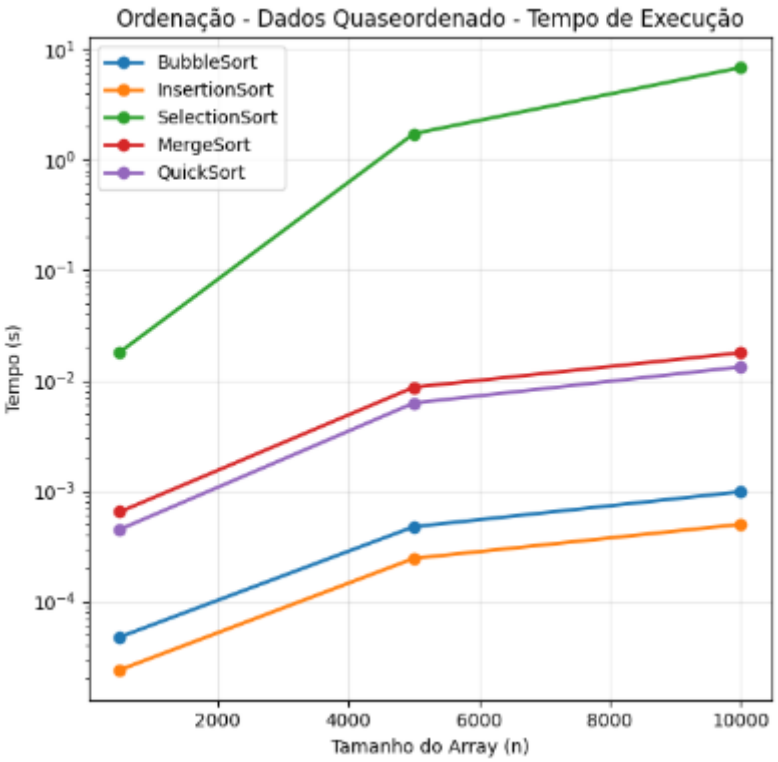
Aleatórios:

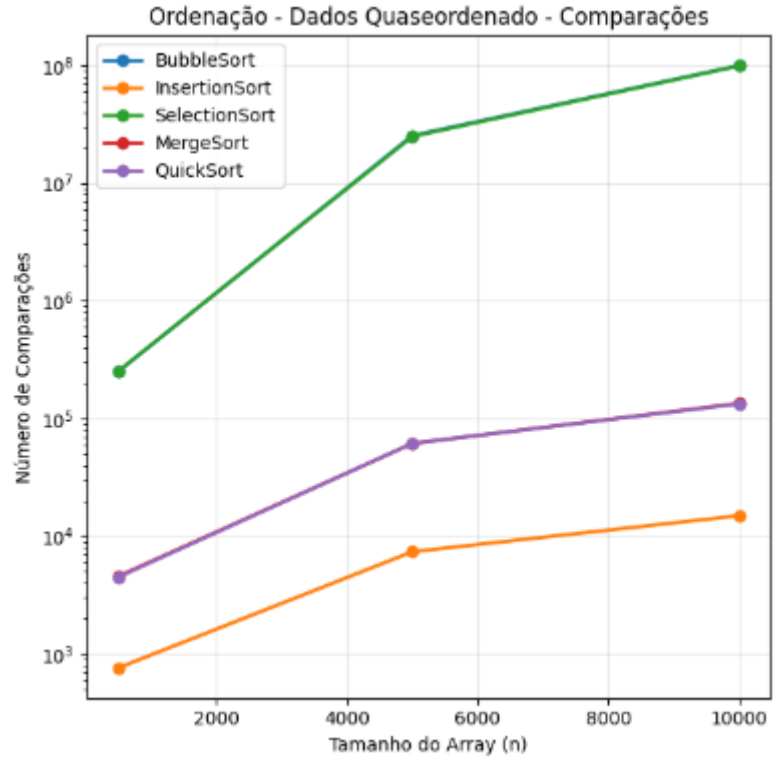


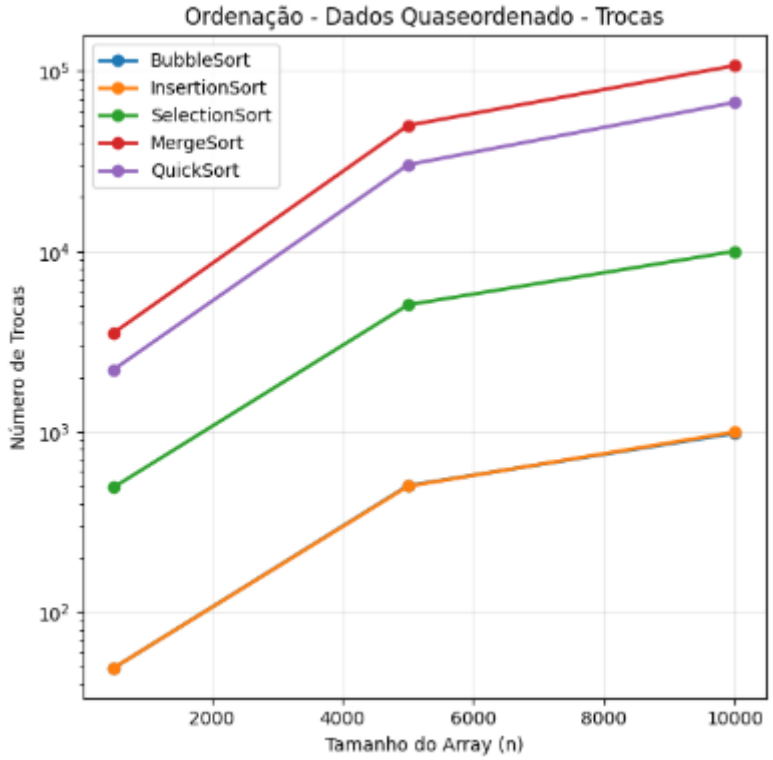

Ordenação - Dados Aleatorio - Comparações
BubbleSort
InsertionSort
SelectionSort
MergeSort
QuickSort
Número de Comparações
Tamanho do Array (n)
10^8
10^7
10^6
10^5
10^4
2000
4000
6000
8000
10000


Ordenação - Dados Aleatorio - Trocas
BubbleSort
InsertionSort
SelectionSort
MergeSort
QuickSort
Número de Trocas
Tamanho do Array (n)
10^7
10^6
10^5
10^4
10^3
2000
4000
6000
8000
10000

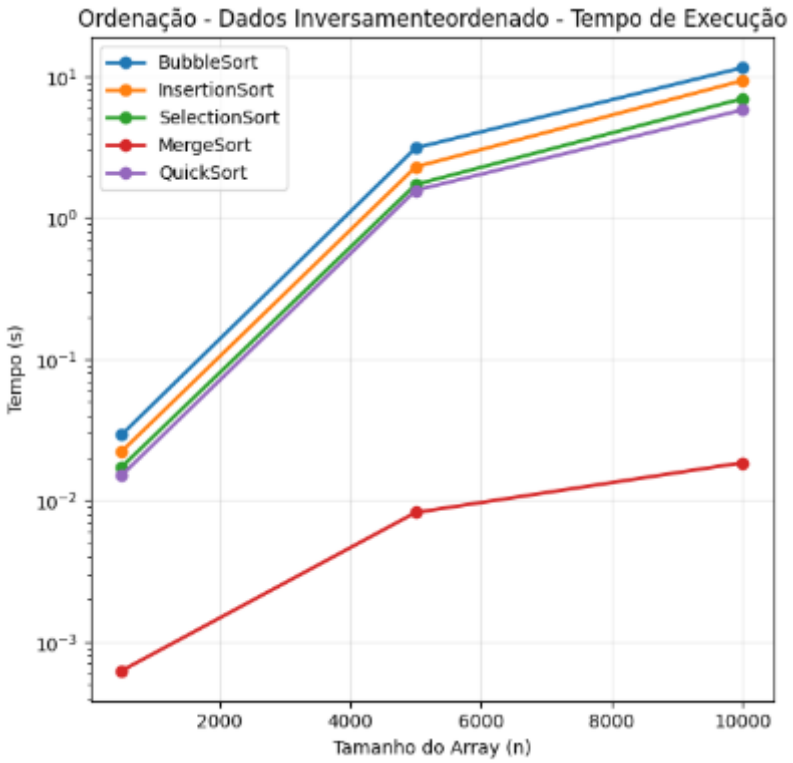
Quase Ordenados:

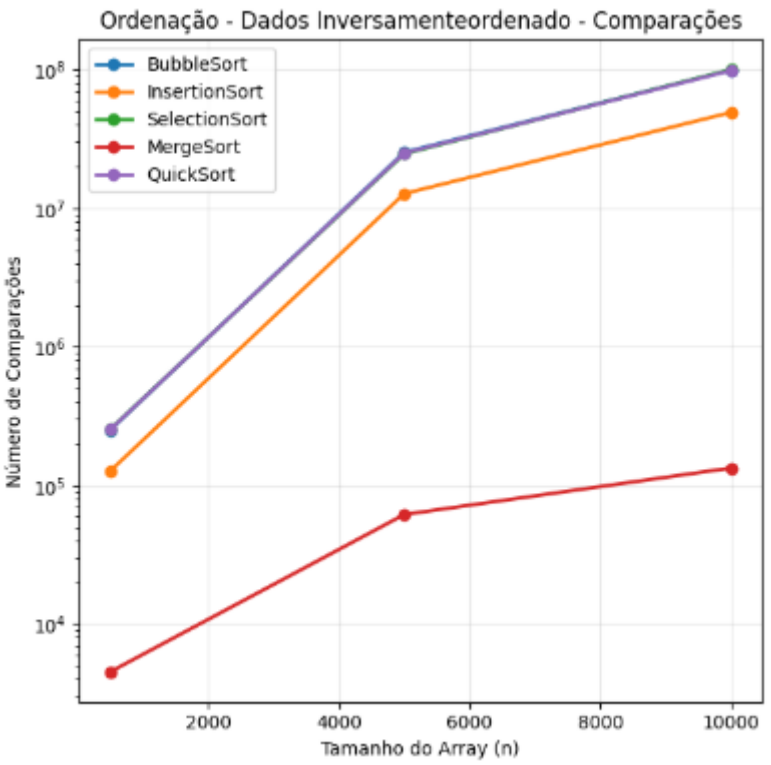


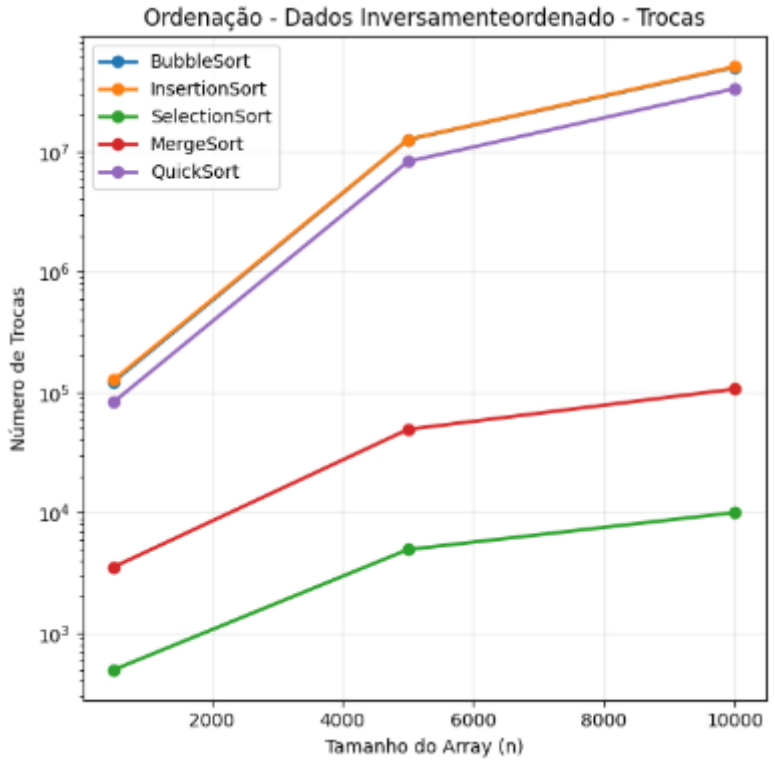




Inversamente Ordenados:




Ordenação - Dados Inversamenteordenado - Comparações
BubbleSort
InsertionSort
SelectionSort
MergeSort
QuickSort
Número de Comparações
Tamanho do Array (n)


Ordenação - Dados Inversamenteordenado - Trocas
BubbleSort
InsertionSort
SelectionSort
MergeSort
QuickSort
Número de Trocas
Tamanho do Array (n)

Resultado e Análise:

Com a coleta dos dados das tabelas e a geração dos gráficos utilizando a biblioteca Matplotlib no python, foi possível obter uma visão mais ampla do comportamento e do desempenho dos principais algoritmos de ordenação citados neste documento. A análise dos resultados permite entender não apenas a eficiência de cada algoritmo em diferentes cenários, mas também como eles se comportam em termos de tempo de execução, número de comparações e quantidade de trocas realizadas.

Ao observar o desempenho dos algoritmos com dados aleatórios, é possível notar que Bubble Sort, Insertion Sort e Selection Sort apresentam tempos de execução consideravelmente mais altos e crescentes conforme o tamanho do vetor aumenta. Isso acontece pois esses algoritmos possuem complexidade quadrática $O(n^2)$ no caso médio, o que os torna pouco eficientes para conjuntos de dados maiores. Por outro lado, Merge Sort e Quick Sort se destacam por apresentarem tempos de execução muito menores, já que a complexidade de ambos é $O(n \log n)$. No que diz respeito ao número de comparações, Selection Sort realiza praticamente a mesma quantidade independentemente da entrada, enquanto Bubble Sort e Insertion Sort realizam um número muito maior de comparações. Merge Sort e Quick Sort são significativamente mais eficientes. O mesmo padrão se repete no número de trocas: Bubble Sort e Insertion Sort fazem muitas trocas, Selection Sort realiza menos, mas continua ineficiente, e Merge Sort e Quick Sort apresentam o melhor desempenho geral.

Já quando analisamos os algoritmos com dados quase ordenados, ocorre uma mudança significativa. Insertion Sort e Bubble Sort melhoram drasticamente seu desempenho, com o Insertion Sort se aproximando de uma complexidade $O(n)$ devido ao fato de que poucos elementos precisam ser movidos para alcançar a ordenação final. Selection Sort, por sua vez, continua com o seu desempenho ruim, pois não aproveita a ordem já existente. Merge Sort e Quick Sort mantêm sua eficiência e constância mesmo com os dados quase ordenados.

Agora com os dados inversamente ordenados, que representa o pior caso para a maioria dos algoritmos de ordenação, Bubble Sort, Insertion Sort e Selection Sort sofrem um aumento expressivo no tempo de execução, apresentando seus piores desempenhos. Merge Sort e Quick Sort continuam bastante eficientes, com tempos mais estáveis. O número de comparações cresce significativamente nos algoritmos quadráticos, enquanto Merge Sort mantém seu comportamento consistente e Quick Sort pode ter um aumento moderado dependendo da escolha do seu pivô. Quanto ao número de trocas, Bubble Sort e Insertion Sort realizam um número muito elevado, enquanto Merge Sort e Quick Sort continuam sendo os mais econômicos.

Por fim, ao analisar todas essas ordens de dados, percebemos que o Merge Sort e o Quick Sort são os algoritmos mais eficientes se forem comparados aos outros, em questão de tempo, comparação e trocas realizadas. Porém, ao depender da demanda e do sistema, os algoritmos Bubble Sort, Insertion Sort e Selection Sort podem ser ótimas opções para o usuário colocar em seu sistema.