

Equipe: Davi Sousa Soares  
Victor Kauan da Silva Miranda  
João Pedro Saleh de Sousa

Professor: Raimundo Santos Moura

NOVEMBRO  
2024

## Conteúdo

1. Introdução
  - 1.1. Objetivos
2. Funcionalidades
  - 2.1. Especificações
  - 2.2. Gráficos
  - 2.3. Conclusões

## 1 Introdução

Esse trabalho será destinado a observar a execução de determinados algoritmos de ordenação em suas tarefas, sendo eles o Heapsort, o Quicksort, o Mergesort, o Shellsort, o Bubblesort, o Insertionsort e o Selectionsort.

### 1.1 Objetivos

O objetivo deste trabalho é medir o tempo gasto pelos algoritmos citados executando em arquivos de 50.000, 100.000, 250.000, 500.000 e 1.000.000 elementos.

Os algoritmos são divididos em 2 grupos, o grupo 1 realiza em média  $O(n \log(n))$  operações, sendo  $n$  o número de elementos no conjunto. Fazem parte desse grupo o Heapsort, o Quicksort, o Mergesort e o Shellsort.

Já o grupo 2 realiza em média  $O(n^2)$  operações. Fazem parte desse grupo o Bubblesort, o Insertionsort, e o Selectionsort.

## 2 Funcionalidades

O código em si é separado em 4 módulos para que seja facilitado entendimento e diminuir a quantidade de linhas do mesmo.

O Algoritmos.py contém todos os algoritmos de ordenação em forma de funções, além de uma função decorator que calcula quanto tempo leva do início ao fim da função.

O Menu contém as operações que podem ser realizadas para medir o tempo, realiza\_operação calcula o tempo gasto por uma função, checa\_se\_ta\_ordenando transfere a lista ordenada para um arquivo para que seja feita a checagem de que o algoritmos está funcionando adequadamente e o calcula\_media que calcula a média de tempos de um algoritmo em determinado arquivo

O sort Quadráticos cria o gráfico dos algoritmos quadráticos

O sort Logarítmicos cria o gráfico dos algoritmos não quadráticos

## 2.1 Especificações

Origem de cada algoritmos:

Heapsort: <https://www.geeksforgeeks.org/heap-sort/>

Quicksort: <https://www.geeksforgeeks.org/quick-sort-algorithm/>

Mergesort: Victor Kauan

Shellsort: [https://pt.wikipedia.org/wiki/Shell\\_sort](https://pt.wikipedia.org/wiki/Shell_sort) modificado por João Pedro Saleh

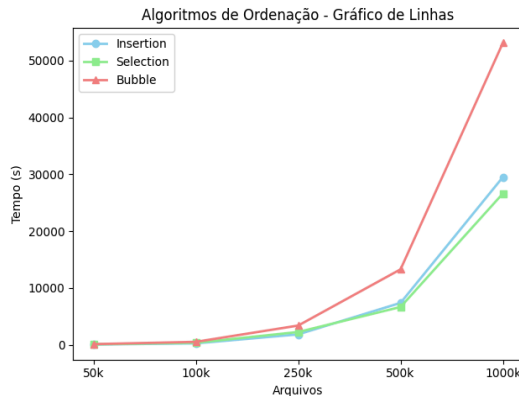
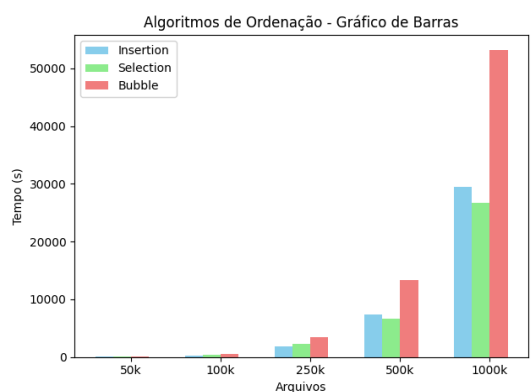
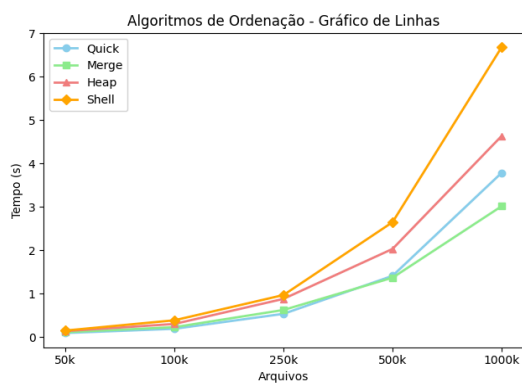
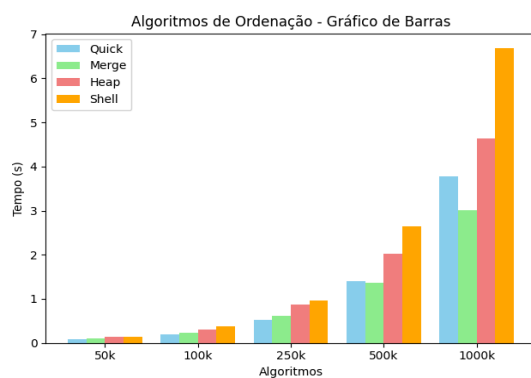
Insertionsort: <https://www.geeksforgeeks.org/insertion-sort-algorithm/> modificado por Davi

Sousa

Selectionsort: <https://www.geeksforgeeks.org/selection-sort-algorithm-2/>

Bubblesort: <https://www.geeksforgeeks.org/bubble-sort-algorithm/>

## 2.2 Gráficos



Testes realizados em um notebook com processador: 12th Gen Intel(R) Core(TM) i7-12700H e sistema operacional linux, usando em média 10% da cpu.

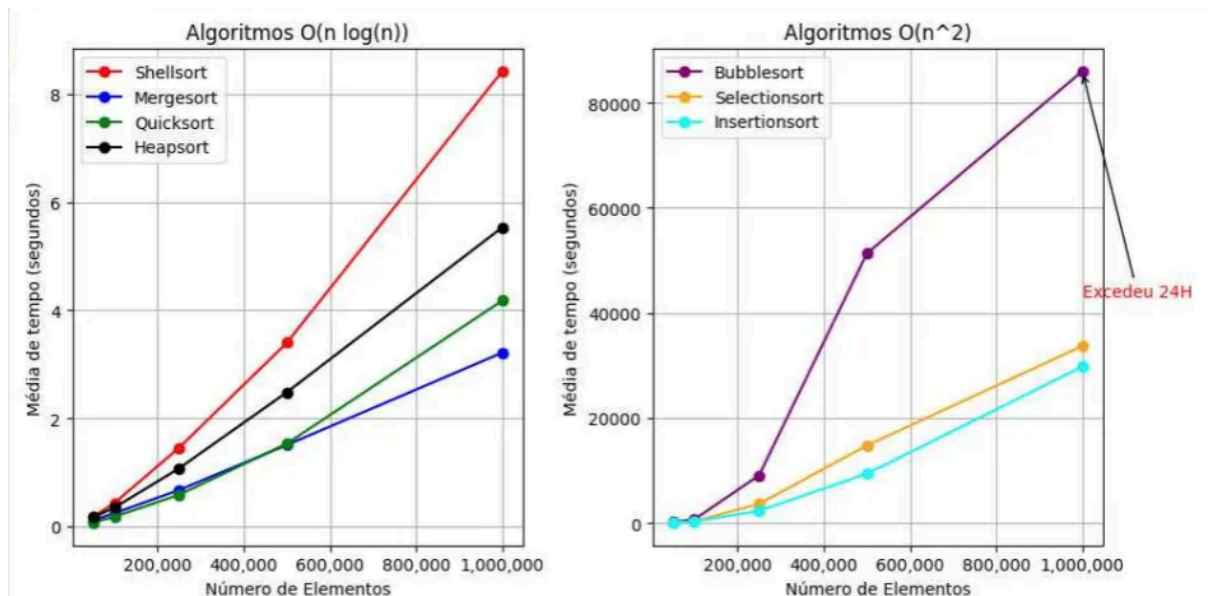
O tempo nos algoritmos quadráticos pode ser lido como 0, 2.7, 5.5, 8.3, 11.11 e 13.8 horas respectivamente para facilitar a leitura.

## 2.3 Conclusões

- Mesmo aumentando significativamente o número de elementos, os algoritmos do grupo 1 não tiveram nenhum aumento de tempo considerável, sendo possível realizar a média de 10 execuções de cada um em pouco menos de 1 minuto.
- O shell é consideravelmente mais lento que seus colegas de grupo, ainda assim percorre 1.000.000 elementos mais rápido que qualquer um do grupo dois percorre 50.000 elementos.

- O grupo 2 se provou difícil de comparar desde o primeiro arquivo de elementos já que os 3 juntos levam cerca de 3 minutos.
- A execução das médias do grupo 2 levou cerca de 4 dias para construir o gráfico, com a exclusão do arquivo de 1.000.000 elementos, que foi feita usando uma estimativa baseada nos resultados dos exemplos anteriores, que tendem a ser aproximadamente o quádruplo de seus antecessores.
- O bubble é o algoritmo mais ineficaz e o quick e o merge variam entre os mais eficazes.

## 2.4 Comparações



Essa seção é destinada a comparação de resultados com o grupo 8, composto por Marcos Vinicius Duarte Sousa, Felipe Gabriel Soares Rodrigues e Cauã Martins da Silva

- **Semelhanças**
  - A colocação da velocidade dos algoritmos é a mesma, com exceção do mergesort no nosso que em alguns casos é melhor que o quick
  - Nenhum dos dois grupos incluiu a execução do bubblesort de 1 milhão de elementos devido a excessivo consumo de tempo e processamento do computador
- **Diferenças**
  - O tempo dos nossos resultados foi superior em todos os casos de testes devido às diferenças de processadores, sendo isso mais evidente no bubblesort
  - Nosso grupo separou as funcionalidades em módulos separados para limitar o número de linhas do código e uma função decorator para realizar a medição do tempo automaticamente em todas as funções