Algoritmos de Ordenação: Análise dos Desempenhos

Marco S. Coutinho, Matheus H. S. Araújo, Rubens E. C. Diniz
Disciplina: Introdução à Computação em Física – FIS616 Turma: TM
Universidade Federal de Minas Gerais – MG, Brasil
Julho de 2021

Resumo: O estudo realizado tem como objetivo analisar o desempenho dos algoritmos de ordenação *Bubble Sort*, *Merge Sort*, *Insertion Sort*, *Shell Sort* e *Selection Sort* para diferentes quantidades de vetores. Para tanto, desenvolveu-se um código em linguagem Python em que tempo de execução dos referidos algoritmos foi medido enquanto ordenavam um determinado número de vetores aleatórios, sendo que essa quantidade é aumentada em intervalos iguais até atingir o número máximo de vetores pré-estabelecido. A fim de otimizar a comparação dos desempenhos, os dados coletados foram utilizados para gerar gráficos do tempo de ordenação em função da quantidade de vetores.

I. INTRODUÇÃO

Problemas são questões propostas em busca de uma solução, e os algoritmos existem, cada qual aplicável em determinada situação, para facilitar essa busca e encontrar a solução com maior eficiência. De modo geral, podemos definir o problema de ordenação como:

- Descrição da entrada: Uma sequência de n itens $R_1, R_2, \dots, R_n$ chamados registros;
- Descrição da tarefa: Ordenar o conjunto de registros em ordem crescente (ou decrescente) de chaves;
- Descrição da saída: Retornar à sequência de entrada, ordenada em ordem crescente ou decrescente.

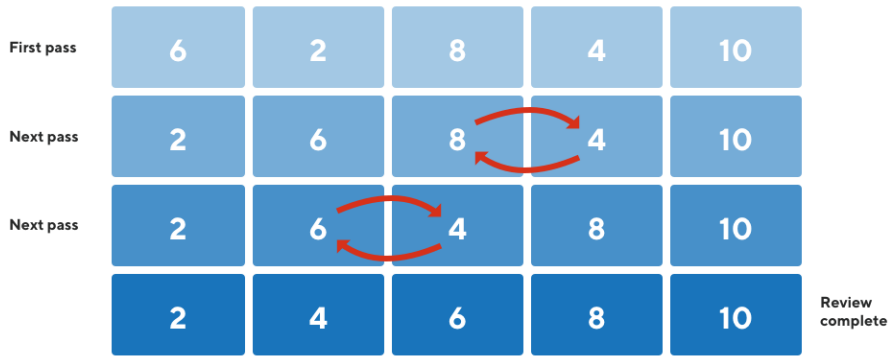
Cada registro é composto por uma chave, ou seja, números inteiros ou strings, e sempre ordenaremos os registros em ordem crescente, a fim de simplificar as operações.

Nesse trabalho, veremos os tipos de ordenação e seus funcionamentos, determinando a eficiência de cada um em determinada situação.

Bubble Sort: Este método (Ordenação Bolha) consiste na realização de múltiplas passagens por uma lista, comparando itens adjacentes e troca aqueles que estão fora de ordem. Cada passagem pela lista coloca o próximo maior valor na sua posição correta. Ou seja, cada item se desloca como uma "bolha" para a posição à qual pertence.

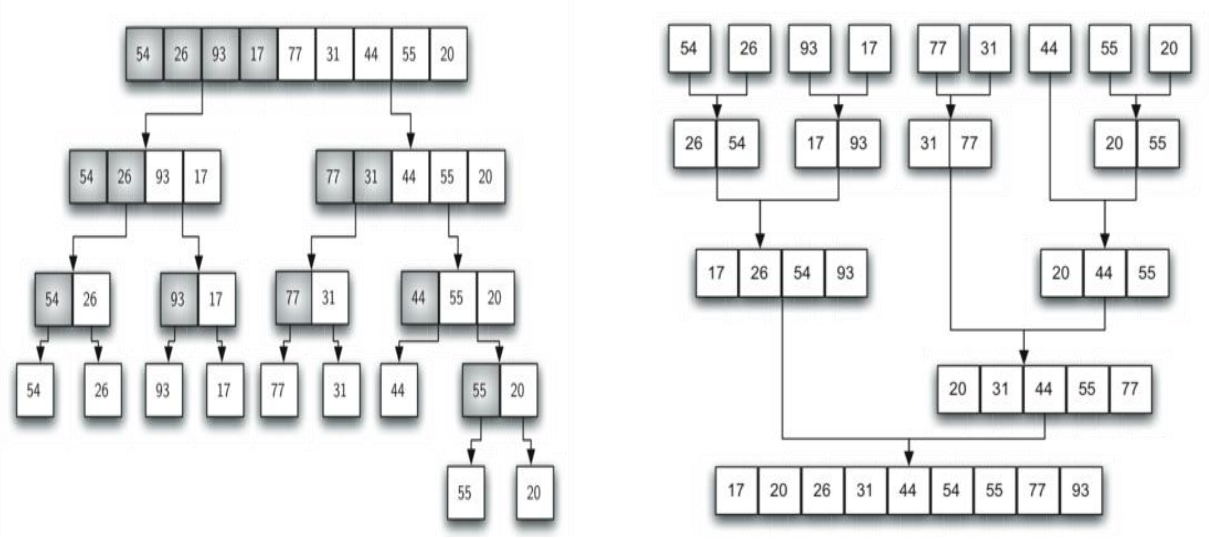
A ideia desse método é ordenar o vetor seguindo os seguintes passos:

1. Primeiro, conduza o maior elemento para a última posição, comparando os elementos de dois em dois;
2. Depois, repita o processo com os próximos termos, até que o vetor esteja completamente ordenado



Merge Sort: Esse método divide as listas de entrada pela metade. Após dividir, é ordenado recursivamente cada metade e por fim, as duas metades são mescladas, ou seja, combinadas em um único arranjo.

A figura a baixo ilustra o funcionamento do Merge Sort:

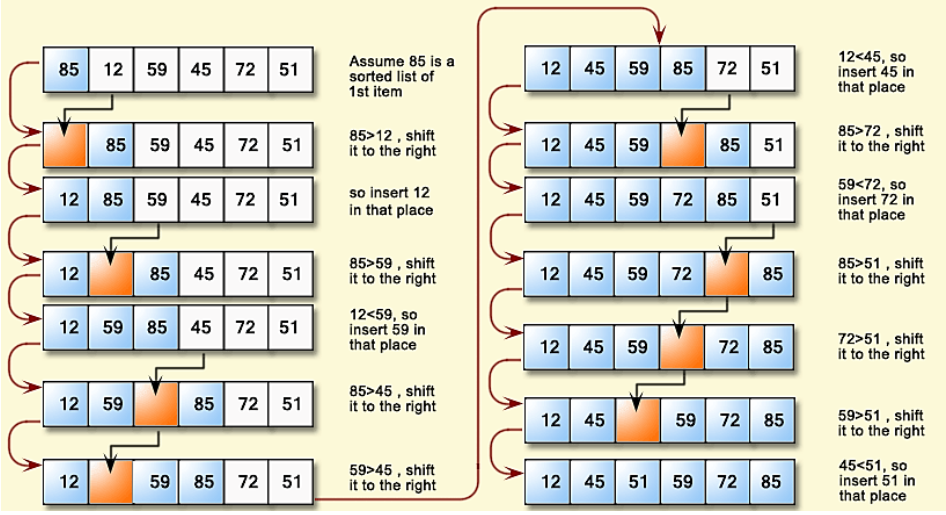


Pode-se notar que a lista é dividida em esquerda e direita até que sobrem dois elementos adjacentes. Após a separação, os interadores [i] e [j] atravessam as duas metades em cada chamada. Se o valor em [i] for menor que o valor em [j], será atribuído ao myList[k]slot e [i] será incrementado. Se não, então right[j] é escolhido. No final deste loop, uma das metades pode não ter sido percorrida completamente. Então, seus valores são simplesmente atribuídos aos slots restantes na lista.

O tempo para realizar o Merge Slot é dado em $O\ n \cdot \log(n)$. Isso ocorre, pois, a lista está sendo dividida em chamadas log (n) e o processo de mesclagem leva um tempo linear em cada chamada.

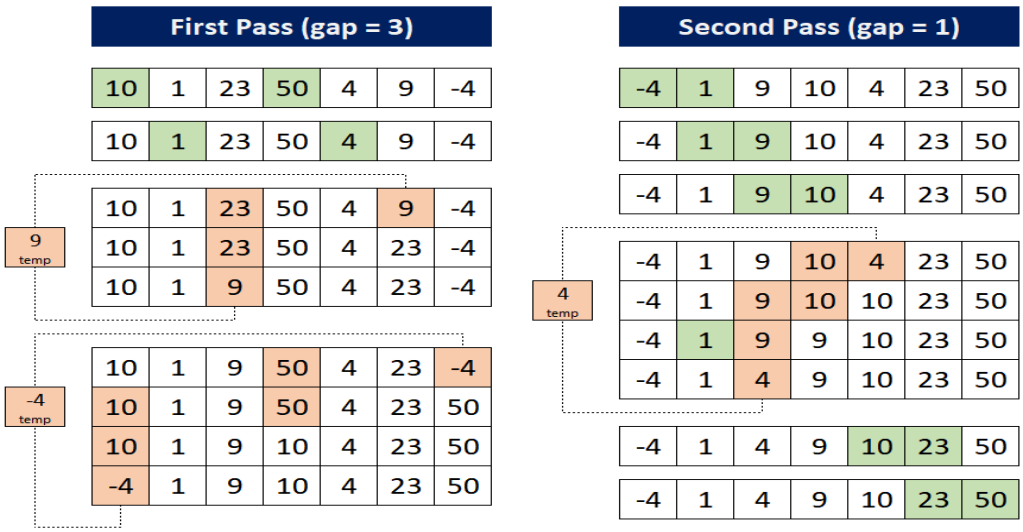
Insertion Sort: É um método de classificação simples que pode ser comparado com a maneira que se classifica as cartas de um baralho nas suas mãos. O array é virtualmente dividido em uma parte ordenada e outra não ordenada. Os valores da parte não classificada são colocados na sua posição correta na parte classificada.

A ideia desse método consiste em classificar uma matriz de tamanho n em ordem crescente. Primeiro ele itera de arr[1] para arr[n] na matriz, depois compara o elemento (chave) com o seu antecessor, se for menor, move os elementos maiores uma posição para cima para liberar espaço para o elemento trocado.



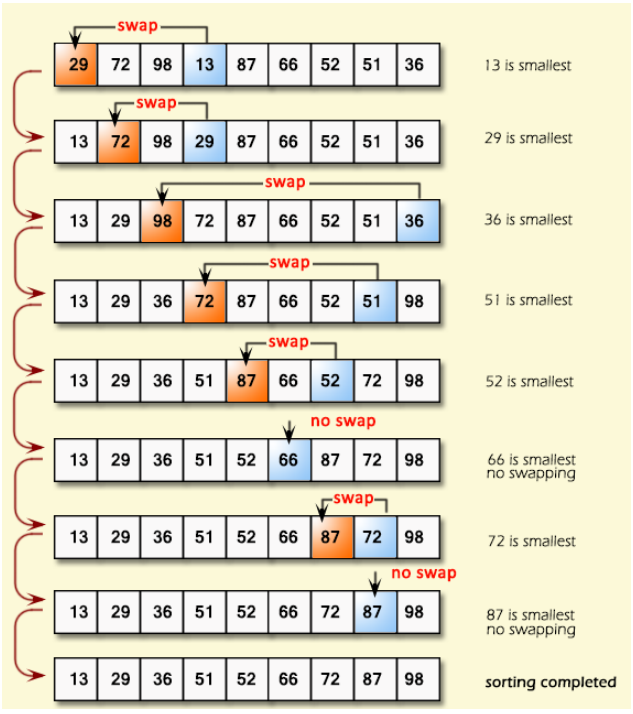
Shell Sort: O Shell Sort melhora a ordenação por inserção, quebrando a lista em um número menor de sublistas, as quais são ordenadas pelo método da inserção.

Além da lista ser quebrada, o Shell Sort incrementa [i] item, para criar uma sublista escolhendo todos os itens que estão afastados de [i] itens uns dos outros.



A forma como os incrementos são escolhidos é uma característica única do Shell Sort. A função usa um conjunto de diferentes incrementos. Nesse caso, começamos com $\frac{n}{2}$ sublistas. No passo seguinte, $\frac{n}{4}$ sublistas são ordenadas. Isso ocorre até ter uma única lista é ordenada com a ordenação por inserção básica.

Selection Sort: O método Selection Sort classifica uma matriz encontrando repetidamente um elemento mínimo da parte não classificada e movendo-o para o início. Esse método divide a lista em duas partes, a parte ordenada (vazia) e a parte não ordenada (aquela que contém os elementos). Então o algoritmo seleciona o valor mínimo de todos os arquivos não classificados e troca pelo primeiro valor não classificado e depois aumenta a parte classificada em um, como na imagem a baixo:



II. ESPECIFICAÇÕES

Para a realização das simulações, a Linguagem Python foi utilizada na codificação, tendo o VSCode e o Google Colab como ambientes de execução. A máquina em que os testes foram executados é um processador Ryzen 5 3600X, com 16 GB de memória RAM 3000 MHz, com sistema operacional Ubuntu 21.04.

III. CÓDIGO

O código utilizado encontra-se disponível no seguinte endereço web:

github.com/coutinhomarco/sorting-algorithms

Os resultados obtidos são referentes à execução desse código nas especificações descritas anteriormente (tópico II. Especificações).

IV. RESULTADOS E DISCUSSÕES

Nas tabelas abaixo estão as aproximações das medições do tempo para 100, 1000, 5000 e 10000 vetores:

Execução no VSCode					
Vetores	Bubble	Insertion	Selection	Merge	Shell
100	$4{,}40 \cdot 10^{-4}\ s$	$2{,}38 \cdot 10^{-4}\ s$	$2{,}38 \cdot 10^{-4}\ s$	$2{,}40 \cdot 10^{-4}\ s$	$1{,}36 \cdot 10^{-4}\ s$
1000	$4{,}08 \cdot 10^{-2}\ s$	$2{,}12 \cdot 10^{-2}\ s$	$1{,}99 \cdot 10^{-2}\ s$	$2{,}78 \cdot 10^{-3}\ s$	$1{,}91 \cdot 10^{-3}\ s$
5000	$1{,}075\ s$	$0{,}482\ s$	$0{,}470\ s$	$1{,}48 \cdot 10^{-2}\ s$	$1{,}26 \cdot 10^{-2}\ s$
10000	$4{,}180\ s$	$1{,}977\ s$	$1{,}920\ s$	$3{,}38 \cdot 10^{-2}\ s$	$2{,}82 \cdot 10^{-2}\ s$

Execução no Google Colab					
Vetores	Bubble	Insertion	Selection	Merge	Shell
100	$1{,}26 \cdot 10^{-3}\ s$	$6{,}39 \cdot 10^{-4}\ s$	$6{,}40 \cdot 10^{-4}\ s$	$5{,}49 \cdot 10^{-4}\ s$	$3{,}48 \cdot 10^{-4}\ s$
1000	$0{,}111\ s$	$5{,}54 \cdot 10^{-2}\ s$	$5{,}86 \cdot 10^{-2}\ s$	$9{,}05 \cdot 10^{-3}\ s$	$5{,}37 \cdot 10^{-3}\ s$
5000	$3{,}028\ s$	$1{,}473\ s$	$1{,}259\ s$	$4{,}10 \cdot 10^{-2}\ s$	$3{,}53 \cdot 10^{-2}\ s$
10000	$11{,}858\ s$	$5{,}803\ s$	$4{,}984\ s$	$0{,}100\ s$	$8{,}35 \cdot 10^{-2}\ s$

Os gráficos a seguir também foram gerados a partir da execução do código desenvolvido:

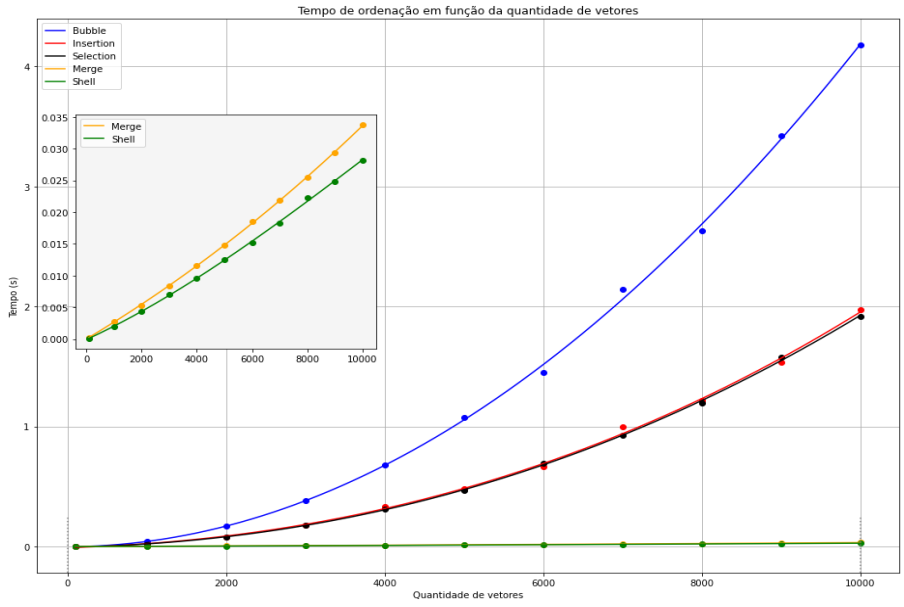


Gráfico 1: Execução no **VSCode** – Tempo de ordenação em função da quantidade de vetores.

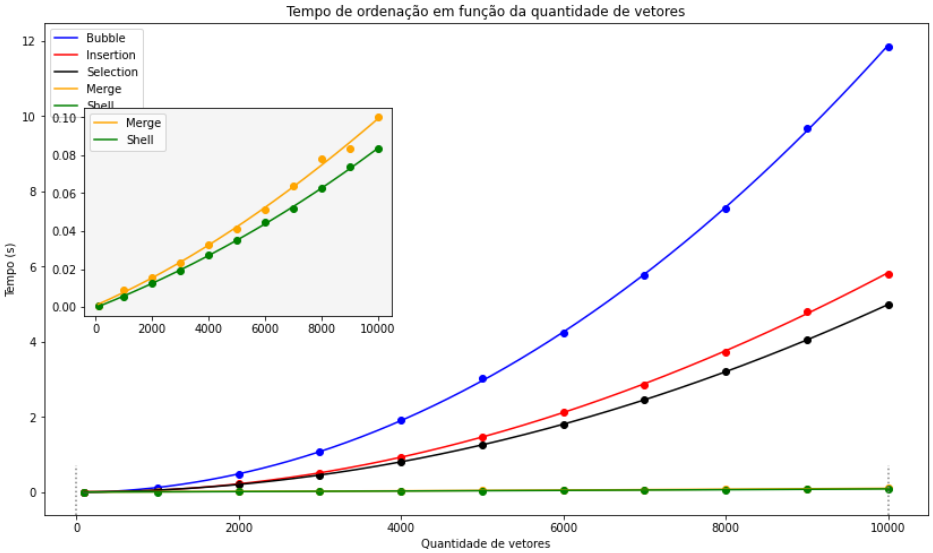


Gráfico 2: Execução no **Google Colab** – Tempo de ordenação em função da quantidade de vetores.

Uma vez que os tempos de ordenação do Merge e do Shell foram bem pequenos e relativamente bem próximos, executou-se então somente os dois algoritmos até uma quantidade de 1 milhão de vetores. Os gráficos abaixo apresentam os desempenhos desses algoritmos:

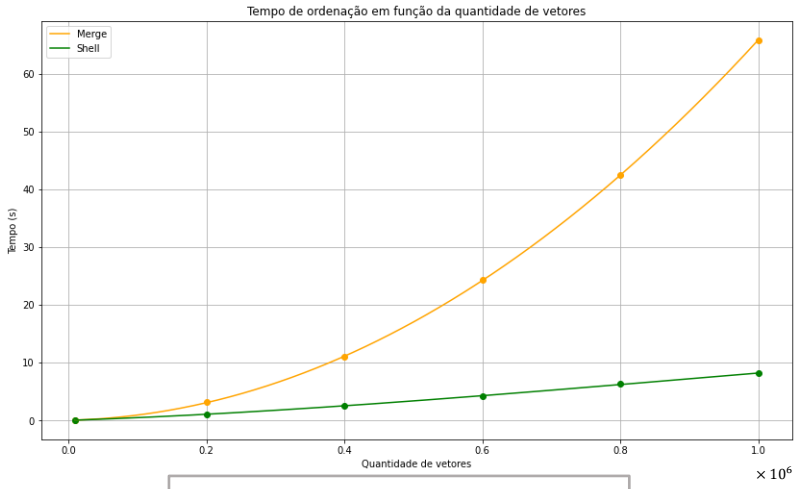


Gráfico 2: Execução no **VSCode** – Merge e Shell.

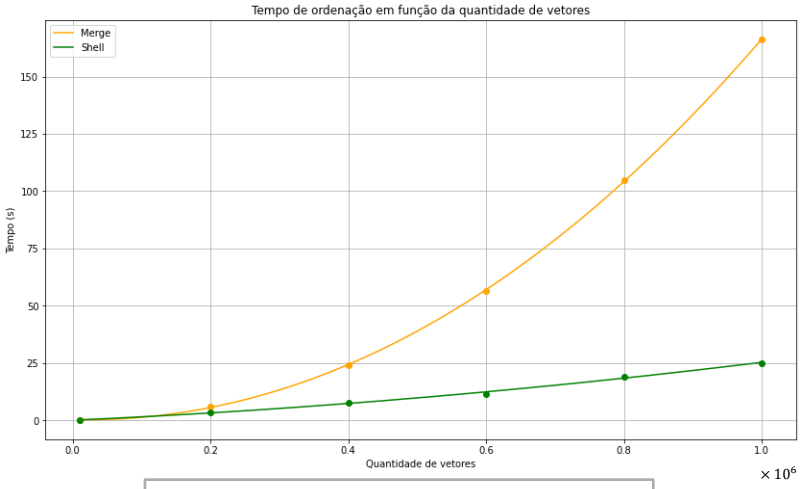


Gráfico 3: Execução no **Google Colab** – Merge e Shell.

V. CONCLUSÃO

De um modo geral, foi possível perceber que o Bubble Sort é o método mais lento de todos e que o Shell Sort é o mais rápido. Pórem, observando o funcionamento dos algoritmos, nota-se que cada um deles tem suas vantagens e desvantagens, tornando-os próprios para usar em determinadas situações:

- O Bubble Sort é o método mais ineficiente, já que ele consiste em realizar a troca sem saber qual vai ser a posição final. Porém, esse método pode ter algumas vantagens, como por exemplo, se durante uma passagem não houver trocas, então sabemos que a lista está ordenada. Portanto, para as listas que requerem apenas algumas passagens, o Bubble Sort pode ter a vantagem de reconhecer a lista ordenada e parar.
- O Merge Sort, apesar de ser uma das ordenações mais eficientes, é lento para tarefas menores, requer um espaço de memória adicional e passa por todo o processo, mesmo se a matriz for classificada.
- O Insertion Sort é um método simples e estável que requer pouco espaço e funciona bem para matrizes pequenas ou quase classificadas.
- O Shell Sort não precisa de realizar muitos deslocamentos, já que cada passagem produz uma lista mais ordenada do que a anterior, tornando a passagem final mais eficiente.
- O Selection Sort é um algoritmo lento, que é usado para classificar sistemas onde a memória é limitada.

Além disso, percebe-se que o ambiente de execução dos algoritmos tem uma influência significativa no tempo de ordenação.

REFERÊNCIAS

- ic.unicamp.br/~mc102/aulas/aula10.pdf
- www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao
- panda.ime.usp.br/pythonds/static/pythonds_pt/05-OrdenacaoBusca/OMergeSort.html
- algoritmosempython.com.br/cursos/algoritmos-python/pesquisa-ordenacao/mergesort/
- www.programiz.com/dsa/insertion-sort
- www.tutorialspoint.com/insertion-sort-in-python-program
- bookdown.org/jessicakubrusly/programacao-estatistica/algoritmos-de-ordenacao.html
- www.geeksforgeeks.org/python-program-for-shellsort/
- www.programiz.com/dsa/shell-sort