
UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

RAFAELLE LARISA DE ARRUDA

# TRABALHO DE COMPARAÇÃO DOS MÉTODOS DE ORDENAÇÃO LINEARÍTMICOS

CHAPECÓ
2021

## INTRODUÇÃO

É de todo conhecimento geral que, quando trabalhamos com listas, existem ocasiões em que necessitamos ordená-las para facilitar as pesquisas e mostrar sua qualidade em alguma precisão, tanto para alto nível quanto para baixo custo. Neste trabalho foi focado entender que existem alguns métodos utilizados para ordenar vetores (listas e/ou matrizes). São eles: Merge Sort, Quick Sort e Heap Sort, cada implementação destes itens tem  diferenças tanto na performance quanto no tempo de processamento para ordenar os dados. Para isso, foi um desafio imenso trabalhar com esses métodos para verificar o tempo estimado de cada ordenação e verificando qual é o melhor em sentido de organização precisa.

**Quick Sort**

A princípio o Quick sort  funciona com vetor ordenado de uma sequência qualquer de valores dividindo em subsequências menores, aplicando recursão para ordenar cada uma destas subsequências e vinculando novamente em uma sequência idêntica à original, porém, já ordenada.

Mas o objetivo desse método é baseado em uma rotina fundamental cujo nome é particionamento.  Essa metodologia de "particionamento" significa que pode escolher um número qualquer presente numa array, chamado o pivot, e colocá-lo em uma posição tal que todos os elementos à esquerda são menores ou iguais à direita que são maiores.

Em análise, podemos observar que  o algoritmo demonstrou os melhores resultados quanto ao tempo de execução, iniciando com valores de tempo “0” são resultados onde o tempo foi consideravelmente muito baixo e ordenando instantâneamente os dados.

| CATEGORIA | TEMPO (s) 10K | TEMPO (s) 50K | TEMPO (s) 100K |
|---|---|---|---|
| Crescente | 0.001366 | 0.005067 | 0.010502 |
| Decrescente | 0.001375 | 0.006631 | 0.013606 |
| Aleatório | 0.003838 | 0.013271 | 0.024678 |

Tabela 1. Tempo do algoritmo Quick sort.

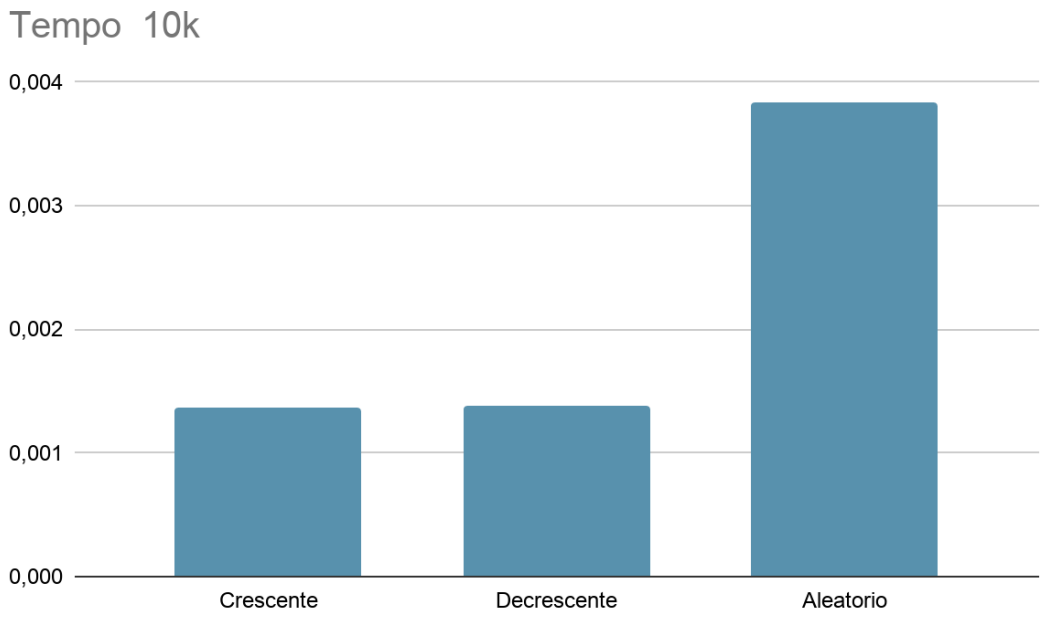


Gráfico 1. Tempo em segundos com dados de 10K.

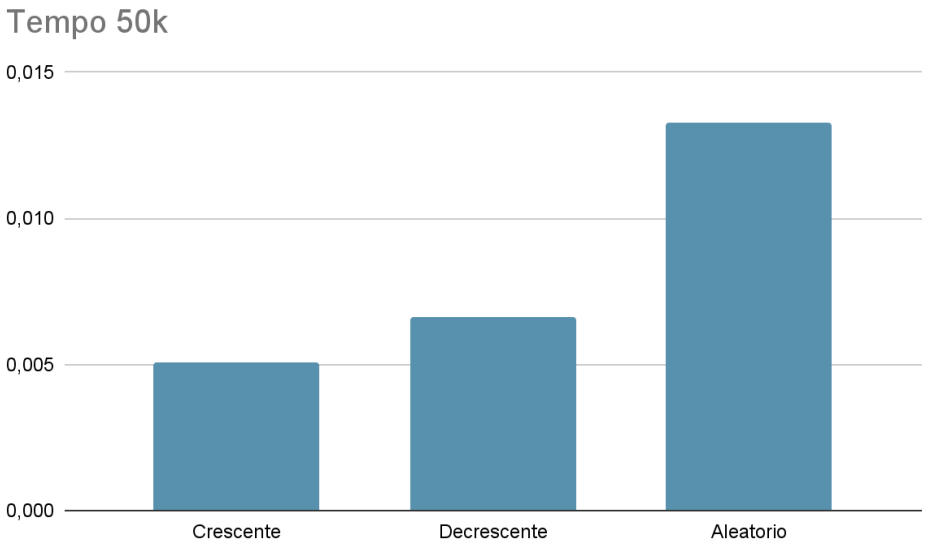


Gráfico 2. Tempo em segundos com dados de 50K.

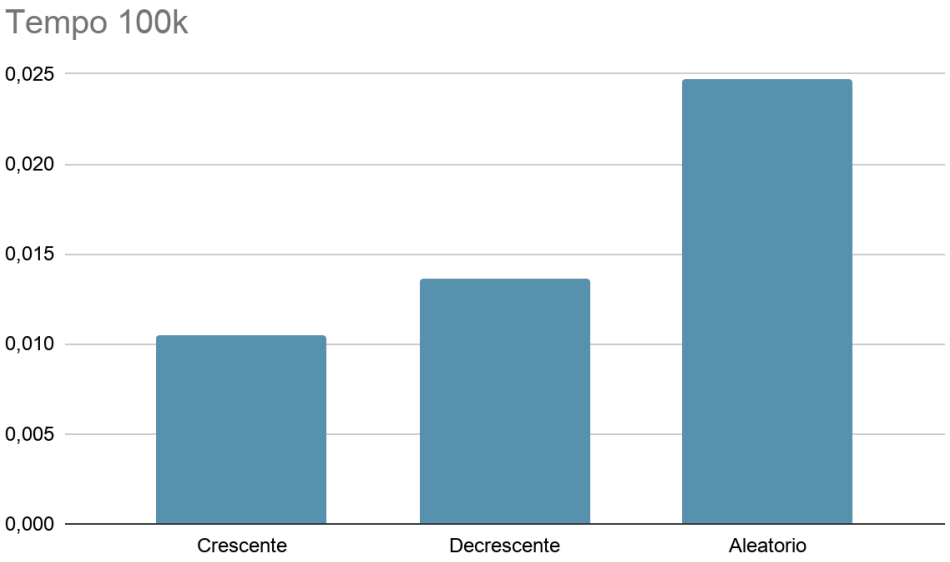


Gráfico 3. Tempo em segundos com dados de 100K.

**Merge Sort**

O intuito do merge sort é basicamente gerar uma sequência ordenada a partir de duas outras também ordenadas. Mas o algoritmo divide a sequência original em pares de dados, agrupando estes pares na ordem desejada e logo depois as agrupa as sequências de pares já ordenados, produzindo uma nova sequência de dados ordenados até ter toda a sequência ordenada.

No caso deste método, há uma característica importante é que sua eficiência é $n*\log n$ para o melhor, pior e para o caso médio. Ou seja, ele não é somente $\Omega(n*\log n)$, mas é $\Theta(n*\log n)$. Com isso temos a garantia de que, independente da disposição dos dados em um array, a ordenação será eficiente.

Mas entender melhor o processo do algoritmo, devemos seguir os seguintes passos que se aplicam ao método, são eles:

I. Dividir os dados em subsequências pequenas, iniciando com a divisão do vetor de n elementos em duas metades, cada uma das metades é novamente dividida em duas novas metades, até que não haja mais possível a divisão entre n vetores que sobrarem.
II. Classificando as duas metades recursivamente aplicando o merge sort.
III. Inserindo duas metades em um único conjunto para completar a ordenação do vetor original de n elementos, faz-se o merge ou a fusão dos subvetores já ordenados.

Em análise o tempo de execução, o algoritmo mostrou no “meio termo” entre o quick sort e o heap sort, mas ainda obteve um tempo de baixo custo comparado com método heap sort.

| CATEGORIA | TEMPO (s) 10K | TEMPO (s) 50K | TEMPO (s) 100K |
|---|---|---|---|
| Crescente | 0.004009 | 0.013264 | 0.023130 |
| Decrescente | 0.004085 | 0.015668 | 0.024139 |
| Aleatório | 0.005008 | 0.019950 | 0.034493 |

Tabela 2. Tempo gasto pelo algoritmo Merge Sort.

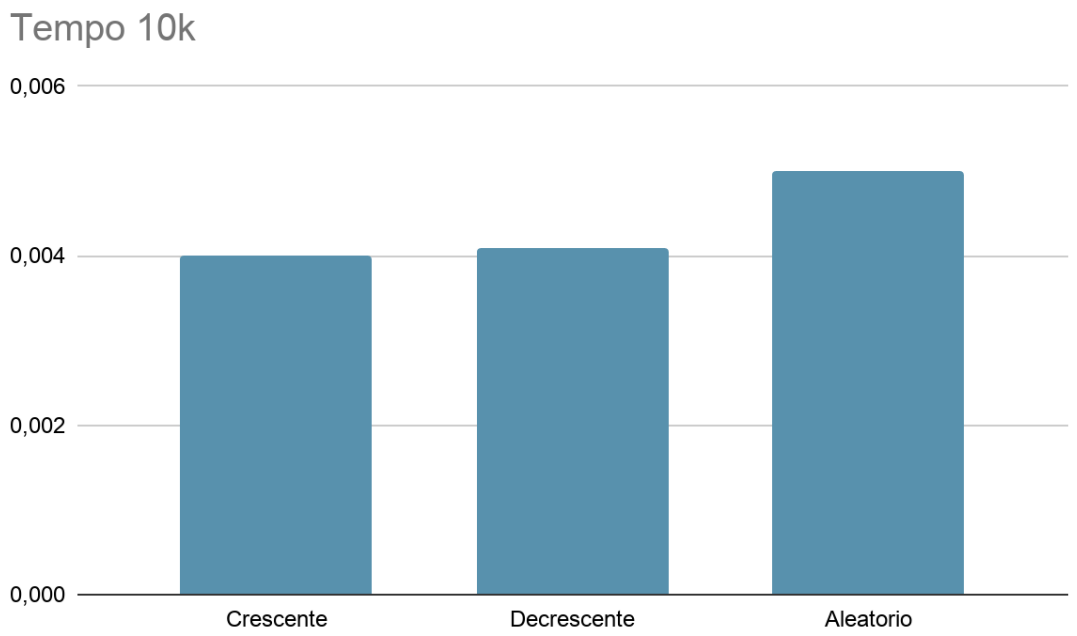


Gráfico 4. Tempo em segundos com dados de 10K.

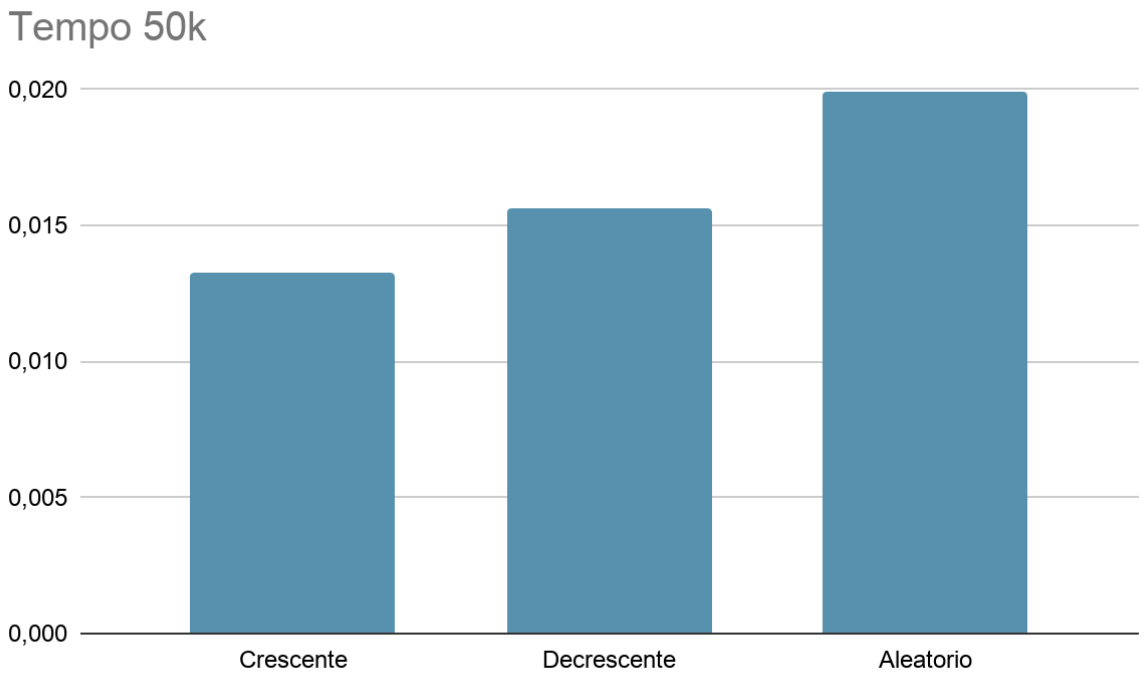


Gráfico 5. Tempo em segundos com dados de 50K.

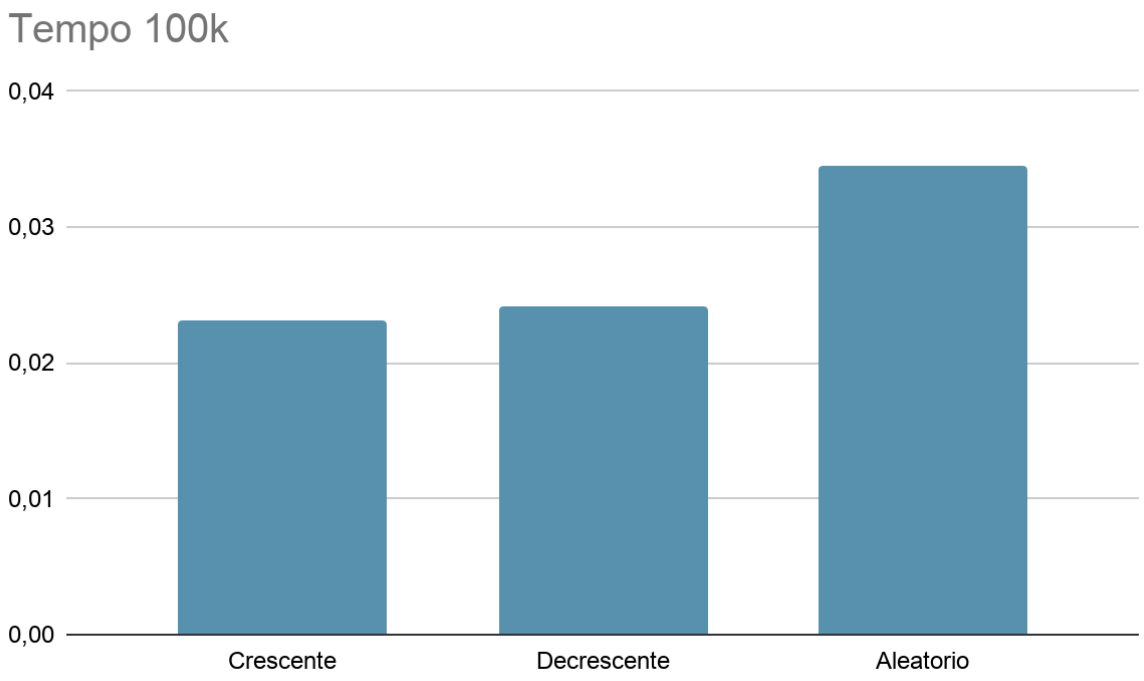


Gráfico 6. Tempo em segundos com dados de 100K.

**Heap Sort**

Analisando este método, podemos perceber que o objetivo é usar uma estrutura de dados chamada heap binário para ordenar os elementos à medida que os insere na estrutura. Com o final das inserções dos elementos, podem ainda ser sucessivamente removidos da raiz, na ordem que deseja. É semelhante à ordenação por seleção, onde primeiro encontramos o elemento mínimo e colocamos o elemento mínimo no início. Repetimos o mesmo processo para os elementos restantes.

O heap binário é uma árvore binária completa que pode ser facilmente representada como uma matriz e sendo eficiente em termos de espaço. Se o nó pai for armazenado no índice I, o filho esquerdo pode ser calculado por 2 * I + 1 e o filho direito por 2 * I + 2 (assumindo que a indexação começa em 0).

Podemos concluir que o algoritmo demonstrou resultados extraordinários e comparados com os demais, porém que o tempo de execução foi estável observando a tabela de execução em segundos.

| CATEGORIA | TEMPO (s)<br>10K | TEMPO (s)<br>50K | TEMPO (s)<br>100K |
|---|---|---|---|
| Crescente | 0.004352 | 0.021662 | 0.042638 |
| Decrescente | 0.005976 | 0.025619 | 0.044584 |
| Aleatório | 0.006734 | 0.027035 | 0.052866 |

Tabela 3. Tempo gasto pelo algoritmo Heap Sort.

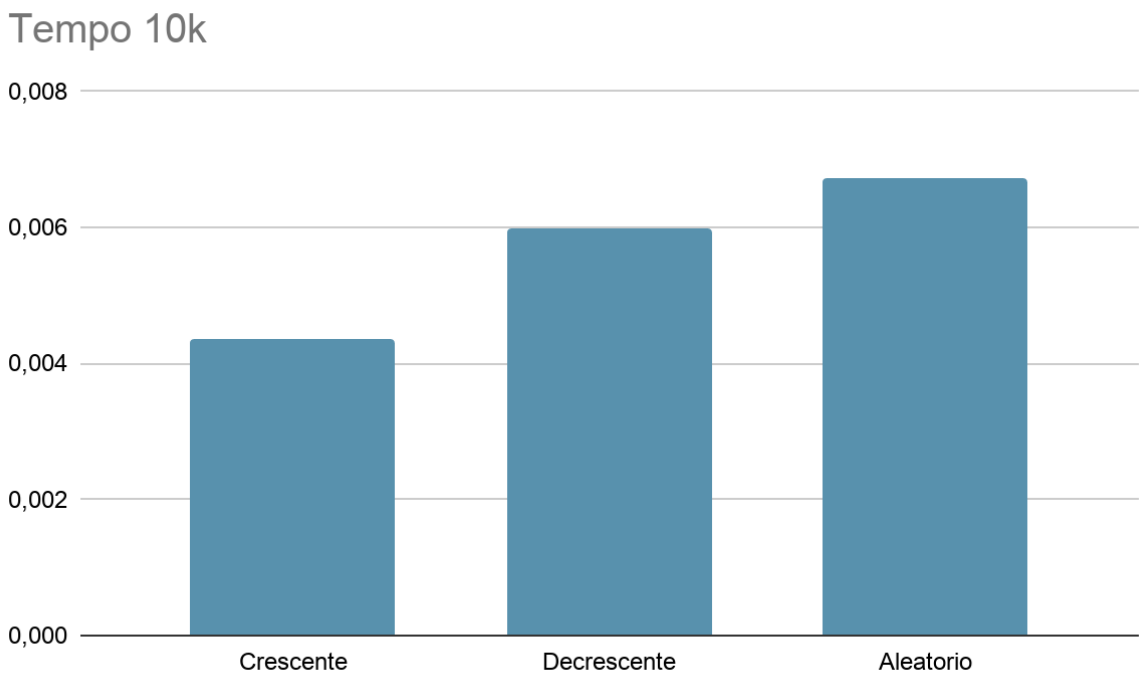


Gráfico 7. Tempo em segundos com dados de 10K.

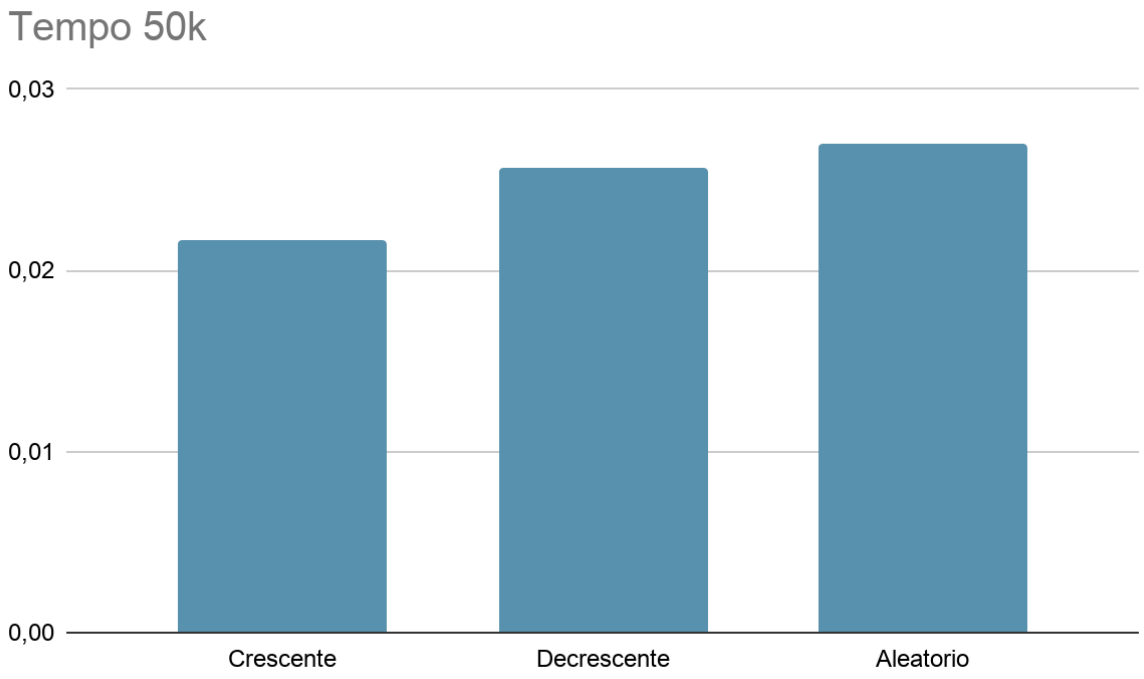


Gráfico 8. Tempo em segundos com dados de 50K.

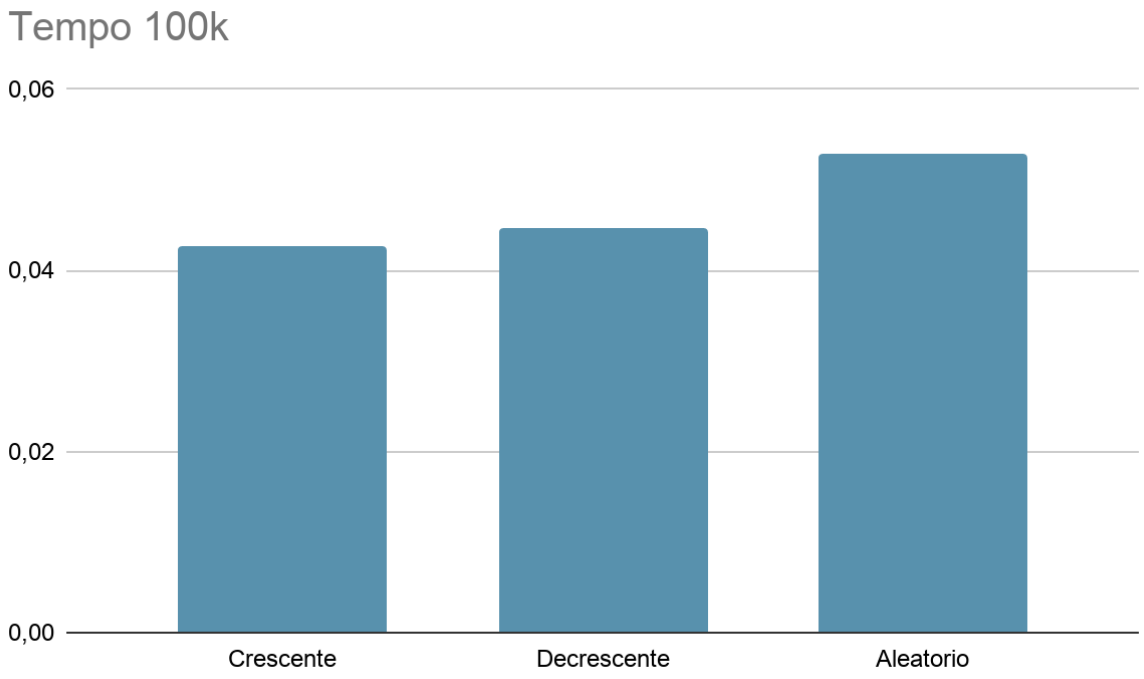


Gráfico 9. Tempo em segundos com dados de 100K.

## Metodologia

O intuito deste trabalho foi entender cada método que foi apresentado nas aulas, saber identificar em qual ocasião em utilizar e lidar com uma importante tarefa, quando se trata de dados ou registro, sempre procuramos demonstrar a melhor performance e rapidez, como podemos perceber que alguns métodos têm pontos positivos e negativos, podendo usar em melhor caso e pior caso. Mas depende muito da tarefa que irá desenvolver. Neste caso como acadêmica, tive que entender cada método e porque sua importância, fazendo pequenas implementações para processar o conhecimento que estava adquirindo, mas analisando a questão do hardware e tempo de execução, juntamente com as informações recolhidas, que para um melhor resultado ao ordenar grandes dados, o melhor e ter um ambiente dedicado a essa tarefa, onde a execução do algoritmo e o tempo de execução do mesmo não sofreria influência de processos externos, obtendo assim maior aproveitamento dos recursos de hardware para determinação função.

## Detalhes

Máquina utilizada para a realização da atividade foi um Notebook Acer, com processador Intel Core i3 5005U 2.0ghz; 4GB de memória RAM e 500 TB de HD, placa de vídeo dedicada intel HD graphics 5500,up to 2005MB dynamic memory e Sistema Operacional Ubuntu 64 bits. Para a compilação e desenvolvimento dos algoritmos de teste foi usado o vscode. No momento dos testes, alguns outros programas estavam em execução em segundo plano, portanto pode haver uma diferença quanto ao tempo de execução uma vez que o processamento e a memória RAM estavam sendo usados por outros aplicativos.

## Conclusão

De acordo com este trabalho, foi estudado a estética e a semântica do tempo de cada função, mas em análise o algoritmo que ganhou destaque foi o algoritmo quicksort que obteve melhor resultados, sendo que os concorrentes também tendo tempos bem baixos. A princípio o equipamento (hardware) usado para gerar os testes influência nos resultados de análise de tempo, em primeiro momento da execução as ordenações em todos os algoritmos, foram realizadas instantaneamente, com isso podendo ter uma desconfiança nos resultados nas análises anteriores do trabalho 1 com bubble sort, insertion sort e selection sort às execuções levavam um tempo considerável para terminar.

De acordo com o tempo baixo de execução dos métodos, podemos perceber que muitas plataformas usam algumas dessas metodologias para agilidade de processamento dados para que chegue ao seu respectivo usuário em segundos.