Algoritmos e Estruturas de Dados II
Laboratório 06 - QuickSort e seu pivô

October 6, 2024

Felipe Barros Ratton de Almeida
Código de Pessoa: 732683
Turma: 9956200

---

1 Proposta

Implementar o algoritmo QuickSort utilizando diferentes estratégias para a escolha do pivô, visando analisar o impacto dessas escolhas no desempenho do algoritmo.

2 Objetivo

O objetivo principal é entender como diferentes estratégias de escolha do pivô afetam o desempenho do QuickSort em diferentes tipos de arrays. Somado a isso, realizar comparações entre as diferentes implementações e relatar os resultados obtidos.

2.1 Estratégias de escolha do pivô

O algoritmo QuickSort foi implementado utilizando as seguintes estratégias para a escolha do pivô:

- **Primeiro elemento**: Utilizar o primeiro elemento do array como pivô.
- **Último elemento**: Utilizar o último elemento do array como pivô.
- **Pivô aleatório**: Selecionar um pivô de forma aleatória a cada execução.
- **Mediana de três elementos**: Utilizar a mediana entre o primeiro, o último e o elemento central do array como pivô.

2.2 Implementação

Cada versão do QuickSort foi implementada em uma função separada, sendo elas as seguintes:

```
static void QuickSortFirstPivot(int[] array, int left, int right);
static void QuickSortLastPivot(int[] array, int left, int right);
static void QuickSortRandomPivot(int[] array, int left, int right);
static void QuickSortMedianOfThree(int[] array, int left, int right);
```

2.3 Análise de Desempenho

A análise de desempenho das diferentes implementações do QuickSort foi realizada da seguinte forma:

- Executando cada uma das versões do QuickSort utilizando arrays de tamanhos variados, como 100, 1.000 e 10.000 elementos.
- Utilizando diferentes tipos de arrays: ordenados, quase ordenados e aleatórios.
- Comparando o tempo de execução de cada estratégia de escolha do pivô para os diferentes tipos e tamanhos de arrays.

3 Funcionamento de Cada Estratégia de Escolha do Pivô

3.1 Primeiro Elemento

Neste método, o primeiro elemento do array é escolhido como pivô. Isso pode levar a um desempenho ruim $O(n^2)$ em arrays já ordenados ou quase ordenados, pois o pivô pode ser sempre o menor ou maior elemento, resultando em divisões desequilibradas.

3.2 Último Elemento

A escolha do último elemento como pivô apresenta problemas semelhantes ao primeiro elemento. Em arrays ordenados, isso também resulta em uma performance ruim, pois pode levar a divisões desiguais.

3.3 Pivô Aleatório

Um pivô escolhido aleatoriamente pode ajudar a evitar os piores casos que ocorrem com as escolhas de primeiro ou último elemento. No entanto, ainda há a possibilidade de divisões desequilibradas, mas em média, essa estratégia oferece um desempenho melhor $O(n \log n)$.

3.4 Mediana de Três Elementos

Este método calcula a mediana de três valores: o primeiro, o do meio e o último do array. Isso tende a fornecer um pivô mais equilibrado e pode melhorar o desempenho em casos de arrays já ordenados, resultando em divisões mais balanceadas.

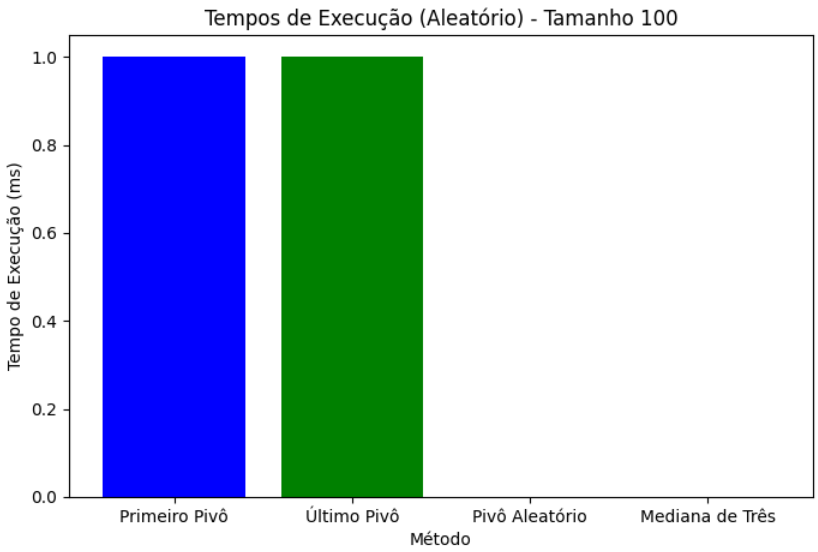
4 Desempenho Observado em Cada Cenário

O código para teste de desempenho do algoritmo foi construído em JAVA, e os gráficos em Python. No final do relatório, se encontram ambos e o arquivo csv com os dados completos do desempenho de cada algoritmo apresentado no documento.

4.1 Array de Tamanho 100

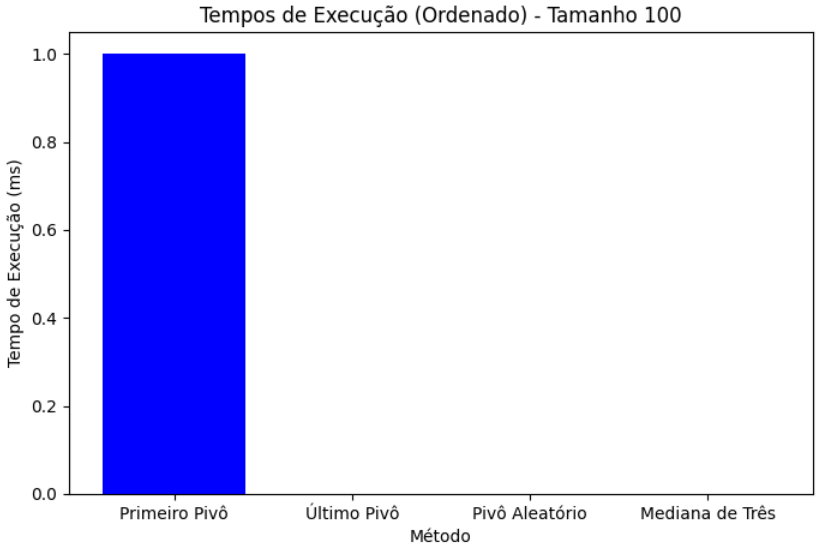
4.1.1 Array Aleatório

Para um array de tamanho 100 com valores dispostos de forma aleatória, os métodos *Mediana de Três* e *Pivô Aleatório* tiveram tempos iguais, ambos com execução extremamente rápida (0 ms). O *Primeiro Pivô* e o *Último Pivô* também apresentaram tempos muito próximos, porém ligeiramente mais altos (1 ms), com o *Último Pivô* sendo marginalmente mais lento.



4.1.2 Array Ordenado

No array ordenado de tamanho 100, tanto o *Primeiro Pivô* quanto o *Último Pivô* tiveram um desempenho de 1 ms e 0 ms, respectivamente, apesar de ambos envolverem muitas comparações internas. O *Mediana de Três* e o *Pivô Aleatório* também tiveram desempenho extremamente eficiente, com ambos os métodos completando a execução em 0 ms.



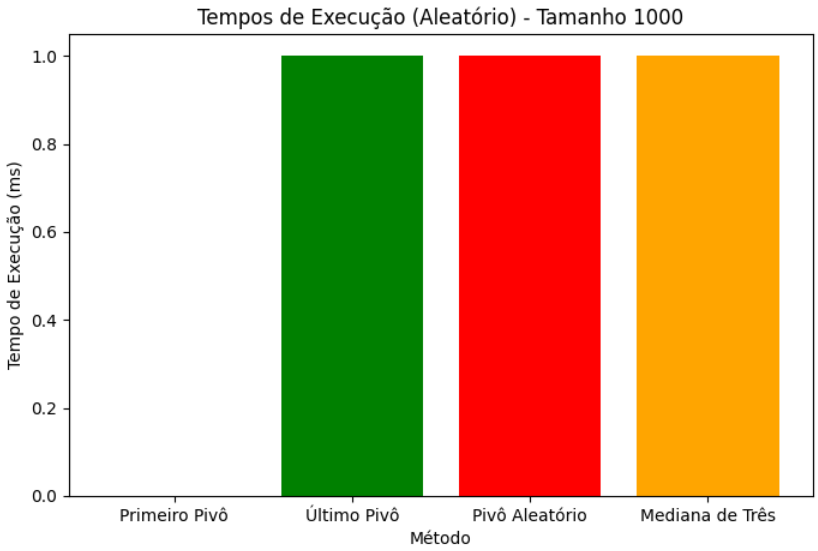
4.1.3 Array Quase Ordenado

Para o array quase ordenado de tamanho 100, todos os métodos se comportaram de forma muito rápida com tempo de 0 ms.

4.2 Array de Tamanho 1000

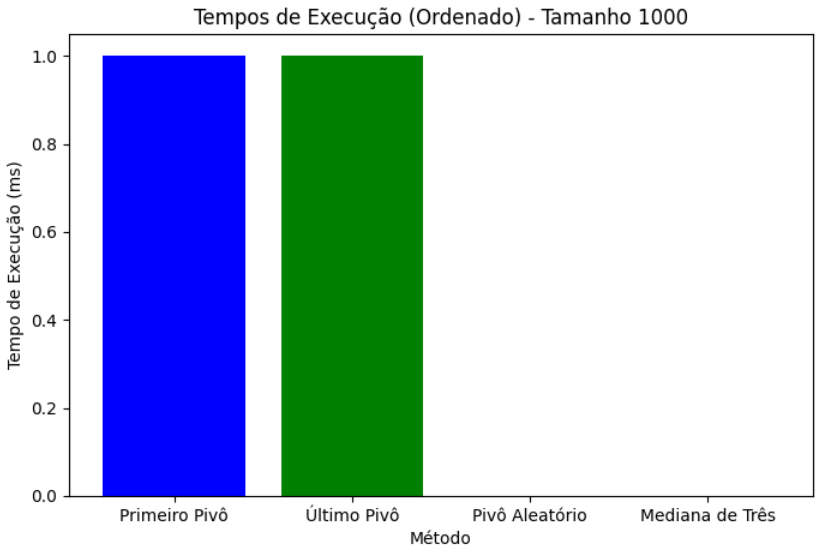
4.2.1 Array Aleatório

Com um aumento no tamanho do array para 1000 elementos aleatórios, os tempos de execução de todos os métodos foram muito próximos, com variações mínimas. A *Mediana de Três*, *Pivô Aleatório* e *Último Pivô* completaram em 1 ms. Já o *Primeiro Pivô*, surpreendentemente completou em 0 ms (se levando em conta em nanosegundos, estaria mais proximo de 1 ms, já que seu numero de comparações e movimentações é próximo dos outros).



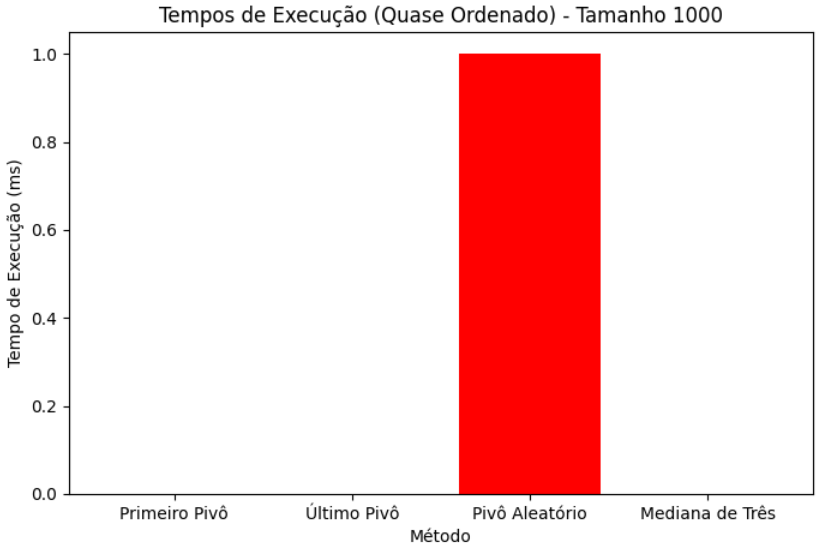
4.2.2 Array Ordenado

Para arrays de 1000 elementos ordenados, o desempenho caiu drasticamente para o métodos *Primeiro Pivô*, onde ele e *Último Pivô* se igualaram, demorando 1 ms, processando muitas comparações. A *Mediana de Três* se destacou, finalizando em 0 ms, e o *Pivô Aleatório* levou 0 ms também.



4.2.3 Array Quase Ordenado

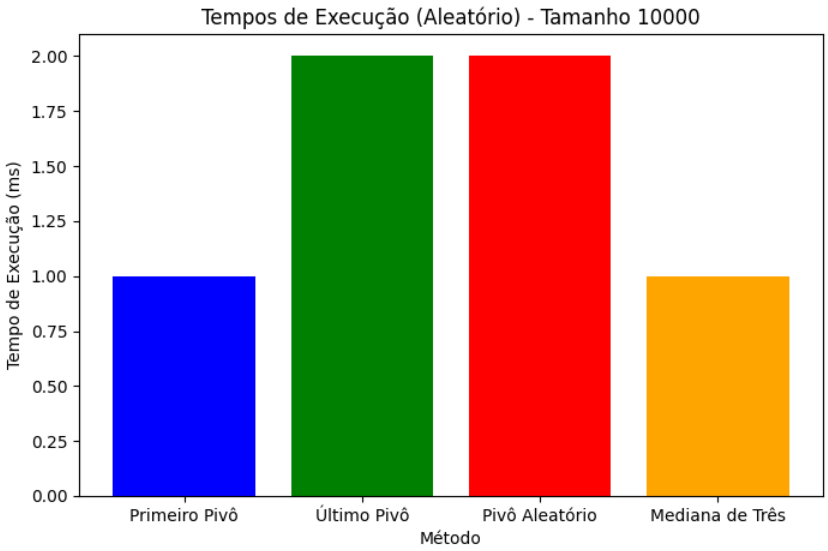
Para arrays quase ordenados de 1000 elementos, a *Mediana de Três* foi novamente o método mais eficiente, levando 0 ms. O *Pivô Aleatório* levou um pouco mais de tempo, finalizando em 1 ms. O *Primeiro Pivô* e o *Último Pivô* tiveram resultados semelhantes, finalizando em 0 ms, porém com bem mais de comparações e movimentações que a mediana.



4.3 Array de Tamanho 10000

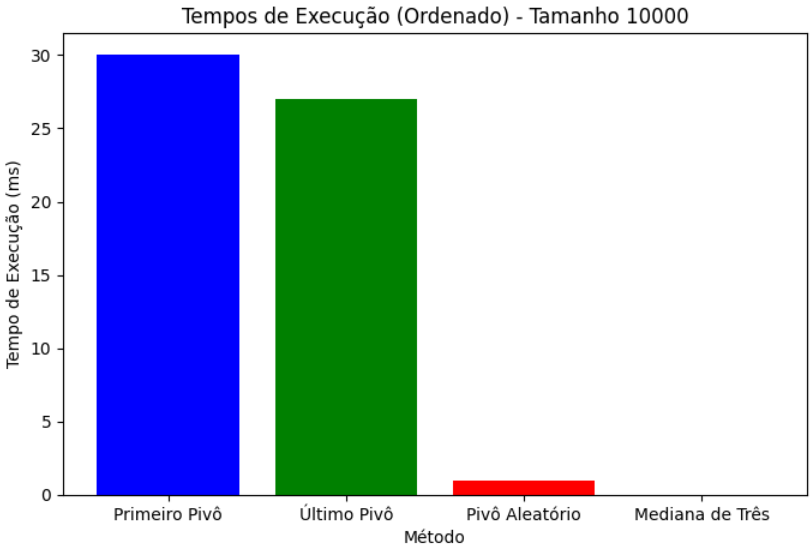
4.3.1 Array Aleatório

Para um array aleatório de tamanho 10000, os métodos *Mediana de Três* e *Primeiro Pivô* foram os mais rápidos, completando em 1 ms. Os métodos *Primeiro Pivô* e *Último Pivô* levaram tempos maiores, sendo os mais lento, levando 2 ms.



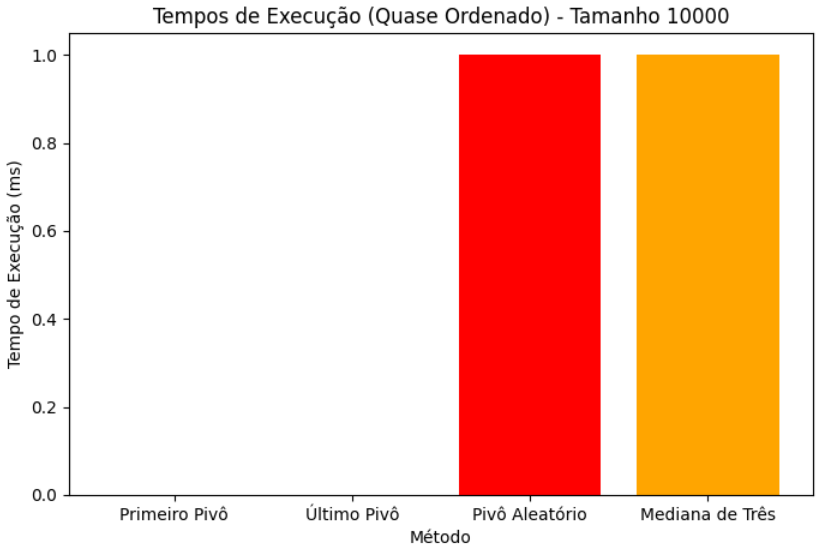
4.3.2 Array Ordenado

Nos arrays ordenados de 10000 elementos, o *Primeiro Pivô* e o *Último Pivô* apresentaram tempos significativamente maiores (30 ms e 27 ms, respectivamente). A *Mediana de Três* foi a mais eficiente, finalizando em 0 ms, e o *Pivô Aleatório* completou em 1 ms.



4.3.3 Array Quase Ordenado

Em arrays quase ordenados de tamanho 10000, a *Mediana de Três* e o*Pivô Aleatório* apresentaram o pior desempenho, completando em 1 ms. Os métodos *Primeiro Pivô* e *Último Pivô* foram ligeiramente mais rapidos, finalizando em 0 ms.



5 Discussão e Conclusão

Com base nos dados fornecidos sobre os diferentes tipos de pivôs usados no algoritmo de ordenação, pode-se concluir o seguinte sobre a eficiência das estratégias:

5.1 Primeiro Pivô e Último Pivô

Ambos os métodos são simples, mas apresentam desvantagens em situações em que o array já está ordenado ou quase ordenado. Isso ocorre porque, nesses casos, o pivô escolhido resulta na maior profundidade de recursão, levando a um comportamento quase quadrático ($O(n^2)$) no tempo de execução, conforme indicado pelos números elevados de comparações e trocas.

5.2 Pivô Aleatório

Este método tem um desempenho mais robusto, especialmente em arrays desordenados, porque randomizar o pivô reduz a chance de escolher um valor ruim repetidamente. Ele também é menos suscetível a problemas com arrays já ordenados, embora não tenha o mesmo desempenho ótimo quanto o método da Mediana de Três em alguns casos.

5.3 Mediana de Três

A estratégia da Mediana de Três se mostrou a mais eficiente na maioria dos casos, principalmente em arrays ordenados e quase ordenados. Ela minimiza os casos ruins (quando o pivô é um valor extremo), resultando em menos comparações e trocas. A escolha do pivô utilizando a mediana dos três valores do início, meio e fim do array é vantajosa pois tende a escolher um valor próximo da mediana real do array, reduzindo a profundidade da recursão.

6 Conclusão

A estratégia mais eficiente, conforme observado nos experimentos, é a *Mediana de Três*, especialmente em arrays que já estão parcialmente ordenados. Isso ocorre porque ela faz uma escolha de pivô mais próxima da mediana real, reduzindo a quantidade de comparações e trocas necessárias. O *Pivô Aleatório* também mostrou bons resultados em arrays completamente desordenados, enquanto os métodos de *Primeiro Pivô* e *Último Pivô* sofrem em cenários ordenados, onde o tempo de execução se degrada para $O(n^2)$. Portanto, para maximizar a eficiência, o uso da *Mediana de Três* seria a escolha preferida em uma implementação de Quicksort para diferentes tipos de dados.

Baixar os códigos aqui!