Relatório de Desempenho do QuickSort e seus pivôs

1. Funcionamento de cada estratégia:

O código testa três estratégias de escolha do pivô para o algoritmo QuickSort, com o objetivo de avaliar o desempenho em diferentes cenários. As três estratégias são:

1. **Pivô sendo o primeiro elemento**
    - O pivô é o primeiro elemento do array. Este método é simples, porém tende a ser ineficiente em arrays já ordenados ou quase ordenados, pois resulta em partições desbalanceadas, levando a um tempo de execução mais próximo do pior caso $\theta(n^2)$
2. **Pivô aleatório**:
    - O pivô é selecionado aleatoriamente dentro do intervalo atual. Esta abordagem visa evitar o pior caso, oferecendo melhor performance em arrays parcialmente ordenados, porém pode ainda apresentar problemas se o array tiver um padrão desfavorável.
3. **Mediana de três elementos:**
    - O pivô é calculado como a mediana de três valores (o primeiro, o último e o do meio). Esta abordagem procura melhorar o balanceamento das partições, evitando escolhas de pivôs que possam gerar partições muito desbalanceadas, especialmente em arrays ordenados.

2. Desempenho observado em cada cenário:

2.1 Teste em Arrays Aleatórios

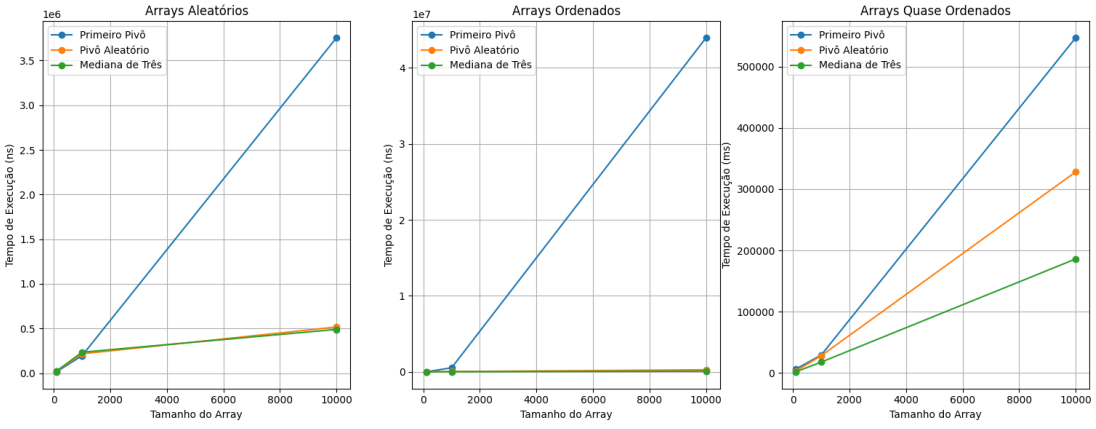
Tamanho	Primeiro Pivô (ns)	Pivô Aleatório (ns)	Mediana de Três (ns)
100	15.042	22.916	24.083
1,000	195.917	216.875	234.708
10,000	3.755.791	517.292	490.167

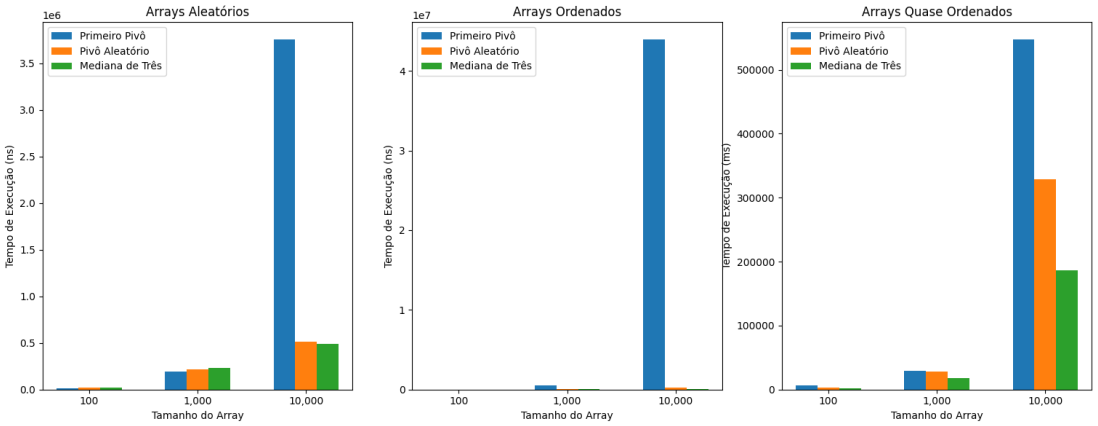
2.2 Teste em Arrays Ordenados

Tamanho	Primeiro Pivô (ns)	Pivô Aleatório (ns)	Mediana de Três (ns)
100	6.750	3.916	2.000
1,000	538.000	40.042	20.583
10,000	43.944.625	278.084	94.625

2.3 Teste em Arrays Quase Ordenados

Tamanho	Primeiro Pivô (ms)	Pivô Aleatório (ms)	Mediana de Três (ms)
100	6.250	3.083	1,.708
1,000	29.166	28.000	17.625
10,000	547.333	328.250	186.083





3. Discussão sobre os dados

3.1 Arrays Aleatórios:

- **Primeiro Pivô**: Esta estratégia foi a mais eficiente para o array de tamanho 10.000, apresentando o menor tempo de execução (3.755.791 ns).
- **Pivô Aleatório**: Foi mais eficiente que a Mediana de Três para arrays de tamanho 100 e 1.000, mas não para o de 10.000, onde o Primeiro Pivô teve um desempenho superior.
- **Mediana de Três**: Embora seja frequentemente considerada uma estratégia mais robusta, teve tempos de execução maiores para os tamanhos 100 e 1.000. No entanto, seu desempenho se aproxima do Pivô Aleatório nos arrays maiores.

> ✅ → **Conclusão**: Nos arrays aleatórios, o **Primeiro Pivô** se mostrou mais eficiente para o tamanho maior (10.000), possivelmente devido à natureza aleatória dos dados, onde a escolha do primeiro elemento como pivô não impacta tanto a performance.

3.2 Arrays Ordenados:

- **Primeiro Pivô**: Desempenho drasticamente pior à medida que o tamanho do array aumenta. Para o array de 10.000, o tempo de execução foi extremamente alto (43.944.625 ns), evidenciando que a escolha do primeiro elemento como pivô em arrays ordenados leva a uma pior divisão dos subarrays (sempre escolhendo o menor ou maior elemento).

- **Pivô Aleatório**: Significativamente mais eficiente do que o Primeiro Pivô, especialmente para arrays maiores (tempo de 278.084 ns para o array de 10.000).
- **Mediana de Três**: Apresentou o melhor desempenho nos arrays ordenados, com tempos menores em todos os tamanhos (2.000 ns para 100 e 94.625 ns para 10.000). Isso ocorre porque a mediana de três minimiza a chance de escolher um pivô ruim, levando a uma melhor divisão.

✅ → **Conclusão**: Nos arrays ordenados, a **Mediana de Três** foi a estratégia mais eficiente, pois evita os problemas que ocorrem quando o pivô é mal escolhido, como no caso do Primeiro Pivô.

3.3 Arrays Quase Ordenados:

- **Primeiro Pivô**: Embora tenha sido a estratégia menos eficiente para os tamanhos maiores (547.333 ms para 10.000), ela ainda manteve um desempenho razoável para os arrays menores.
- **Pivô Aleatório**: Mostrou-se consistentemente bom para arrays de tamanho intermediário (28 ms para 1.000), mas não foi a melhor estratégia para os arrays maiores.
- **Mediana de Três**: Novamente, foi a mais eficiente para os arrays quase ordenados, especialmente no tamanho maior (186.083 ms para 10.000), seguindo a mesma lógica dos arrays ordenados.

✅ → **Conclusão**: Nos arrays quase ordenados, a **Mediana de Três** também se destacou como a estratégia mais eficiente, por ser capaz de lidar melhor com a proximidade da ordenação sem depender de uma única escolha ruim de pivô.

4 Conclusão:

- **Primeiro Pivô** funciona bem em arrays pequenos e aleatórios, mas falha drasticamente em arrays ordenados ou quase ordenados, onde escolhe pivôs inadequados e gera piores casos de complexidade.

- **Pivô Aleatório** é uma estratégia intermediária, tendo um desempenho razoável em todos os casos, mas sem ser a melhor em arrays ordenados ou quase ordenados.
- **Mediana de Três** é a estratégia mais robusta, sendo eficiente especialmente em arrays ordenados e quase ordenados. Apesar de ter um pequeno overhead em arrays menores, sua capacidade de escolher pivôs melhores faz com que seja mais eficiente em casos onde a ordenação inicial é um problema.

Portanto, a **Mediana de Três** foi a estratégia mais eficiente para arrays ordenados e quase ordenados, enquanto o **Primeiro Pivô** teve melhor desempenho apenas em arrays aleatórios de grandes tamanhos.