# Relatório de Desempenho

Miniprojeto de Ordenação

**Disciplina:** Estrutura de Dados
**Docente**: Prof. Ricardo Rubens
**Discentes:** Igor Santana Batista, Rayanne Rayssa Gomes Ferreira dos Santos e Vinicius Aranda Lima da Silva

## 1. Descrição do Algoritmo Híbrido

O algoritmo híbrido foi desenvolvido conforme instruções descritas para desenvolvimento do projeto.

Criamos um classe chamada **AlgoHybridSort**, no método **__init__** adicionamos o parâmetro ***limitSublist*** com um valor **default de 164**, caso o usuário não passe essa informação.

Esse parâmetro ***limitSublist*** é usada especificamente para quando o tamanho da sublista atinge o limite ou um valor menor que o limite, para que seja utilizado ao algoritmo do **SelectionSort**, quando o tamanho da sublista for maior que o limite será utilizado o algoritmo **QuickSort** para ordenar.

Criamos a função **quickSortHelper** que vai ser a responsável por processar a lógica para definir qual algoritmo vai ser usado na interação. Quando o algoritmo de Quick sor vai ser usado, é realizado um particionamento da lista com base em um pivô selecionado.

O pivô está sendo selecionado buscando o valor mediano entre o primeiro item, o item do meio e o último item da lista.

A função ***quickSortHelper*** é chamada recursivamente para que possa ser processada cada sublista.

Analisando o desempenho do algoritmo híbrido, ele usa como base o quicksort, no melhor caso o desempenho será O( n log n), se o limite estipulado para a utilização do selectionsort na sublista for pequeno, não haverá um impacto significativo, sendo assim, ainda garantimos a complexidade de O( n log n).
Para o caso médio o desempenho para o QuickSort ainda será O( n log n), o selectionsort ainda não afetará significativamente se escolhido bem.

Para o pior caso o desempenho para o QuickSort quando o pivô é escolhido como o maior elemento ou menor elemento da lista a complexidade será O(n^2).

## 2. Metodologia de Teste de Desempenho

Criamos inicialmente um **dataset** na estrutura de **dicionário** do python, onde a **chave** é a quantidade do conjunto de dados e o **valor** é um lista do tamanho referente a quantidade com número aleatórios gerados com **random.randint(0, 1000000)**. Decidimos realizar dessa forma para que fosse gerado apenas uma vez o conjunto de dados para teste.

Foi criado também um dicionário chamado ***algorithms*** onde a chave é o nome do algoritmo e o **valor** é a função responsável pelo teste do algoritmo, segue exemplo:

```
algorithms = {
    'SelectionSort': testSelectionSort,
    'QuickSort': testQuickSort,
    'MergeSort': testMergeSort,
    'AlgoHybridSort with limit 16': lambda arr: testHybridSort(arr, 16),
    'AlgoHybridSort with limit 64': lambda arr: testHybridSort(arr, 64),
    'AlgoHybridSort with limit 256': lambda arr: testHybridSort(arr, 256),
}
```

Decidimos fazer dessa forma para facilitar na criação de uma única função para rodar os testes de performance e também para facilitar na geração de informações de desempenho de cada algoritmo.

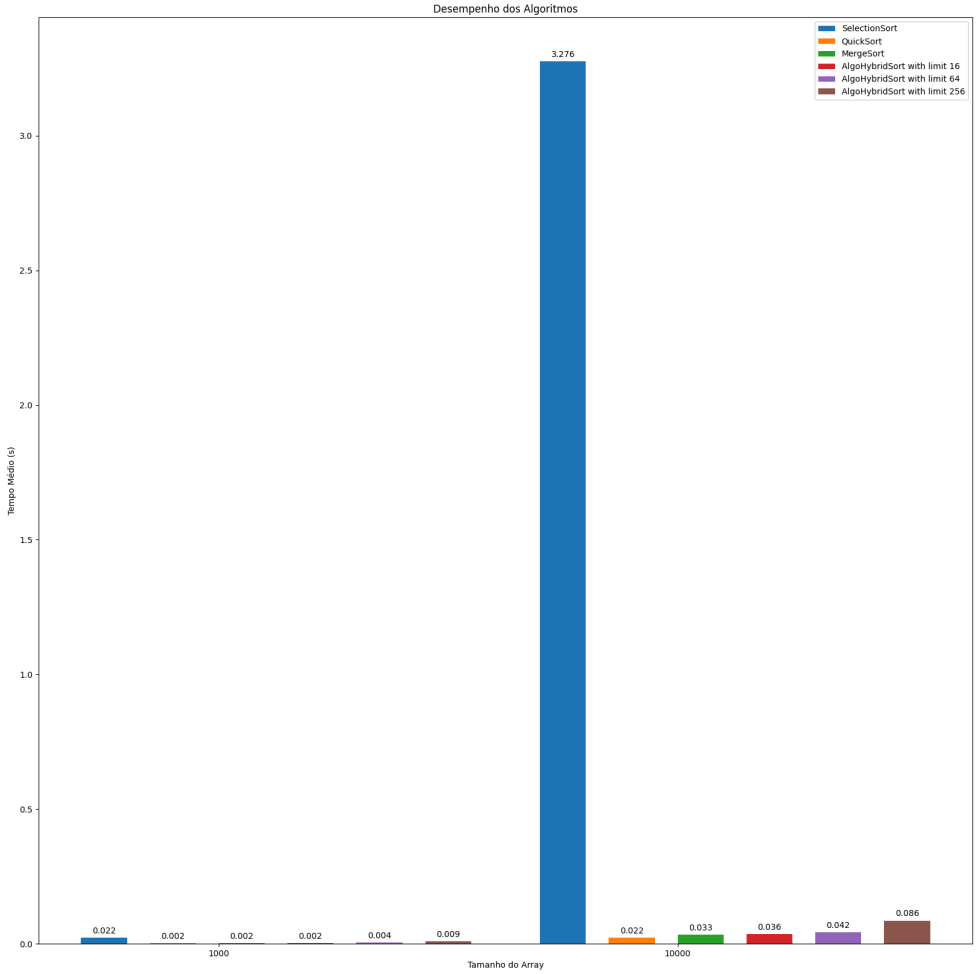
Após as etapas acima, criamos uma função chamada **testPerformance**, que percorrer o dicionário de ***algorithms*** executando cada valor que é uma função referente ao teste do algoritmo, no momento de passar os dados para ordenação, utilizamos o copy(), para garantir uma comparação justa entre os algoritmos.

Foi utilizado o **timeit** para medir o tempo de execução e utilizamos como parâmetro de repetições o valor 3, após isso, calculamos o tempo médio e salvamos todos os resultados em um dicionário ***results***, onde a **chave** é o nome do algoritmo e o **valor** é uma lista com o tempo médio de execução para cada conjunto de dados.

Com os resultados armazenados em ***results***, executamos algumas funções existentes em uma classe **GenerateInfo**, que criamos algumas funções de geração de txt, e gráficos com as informações de desempenho que iremos apresentar no tópico 3.

## 3. Resultados do Teste de Desempenho

Rodamos os testes com todos os conjuntos de dados inicialmente, 1k, 10k, 50k e 500k, no entanto, o algoritmo não terminou de rodar, então decidimos fazer a análise primeiro com 1k e 10k.



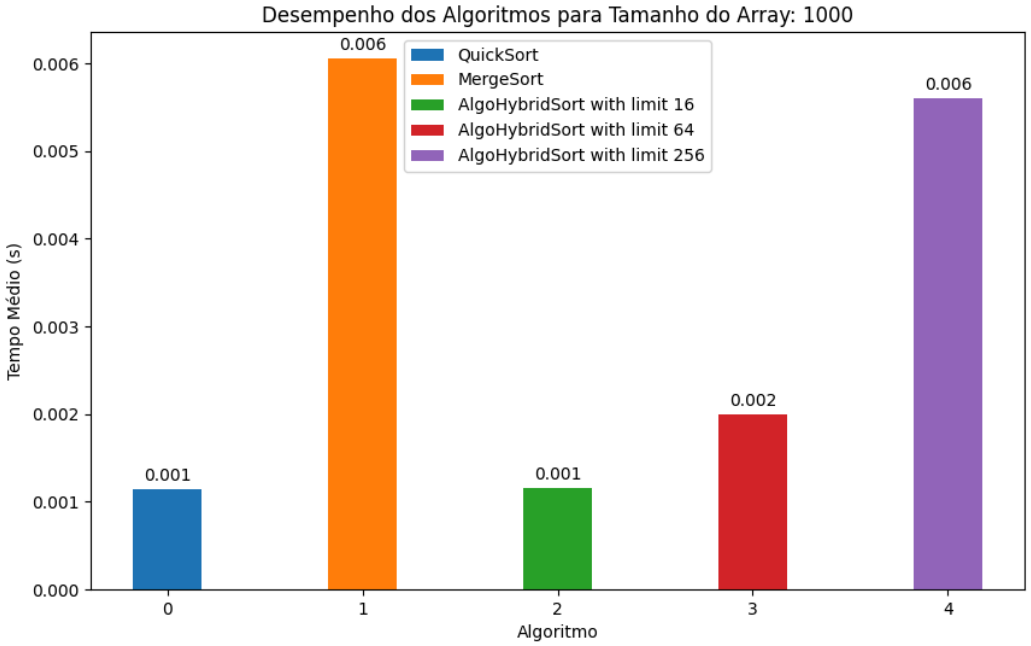

No gráfico de barras, podemos perceber que a visualização ficava inviável com o algoritmo SelectionSort, pois demora muito mais que os outros.
Então resolvemos analisar o txt, que retornou o seguinte cenário:

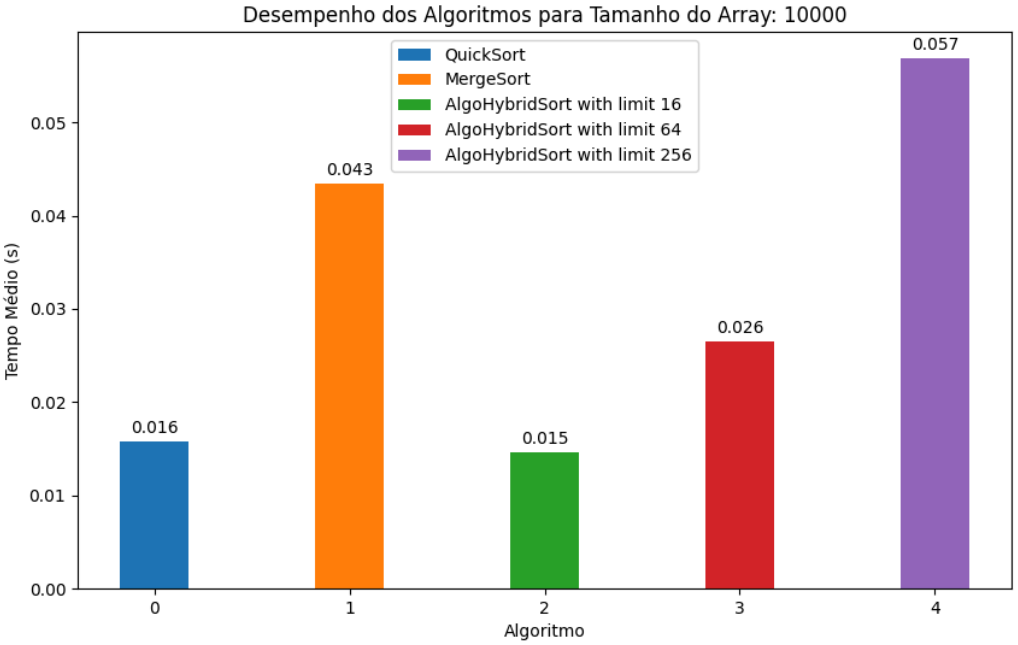
Resultados do Teste de Desempenho:
==========
Tamanho do Array: 1000
SelectionSort: 0.021789 segundos
QuickSort: 0.001506 segundos
MergeSort: 0.002323 segundos
AlgoHybridSort with limit 16: 0.002326 segundos
AlgoHybridSort with limit 64: 0.004490 segundos
AlgoHybridSort with limit 256: 0.008606 segundos
==========
Tamanho do Array: 10000

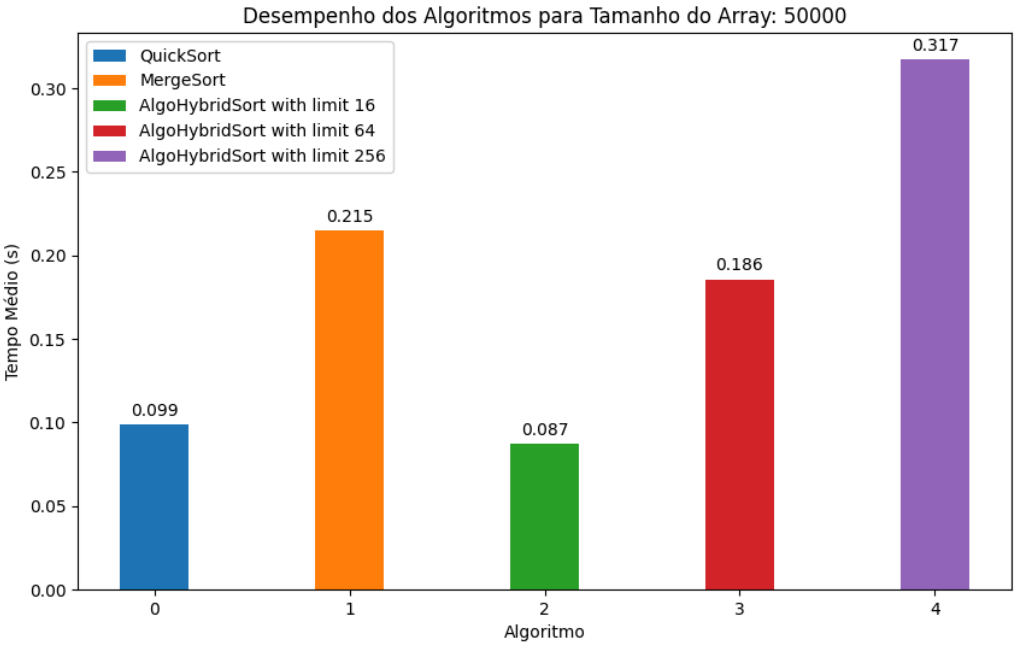
SelectionSort: 3.276043 segundos
QuickSort: 0.021590 segundos
MergeSort: 0.033271 segundos
AlgoHybridSort with limit 16: 0.035868 segundos
AlgoHybridSort with limit 64: 0.042317 segundos
AlgoHybridSort with limit 256: 0.085898 segundos
==========

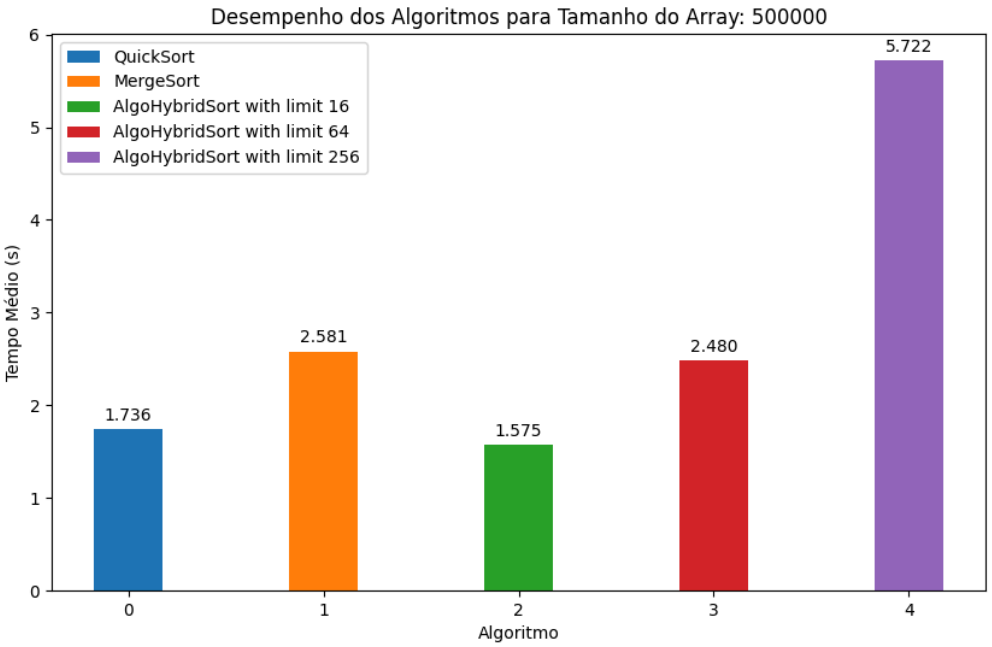
Podemos perceber que com o **SelectionSort** demora muito tempo mais que os outros, então já resolvemos descartar dos demais testes, pois ele é de longe o algoritmo mais custoso quando se trata de uma quantidade grande de dados.

Segue abaixo então os resultados aplicando os conjuntos de 1k, 10k, 50k e 500k, para os algoritmos QuickSort, MergeSort, AlgoHybridSort com limite 16, AlgoHybridSort com limite 64 e AlgoHybridSort com limite 256.




Desempenho dos Algoritmos para Tamanho do Array: 10000
QuickSort
MergeSort
AlgoHybridSort with limit 16
AlgoHybridSort with limit 64
AlgoHybridSort with limit 256
Tempo Médio (s)
Algoritmo
0.016
0.043
0.015
0.026
0.057
0.00
0.01
0.02
0.03
0.04
0.05
0
1
2
3
4


Desempenho dos Algoritmos para Tamanho do Array: 50000
QuickSort
MergeSort
AlgoHybridSort with limit 16
AlgoHybridSort with limit 64
AlgoHybridSort with limit 256
Tempo Médio (s)
Algoritmo
0.099
0.215
0.087
0.186
0.317
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
1
2
3
4




Segue também em formato de texto:

Resultados do Teste de Desempenho:

==========

Tamanho do Array: 1000

QuickSort: 0.001136 segundos

MergeSort: 0.006056 segundos

AlgoHybridSort with limit 16: 0.001150 segundos

AlgoHybridSort with limit 64: 0.001993 segundos

AlgoHybridSort with limit 256: 0.005600 segundos

==========

Tamanho do Array: 10000

QuickSort: 0.015765 segundos

MergeSort: 0.043437 segundos

AlgoHybridSort with limit 16: 0.014624 segundos

AlgoHybridSort with limit 64: 0.026452 segundos

AlgoHybridSort with limit 256: 0.056896 segundos

==========

Tamanho do Array: 50000

QuickSort: 0.098514 segundos

MergeSort: 0.214706 segundos

AlgoHybridSort with limit 16: 0.087253 segundos

AlgoHybridSort with limit 64: 0.185685 segundos

AlgoHybridSort with limit 256: 0.317309 segundos

==========

Tamanho do Array: 500000

QuickSort: 1.736274 segundos

MergeSort: 2.581242 segundos

AlgoHybridSort with limit 16: 1.574800 segundos

AlgoHybridSort with limit 64: 2.480166 segundos

AlgoHybridSort with limit 256: 5.722149 segundos

==========

## 4. Discussão dos resultados

O objetivo é analisar e comparar os resultados do algoritmo híbrido em relação aos demais, porém já podemos novamente falar sobre o SelectionSort, individualmente é o pior desempenho entre os algoritmos utilizados, ele é bastante eficaz para conjuntos pequenos, mas à medida que o conjunto de dados vai aumentando, seu desempenho piorando drasticamente.

Analisando o algoritmo híbrido, é possível perceber que o tempo médio de execução em comparação aos demais vai depender do limite utilizado para acionar o SelectionSort na sublista, nos casos onde utilizamos os limites 16, 64 e 256, é possível ver que 16 é o melhor caso, na medida que aumenta o limite para 64, o tempo de execução também aumenta, ou seja, o desempenho cai.

Dito isso, o algoritmo híbrido com limite 16, é o ideal para os nossos testes com conjunto de tamanhos 1k, 10k, 50k e 500k. Comparando ele com os demais algoritmos podemos ver que ele é melhor que o MergeSort em todos os casos, porém ele se aproxima muito do desempenho do QuickSort em todos os casos também. Porém o algoritmo híbrido supera o desempenho do do QuickSort, pois quando aplicamos o SelectionSort nas sublistas de tamanho até 16, temos um ganho de desempenho.

Resumindo, segue um ranking do melhor ao pior algoritmo com base em nossos testes:

| Ranking | Algoritmo |
|---|---|
| 1º | AlgoHybridSort with limit 16 |
| 2º | QuickSort |
| 3º | AlgoHybridSort with limit 64 |
| 4º | MergeSort |
| 5º | AlgoHybridSort with limit 256 |
| 6º | SelectionSort |