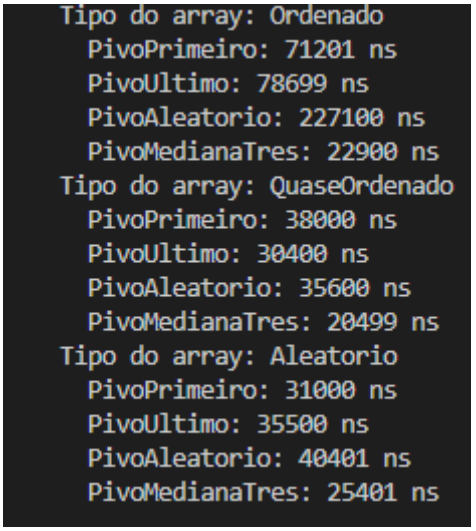
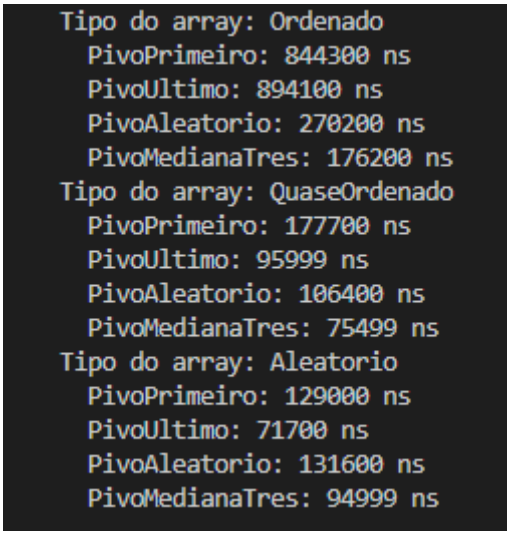
Nome: Gabriel Henrique Vieira de Oliveira
Data: 08/10/2024
Matéria: AEDs 2 - **LAB 06 - Quicksort e seu pivô**

---

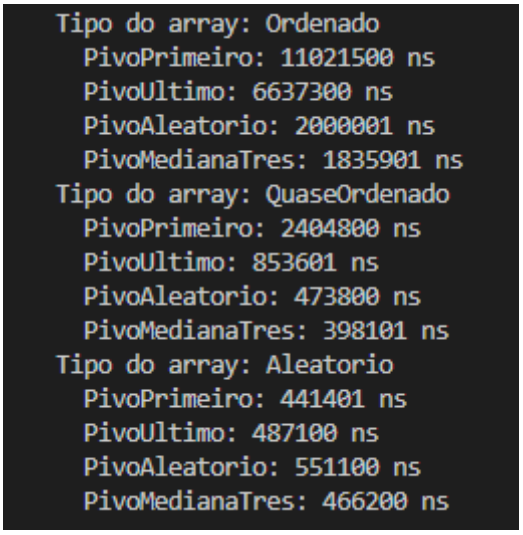
1) Tamanho do Array em 100 (e seus resultados):



2) Tamanho do Array em 1000 (e seus resultados):



3) Tamanho do Array em 10000 (e seus resultados):



CONCLUSÕES

Funcionamento das Estratégias de Escolha de Pivô:

1. **Pivô Primeiro**: Essa estratégia seleciona o primeiro elemento do array como pivô. Esse método é simples e eficiente em algumas situações, mas pode ser ineficiente quando o array já está ordenado ou quase ordenado, resultando em um desempenho ruim, já que não divide o array de forma equilibrada.

2. **Pivô Último**: O último elemento do array é escolhido como pivô. Assim como o pivô primeiro, essa abordagem é vulnerável quando o array está ordenado ou quase ordenado, causando um comportamento degradado (complexidade quadrática O(n2), O(n^2), O(n2) devido à má divisão dos sub-arrays.

3. **Pivô Aleatório**: Seleciona um elemento aleatório como pivô. Essa estratégia visa reduzir a chance de pior caso no QuickSort, pois a escolha aleatória evita padrões específicos no array, aumentando a probabilidade de dividir o array de forma mais equilibrada.

4. **Pivô Mediana de Três**: Escolhe como pivô a mediana entre o primeiro, o último é um elemento do meio. Essa abordagem é considerada uma das mais robustas, pois tenta garantir uma divisão mais equilibrada, minimizando a ocorrência de sub-arrays desequilibrados.

Discussão da Eficiência das Estratégias

Com base nos tempos exibidos nas imagens para os três tipos de array (Ordenado, QuaseOrdenado e Aleatório), podemos fazer uma análise da eficiência de cada estratégia:

1. Array Ordenado:

- **Pivô Primeiro** apresentou consistentemente o pior tempo em todos os tamanhos testados. Isso ocorre porque, em um array já ordenado, escolher o primeiro elemento como pivô resulta em partições desbalanceadas, causando pior desempenho.

- **Pivô Último** também mostrou desempenhos ruins, similares ao Pivô Primeiro, pelas mesmas razões: a escolha do pivô não divide adequadamente o array.

- **Pivô Aleatório** teve um desempenho muito melhor, principalmente em arrays maiores, pois a escolha aleatória permite divisões mais equilibradas, reduzindo o tempo de execução.

- **Pivô Mediana de Três** foi a estratégia mais eficiente em arrays ordenados, pois a mediana tende a ser um bom pivô, garantindo uma melhor divisão dos subarrays.

2. Array Quase Ordenado:

- **Pivô Primeiro** novamente teve um desempenho ruim, mas melhor do que no caso de array totalmente ordenado.

- **Pivô Último** apresentou tempos melhores que o Pivô Primeiro, mas ainda assim não foi tão eficiente quanto as outras estratégias
.
- **Pivô Aleatório** teve um desempenho satisfatório, embora o aumento no tamanho do array tenha impactado mais o tempo de execução em comparação ao Pivô Mediana de Três.

- **Pivô Mediana de Três** mostrou-se novamente como a estratégia mais rápida, devido à sua capacidade de encontrar um pivô mais equilibrado, mesmo em arrays quase ordenados.

3. Array Aleatório:

- **Pivô Primeiro** e **Pivô Último** tiveram desempenhos semelhantes, ambos piores que as estratégias mais sofisticadas, mas menos problemáticos em comparação com os arrays ordenados.

- **Pivô Aleatório** foi bastante eficiente, aproveitando a distribuição aleatória dos elementos, o que reduziu as chances de divisões desbalanceadas.

- **Pivô Mediana de Três** apresentou o melhor desempenho neste cenário, continuando a ser a estratégia mais robusta.

Conclusão:

- A **estratégia de Pivô Mediana de Três** foi consistentemente a mais eficiente em todos os casos, especialmente para arrays ordenados e quase ordenados, devido à sua capacidade de equilibrar as partições de maneira eficaz.

- A **estratégia de Pivô Aleatório** também apresentou bons resultados, especialmente em arrays de ordem aleatória, sendo uma escolha sólida para garantir um bom desempenho em diferentes cenários.

- As estratégias de **Pivô Primeiro** e **Pivô Último** tiveram desempenhos significativamente inferiores, particularmente em arrays ordenados e quase ordenados, e devem ser evitadas nesses casos.

Esse comportamento é esperado, já que o QuickSort depende de uma boa escolha de pivô para garantir a divisão equilibrada dos sub-arrays. Escolhas ruins de pivô (como o primeiro ou o último em arrays ordenados) resultam em um desempenho muito degradado.