Relatório TDE 3 – Algoritmos de ordenação

João Pedro Igeski Morais

Pontifícia Universidade Católica do Paraná - PUCPR

1. Introdução

Esse relatório tem como objetivo realizar uma análise dos resultados obtidos em relação a trocas, iterações e tempo médio de execução dos algoritmos de ordenação Insertion Sort, Shell Sort, Merge Sort e Bubble Sort.

Link do Github: https://github.com/jpedro1711/TDE-AlgoritmosOrdena-o

2. Tempo de execução

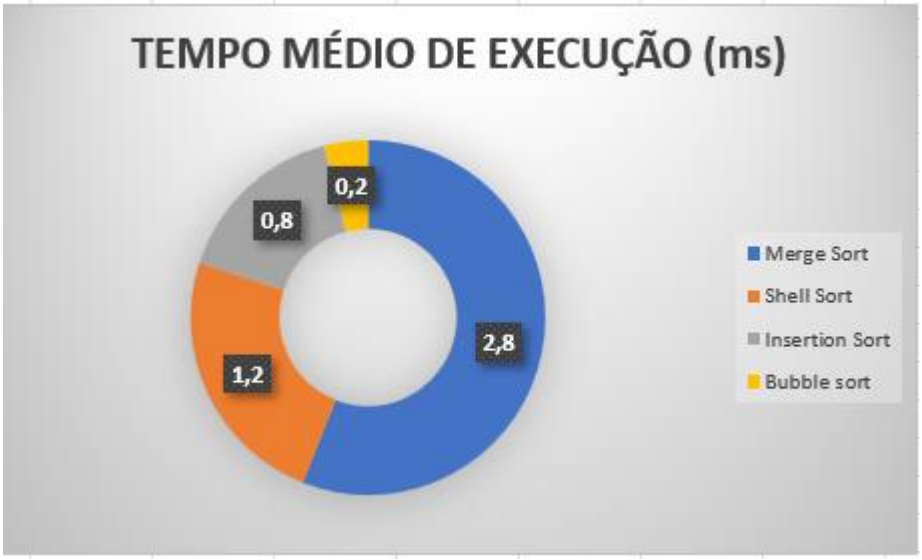
O tempo de execução médio de cada algoritmo foi variável, conforme o número de elementos considerados para cada vetor. Vale ressaltar, que foram considerados o mesmo conjunto de dados para avaliação de cada quesito.

2.1. Algoritmo Merge Sort

Para o algoritmo Merge Sort, o tempo de execução foi maior no caso de menos elementos, isso se dá por conta de que a complexidade do algoritmo é maior se comparada aos outros, exigindo um número de passos maior, logo, para um conjunto pequeno, de **50 elementos**, ele foi menos performático, sendo representado abaixo pelo gráfico.

50 elementos

Algoritmo	Número de elementos	Tempo médio de execução
Merge Sort	50	2,8
Shell Sort	50	1,2
Insertion Sort	50	0,8
Bubble sort	50	0,2

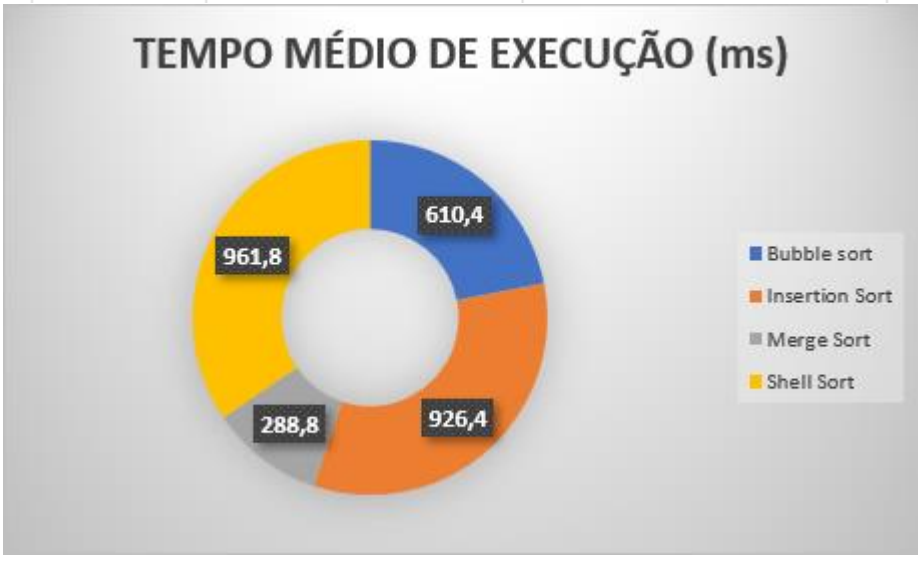


No entanto, para conjuntos maiores de elementos, ele passou a ter uma menor diferença de tempo de execução se comparado a outros algoritmos, como no caso de **500 elementos**.

Algoritmo	Número de elementos	Tempo médio de execução
Bubble sort	500	14,6
Insertion Sort	500	19,6
Merge Sort	500	27
Shell Sort	500	14

Para o maior conjunto de dados, sendo 10000 elementos, ele apresentou o melhor tempo de execução se comparado a outros algoritmos, ou seja, podemos chegar a conclusão que quanto maior o número de elementos, torna-se mais recomendado o uso do Merge Sort, sendo esse cenário representado abaixo.

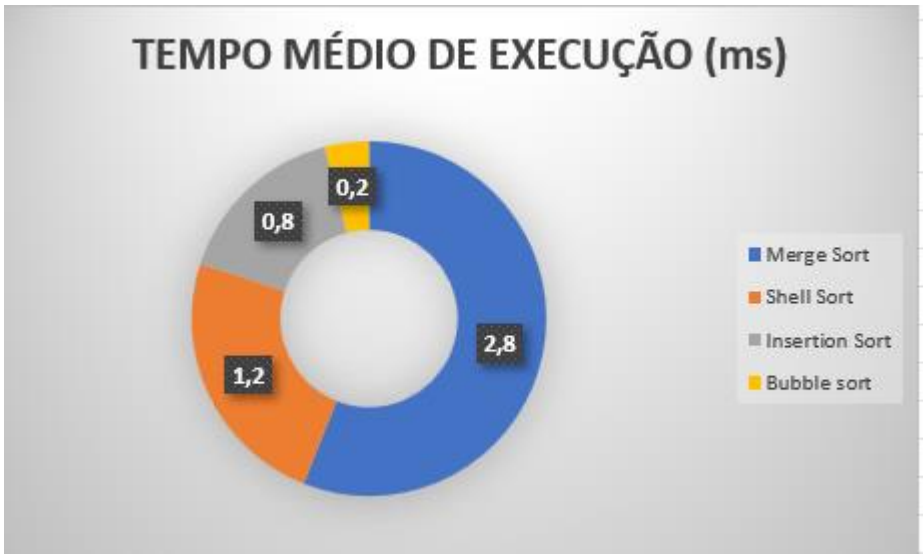
Algoritmo	Número de elementos	Tempo médio de execução
Bubble sort	10000	610,4
Insertion Sort	10000	926,4
Merge Sort	10000	288,8
Shell Sort	10000	961,8



2.2. Algoritmo Insertion Sort e Algoritmo Bubble Sort

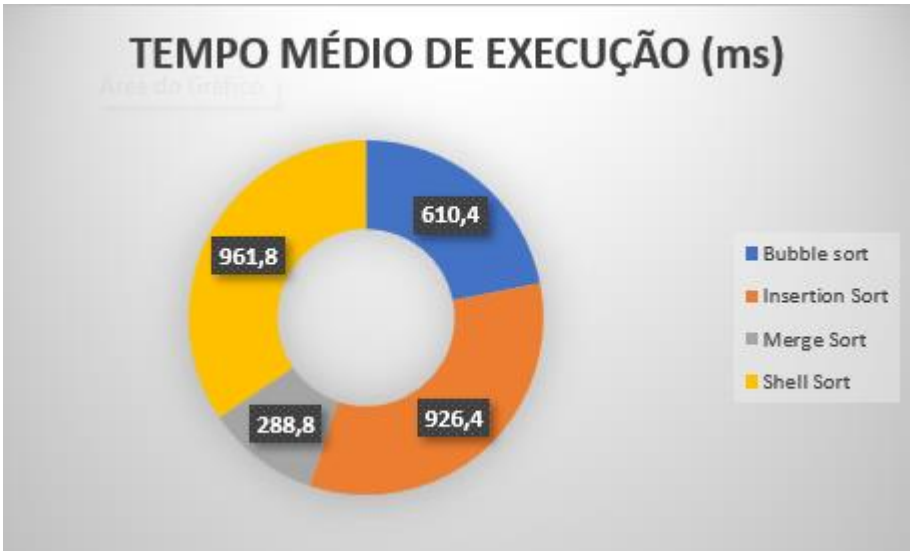
Esses algoritmos, possuem uma complexidade menor, ou seja, em um conjunto menor de dados, serão bem aproveitados, por outro lado, não serão tão performáticos no quesito tempo quando conjuntos maiores de dados devem ser ordenados. Isso é representado pelo gráfico de tempo de execução para 50 elementos.

Algoritmo	Número de elementos	Tempo médio de execução
Merge Sort	50	2,8
Shell Sort	50	1,2
Insertion Sort	50	0,8
Bubble sort	50	0,2



No entanto, para 10000 elementos, o tempo de execução deles foi bem maior se comparado aos outros algoritmos.

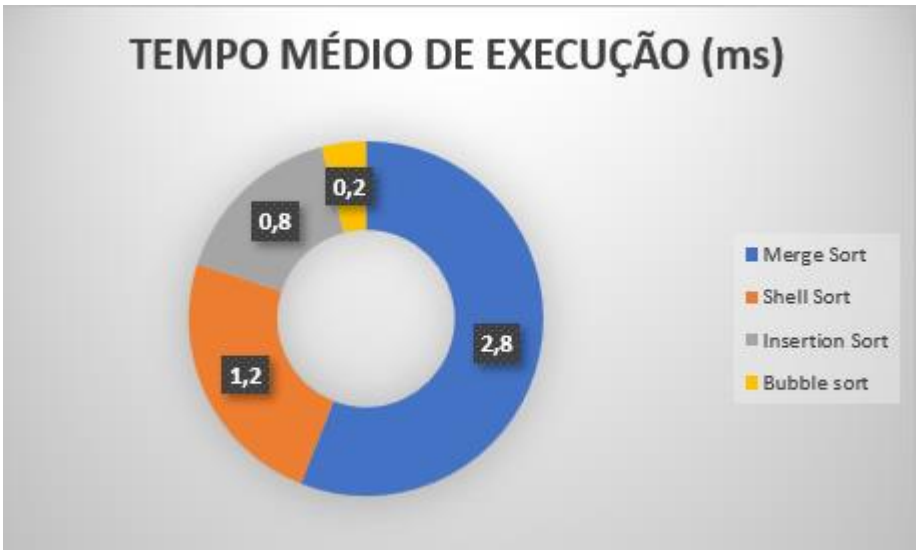
Algoritmo	Número de elementos	Tempo médio de execução
Bubble sort	10000	610,4
Insertion Sort	10000	926,4
Merge Sort	10000	288,8
Shell Sort	10000	961,8



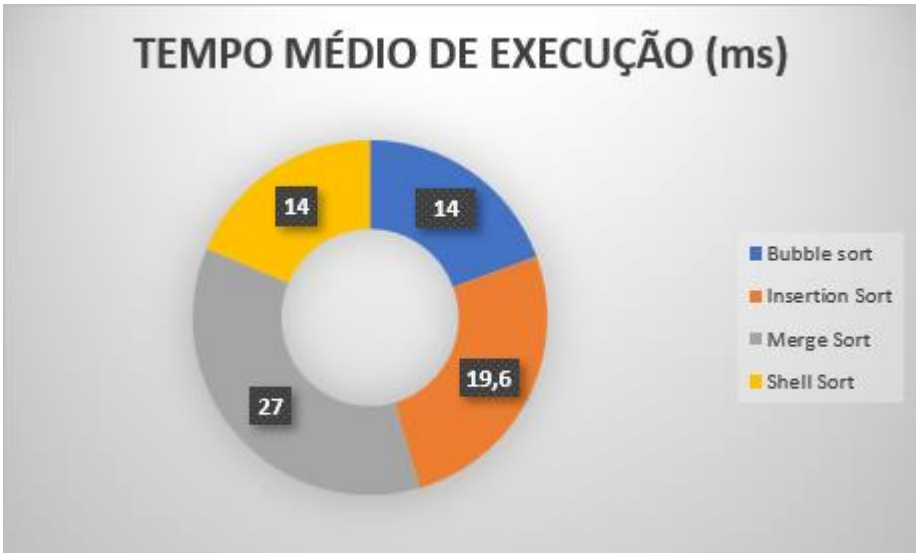
2.3. Algoritmo Shell Sort

O algoritmo Shell Sort, que é muito parecido com o algoritmo Insertion Sort, apresentou um dos melhores desempenhos quando o número de elementos do conjunto de dados foi menor, sendo representado abaixo pelo gráfico de 50 elementos e 500 elementos.

50 elementos

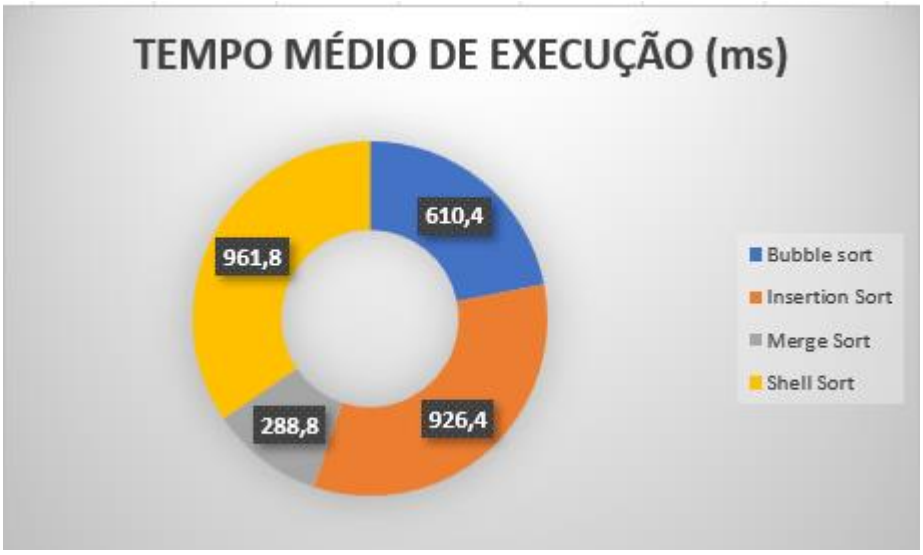


500 elementos



No entanto, quando o número de elementos é maior (10000 elementos), ele deixa de ter um dos melhores tempos de execução, para ter o pior tempo de execução, o que é representado abaixo.

Algoritmo	Número de elementos	Tempo médio de execução
Merge Sort	10000	288,8
Bubble sort	10000	610,4
Insertion Sort	10000	926,4
Shell Sort	10000	961,8



3. Número de trocas

3.1. Algoritmo Merge Sort

No quesito trocas, o algoritmo Merge Sort sempre foi um dos melhores, senão o melhor, exigindo menos trocas para ordenação completa do vetor. Isso é representado pelo gráfico de trocas para 50 elementos e 10000 elementos.

50 elementos

Algoritmo	Número de elementos	Total de trocas
Shell Sort	50	561
Merge Sort	50	572
Insertion Sort	50	691
Bubble sort	50	691

10000 elementos

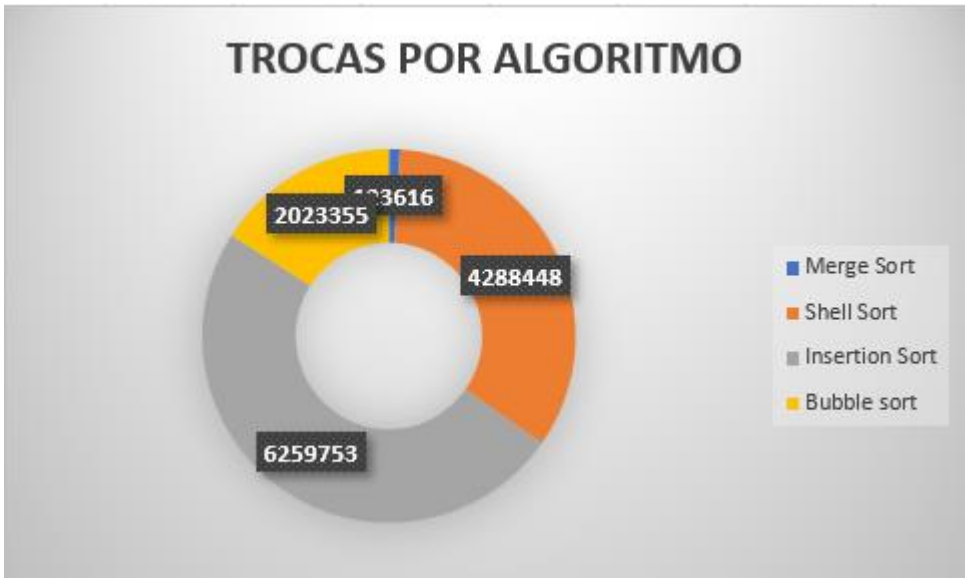
Algoritmo	Número de elementos	Total de trocas
Insertion Sort	10000	25103283
Shell Sort	10000	17154398
Bubble sort	10000	4523425
Merge Sort	10000	267232

3.2. Algoritmos Insertion Sort e Shell Sort

Esses algoritmos possuem princípios muito parecidos, sendo que, a única diferença substancial na implementação é o um intervalo para percorrer o vetor no algoritmo Shell Sort, que vai diminuindo com o tempo. Dessa maneira, o número de trocas exigido é menor se comparado ao Insertion Sort, tanto para coleções menores ou maiores.

50 elementos

5000 elementos



3.3. Algoritmo Bubble Sort

Esse é um algoritmo de complexidade básica, mas que exige um número maior de trocas, sendo esse cenário representado pelos gráficos abaixo. No entanto, apesar de não ser o melhor nesse quesito, ainda pode ser aproveitado para conjuntos de dados menores.

50 elementos

1000 elementos

4. Iterações

4.1. Algoritmo Merge Sort

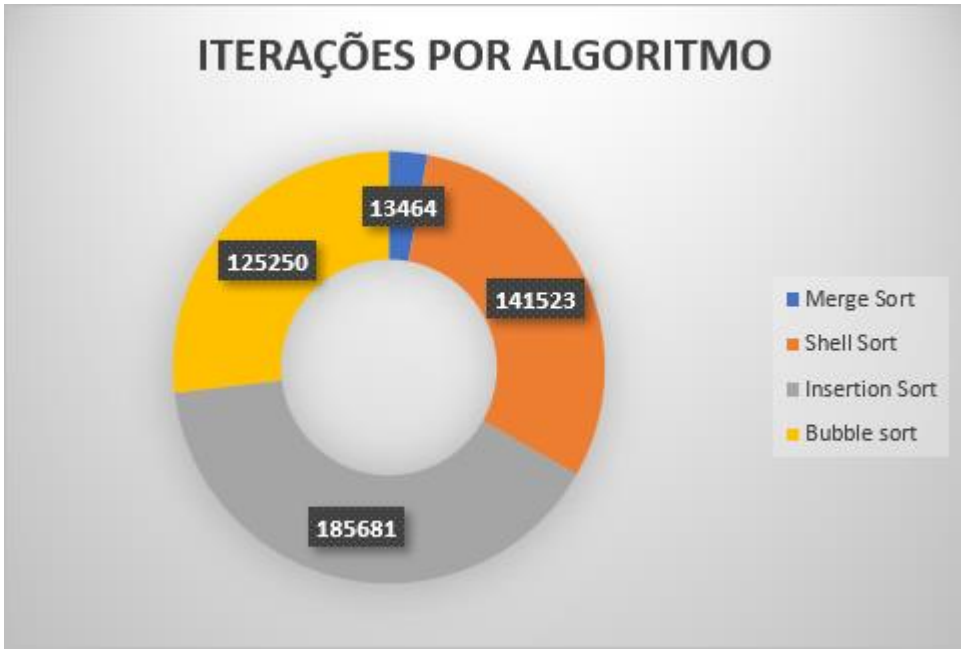
Novamente, esse algoritmo resultou em um melhor desempenho, precisando de menos iterações para ordenação completa do vetor. Isso se explica, usando a notação Big O, por conta de que a esse algoritmo apresenta uma complexidade menor se comparada aos outros, no pior dos casos.

50 elementos

Algoritmo	Número de elementos	Total de iterações
Merge Sort	50	858
Shell Sort	50	1425
Insertion Sort	50	1860
Bubble sort	50	1275

500 elementos

Algoritmo	Número de elementos	Total de iterações
Merge Sort	500	13464
Shell Sort	500	141523
Insertion Sort	500	185681
Bubble sort	500	125250



5000 elementos

Algoritmo	Número de elementos	Total de iterações
Merge Sort	5000	185424
Shell Sort	5000	16585695
Insertion Sort	5000	18743342
Bubble sort	5000	12502500

4.2. Algoritmos Shell Sort, Insertion Sort e Bubble Sort

Esses algoritmos foram os que precisaram de um número maior de iterações para ordenação dos vetores, que foi muito parecido entre si. Isso ocorre, por conta de que na implementação dos três, foi necessário o uso de loops for para ordenação. Outra forma de explicar esse fato, é a mesma complexidade Big O dos algoritmos, nesse caso, O (N^2).

50 elementos

Algoritmo	Número de elementos	Total de iterações
Merge Sort	50	858
Shell Sort	50	1425
Insertion Sort	50	1860
Bubble sort	50	1275

10000 elementos

Algoritmo	Número de elementos	Total de iterações
Merge Sort	10000	400848
Shell Sort	10000	66322561
Insertion Sort	10000	75050919
Bubble sort	10000	50005000

5. Conclusão

Ao analisar o tempo de execução médio de cada algoritmo, o número de trocas e iterações necessários para ordenação completa de um mesmo conjunto de dados, chegamos a conclusão de que o que apresentou melhor desempenho, é o algoritmo MergeSort, que possui uma complexidade menor se comparado aos outros algoritmos que possuem implementação parecida, usando dois loops For e possuindo uma complexidade, nos piores casos, de O (N^2), enquanto o MergeSort, nos piores casos possui complexidade O(n Log(n)). Essa complexidade do MergeSort, se dá por conta do algoritmo dividir o vetor original em menores, e depois combiná-los, em vez de percorres realizar inúmeras iterações e trocas.