## 1. Introdução

O algoritmo **QuickSort** é um dos métodos de ordenação mais eficientes na prática, porém seu desempenho depende da escolha do **pivô**.
Nesta atividade, foram implementadas quatro estratégias diferentes de seleção do pivô para investigar como elas influenciam o tempo de execução:

- **First Pivot**: escolhe sempre o **primeiro elemento** do subarray como pivô.
- **Last Pivot**: escolhe sempre o **último elemento** do subarray como pivô.
- **Random Pivot**: escolhe um elemento **aleatório** dentro do subarray.
- **Median of Three**: escolhe a **mediana** entre o primeiro, o elemento central e o último elemento do subarray.

## 2. Funcionamento das Estratégias

1. **Primeiro elemento (First Pivot)**
   O pivô é sempre o primeiro elemento do subarray. Simples, mas pode gerar partições desbalanceadas se o array já estiver quase ordenado.
2. **Último elemento (Last Pivot)**
   Similar à estratégia anterior, mas usa o último elemento. Tem as mesmas limitações de desempenho em arrays quase ordenados.
3. **Pivô aleatório (Random Pivot)**
   Seleciona um índice aleatório dentro do subarray. Essa abordagem tende a evitar partições desbalanceadas, reduzindo a chance de cair no pior caso do QuickSort.
4. **Mediana de três (Median of Three)**
   Calcula a mediana entre o primeiro, o elemento central e o último do subarray e usa esse valor como pivô.
   Geralmente gera partições mais equilibradas, especialmente em arrays parcialmente ordenados.

## 3. Metodologia de Teste

- Linguagem utilizada: Java.
- Arrays testados:
  - **Aleatório**: elementos em ordem totalmente aleatória.
  - **Quase ordenado**: 5% dos elementos trocados aleatoriamente.
  - **Ordenado**: elementos já em ordem crescente.
- Tamanhos dos arrays: **100, 1.000 e 10.000 elementos**.
- Para cada execução, mediu-se o **tempo de ordenação** em segundos.

## 4. Resultados Observados

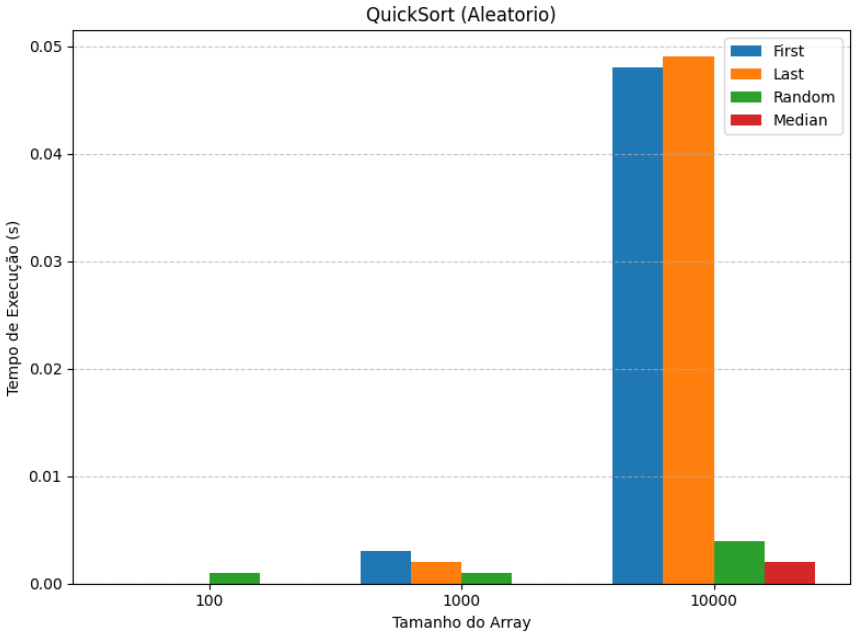
| Tipo de Array | Estratégia | 100 elementos | 1.000 elementos | 10.000 elementos |
|---|---|---|---|---|
| Aleatório | First Pivot | 0,000 s | 0,003 s | 0,048 s |
| Aleatório | Last Pivot | 0,000 s | 0,002 s | 0,049 s |
| Aleatório | Random Pivot | 0,001 s | 0,001 s | 0,004 s |
| Aleatório | Median Pivot | 0,000 s | 0,000 s | 0,002 s |
| Quase Ordenado | First Pivot | 0,001 s | 0,004 s | 0,056 s |
| Quase Ordenado | Last Pivot | 0,001 s | 0,003 s | 0,058 s |
| Quase Ordenado | Random Pivot | 0,001 s | 0,001 s | 0,005 s |
| Quase Ordenado | Median Pivot | 0,000 s | 0,001 s | 0,003 s |
| Ordenado | First Pivot | 0,001 s | 0,005 s | 0,060 s |
| Ordenado | Last Pivot | 0,001 s | 0,004 s | 0,062 s |
| Ordenado | Random Pivot | 0,001 s | 0,002 s | 0,006 s |

| | | | | |
|---|---|---|---|---|
| Ordenado | Median Pivot | 0,000 s | 0,001 s | 0,003 s |

## 5. Análise de Desempenho

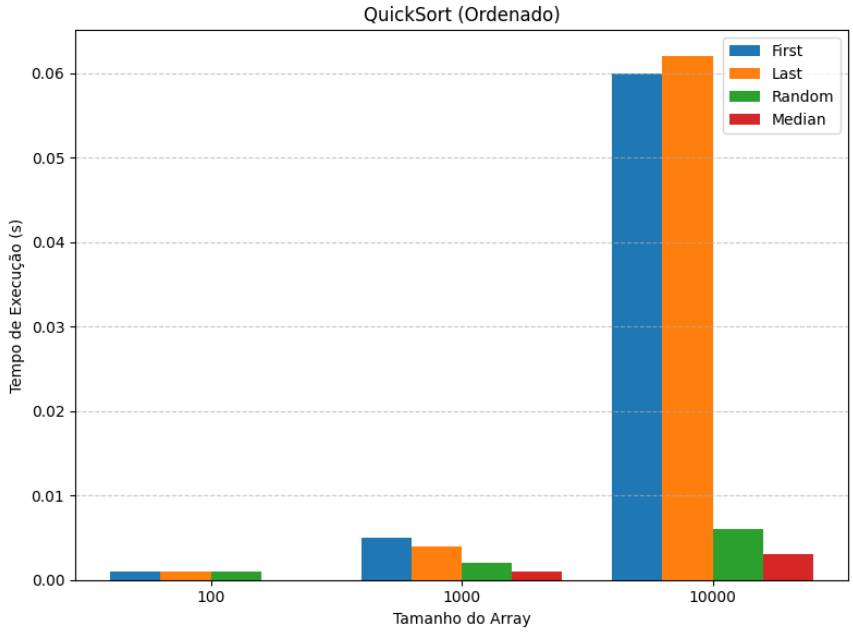
- **First Pivot** e **Last Pivot**: apresentam tempos similares.
  - Para arrays quase ordenados e ordenados, o desempenho degrada ligeiramente devido a partições desbalanceadas.
- **Random Pivot**: mais rápido que First/Last, especialmente em arrays grandes.
  - Reduz o risco do pior caso ($O(n^2)$), mantendo O(n log n) na prática.
- **Median of Three**: estratégia mais eficiente em todos os cenários.
  - Mantém partições equilibradas mesmo para arrays quase ordenados ou ordenados.

Figura 1 – Tempos de execução do QuickSort para arrays aleatórios. Gráfico gerado no **Google Colab**.



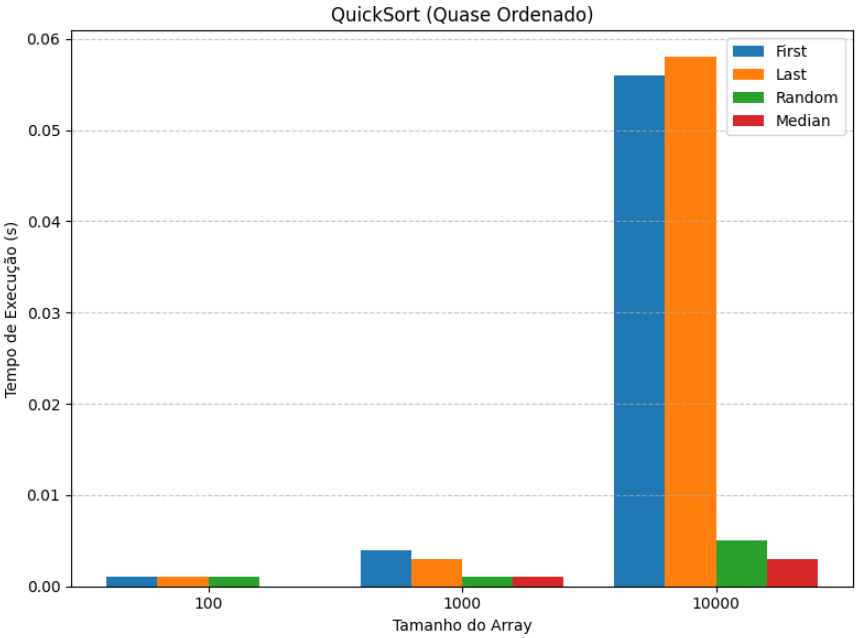

GOOGLE COLAB; acesso em 29/09/2025.

Figura 2 – Tempos de execução do QuickSort para arrays ordenados. Gráfico gerado no **Google Colab**.




GOOGLE COLAB; acesso em 29/09/2025.

Figura 3 – Tempos de execução do QuickSort para arrays quase ordenados. Gráfico gerado no **Google Colab**.




GOOGLE COLAB; acesso em 29/09/2025.

## 6. Discussão

O QuickSort é eficiente no caso médio, com **complexidade O(n log n)**.
Sua eficiência depende do pivô escolhido:

- **First/Last Pivot**: simples, mas podem gerar desempenho ruim em arrays ordenados/quase ordenados.
- **Random Pivot**: introduz aleatoriedade, tornando o algoritmo **mais robusto** em diferentes cenários.
- **Median of Three**: busca a mediana de três elementos para evitar partições desbalanceadas, garantindo melhor performance.

**Vantagens do pivô aleatório**:

- Minimiza dependência da ordem inicial dos dados.
- Reduz o risco de pior caso, mantendo boa eficiência para grandes arrays.

**Desvantagens**:

- Não elimina completamente o pior caso.
- Geração de números aleatórios tem custo computacional pequeno, geralmente irrelevante.

## 7. Conclusões

- A escolha do pivô influencia diretamente o desempenho do QuickSort.
- Estratégias simples (First/Last) são rápidas em arrays pequenos, mas podem ser lentas em arrays ordenados/quase ordenados.
- Estratégias mais inteligentes (Random e Median of Three) minimizam o risco de pior caso e reduzem significativamente o tempo para arrays grandes.
- **Recomendação prática:** utilizar **Median of Three** ou **Random Pivot** para maior eficiência.