Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0029 **- Estrutura de Dados Básicas**

Aluna: Bruna Dantas **Matrícula**: 20240069703

Análise Comparativa de Algoritmos de Ordenação

Introdução

O que são algoritmos de ordenação?

Já sabemos que algoritmos são instruções de um código na área da computação, portanto, algoritmos de ordenação são procedimentos que organizam elementos de uma lista ou conjunto de dados em uma ordem específica, geralmente numérica ou lexicográfica. Essa ordem pode ser crescente ou decrescente. Eles são fundamentais para tornar a manipulação, análise e organização de dados mais eficiente.

Este trabalho fará uma listagem e análise de cada tipo de algoritmos que foram estudados durante as aulas

Principais exemplos de algoritmos de ordenação:

- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quick Sort

1. Selection Sort:

Este algoritmo divide a lista em 2 partes, uma parte ordenada e outra não. A cada iteração, o algoritmo seleciona um parâmetro e faz a troca de lugar com um outro dado, até que a lista esteja ordenada.

Complexidade:

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

Não tem estabilidade, ordena in-place, não usa espaço extra significativo, é simples de implementar, é ineficiente para listas grandes e não é adaptativo. Melhor usado quando a simplicidade é mais importante que a eficiência (onde situações onde espaço de memória é limitado)

2. Insertion Sort:

Este algoritmo que também divide a lista em duas partes dividindo em uma parte ordenada e não ordenada, semelhante às cartas de um baralho, o algoritmo a

cada iteração, pega um elemento da parte não ordenada e insere na posição correta dentro da parte ordenada.

Complexidade:

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

É estável, é simples de implementar, ordena in place, é eficiente para listas pequenas ou quase ordenadas, contudo é ineficiente para listas grandes (quadrático é o pior caso), quando simplicidade e estabilidade são importantes e por fim, bom usar em sistemas onde há baixo overhead de memória.

3. **Bubble Sort:**

Este algoritmo compara e troca elementos ao lado repetidamente, trocando de posição aqueles que estiverem fora de ordem determinando o dado de maior ordem a cada fim de ciclo, o processo se repete até que a lista esteja ordenada.

Complexidade:

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

Simples de entender e de implementar, é estável (não altera a ordem de elementos iguais), ordenação in-place, lento para grandes conjuntos de dados e ineficiente comparado a outros algoritmos.

4. **Merge Sort**

Este algoritmo usa a estratégia “divide and conquer”, ou seja, ele irá dividir a lista em subsistemas menores, ordenando cada uma e em seguida misturando as sub listas ordenadas para produzir a lista final ordenada.

Complexidade:

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

É estável, eficiente para grandes conjuntos de dados pois tem a mesma complexidade em todos os casos, previsível, contudo, não é in-place (usa memória adicional O(n) para fazer as sublistas) e é mais lento que o quicksort em casos médios devido ao overhead de memória.

5. Quick Sort

Este algoritmo é semelhante ao Merge Sort, usa a estratégia “Divide and conquer”. Exaltado por sua rapidez, o Quicksort escolhe um elemento pivô e organiza os dados à sua direita e à sua esquerda de acordo com este pivô, fazendo iterações até a lista estar completamente ordenada. (sua complexidade depende da escolha do pivô).

Complexidade:

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

É rápido em média, in-place, prático para grandes conjuntos de dados, contudo tem pior caso quando o pivô é mal escolhido, não é estável ou seja pode trocar elementos iguais, melhor utilizado em ambientes com restrição de memória (otimizável).

Relatório dos testes

1.Tabelas

Especificações de dados:

- **Tamanhos**: 100, 1000, 5000
- **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 decorrido entre o início e o fim da execução de cada algoritmo. As métricas de desempenho, como número de comparações e trocas, foram contabilizadas manualmente dentro dos algoritmos, por meio de variáveis incrementadas a cada operação relevante.

Para Tamanho 100

Tipo aleatório:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	100	3.6e-05s	4950	2444
Insertion Sort	100	8e-06s	2541	2444
Selection Sort	100	1.6e-05s	4950	96
Merge Sort	100	1.3e-05s	542	254
Quick Sort	100	6e-06s	690	493

Tipo quase ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	100	1.6e-05s	4950	524
Insertion Sort	100	3e-06s	623	524
Selection Sort	100	1.5e-05s	4950	10
Merge Sort	100	1e-05s	520	189
Quick Sort	100	6e-06s	1395	1143

Tipo inversamente ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	100	4.4e-05s	4950	4950
Insertion Sort	100	1.3e-05s	4950	4950
Selection Sort	100	1.8e-05s	4950	50
Merge Sort	100	1.6e-05s	316	316
Quick Sort	100	1.6e-05s	4950	2549

Para Tamanho 1000

Tipo aleatório:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	1000	0.002961s	499500	258017
Insertion Sort	1000	0.000576s	259007	258017
Selection Sort	1000	0.001464s	499500	993
Merge Sort	1000	0.00012s	8722	4314
Quick Sort	1000	4.8e-05s	10470	5153

Tipo quase ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	1000	0.001549s	499500	63068
Insertion Sort	1000	0.000153s	64067	63068
Selection Sort	1000	0.001572s	499500	100
Merge Sort	1000	7.7e-05s	8035	3225
Quick Sort	1000	4.4e-05s	15483	9590

Tipo inversamente ordenado:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	1000	0.004506s	499500	499500
Insertion Sort	1000	0.00145s	499500	499500
Selection Sort	1000	0.001414s	499500	500
Merge Sort	1000	6.2e-05s	4932	4932
Quick Sort	1000	0.001061s	499500	250499

Para tamanho 5000

Tipo aleatório:

ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	5000	0.056741s	12497500	6213514
Insertion Sort	5000	0.012184s	6218507	6213514
Selection Sort	5000	0.031943s	12497500	4982
Merge Sort	5000	0.000571s	55197	27071
Quick Sort	5000	0.000265s	65663	36482

Tipo quase ordenado:

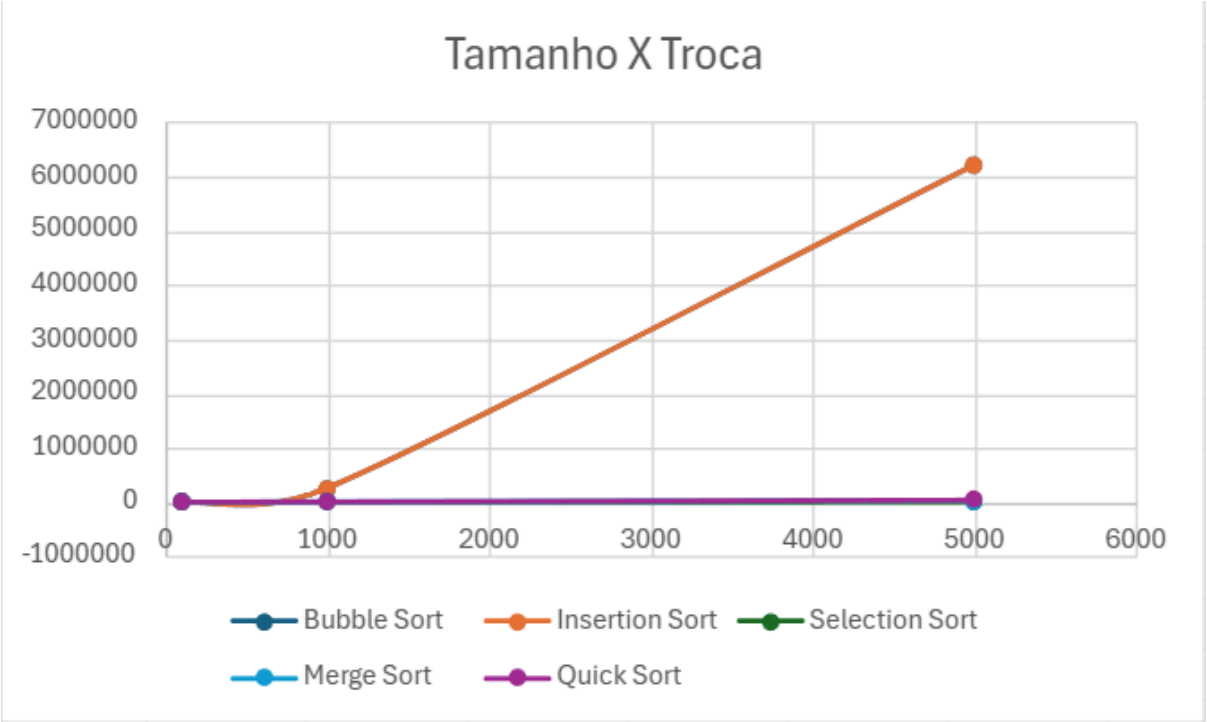
ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	5000	0.029335s	12497500	1373893
Insertion Sort	5000	0.002682s	1378892	1373893
Selection Sort	5000	0.03455s	12497500	499
Merge Sort	5000	0.000392s	52548	21964
Quick Sort	5000	0.000579s	276598	184208

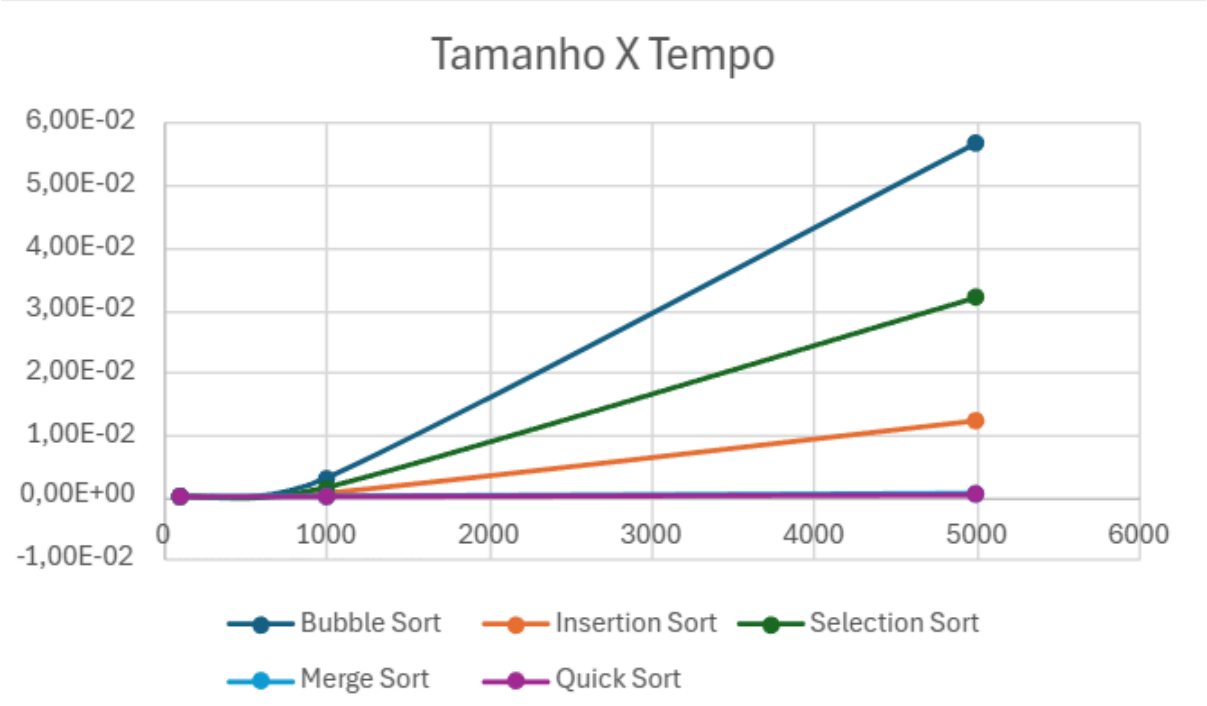
Tipo inversamente ordenado:

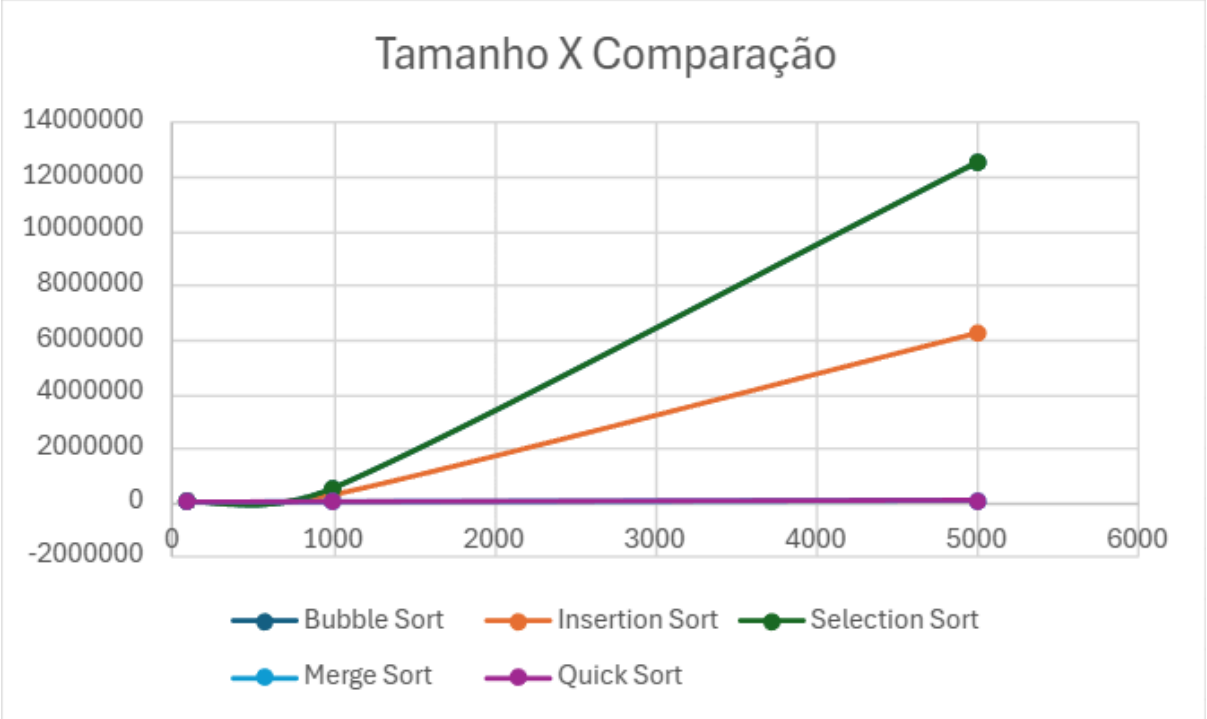
ALGORITMO	TAMANHO	TEMPO	COMPARAÇÕES	TROCAS
Bubble Sort	5000	0.085534s	12497500	12497500
Insertion Sort	5000	0.024574s	12497500	12497500
Selection Sort	5000	0.030862s	12497500	2500
Merge Sort	5000	0.0003s	29804	29804
Quick Sort	5000	0.024306s	12497500	6252499

2. Gráficos

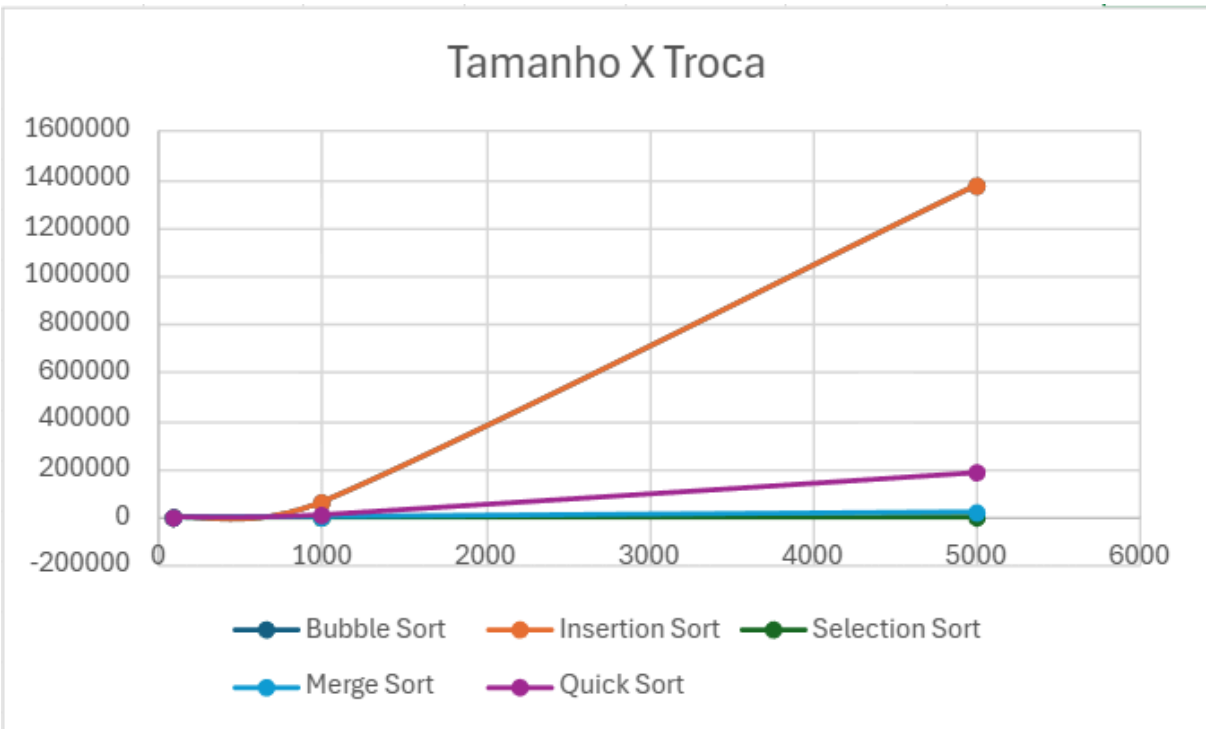
Para dados aleatórios

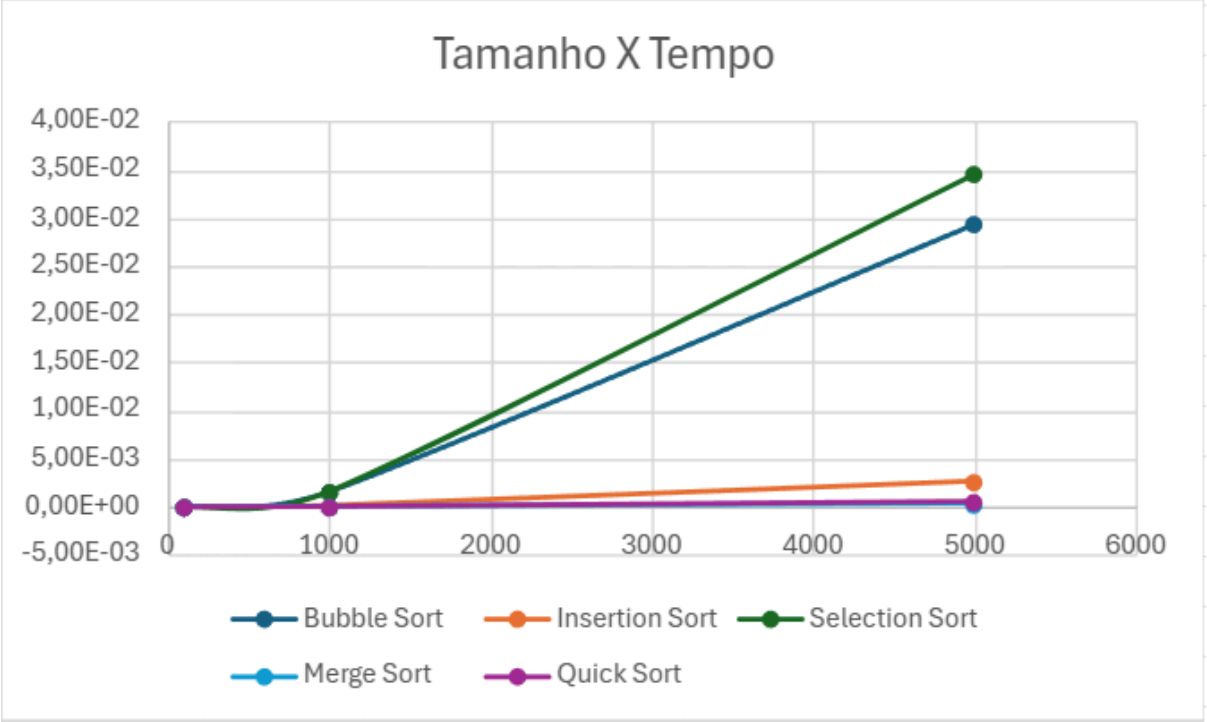


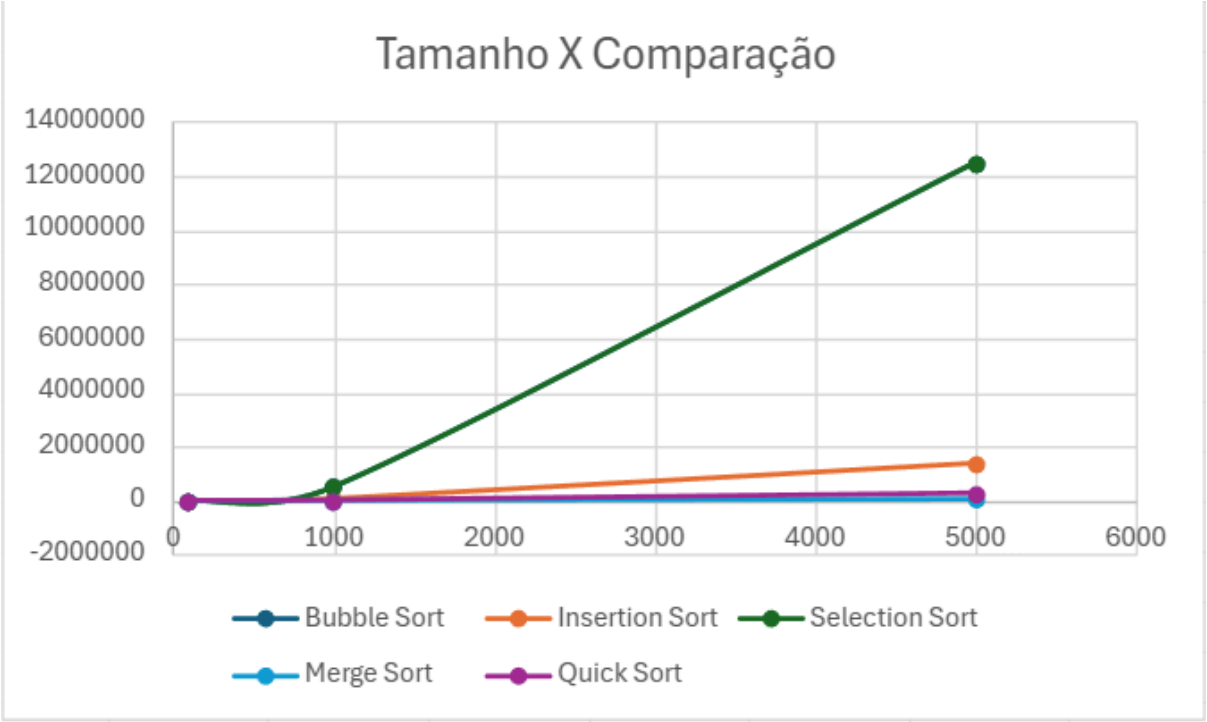




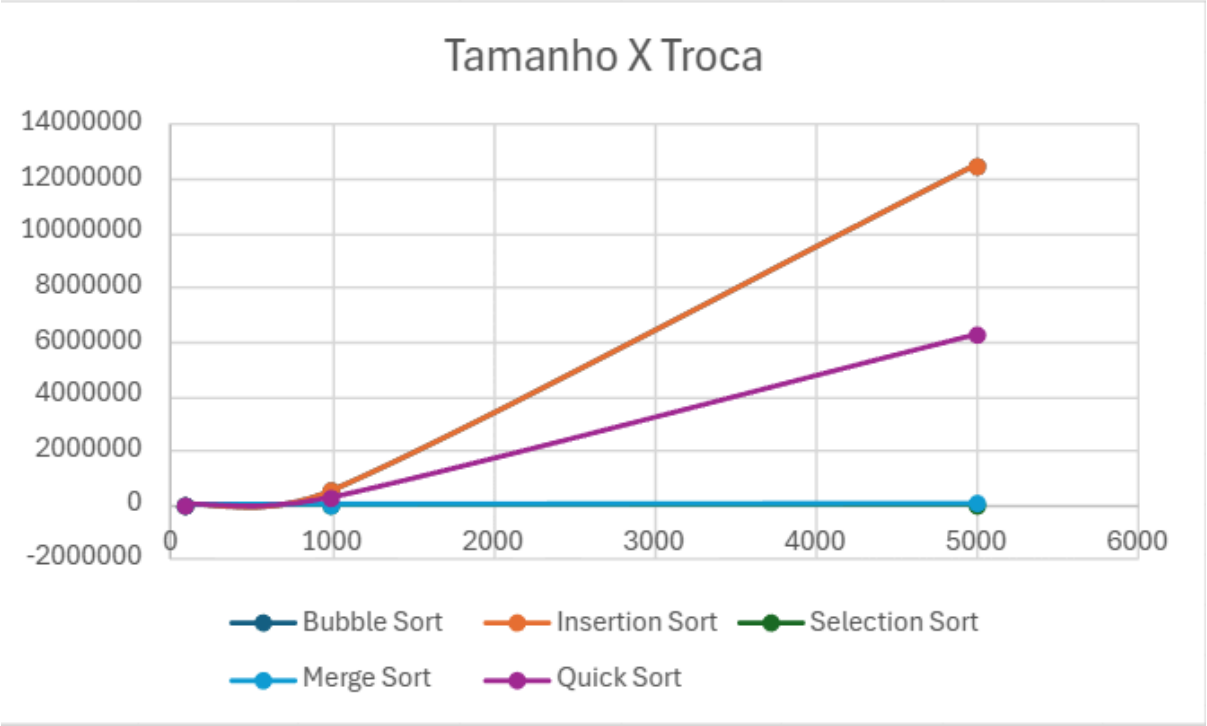
Para dados Quase Ordenados

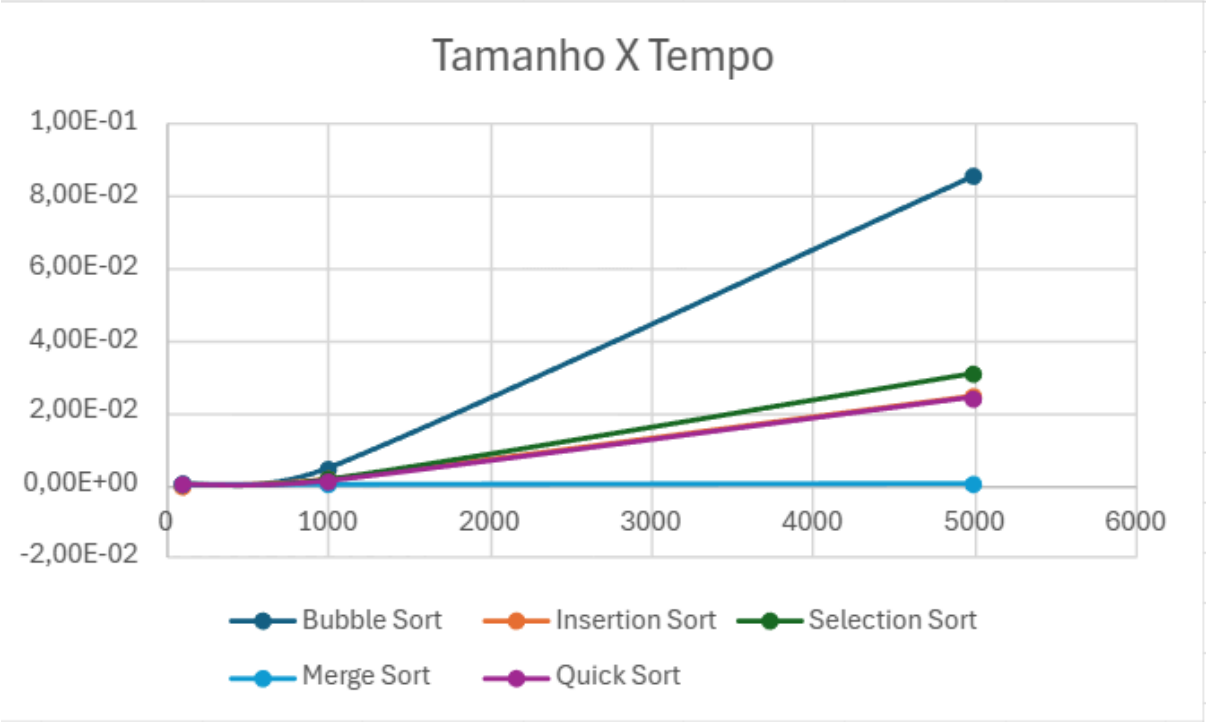


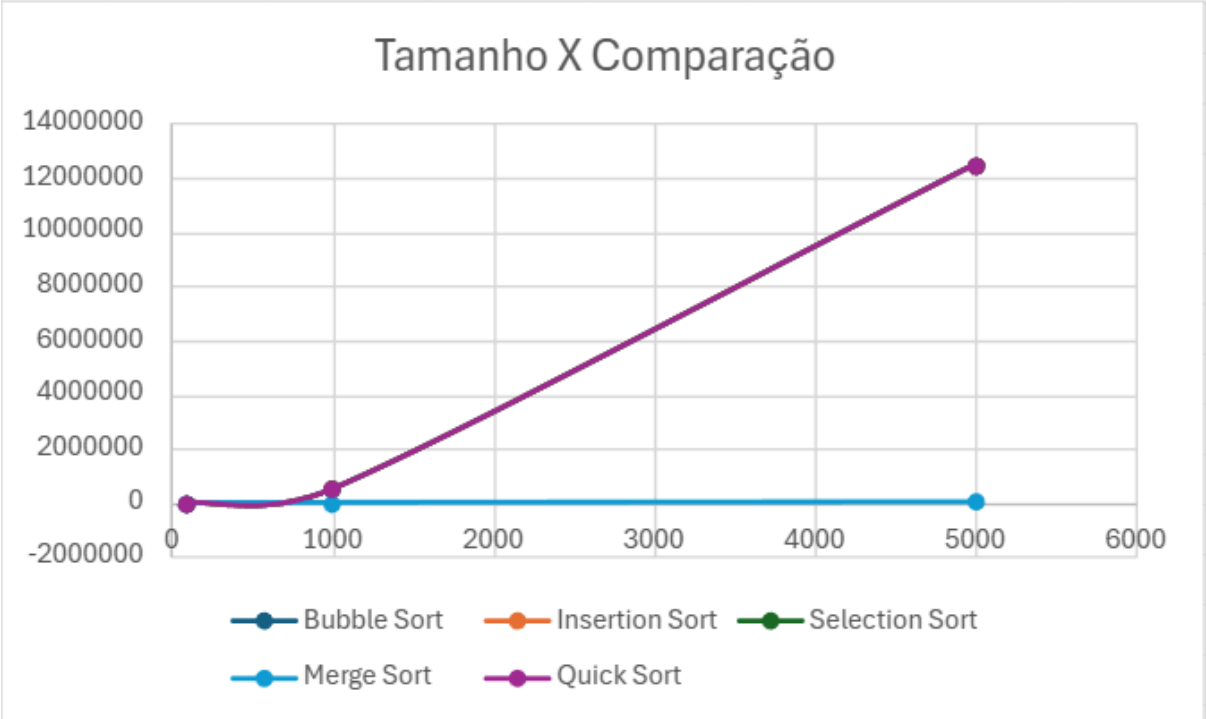

Tamanho X Tempo
4,00E-02
3,50E-02
3,00E-02
2,50E-02
2,00E-02
1,50E-02
1,00E-02
5,00E-03
0,00E+00
-5,00E-03
0
1000
2000
3000
4000
5000
6000
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Quick Sort


Tamanho X Comparação
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
-2000000
0
1000
2000
3000
4000
5000
6000
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Quick Sort

Para dados Inversamente Ordenados







3. Análise

Com todos esses dados em mente podemos começar as análises, estes (dados) afirmam, em grande parte, o comportamento teórico esperado de cada algoritmo:

- Ao longo que o vetor cresce, é notável o desempenho inferior do Bubble Sort, Selection Sort e Insertion Sort. Tendo todos complexidade $O(n^2)$ no pior caso, o que se reflete nos testes com vetores de 1000 e 5000, especialmente com dados do tipo aleatório e inversamente ordenados.
- Insertion Sort foi claramente o mais eficiente entre os algoritmos quadráticos quando o vetor estava quase ordenado. Isso também era previsto, já que seu melhor caso é $O(n)$, exatamente a situação em que ele se destacou.
- Merge Sort e Quick Sort apresentaram desempenho muito superior em todos os cenários, o que confirma suas complexidades de tempo $O(n \log n)$. O tempo de execução desses algoritmos se manteve baixo mesmo para tamanhos maiores (5000).
- O Quick Sort superou o Merge Sort em algumas situações práticas, embora sua complexidade no pior caso seja $O(n^2)$. Isso se explica pelo seu excelente desempenho médio, especialmente em entradas aleatórias.

OBS: É válido ressaltar que o algoritmo Quick Sort foi implementado utilizando o primeiro elemento como pivô. Essa estratégia pode levar ao pior caso em vetores já ordenados ou inversamente ordenados, o que se refletiu nos resultados com maiores tempos de execução e número elevado de comparações nesses cenários.

Além de que a aluna estava à beira de um surto para fazer os outros tipos de exemplos de pivô.

Alguns resultados que chamaram atenção por destoar ligeiramente da teoria:

- O Quick Sort teve número de comparações ou trocas bastante elevado em vetores inversamente ordenados. Justamente por que esses resultados vêm diretamente do tipo de escolha do pivô que foi selecionado para este trabalho.
- Embora o Selection Sort apresente um número muito baixo de trocas em alguns casos, ele não é um algoritmo estável. A estratégia de selecionar o menor elemento e trocá-lo pode alterar a ordem relativa de elementos com valores iguais.
- O tipo de entrada teve grande influência nos algoritmos quadráticos.
- Bubble Sort e Insertion Sort tiveram desempenho muito melhor com vetores quase ordenados.
- Quick Sort teve desempenho mais instável nos inversamente ordenados, mostrando que sua eficiência depende do balanceamento da partição.
- Os algoritmos eficientes (Merge e Quick) foram pouco afetados pelo tipo de entrada, mostrando maior robustez.

Conclusões

1. **Ranking de Desempenho**

Entrada aleatória:

1. Quick Sort 👑
2. Merge Sort
3. Insertion Sort
4. Selection Sort
5. Bubble Sort

Entrada quase ordenada:

1. Insertion Sort 👑
2. Quick Sort
3. Merge Sort
4. Selection Sort
5. Bubble Sort

Entrada inversamente ordenada:

2. Merge Sort 👑
3. Quick Sort
4. Selection Sort
5. Insertion Sort
6. Bubble Sort

2. Aplicações por cenário

- **Para vetores aleatórios**, Quick Sort e Merge Sort possuem um bom desempenho, destacando sua rapidez e constância mesmo em cenários de pior caso.
- **Para vetores quase ordenados**, Insertion Sort prova-se a ser uma ótima opção levando em consideração a sua complexidade de O(n) no melhor caso
- Para vetores inversamente ordenados, O Merge Sort se destaca como melhor opção, possuindo um desempenho estável independente da entrada.
- **Para conjuntos menores/simples**, Insertion Sort se faz simples, leve e eficiente.
- **E para grandes conjuntos grandes/complexos**, Merge ou Quick Sort escalam melhor com 0(n log n), enquanto outros algoritmos se tornam inviáveis com $O(n^2)$.

FIM