TDE 03 - Problemas Estruturados em Computação

Alunos

- Gustavo Silveira e Silva
- Vinicius de Andrade Deolindo

Introdução

Este documento diz respeito ao TDE 03 da disciplina de Resolução de Problemas Estruturados em Computação. A proposta para o projeto iniciava solicitando que a **dupla** escolhesse:

```
- 1 algoritmo do grupo A (Insert Sort, Selection Sort, Comb Sort, Bogo Sort)
- 2 algoritmos do grupo B (Merge Sort , Quick Sort, Shell Sort, Heap Sort)
- 1 algoritmo do grupo C (Radix sort, Gnome Sort, Counting sort, Bucket sort,
Cocktail sort , Timsort, Stooge Sort)
```

Assim, a **dupla** escolheu:

```
- Grupo A - Bogo Sort
- Grupo B - Merge Sort, Shell Sort
- Grupo C - Cocktail Sort
```

Para esse grupo de algoritmos, devem ser realizados **testes com vetores de inteiros**, sendo necessários testes com os tamanhos de **1.000, 10.000, 100.000 e 1.000.000** para, depois de **5 rodadas de teste para cada quantidade**, ser apresentada uma média. Assim, poderemos realizar um **relatório comparativo dos algoritmos**. No relatório, teremos comparações de **tempo de execução, número de iterações e número de trocas**.

Por último, existe a seguinte limitação: "Só será permitida a utilização de vetores, estruturas de nó, tipos primitivos (int, float, boolean), String, estruturas de matrizes (não a função pronta), random, seeds, formas para ler os dados como buffer e scanner e bibliotecas de representação visual e exportação dos dados para análise e construção dos gráficos. Os gráficos podem ser feitos em qualquer ferramenta."

Para o Bogo Sort, a equipe optou por incluir apenas testes para vetores de até 50 números. Por consistir de tentativas aleatórias de sequências, seria inviável esperar pelo resultado com um milhão de elementos. Para compensar, incluímos o algoritmo de Insertion Sort. Também é útil para comparação com o Shell Sort, que usa de um Insertion Sort em uma das etapas.

Quanto ao Cocktail Sort, foi perceptível a má performance, por isso optamos por incluir o Bubble Sort para comparação. Já que o Cocktail Sort é apenas um Bubble Sort bidirecional, a comparação é interessante.

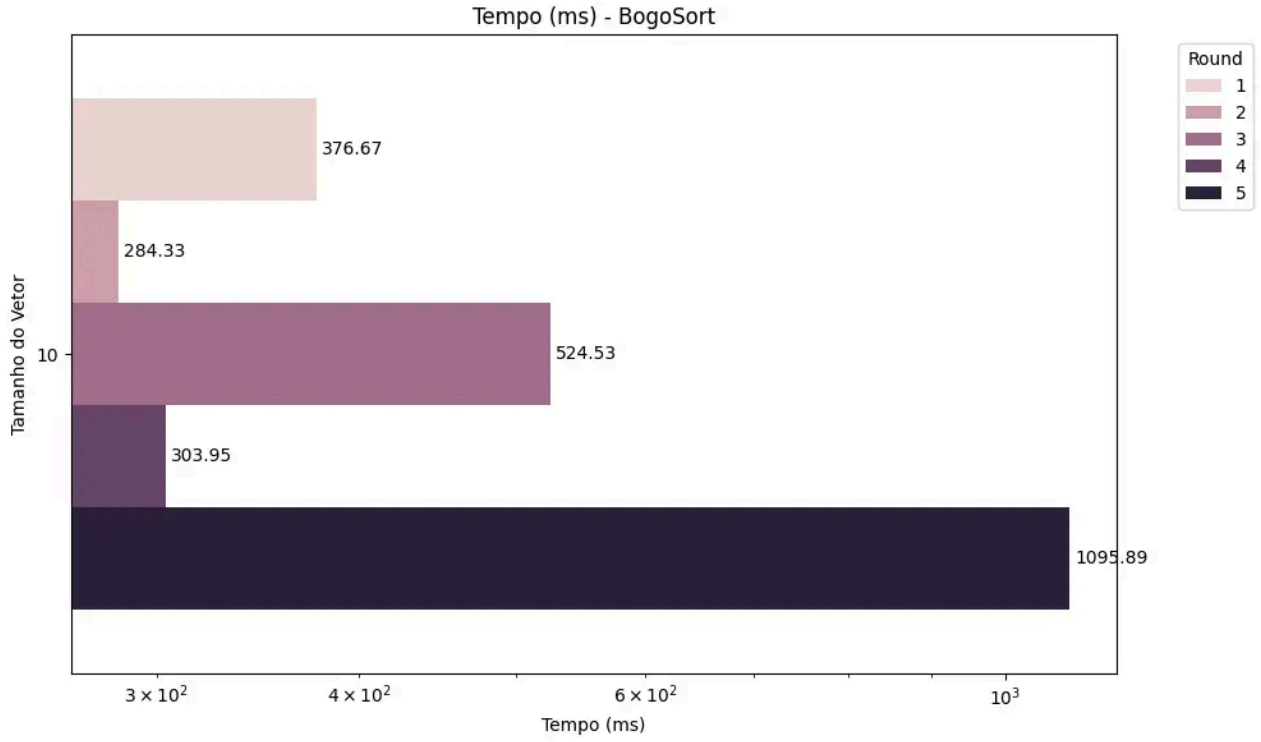
A1. Bogo Sort

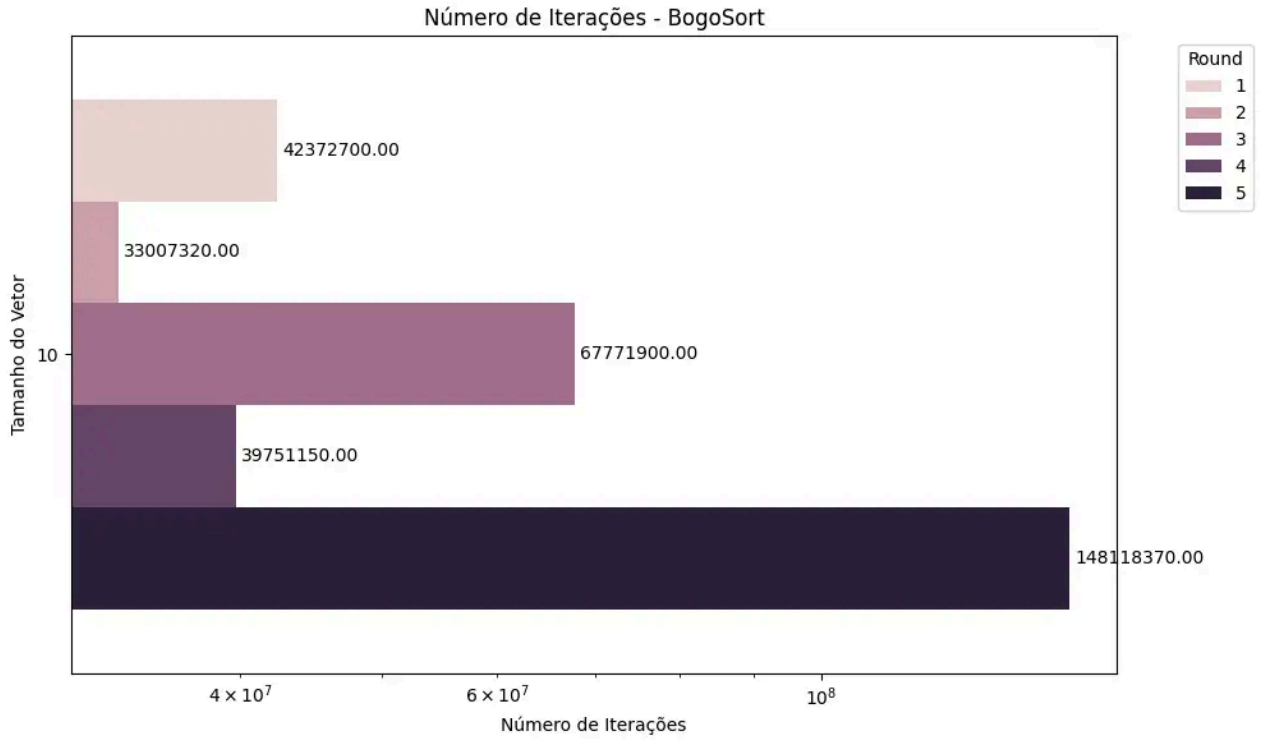
A1.1 Explicação

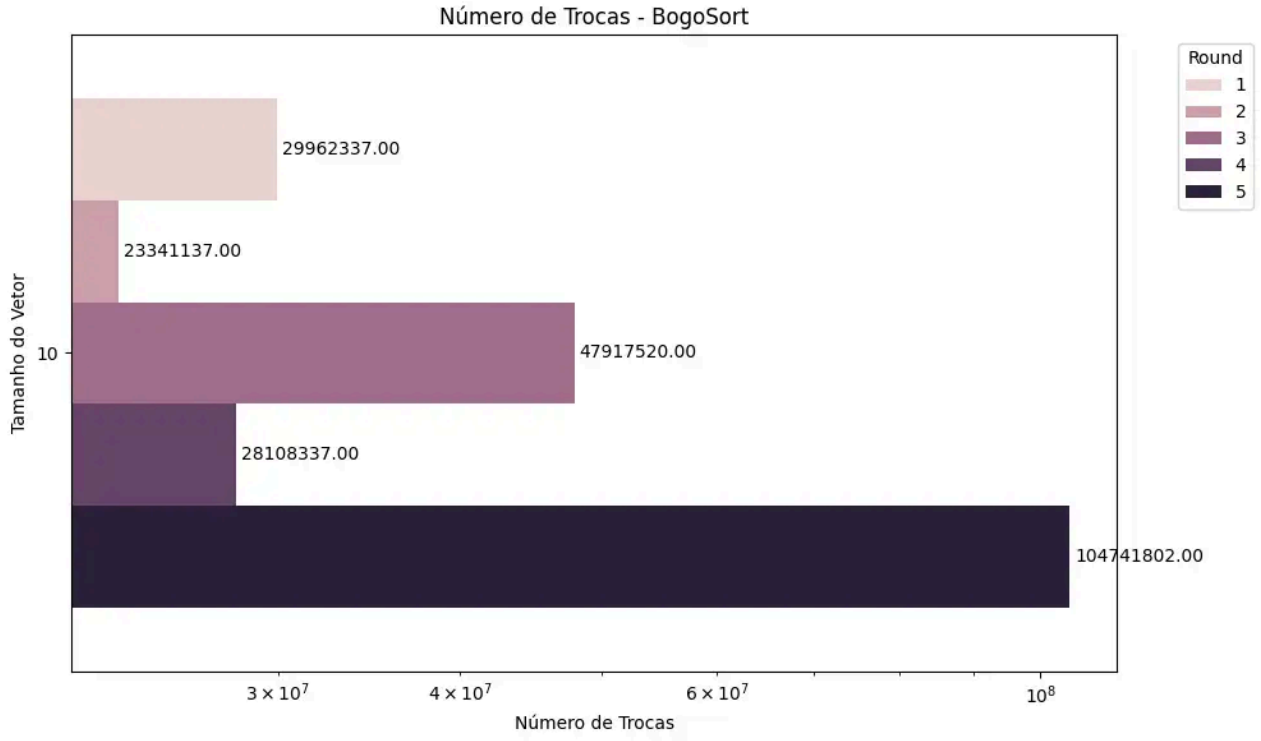
O Bogo Sort é um algoritmo de ordenação extremamente ineficiente. Seu funcionamento básico está em permutações aleatórias de um vetor até que ele esteja ordenado. Após cada permutação, o algoritmo precisa verificar a ordenação. É interessante notar que, dado um número ilimitado de tentativas, o algoritmo acertar a ordem na primeira permutação.

A1.2 Resultados

Por sua natureza imprevisível, o Bogo Sort se mostrou inviável para testes. O que decidimos fazer foi apenas incluir um teste com 10 elementos, demonstrando a instabilidade, com 5 rodadas com resultados completamente diferentes. O mesmo aconteceu para número de iterações e de trocas. Por já apresentar resultados extremamente demorados e impervisíveis, outros testes deixaram esse algoritmo de fora.







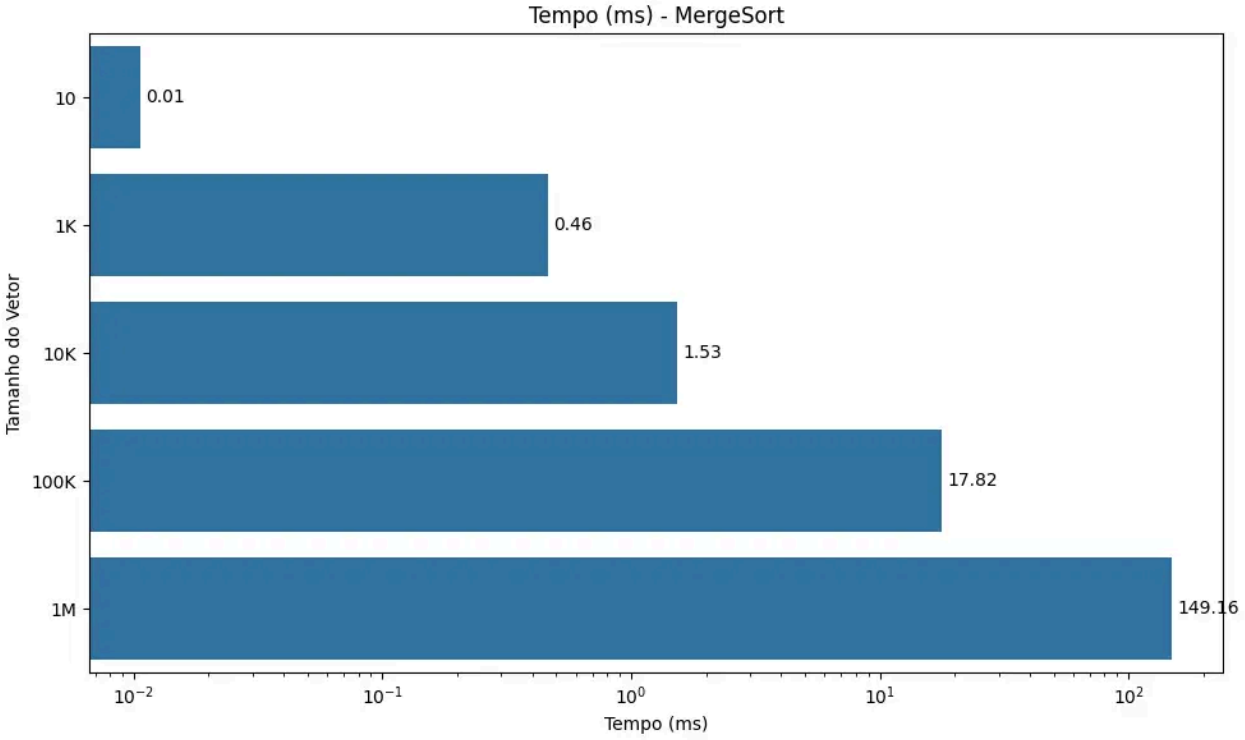
B1 Merge Sort

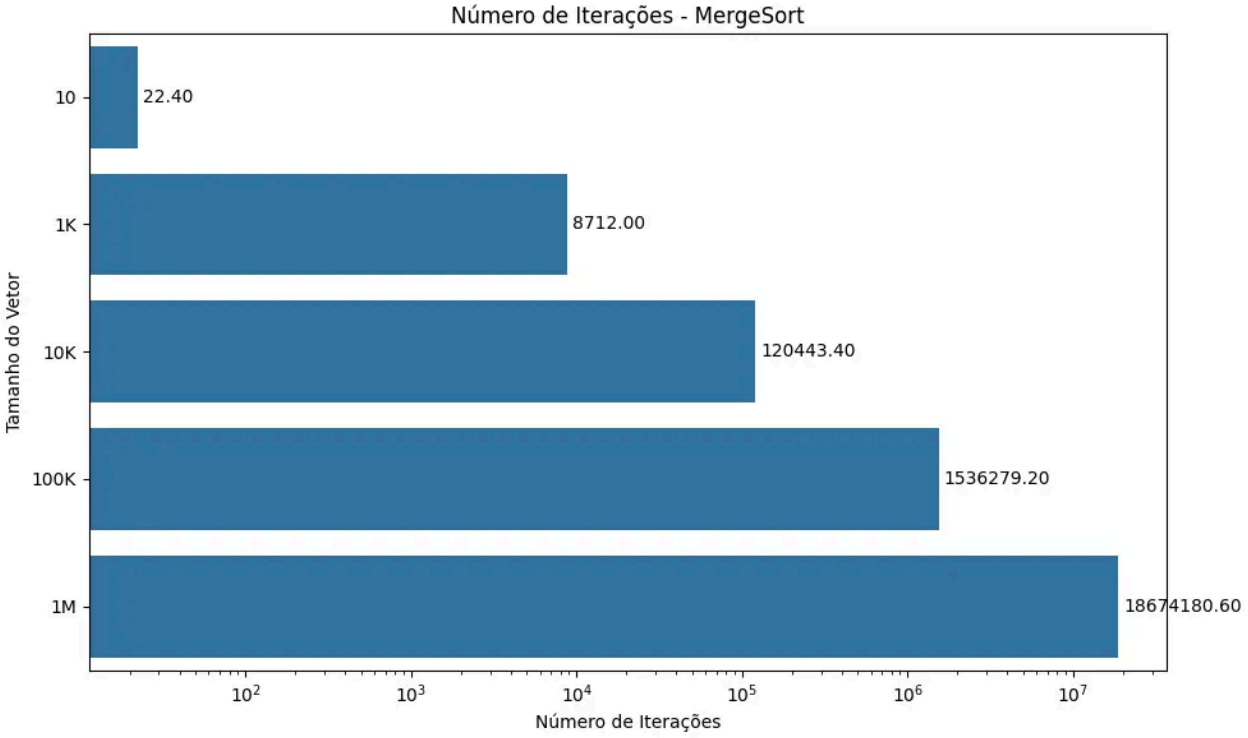
B1.1 Explicação

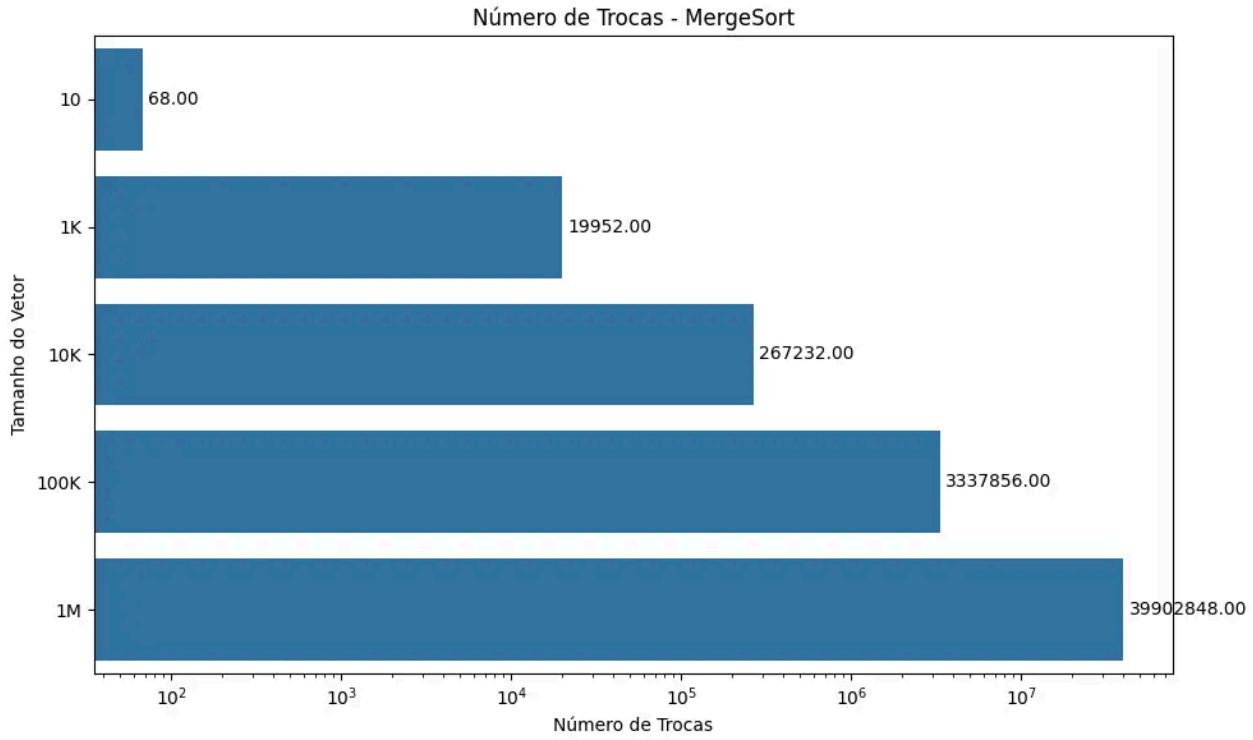
O Merge Sort é um algoritmo de ordenação eficiente. Seu funcionamento básico está em dividir o vetor na metade recursivamente até ter arrays unitários, que são naturalmente ordenados. Em seguida, o algoritmo combina os "sub-vetores" de forma ordenada (por isso o nome "merge"), construindo vetores maiores, até que todo o vetor original esteja ordenado. Essa estratégia é comum, e normalmente apelidada de divisão e conquista.

B1.2 Resultados

O Merge Sort apresentou ótimos resultados consistentemente







B2 Shell Sort

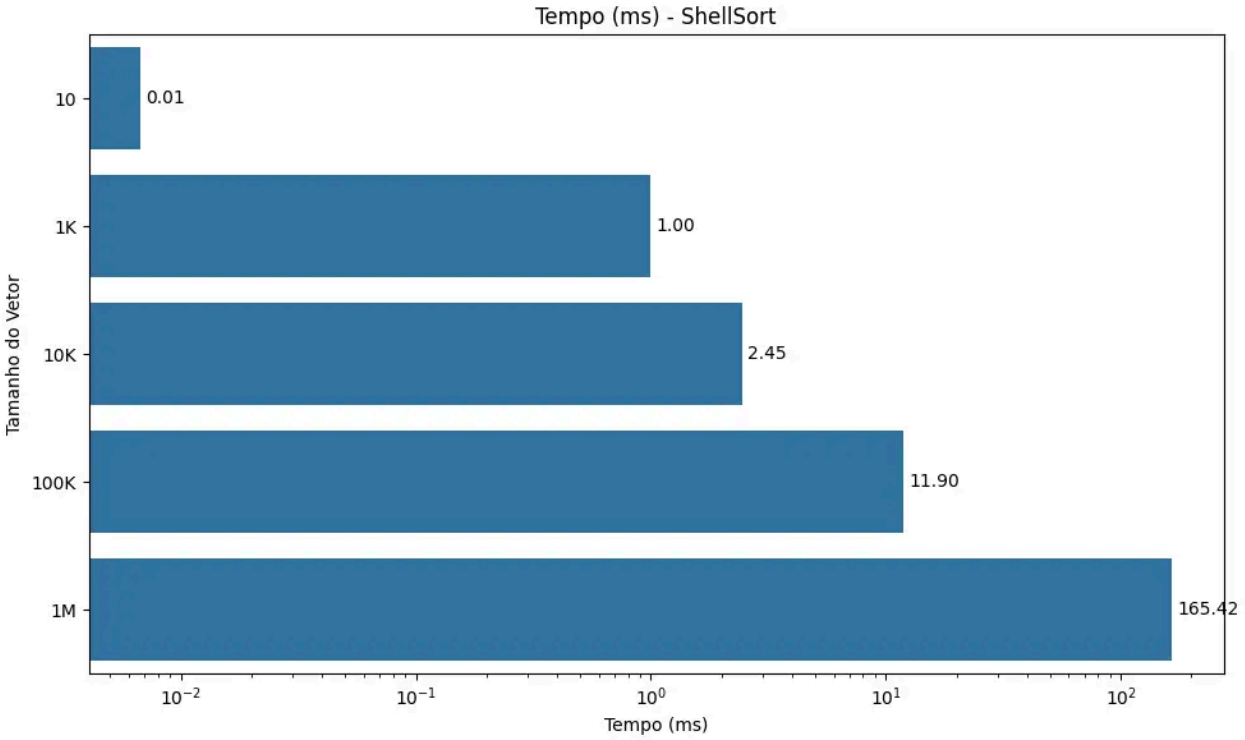
B2.1 Explicação

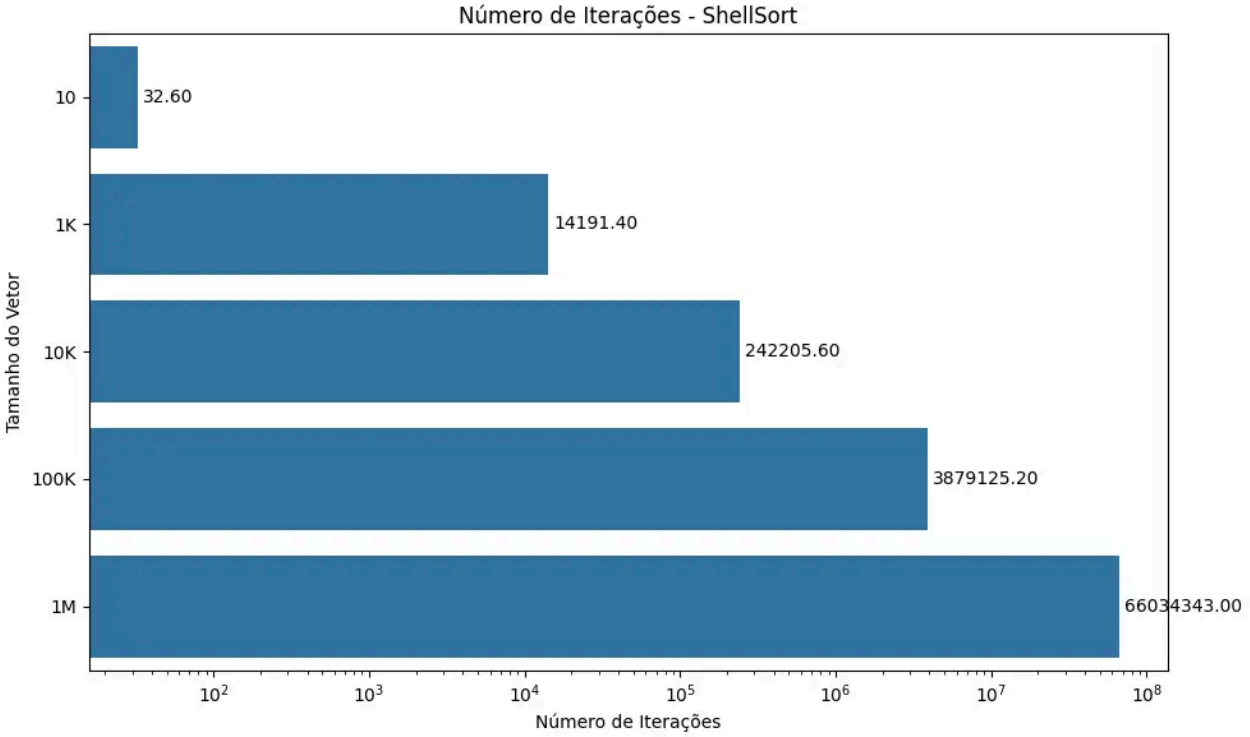
O Shell Sort usa como base o Insertion Sort. O algoritmo funciona dividindo o vetor em grupos menores, que são ordenados usando o Insertion Sort. Inicialmente, os elementos distantes são comparados e trocados se necessário. O tamanho entre comparações é gradualmente reduzido até chegar a 1, onde o algoritmo se comporta como um Insertion Sort comum. O interessante é que podem ser usadas várias sequências de distâncias diferentes. Uma das sequências mais comuns é a de autoria de Donald Knuth, definida por:

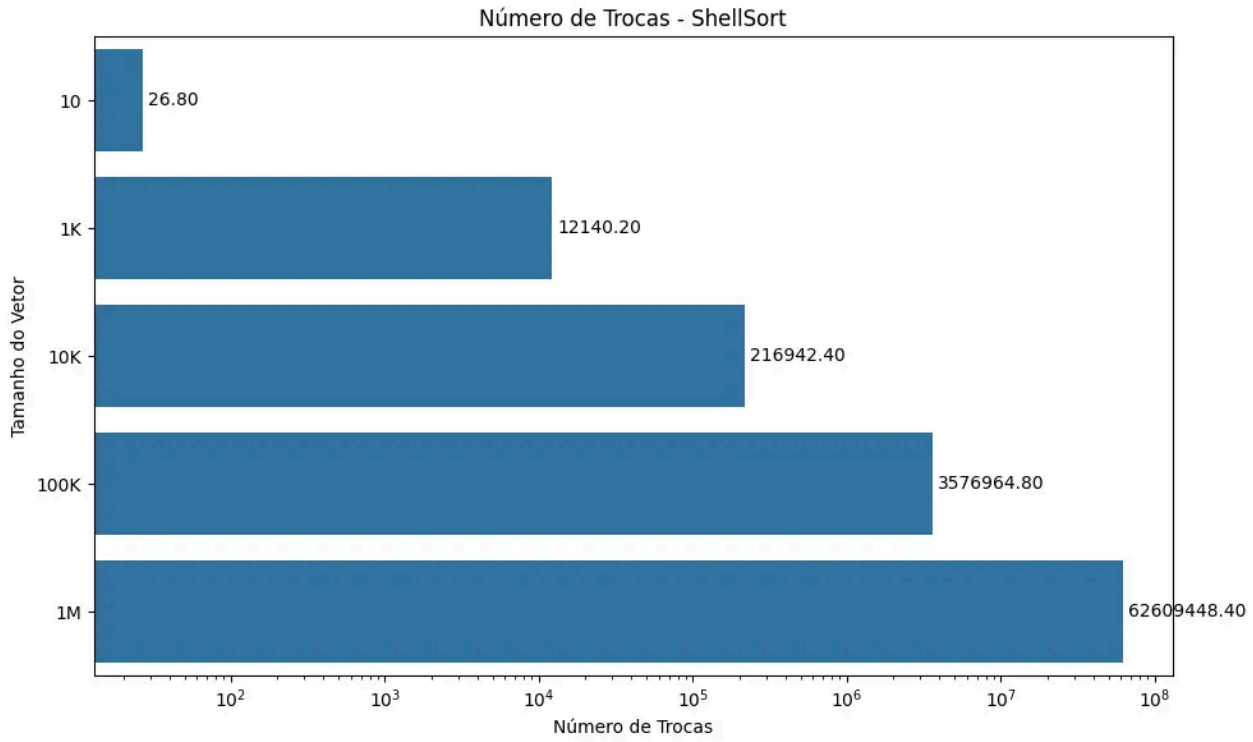
$$\frac{3^k - 1}{2}\text{, não maior que } [\frac{N}{3}]$$

B2.2 Resultados

O Shell Sort também apresentou resultados ótimos, sendo até melhores que o Merge Sort para certas quantidades.


Tempo (ms) - ShellSort
Tamanho do Vetor
10
1K
10K
100K
1M
0.01
1.00
2.45
11.90
165.42
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Tempo (ms)


Número de Iterações - ShellSort
Tamanho do Vetor
10
1K
10K
100K
1M
32.60
14191.40
242205.60
3879125.20
66034343.00
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}
10^{8}
Número de Iterações



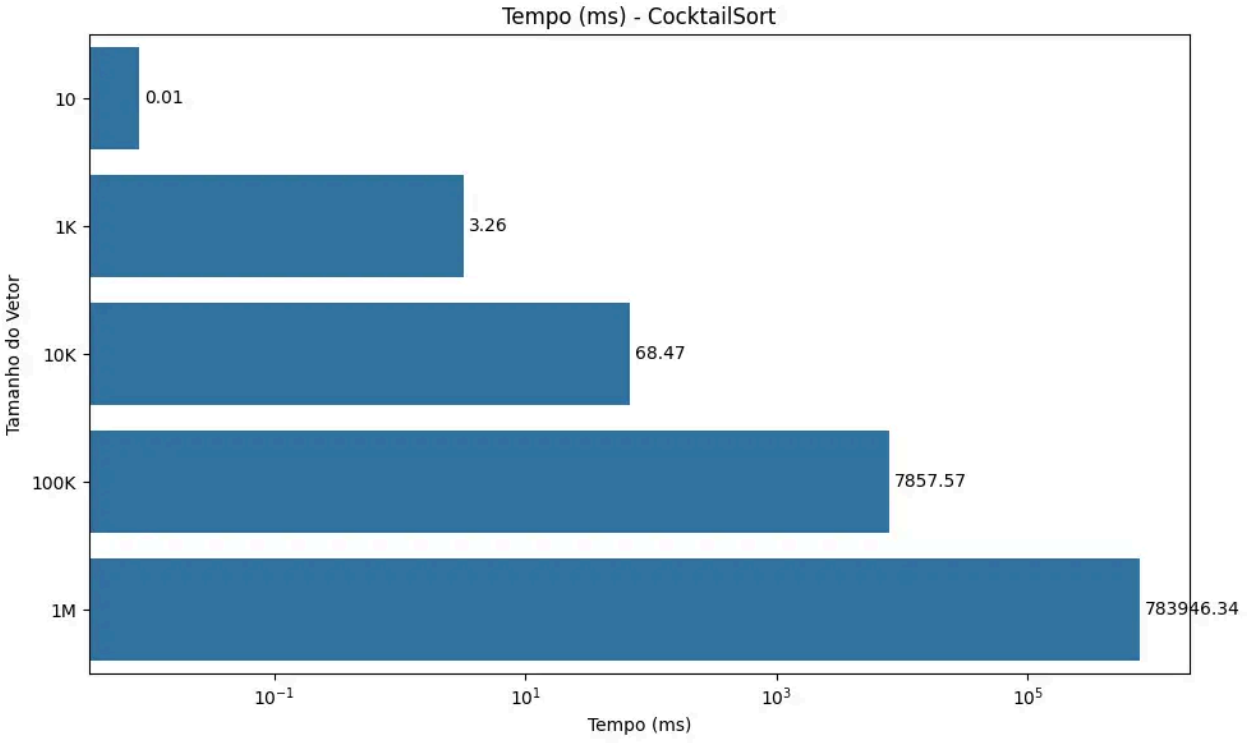
C1 Cocktail Sort

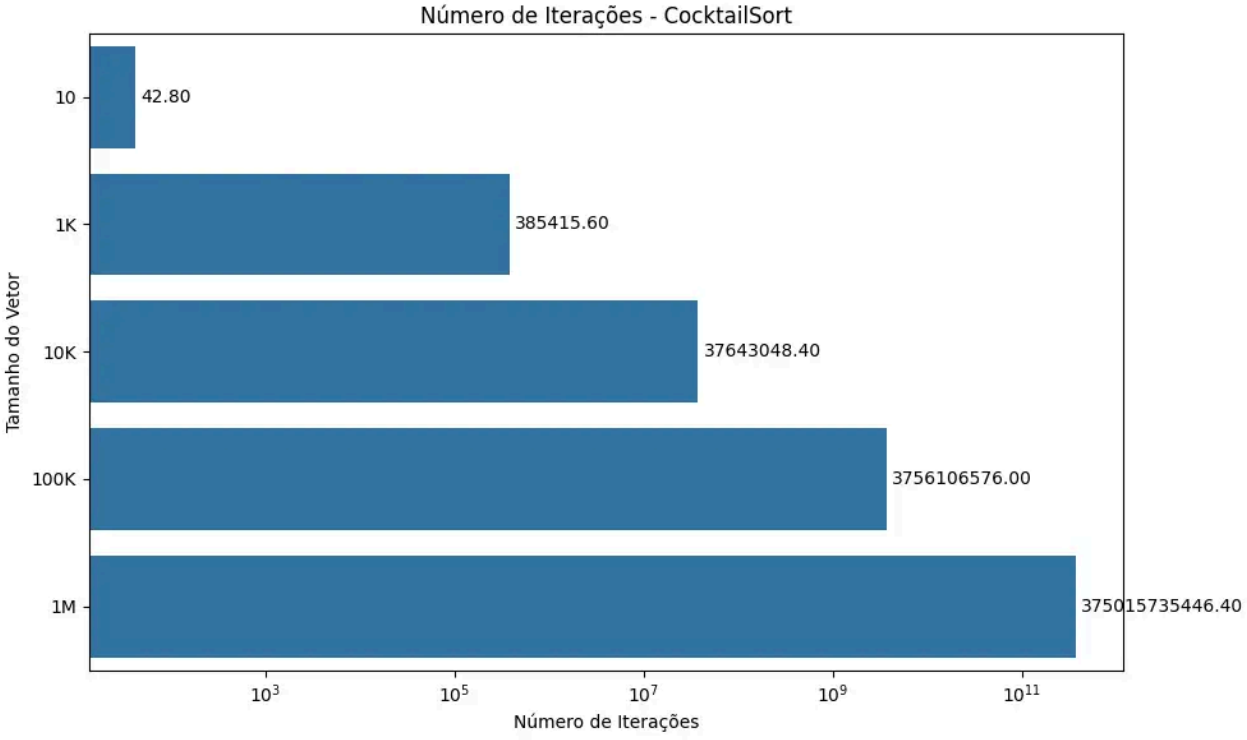
C1.1 Explicação

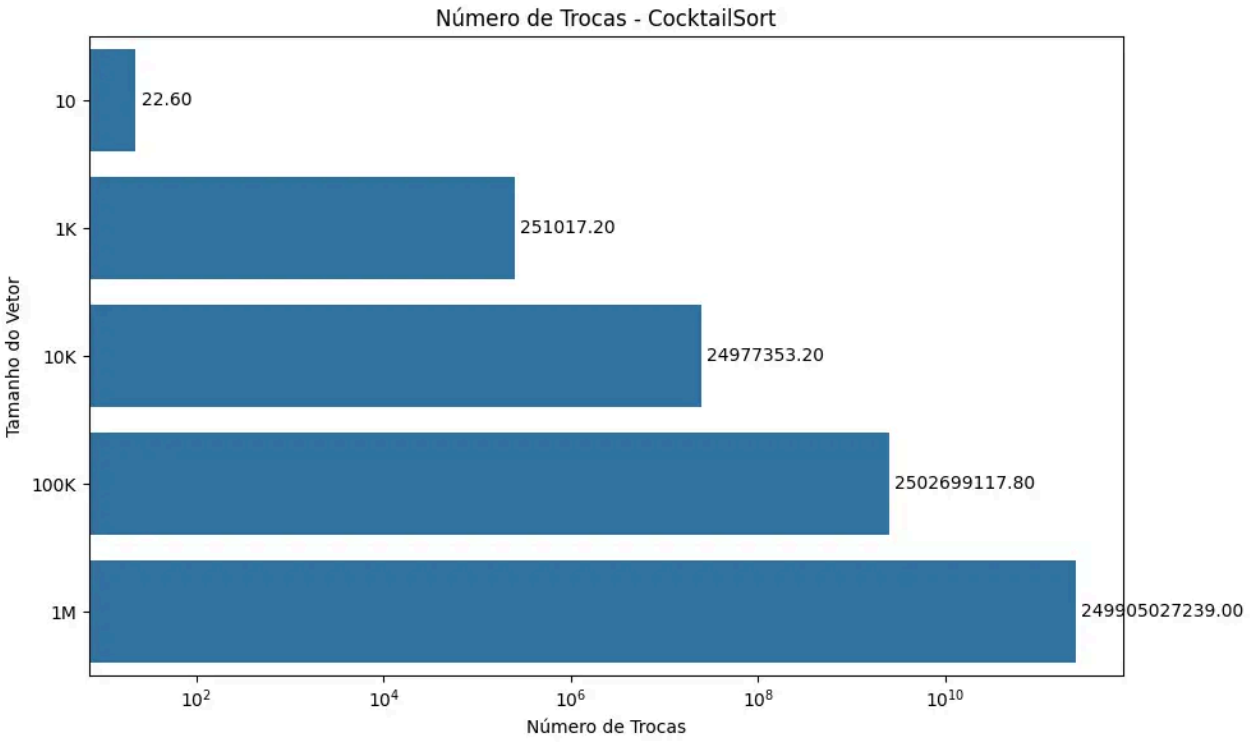
O Cocktail Sort, também conhecido como Bidirectional Bubble Sort, é uma variação do Bubble Sort. Ele funciona por percorrer o vetor em ambas as direções em cada iteração. Por ir passando os elementos maiores para o fim e menores para o começo em cada iteração, pode fazer menos comparações, evitando os primeiros e últimos elementos. Assim, possui performance superior ao Bubble Sort convencional.

C1.2 Resultados

Já o Cocktail Sort apresentou uma performance ruim, com os resultados de 100k tendo próximo dos 8 segundos de execução, e 1 milhão, 12 minutos.


Tempo (ms) - CocktailSort
Tamanho do Vetor
10
1K
10K
100K
1M
0.01
3.26
68.47
7857.57
783946.34
10^{-1}
10^{1}
10^{3}
10^{5}
Tempo (ms)


Número de Iterações - CocktailSort
Tamanho do Vetor
10
1K
10K
100K
1M
42.80
385415.60
37643048.40
3756106576.00
375015735446.40
10^{3}
10^{5}
10^{7}
10^{9}
10^{11}
Número de Iterações



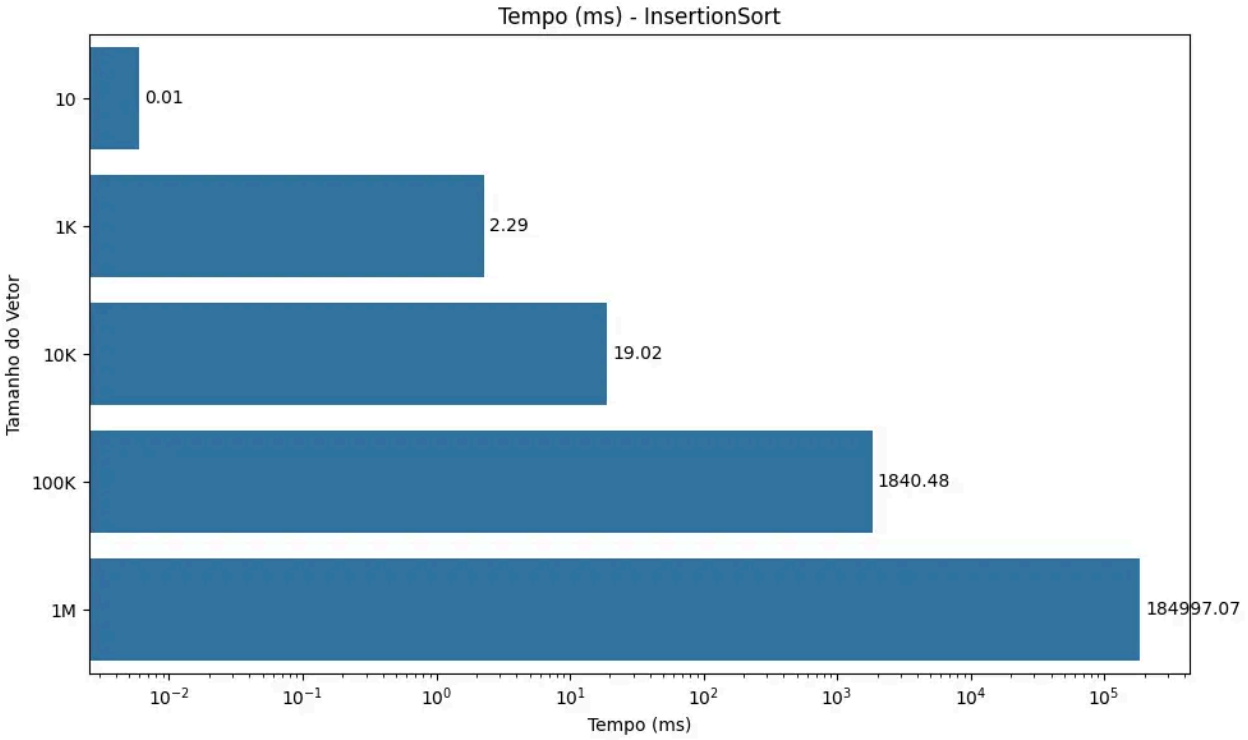
A2 Insertion Sort

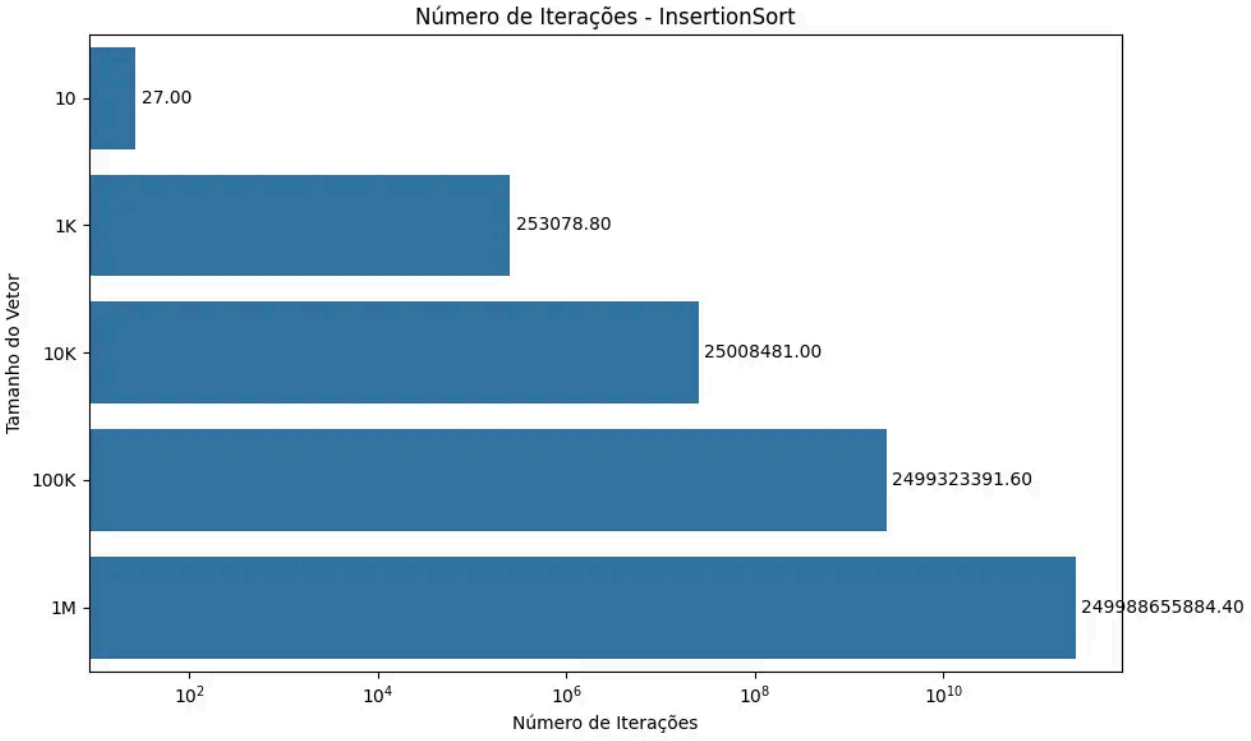
A2.1 Explicação

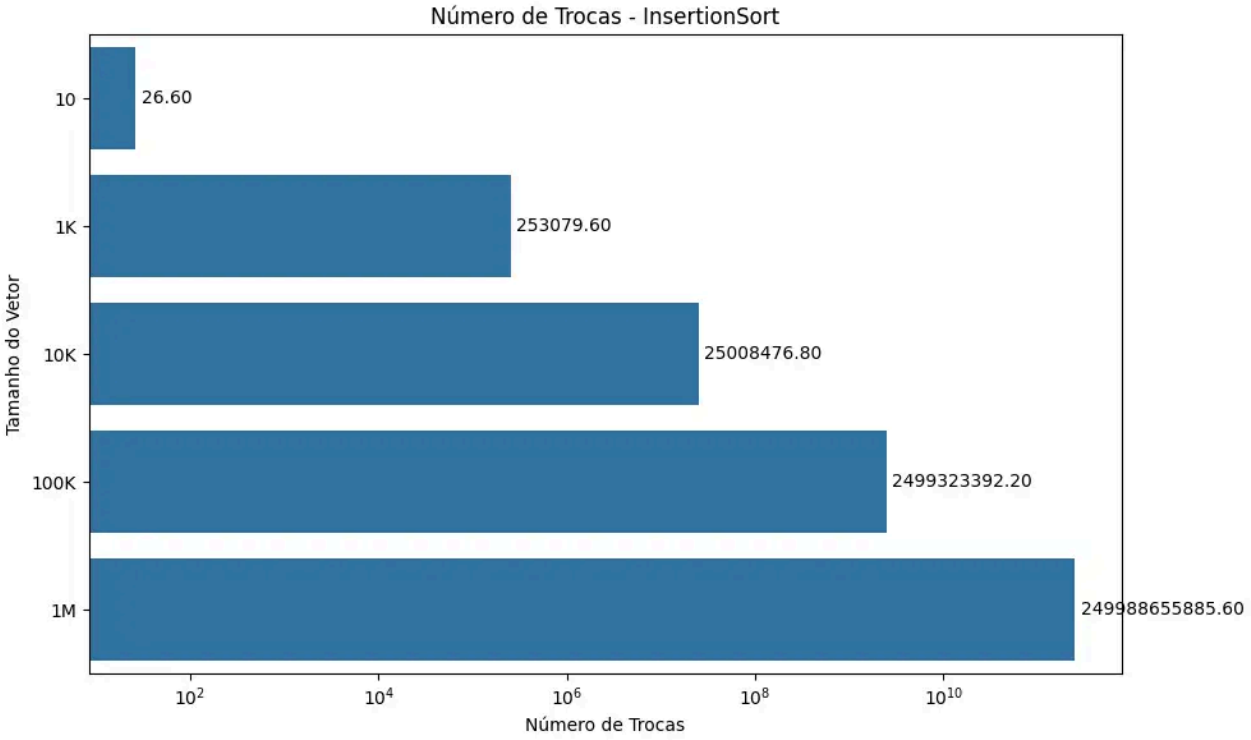
O Insertion Sort é um algoritmo simples que funciona construindo o vetor ordenado um elemento por vez, sempre mantendo a porção já verificada do vetor em ordem. Para cada iteração, um elemento é comparado aos anteriores, já ordenados. Em seguida, move os maiores uma posição para frente, e insere o elemento naquela posição.

A2.1 Resultados

O Insertion Sort apresentou resultados relativamente bons, comparado à outros implementados. Dado sua simplicidade, é bastante positivo


Tempo (ms) - InsertionSort
Tamanho do Vetor
10
1K
10K
100K
1M
0.01
2.29
19.02
1840.48
184997.07
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
Tempo (ms)


Número de Iterações - InsertionSort
Tamanho do Vetor
10
1K
10K
100K
1M
27.00
253078.80
25008481.00
2499323391.60
249988655884.40
10^{2}
10^{4}
10^{6}
10^{8}
10^{10}
Número de Iterações



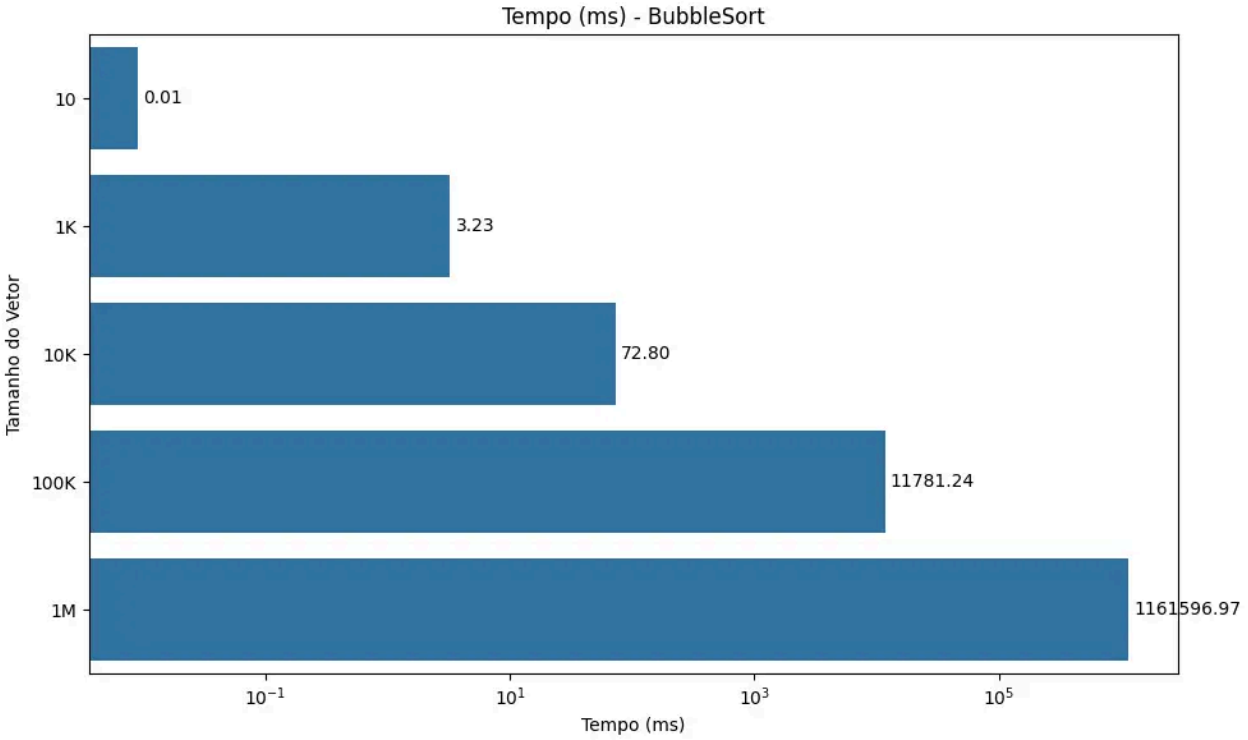
Bubble Sort

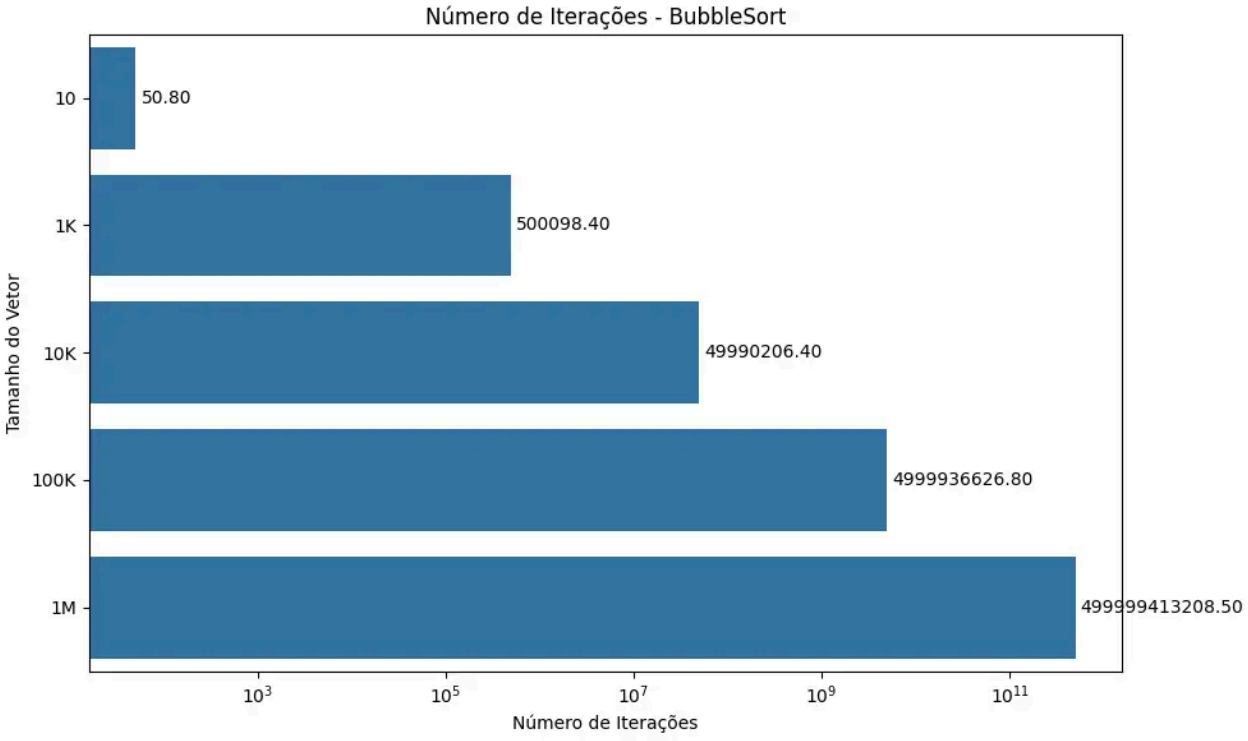
Explicação

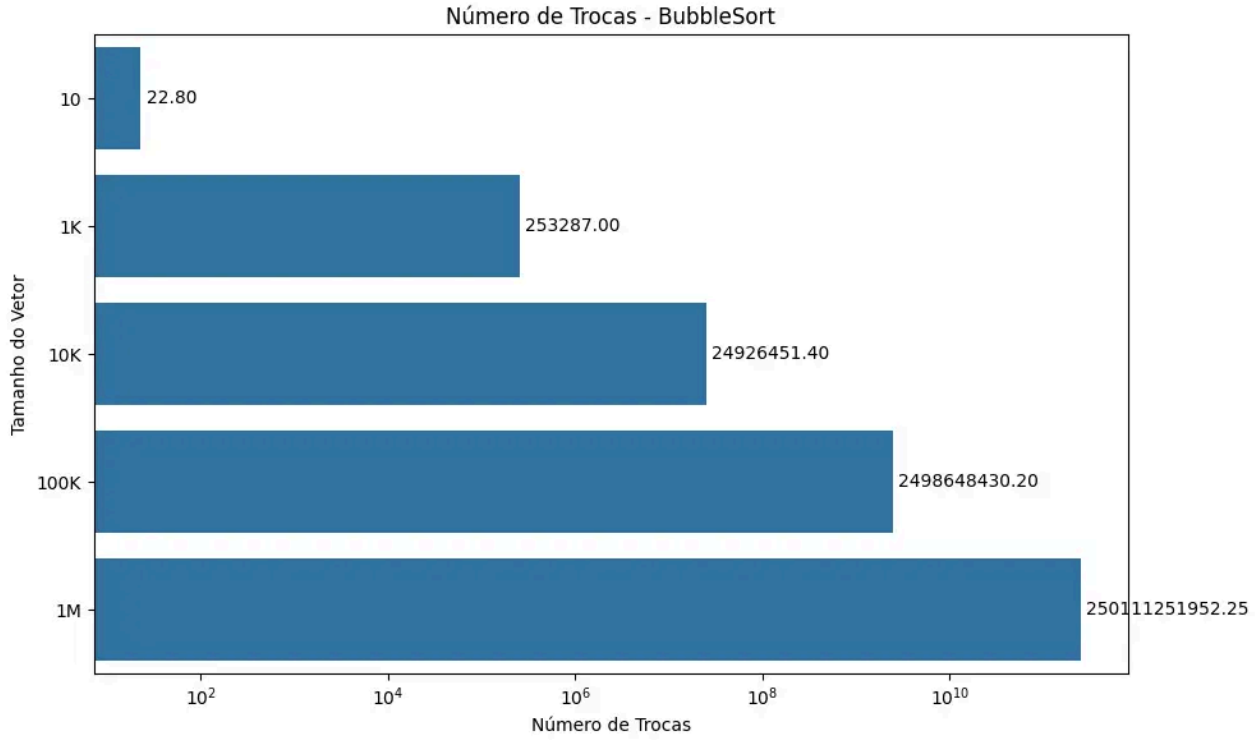
O Bubble Sort é um algoritmo que faz com que os maiores valores "borbulhem" gradualmente para o final do vetor. A cada iteração, compara pares de elementos consecutivos e os troca se estiverem fora de ordem. O processo é repetido até que o maior elemento esteja no final. Quando não forem mais necessárias trocas, o vetor está ordenado.

Resultados

Dos algoritmos implementados pela equipe, apresentou a pior performance, com 100k levando 12 segundos e 1 milhão, 20 minutos

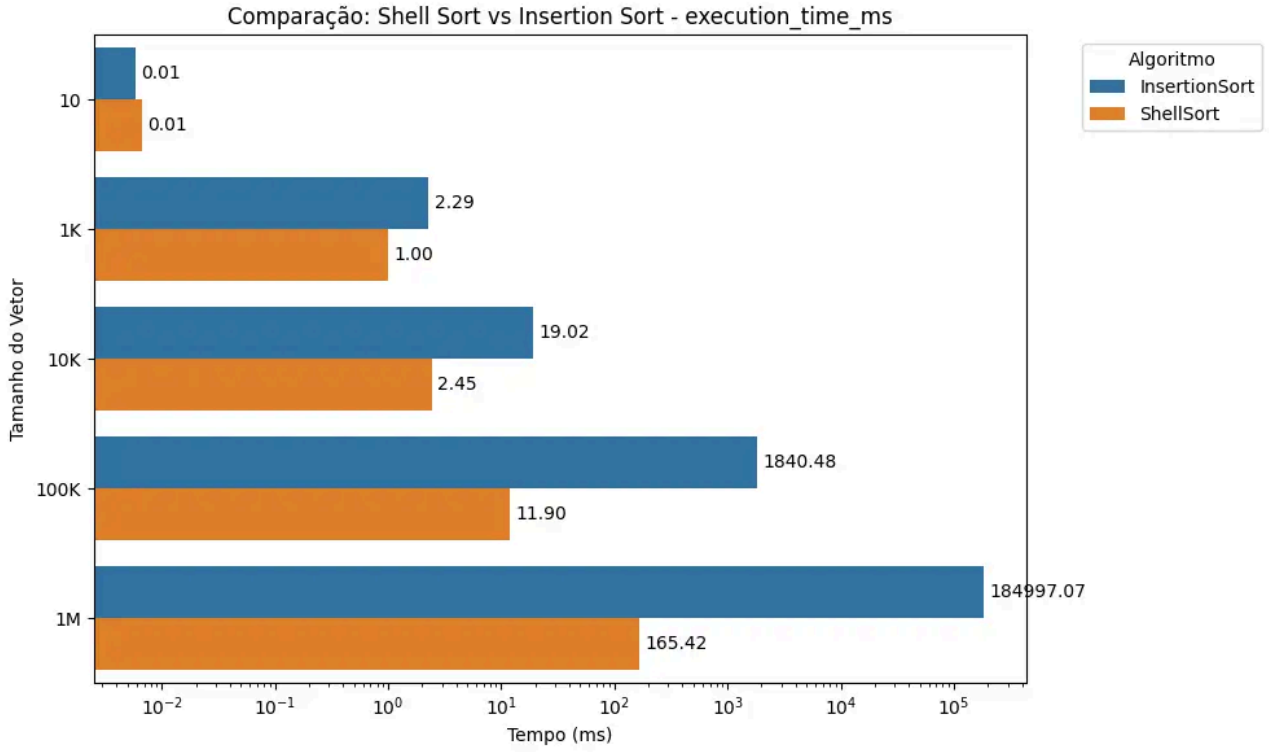

Tempo (ms) - BubbleSort
Tamanho do Vetor
10
1K
10K
100K
1M
0.01
3.23
72.80
11781.24
1161596.97
10^{-1}
10^{1}
10^{3}
10^{5}
Tempo (ms)

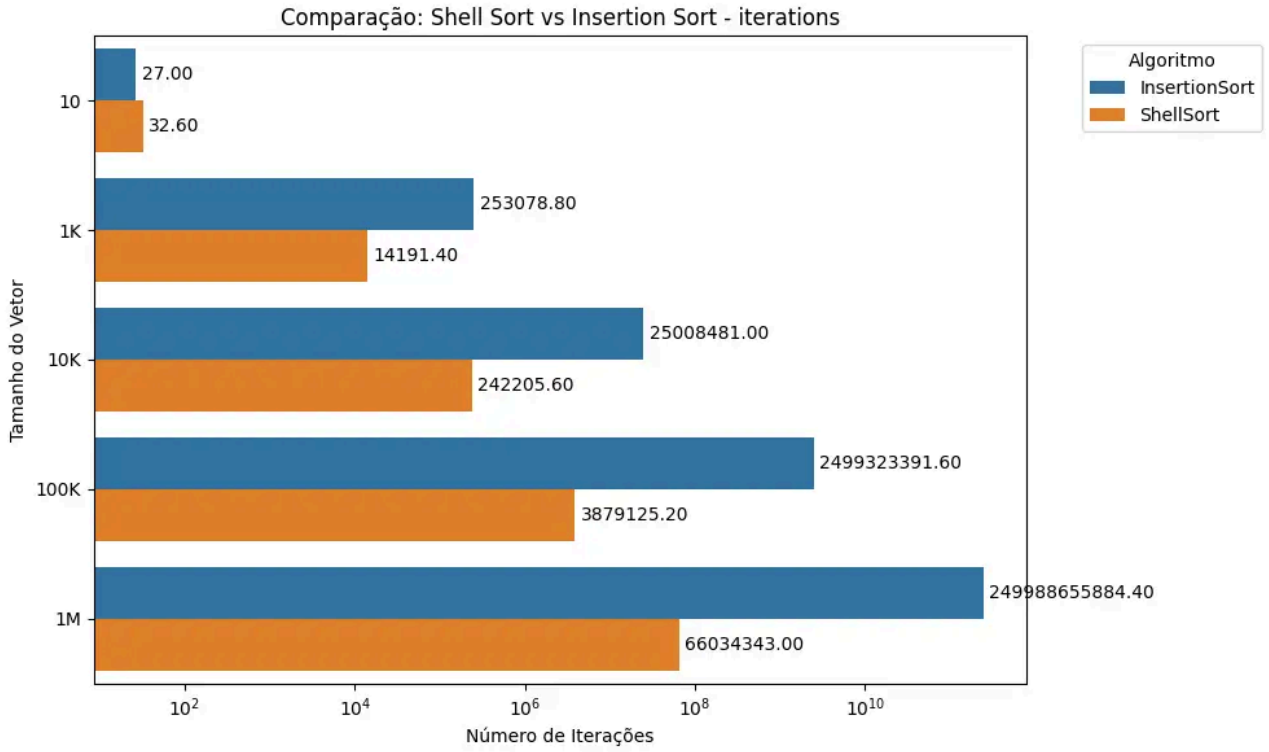

Número de Iterações - BubbleSort
Tamanho do Vetor
10
1K
10K
100K
1M
50.80
500098.40
49990206.40
4999936626.80
499999413208.50
10^{3}
10^{5}
10^{7}
10^{9}
10^{11}
Número de Iterações

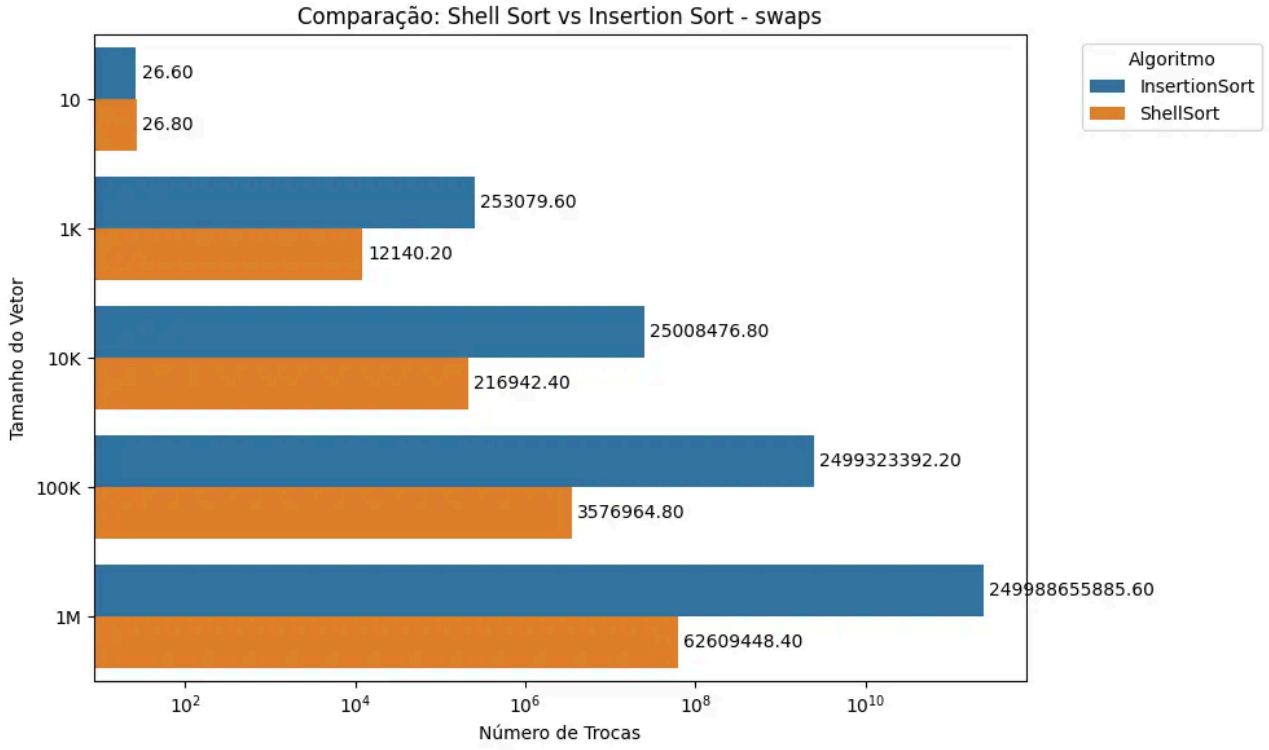


Comparativos Finais

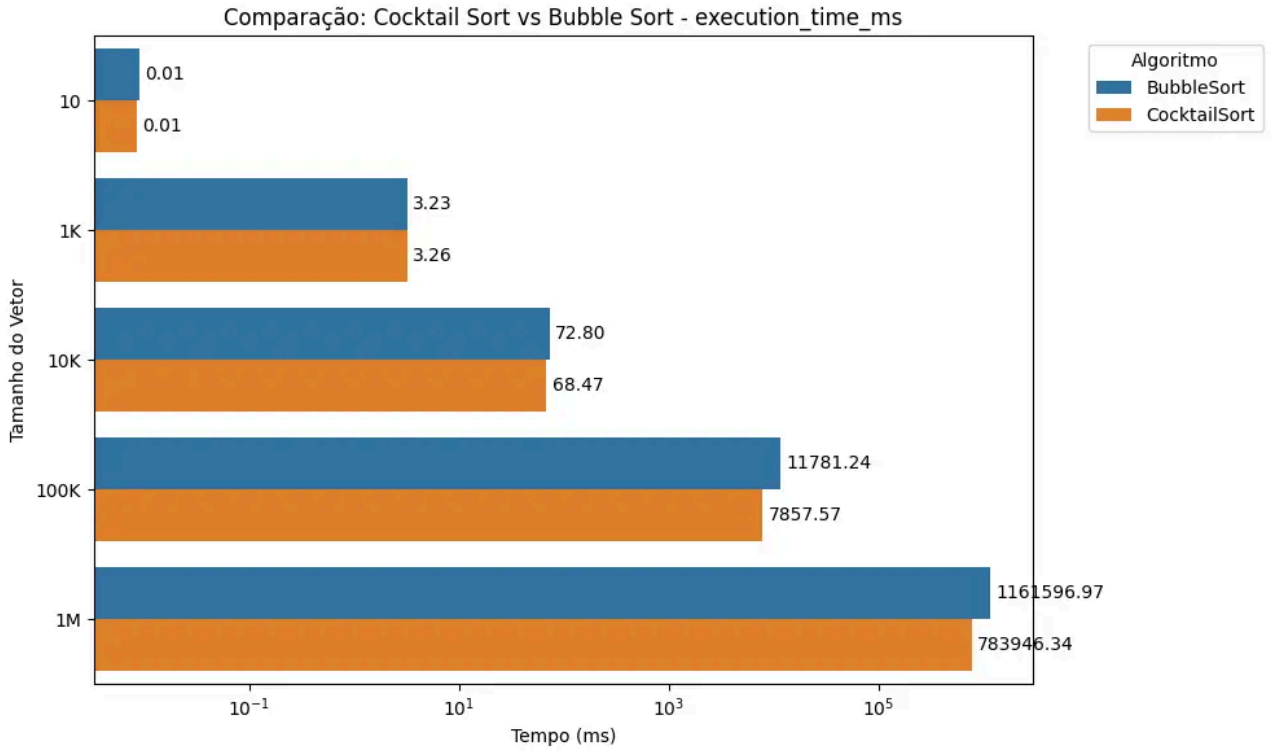
Pela similaridade dos algoritmos Shell e Insertion Sort, fizemos também uma comparação entre os dois:

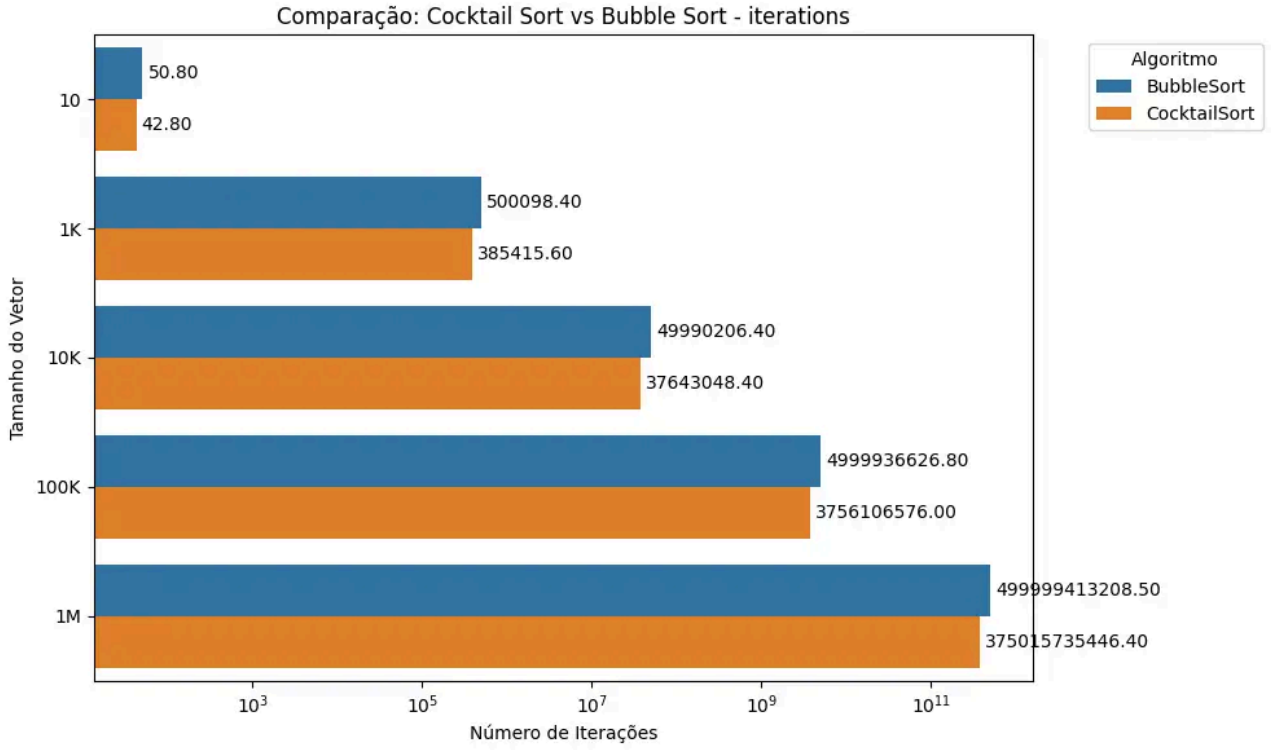


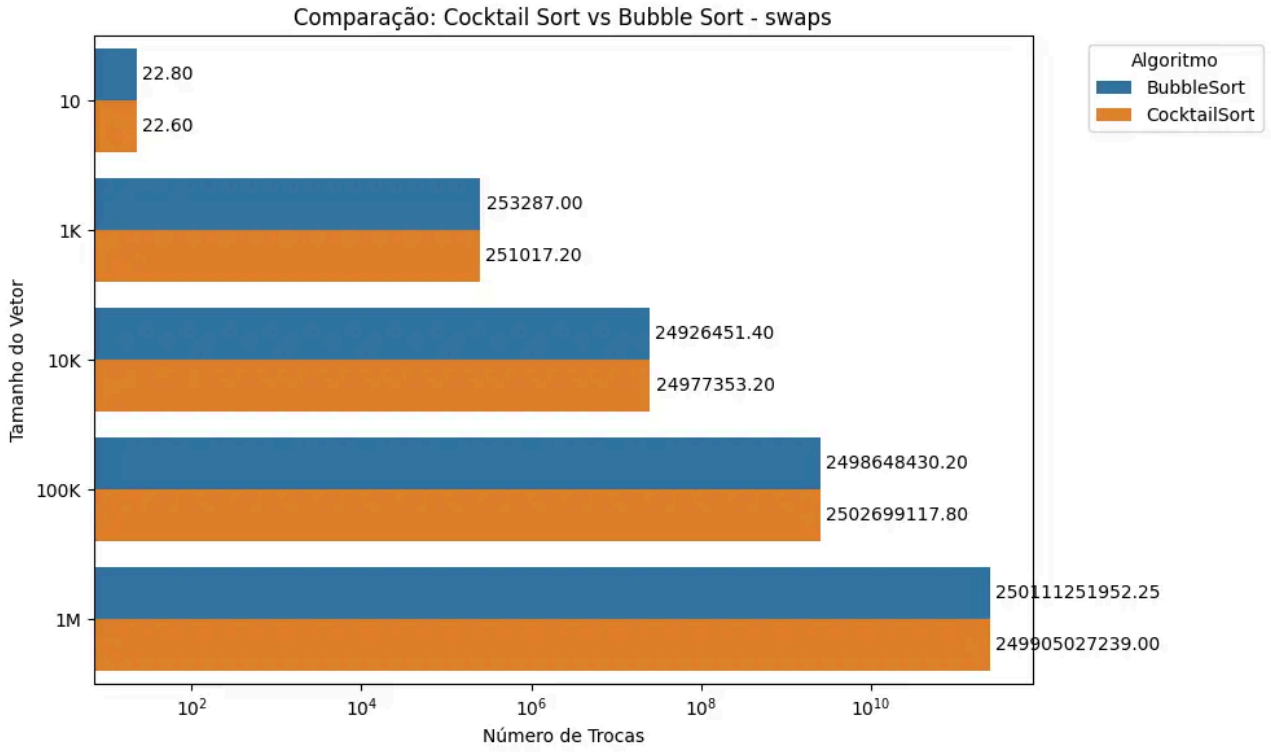




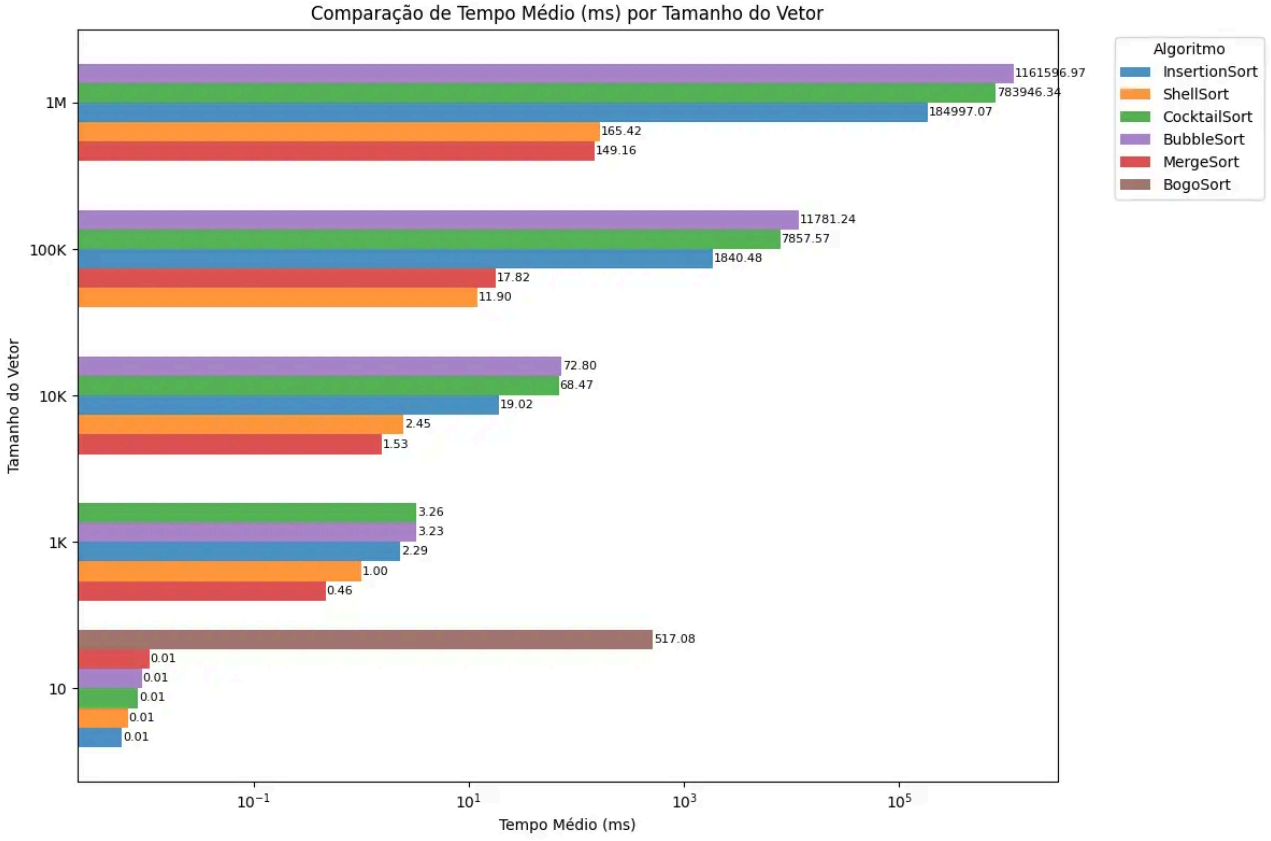
Igualmente, comparamos o Cocktail e Bubble Sort:

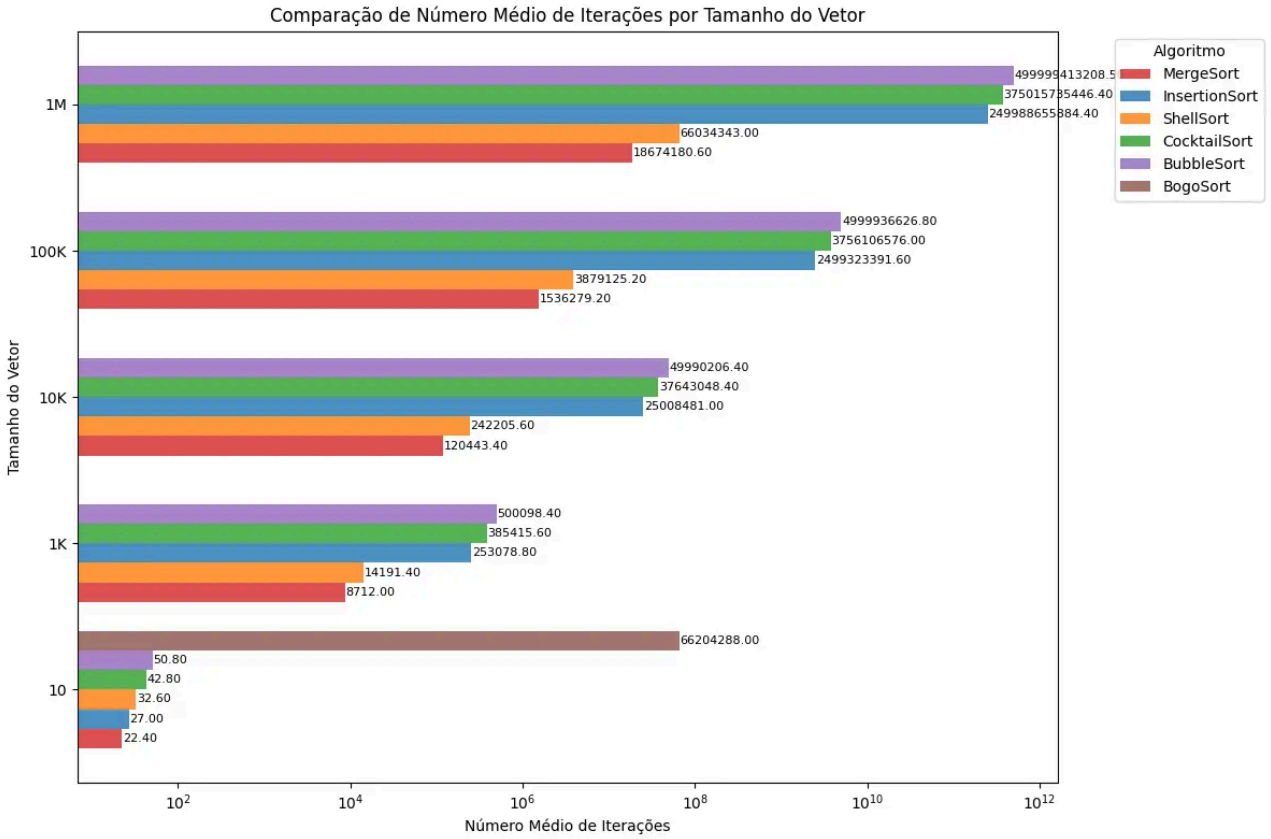

Comparação: Cocktail Sort vs Bubble Sort - execution_time_ms
Algoritmo
BubbleSort
CocktailSort
Tamanho do Vetor
10
1K
10K
100K
1M
0.01
0.01
3.23
3.26
72.80
68.47
11781.24
7857.57
1161596.97
783946.34
10^{-1}
10^{1}
10^{3}
10^{5}
Tempo (ms)

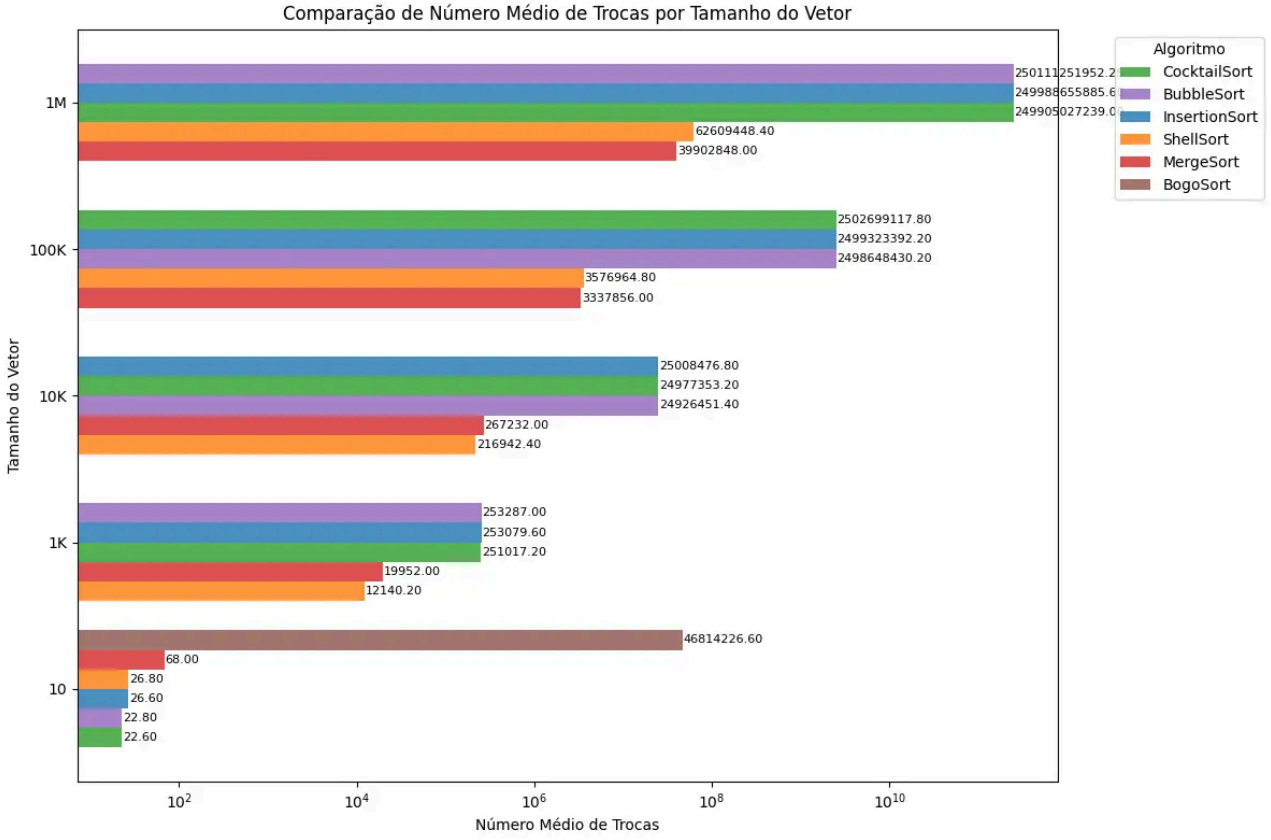

Comparação: Cocktail Sort vs Bubble Sort - iterations
Algoritmo
BubbleSort
CocktailSort
Tamanho do Vetor
10
1K
10K
100K
1M
50.80
42.80
500098.40
385415.60
49990206.40
37643048.40
4999936626.80
3756106576.00
499999413208.50
375015735446.40
10^{3}
10^{5}
10^{7}
10^{9}
10^{11}
Número de Iterações



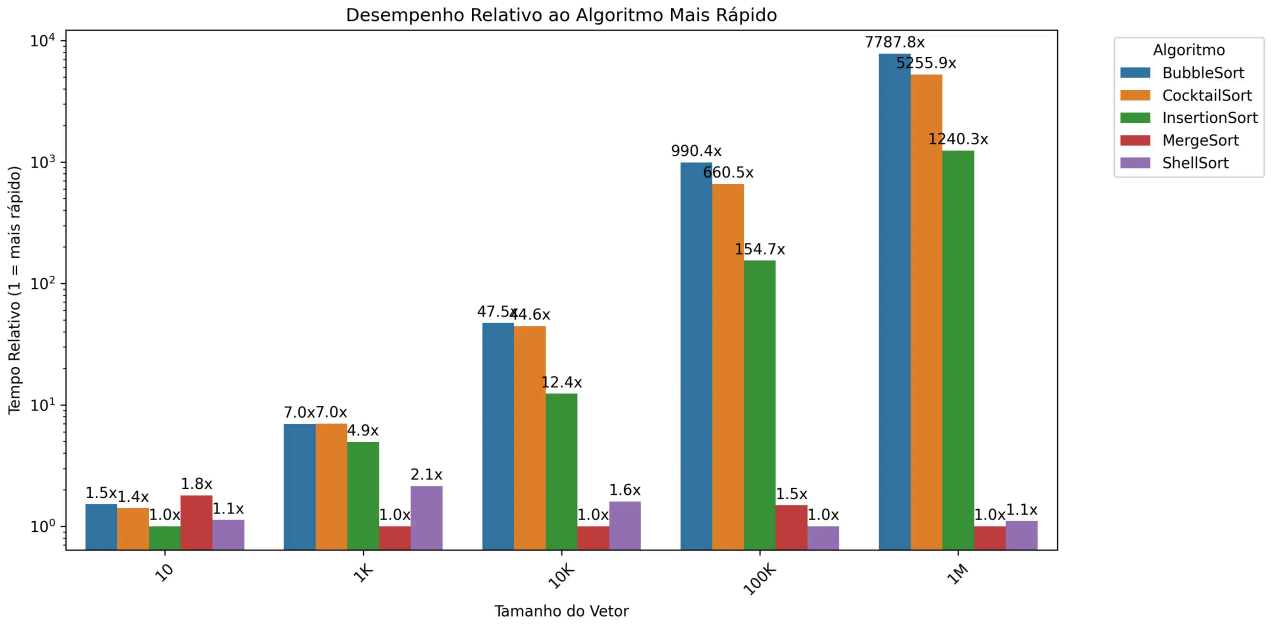
Para análise os comparativos gerais, comparamos todos os algoritmos, entre diferentes quantidades. Podemos concluir que, dadas as condições dos algoritmos (Shell usando sequência de Knuth) e os tamanhos especificados, os algoritmos Shell e Merge Sort têm melhor performance. Sendo que na maioria dos casos o Merge Sort é mais veloz.







Também comparamos os tempos de execução de forma relativa. Usando o resultado mais rápido como ponto de referência, analisamos a performance relativa dos outros.



Comparação de Tempo de Execução - Tamanho 10

Algoritmo	Tempo (ms)	Tempo Relativo	Posição
InsertionSort	0.01	1.00x	1º
ShellSort	0.01	1.13x	2º
CocktailSort	0.01	1.42x	3º
BubbleSort	0.01	1.53x	4º
MergeSort	0.01	1.79x	5º

Comparação de Tempo de Execução - Tamanho 1K

Algoritmo	Tempo (ms)	Tempo Relativo	Posição
MergeSort	0.46	1.00x	1º
ShellSort	1.00	2.15x	2º
InsertionSort	2.29	4.94x	3º
BubbleSort	3.23	6.96x	4º
CocktailSort	3.26	7.01x	5º

Comparação de Tempo de Execução - Tamanho 10K

Algoritmo	Tempo (ms)	Tempo Relativo	Posição
MergeSort	1.53	1.00x	1º
ShellSort	2.45	1.60x	2º
InsertionSort	19.02	12.40x	3º

Algoritmo	Tempo (ms)	Tempo Relativo	Posição
CocktailSort	68.47	44.64x	4º
BubbleSort	72.80	47.47x	5º

Comparação de Tempo de Execução - Tamanho 100K

Algoritmo	Tempo (ms)	Tempo Relativo	Posição
ShellSort	11.90	1.00x	1º
MergeSort	17.82	1.50x	2º
InsertionSort	1840.48	154.72x	3º
CocktailSort	7857.57	660.53x	4º
BubbleSort	11781.24	990.37x	5º

Comparação de Tempo de Execução - Tamanho 1M

Algoritmo	Tempo (ms)	Tempo Relativo	Posição
MergeSort	149.16	1.00x	1º
ShellSort	165.42	1.11x	2º
InsertionSort	184997.07	1240.29x	3º
CocktailSort	783946.34	5255.87x	4º
BubbleSort	1161596.97	7787.78x	5º

Análise de Crescimento dos Algoritmos

Algoritmo	Tamanho	Fator de Crescimento
BubbleSort	10	1.00x
BubbleSort	1K	356.75x
BubbleSort	10K	8031.18x
BubbleSort	100K	1299640.07x
BubbleSort	1M	128140867.95x
CocktailSort	10	1.00x
CocktailSort	1K	387.31x
CocktailSort	10K	8142.95x
CocktailSort	100K	934423.68x
CocktailSort	1M	93227059.62x
InsertionSort	10	1.00x
InsertionSort	1K	386.45x
InsertionSort	10K	3204.28x

Algoritmo	Tamanho	Fator de Crescimento
InsertionSort	100K	310043.83x
InsertionSort	1M	31164223.83x
MergeSort	10	1.00x
MergeSort	1K	43.63x
MergeSort	10K	144.10x
MergeSort	100K	1674.25x
MergeSort	1M	14013.97x
ShellSort	10	1.00x
ShellSort	1K	148.46x
ShellSort	10K	364.55x
ShellSort	100K	1770.58x
ShellSort	1M	24620.73x

Métricas Médias por Algoritmo e Tamanho

Algoritmo	Tamanho	Tempo (ms)	Trocas	Iterações
BogoSort	10	517.08	46814227	66204288
BubbleSort	10	0.01	23	51
BubbleSort	1K	3.23	253287	500098
BubbleSort	10K	72.80	24926451	49990206
BubbleSort	100K	11781.24	2498648430	4999936627
BubbleSort	1M	1161596.97	250111251952	499999413208
CocktailSort	10	0.01	23	43
CocktailSort	1K	3.26	251017	385416
CocktailSort	10K	68.47	24977353	37643048
CocktailSort	100K	7857.57	2502699118	3756106576
CocktailSort	1M	783946.34	249905027239	375015735446
InsertionSort	10	0.01	27	27
InsertionSort	1K	2.29	253080	253079
InsertionSort	10K	19.02	25008477	25008481
InsertionSort	100K	1840.48	2499323392	2499323392
InsertionSort	1M	184997.07	249988655886	249988655884
MergeSort	10	0.01	68	22
MergeSort	1K	0.46	19952	8712
MergeSort	10K	1.53	267232	120443
MergeSort	100K	17.82	3337856	1536279
MergeSort	1M	149.16	39902848	18674181
ShellSort	10	0.01	27	33
ShellSort	1K	1.00	12140	14191
ShellSort	10K	2.45	216942	242206
ShellSort	100K	11.90	3576965	3879125

Algoritmo	Tamanho	Tempo (ms)	Trocas	Iterações
ShellSort	1M	165.42	62609448	66034343