Funcionamento de cada Estratégia de Escolha do Pivô

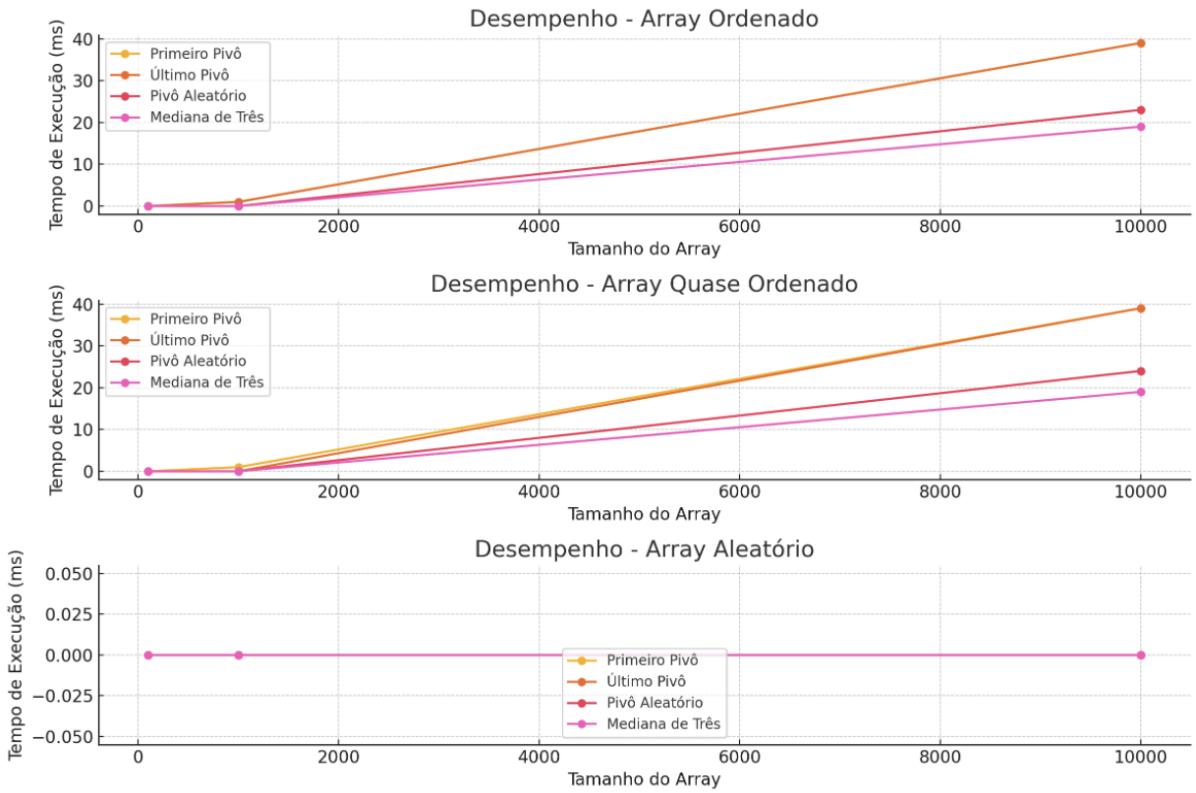
1. **Primeiro Pivô**:
    - Método é fácil de implementar, mas pode levar a partições muito desbalanceadas.
    - **Problema**: Quando o array já está ordenado, a escolha do primeiro pivô resulta em uma divisão altamente desbalanceada, criando uma estrutura semelhante a uma lista encadeada e aumentando o número de comparações para O(nˆ2)

2. **Último Pivô**:
    - Assim como o primeiro pivô, é uma escolha simples, mas também corre o risco de causar partições desbalanceadas.
    - **Problema**: Arrays ordenados ou quase ordenados enfrentam o mesmo problema de partições desbalanceadas, tornando o desempenho pior.

3. **Pivô Aleatório**:
    - Um elemento é selecionado aleatoriamente como pivô em cada partição. Esta estratégia diminui a probabilidade de gerar uma partição desbalanceada e, em média, oferece um bom desempenho.
    - **Vantagem**: A aleatoriedade impede que o algoritmo tenha um pior caso consistente (como ocorre com o primeiro e último pivô em arrays ordenados). Isso resulta em um desempenho mais estável, aproximando-se de O(nlogn).

4. **Mediana de Três (Início, Meio e Fim)**:
    - Esta estratégia calcula o pivô tomando a mediana de três elementos: o primeiro, o elemento do meio e o último elemento do subarray. O objetivo é melhorar a chance de uma partição balanceada.
    - **Vantagem**: A mediana de três tende a fornecer uma partição mais equilibrada do que o primeiro ou último pivô, especialmente em arrays que já estão ordenados ou quase ordenados, prevenindo o pior caso.

Desempenho Observado em Cada Cenário



Gráficos de Desempenho

1. **Array Ordenado**:
    - Para arrays ordenados, a mediana de três foi a estratégia mais eficiente, com tempos de execução significativamente menores. O primeiro e o último pivô foram as estratégias mais lentas.
2. **Array Quase Ordenado**:
    - Novamente, a mediana de três foi a estratégia mais eficiente, com o pivô aleatório apresentando um desempenho um pouco inferior, mas ainda bom. O primeiro e último pivô tiveram desempenhos pobres.
3. **Array Aleatório**:
    - Para arrays aleatórios, todas as estratégias tiveram tempos de execução muito semelhantes, mostrando que a escolha do pivô é menos crítica quando o array está desordenado.

Estratégia Mais Eficiente

1. **Array Ordenado e Quase Ordenado**:
    - A mediana de três foi a estratégia mais eficiente, pois garantiu uma partição mais equilibrada mesmo em cenários adversos, como arrays já ordenados. Isso reduziu o número de comparações, resultando em menor tempo de execução.
    - O pivô aleatório também teve um bom desempenho, sendo uma estratégia que previne o pior caso em arrays ordenados, mas não foi tão eficiente quanto a mediana de três.
    - As estratégias de primeiro e último pivô tiveram os piores desempenhos, devido ao fato de resultarem em partições desbalanceadas em arrays ordenados, aumentando a complexidade do algoritmo para O(n^2).

2. **Array Aleatório**:
    - Em arrays aleatórios, a escolha do pivô foi menos relevante, já que qualquer pivô tende a resultar em partições equilibradas. As quatro estratégias tiveram tempos de execução muito semelhantes, com a mediana de três e o pivô aleatório ligeiramente mais eficientes em arrays maiores.

Conclusão

A estratégia de mediana de três foi consistentemente a mais eficiente em arrays ordenados e quase ordenados, pois garante partições balanceadas, mesmo em casos extremos. O pivô aleatório é uma boa escolha geral para manter a eficiência do QuickSort sem o risco de pior caso. As estratégias de primeiro e último pivô devem ser evitadas em arrays ordenados ou quase ordenados, pois podem causar o pior desempenho possível.