Algoritmos e Estrutura de Dados II
Relatório - Quicksort e seu pivô

September 26, 2025

Matheus de Almeida Moreira
848813

Estratégias de escolha do pivô

No Quicksort, a escolha do pivô é um fator determinante no desempenho do algoritmo. As estratégias avaliadas foram:

- **FirstPivot**: escolhe sempre o primeiro elemento do array como pivô.
- **LastPivot**: escolhe sempre o último elemento do array como pivô.
- **RandomPivot**: escolhe aleatoriamente um elemento do array como pivô.
- **MedianOfThree**: seleciona o pivô como a mediana entre o primeiro, o último e o elemento do meio.

Bônus: QuickWithInsertion

Como abordagem bônus, foi implementada a combinação do Quicksort com o Insertion Sort. Essa técnica utiliza o Insertion Sort quando o tamanho do subarray é menor ou igual a 16 elementos, aproveitando sua eficiência em pequenas partições.

Algoritmos

1. Pivô: Primeiro elemento

```java
public void quicksortFirstPivot(int esq, int dir) {
    int i = esq, j = dir;
    int pivo = array[esq];
    while (i <= j) {
        while (array[i] < pivo) i++;
        while (array[j] > pivo) j--;
        if (i <= j) {
            swap(i, j);
            i++;
            j--;
        }
    }
    if (esq < j) quicksortFirstPivot(esq, j);
    if (i < dir) quicksortFirstPivot(i, dir);
}
```

2. Pivô: Último elemento

```java
public void quicksortLastPivot(int esq, int dir) {
    int i = esq, j = dir;
    int pivo = array[dir];
    while (i <= j) {
        while (array[i] < pivo) i++;
        while (array[j] > pivo) j--;
        if (i <= j) {
            swap(i, j);
            i++;
            j--;
        }
    }
    if (esq < j) quicksortLastPivot(esq, j);
    if (i < dir) quicksortLastPivot(i, dir);
}
```

3. Pivô: Aleatório

```
public void quicksortRandomPivot(int esq, int dir) {
    int i = esq, j = dir;
    int randomIndex = esq + (int)(Math.random() * (dir - esq + 1));
    int pivo = array[randomIndex];
    while (i <= j) {
        while (array[i] < pivo) i++;
        while (array[j] > pivo) j--;
        if (i <= j) {
            swap(i, j);
            i++;
            j--;
        }
    }
    if (esq < j) quicksortRandomPivot(esq, j);
    if (i < dir) quicksortRandomPivot(i, dir);
}
```

4. Pivô: Mediana de Três

```
public void quicksortMedianOfThree(int esq, int dir) {
    int i = esq, j = dir;
    int meio = (esq + dir) / 2;
    if (array[esq] > array[meio]) swap(esq, meio);
    if (array[esq] > array[dir]) swap(esq, dir);
    if (array[meio] > array[dir]) swap(meio, dir);
    int pivo = array[meio];
    while (i <= j) {
        while (array[i] < pivo) i++;
        while (array[j] > pivo) j--;
        if (i <= j) {
            swap(i, j);
            i++;
            j--;
        }
    }
    if (esq < j) quicksortMedianOfThree(esq, j);
    if (i < dir) quicksortMedianOfThree(i, dir);
}
```

Bônus: QuickWithInsertion

```java
public void quicksortWithInsertion(int esq, int dir) {
    int LIMITE = 16;
    if (dir - esq + 1 <= LIMITE) {
        insertionSort(esq, dir);
    } else {
        int i = esq, j = dir;
        int pivo = array[(esq + dir) / 2];
        while (i <= j) {
            while (array[i] < pivo) i++;
            while (array[j] > pivo) j--;
            if (i <= j) {
                swap(i, j);
                i++;
                j--;
            }
        }
        if (esq < j) quicksortWithInsertion(esq, j);
        if (i < dir) quicksortWithInsertion(i, dir);
    }
}

private void insertionSort(int esq, int dir) {
    for (int i = esq + 1; i <= dir; i++) {
        int tmp = array[i];
        int j = i - 1;
        while (j >= esq && array[j] > tmp) {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = tmp;
    }
}
```

Resultados e Discussão

Foram realizados testes com três cenários distintos: array **Ordenado**, **Quase Ordenado** e **Aleatório**, nos tamanhos 100, 1000 e 10000. Os gráficos abaixo mostram os tempos de execução em escala logarítmica.

Cenário Ordenado

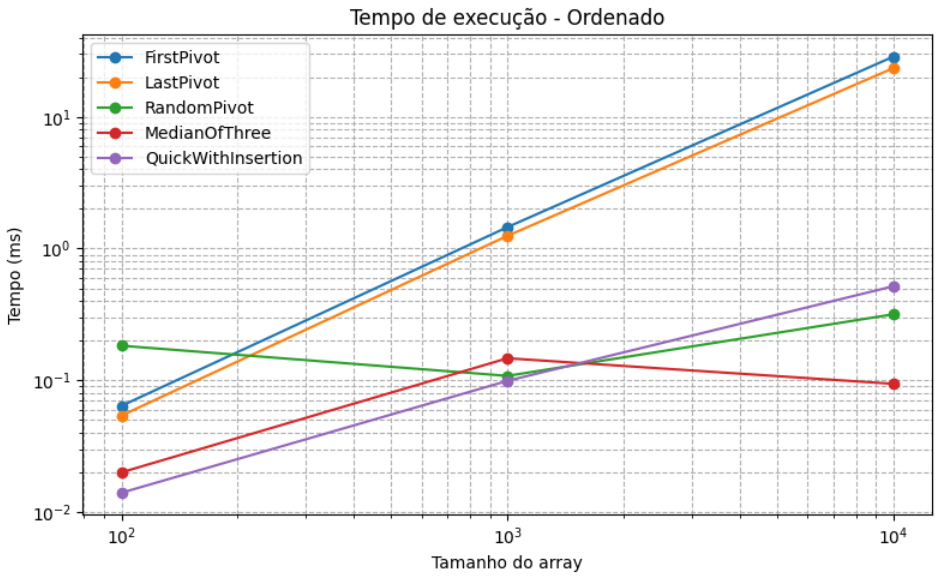

Figura 1: Desempenho das estratégias no cenário ordenado

Observa-se que as estratégias **FirstPivot** e **LastPivot** apresentam um desempenho significativamente pior para arrays ordenados, pois geram partições muito desbalanceadas, aproximando o custo de $O(n^2)$. Já a estratégia **MedianOfThree** apresentou boa eficiência. A abordagem bônus **QuickWithInsertion** manteve tempos baixos mesmo para $n = 10000$, mostrando-se especialmente vantajosa em subarrays pequenos. O **RandomPivot** também se mostrou consistente, mas um pouco mais instável.

Cenário Quase Ordenado

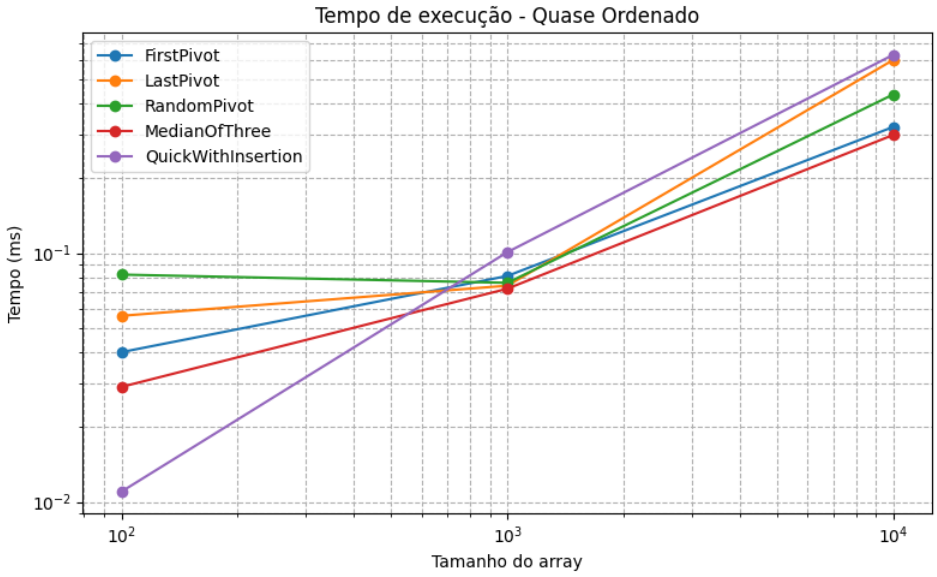

Figura 2: Desempenho das estratégias no cenário quase ordenado

Neste caso, o comportamento foi semelhante ao cenário ordenado, porém com impacto menor. As estratégias **MedianOfThree** e **QuickWithInsertion** novamente apresentaram melhor desempenho, mostrando maior robustez. O **RandomPivot** teve resultados medianos e estáveis, enquanto **FirstPivot** e **LastPivot** continuaram piores para tamanhos grandes.

Cenário Aleatório

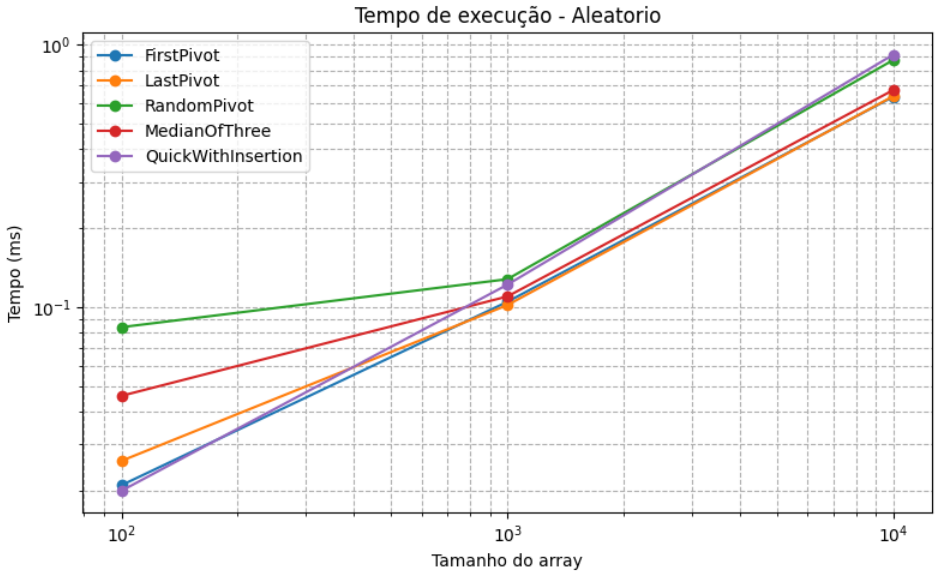

Figura 3: Desempenho das estratégias no cenário aleatório

Para arrays aleatórios, todas as estratégias apresentaram desempenho muito próximo, pois a distribuição dos elementos já é favorável. Ainda assim, **MedianOfThree** e **QuickWithInsertion** mantiveram uma leve vantagem de estabilidade e menor tempo médio.

Conclusão

- Em **arrays ordenados** e **quase ordenados**, **FirstPivot** e **LastPivot** foram ineficientes, confirmando sua fragilidade nesses casos.
- O **RandomPivot** apresentou desempenho aceitável em todos os cenários, mas sem ser o mais eficiente.
- A estratégia da **Mediana de Três** (*MedianOfThree*) foi a que apresentou maior consistência, reduzindo os piores casos e garantindo tempos estáveis.
- O bônus **QuickWithInsertion** mostrou-se muito eficiente, especialmente em subarrays pequenos (tamanho máximo 16), aproveitando a eficiência do Insertion Sort nessas condições.

Portanto, conclui-se que a **Mediana de Três** é a melhor estratégia geral para escolha de pivô no Quicksort, seguida de perto pela combinação com Insertion Sort.