Laboratório 06 - Quicksort e seu pivô

Aluno: Guilherme Soares Silva
Matrícula: 863485

Implementação do Quicksort para os pivôs:

- **Primeiro elemento**
- **´Ultimo elemento**
- **Pivô aleatório**
- **Mediana de três elementos (início, meio e fim)**

Código:
/*
* O código foi criado com base em uma classe que cria os arrays de acordo com a
* necessidade do teste, cada implementação do QuickSort está em uma classe
* distinta. A classe Main instancia um objeto do tipo GenerationArrayInt que por sua
* vez faz requisitos às classes filhas para realizar os testes.
*/

- Classe GenerationArrayInt

// Criação de Arrays

```java
/**
* Geracao de elementos de um array de inteiros
* @author Guilherme Soares Silva
* @version 1 10/2024
*/

import java.util.*;

public class GenerationArrayInt {
/*
   * Leitor de entrada
   */
   Scanner scan = new Scanner(System.in);

   /**
    * Atributos da classe
   */
   protected int[] array;
   protected int n;

   /**
    * Cria um array com base no tipo especificado.
```

```
    * @param size Tamanho do array.
    * @param type Tipo do array: 0 (aleatório), 1 (ordenado), 2 (quase
ordenado).
    * @return O array gerado.
    */
   public int[] createArray(int size, int type) {
       int[] array = new int[size];
       Random rand = new Random();

       switch (type) {
           case 0: // Aleatório
               for (int i = 0; i < size; i++) {
                   array[i] = rand.nextInt(size);
               }
               break;
           case 1: // Ordenado
               for (int i = 0; i < size; i++) {
                   array[i] = i;
               }
               break;
           case 2: // Quase ordenado
               for (int i = 0; i < size; i++) {
                   array[i] = i;
               }
               // Trocando elementos
               for (int i = 0; i < size / 10; i++) {
                   int index1 = rand.nextInt(size);
                   int index2 = rand.nextInt(size);
                   swapArray(array, index1, index2);
               }
               break;
           default:
               throw new IllegalArgumentException("Tipo desconhecido: "
+ type);
       }

       return array;
   }

   /**
    * Troca o conteudo de duas posicoes do array
    * @param array O array onde a troca ocorrerá.
    * @param i int primeira posicao
```

```
     * @param j int segunda posicao
     */
    private void swapArray(int[] array, int i, int j) {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }

    /**
     * Metodo Construtor com tamanho padrao
    */
    public GenerationArrayInt(){
        array = new int[100];
        n = array.length;
    }
    /**
     * Metodo Construtor com tamanho definido
    * @param int tamanho do array de numeros inteiros
    */
    public GenerationArrayInt(int size){
        array = new int[size];
        n = array.length;
    }


    /**
     * Produz um array ordenado de modo crescente
    */
    public void crescentArrayInt() {
        for (int i = 0; i < n; i++) {
            array[i] = i;
        }
    }
    /**
     * Produz um array ordenado de modo decrescente
    */
    public void decrescentArrayInt() {
        for (int i = 0; i < n; i++) {
            array[i] = n - 1 - i;
        }
    }
    /**
     * Produz um array com numeros aleatorios
```

```
    */
    public void randomArrayInt() {
        Random rand = new Random();
        crescentArrayInt();
        for (int i = 0; i < n; i++) {
            swap(i, Math.abs(rand.nextInt()) % n);
        }
    }


    /**
     * Efetua a leitura dos elementos via entrada padrao
    */
    public void defaultEntry() {
        n = scan.nextInt();
        array = new int[n];

        for (int i = 0; i < n; i++) {
            array[i] = scan.nextInt();
        }
    }

    /**
     * Recebe um array e efetua a leitura dos elementos via entrada
padrao
    */
    public void entry(int[] vet){
        n = vet.length;
        array = new int[n];
        for(int i = 0; i < n; i++){
            array[i] = vet[i];
        }
    }


    /**
     * Mostra os k primeiros elementos do array
    * @param int k indica a quantidade de elementos do array a serem
mostrados.
    */
    public void printKArrayInt(int k) {
        System.out.print("[");
```

```
        for (int i = 0; i < k; i++) {
            System.out.print(" ("+i+")" + array[i]);
        }

        System.out.println("]");
    }


    /**
     * Mostra os elementos do array.
    */
    public void printArrayInt() {
        System.out.print("[");

        for (int i = 0; i < n; i++) {
            System.out.print(" ("+i+")" + array[i]);
        }

        System.out.println("]");
    }


    /**
     * Troca o conteudo de duas posicoes do array
    * @param i int primeira posicao
    * @param j int segunda posicao
    */
    public void swap(int i, int j) {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }


    /**
     * Retorna o timestamp atual
    * @return timestamp atual
    */
    public long now(){
        return new Date().getTime();
    }
```

```
    /**
     * Retorna verdadeiro/falso indicando se o array esta ordenado
    * @return boolean indicando se o array esta ordenado
    */
    public boolean isOrdenedArrayInt(){
        boolean resp = true;
        for(int i = 1; i < n; i++){
            if(array[i] < array[i-1]){
                i = n;
                resp = false;
            }
        }
        return resp;
    }

    /*
    * Metodo a ser implementado nas subclasses
    */
    public void sort(){
        System.out.println("Método a ser implementado nas subclasses.");
    }
}
```

- Classe QuicksortFirstPivot

// Ordenação com pivô no primeiro elemento

```
/**
* Algoritmo de ordenacao QuicksortFirstPivot
* @author Guilherme Soares Silva
* @version 1 10/2024
*/

public class QuicksortFirstPivot extends GenerationArrayInt {

  /**
   * Construtor
   */
  public QuicksortFirstPivot(){
     super();
  }
  /**
   * Construtor com tamnho do array definido
   * @param int tamanho do array de numeros inteiros.
```

```
   */
  public QuicksortFirstPivot(int size){
     super(size);
  }


  /**
   * Algoritmo de ordenacao QuicksortFirstPivot
   */
  @Override
  public void sort() {
     quicksortFirstPivot(0, n-1);
  }

  /**
   * Algoritmo de ordenacao QuicksortFirstPivot
   * @param int left inicio do array a ser ordenado
   * @param int right fim do array a ser ordenado
   */
   private void quicksortFirstPivot(int left, int right) {
       int i = left, j = right;
       int pivot = array[left];
       while (i <= j) {
           while (array[i] < pivot) i++;
           while (array[j] > pivot) j--;
           if (i <= j) {
               swap(i, j);
               i++;
               j--;
           }
       }
       if (left < j)  quicksortFirstPivot(left, j);
       if (i < right)  quicksortFirstPivot(i, right);
   }
}
```

- Classe QuicksortLastPivot

// Ordenação com pivô no último elemento

```
/**
* Algoritmo de ordenacao QuicksortLastPivot
* @author Guilherme Soares Silva
* @version 1 10/2024
```

```java
*/

public class QuicksortLastPivot extends GenerationArrayInt {

  /**
   * Construtor
   */
  public QuicksortLastPivot(){
     super();
  }
  /**
   * Construtor com tamnho do array definido
   * @param int tamanho do array de numeros inteiros.
   */
  public QuicksortLastPivot(int size){
     super(size);
  }



  /**
   * Algoritmo de ordenacao QuicksortLastPivot
   */
  @Override
  public void sort() {
     quicksortLastPivot(0, n-1);
  }

  /**
   * Algoritmo de ordenacao QuicksortLastPivot
   * @param int left inicio do array a ser ordenado
   * @param int right fim do array a ser ordenado
   */
   private void quicksortLastPivot(int left, int right) {
       int i = left, j = right;
       int pivot = array[right];
       while (i <= j) {
           while (array[i] < pivot) i++;
           while (array[j] > pivot) j--;
           if (i <= j) {
               swap(i, j);
               i++;
               j--;
           }
```

```
        }
        if (left < j)  quicksortLastPivot(left, j);
        if (i < right)  quicksortLastPivot(i, right);
    }
}
```

- Classe QuicksortMedianOfThree

// Ordenação com pivô Mediana do primeiro elemento, central e último

```
/**
* Algoritmo de ordenacao QuicksortMedianOfThree
  * @author Guilherme Soares Silva
  * @version 1 10/2024
  */

public class QuicksortMedianOfThree extends GenerationArrayInt {

  /**
   * Construtor
   */
  public QuicksortMedianOfThree(){
     super();
  }
  /**
   * Construtor com tamnho do array definido
   * @param int tamanho do array de numeros inteiros.
   */
  public QuicksortMedianOfThree(int size){
     super(size);
  }



  /**
   * Algoritmo de ordenacao QuicksortMedianOfThree
   */
  @Override
  public void sort() {
     quicksortMedianOfThree(0, n-1);
  }

  /**
   * Metodo que retorna a mediana de 3 elementos
   * @param x
```

```
   * @param y
   * @param z
   * @return int median
   */
  private int median(int x, int y, int z) {
     if((x < y) && (x < z)) {
        if(y < z) {
           return y;
        } else {
           return z;
        }
     } else if ((x < y) && (x > z)) {
        return x;
     } else if (y < z){
        return y;
     } else {
        return z;
     }
  }

  /**
   * Algoritmo de ordenacao QuicksortMedianOfThree
   * @param int left inicio do array a ser ordenado
   * @param int right fim do array a ser ordenado
   */
  private void quicksortMedianOfThree(int left, int right) {
     int i = left, j = right;
     int pivot = median(array[left], array[right], array[(left +
right)/2]);
     while (i <= j) {
           while (array[i] < pivot) i++;
           while (array[j] > pivot) j--;
           if (i <= j) {
              swap(i, j);
              i++;
              j--;
           }
     }
     if (left < j)  quicksortMedianOfThree(left, j);
     if (i < right)  quicksortMedianOfThree(i, right);
  }
}
```

- Classe QuicksortRandom

// Ordenação com pivô aleatório

```java
/**
* Algoritmo de ordenacao QuicksortRandomPivot
* @author Guilherme Soares Silva
* @version 1 10/2024
*/

import java.util.Random;

public class QuicksortRandomPivot extends GenerationArrayInt {

  /**
   * Construtor
   */
  public QuicksortRandomPivot(){
     super();
  }
  /**
   * Construtor com tamnho do array definido
   * @param int tamanho do array de numeros inteiros.
   */
  public QuicksortRandomPivot(int size){
     super(size);
  }



  /**
   * Algoritmo de ordenacao Quicksort com pivo aleatorio
   */
  @Override
  public void sort() {
     quicksortRandomPivot(0, n-1);
  }

  /**
   * Algoritmo de ordenacao QuicksortRandomPivot
   * @param int left inicio do array a ser ordenado
   * @param int right fim do array a ser ordenado
   */
   private void quicksortRandomPivot(int left, int right) {
       Random rand = new Random();
```

```
        int i = left, j = right;
        int pivot = array[left + Math.abs(rand.nextInt()) % (right -
left + 1)];
        while (i <= j) {
            while (array[i] < pivot) i++;
            while (array[j] > pivot) j--;
            if (i <= j) {
                swap(i, j);
                i++;
                j--;
            }
        }
        if (left < j)  quicksortRandomPivot(left, j);
        if (i < right)  quicksortRandomPivot(i, right);
    }
}
```

- Classe Main

// Requisita a criação de arrays, ordenação e geração de um CSV com os tempos decorridos

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        int[] sizes = {100, 1000, 10000};
        int[] testTypes = {0, 1, 2}; // 0: aleatório, 1: ordenado, 2:
quase ordenado

        GenerationArrayInt generator = new GenerationArrayInt();

        // Criar o arquivo CSV
        try (FileWriter writer = new
FileWriter("desempenho_quicksort.csv")) {
            // Cabeçalho do CSV
            writer.append("Tamanho,Tipo,Primeiro Pivô,Último Pivô,Pivô
Aleatório,Mediana de Três\n");

            for (int size : sizes) {
                for (int type : testTypes) {
```

```
                    int[] array = generator.createArray(size, type);
                    String typeStr = (type == 0) ? "Aleatório" : (type
== 1) ? "Ordenado" : "Quase ordenado";

                    List<Long> timesFirstPivot = new ArrayList<>();
                    List<Long> timesLastPivot = new ArrayList<>();
                    List<Long> timesRandomPivot = new ArrayList<>();
                    List<Long> timesMedianOfThree = new ArrayList<>();

                    for (int i = 0; i < 10; i++) {
                        // Quicksort com Primeiro Pivô
                        QuicksortFirstPivot quicksortFirst = new
QuicksortFirstPivot(size);
                        quicksortFirst.entry(array.clone());
                        long startTime = System.nanoTime();
                        quicksortFirst.sort();
                        long endTime = System.nanoTime();
                        timesFirstPivot.add(endTime - startTime);

                        // Quicksort com Último Pivô
                        QuicksortLastPivot quicksortLast = new
QuicksortLastPivot(size);
                        quicksortLast.entry(array.clone());
                        startTime = System.nanoTime();
                        quicksortLast.sort();
                        endTime = System.nanoTime();
                        timesLastPivot.add(endTime - startTime);

                        // Quicksort com Pivô Aleatório
                        QuicksortRandomPivot quicksortRandom = new
QuicksortRandomPivot(size);
                        quicksortRandom.entry(array.clone());
                        startTime = System.nanoTime();
                        quicksortRandom.sort();
                        endTime = System.nanoTime();
                        timesRandomPivot.add(endTime - startTime);

                        // Quicksort com Mediana de Três
                        QuicksortMedianOfThree quicksortMedian = new
QuicksortMedianOfThree(size);
                        quicksortMedian.entry(array.clone());
                        startTime = System.nanoTime();
                        quicksortMedian.sort();
```

```
                    endTime = System.nanoTime();
                    timesMedianOfThree.add(endTime - startTime);
                }

                // Escreve os tempos no CSV
                for (int i = 0; i < 10; i++) {
                    writer.append(size + "," + typeStr + "," +
                            timesFirstPivot.get(i) + "," +
                            timesLastPivot.get(i) + "," +
                            timesRandomPivot.get(i) + "," +
                            timesMedianOfThree.get(i) + "\n");
                }
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

- plot.py

// Plota o gráfico com base nos tempos de execução

```
import matplotlib.pyplot as plt
import pandas as pd

# Lê os dados do arquivo CSV
data = pd.read_csv('desempenho_quicksort.csv')

# Extraindo os dados
tamanhos = data['Tamanho'].unique()
tipos = data['Tipo'].unique()

# Configurando o gráfico
plt.figure(figsize=(10, 6))

# Plota os tempos para cada tipo de pivô
for tipo in tipos:
   subset = data[data['Tipo'] == tipo]
```

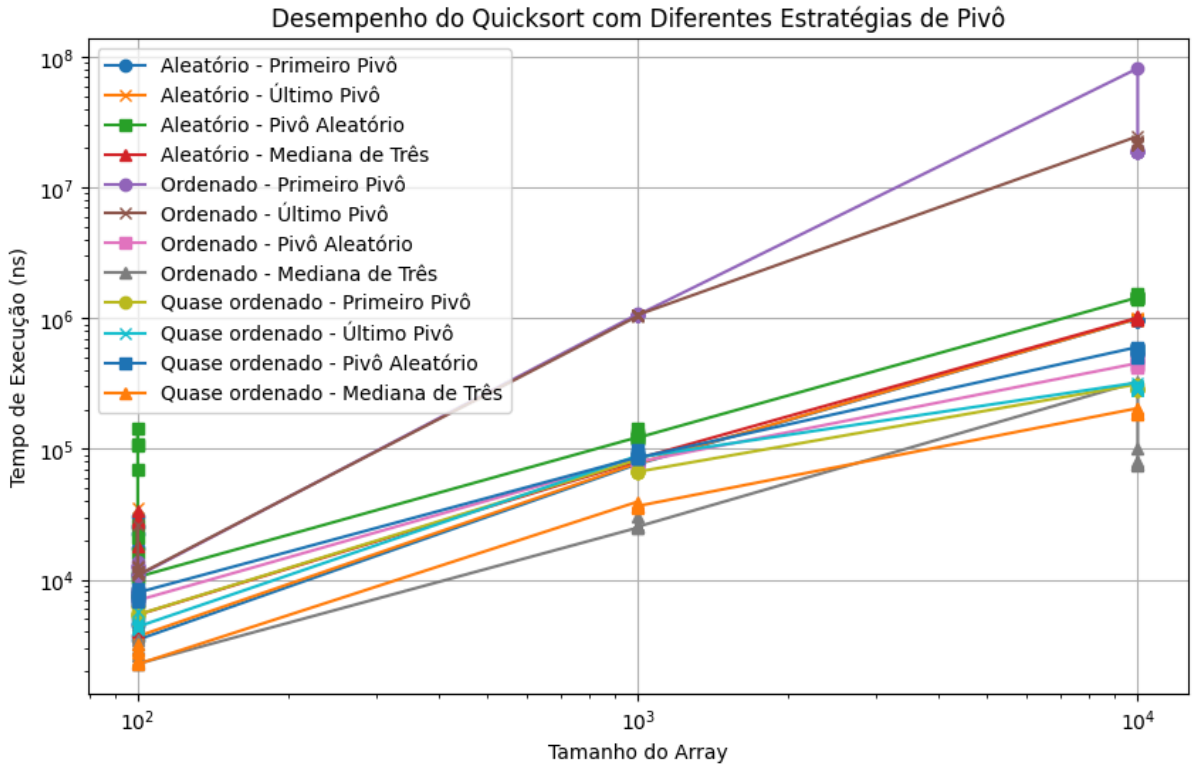
```
   plt.plot(subset['Tamanho'], subset['Primeiro Pivô'], marker='o',
label=f'{tipo} - Primeiro Pivô')
   plt.plot(subset['Tamanho'], subset['Último Pivô'], marker='x',
label=f'{tipo} - Último Pivô')
   plt.plot(subset['Tamanho'], subset['Pivô Aleatório'], marker='s',
label=f'{tipo} - Pivô Aleatório')
   plt.plot(subset['Tamanho'], subset['Mediana de Três'], marker='^',
label=f'{tipo} - Mediana de Três')

# Configurando o gráfico
plt.title('Desempenho do Quicksort com Diferentes Estratégias de Pivô')
plt.xlabel('Tamanho do Array')
plt.ylabel('Tempo de Execução (ns)')
plt.xscale('log')
plt.yscale('log')
plt.xticks(tamanhos)
plt.legend()
plt.grid(True)

# Salva o gráfico como imagem
plt.savefig('desempenho_quicksort.png', bbox_inches='tight')
plt.close()

print("Gráfico salvo como 'desempenho_quicksort.png'")
```

Plot do gráfico com auxílio da biblioteca Matplotlb em Python
(tempo de execução, tamanho do array e casos de teste)



Complexidades:

1. Primeiro elemento como Pivô

- Melhor Caso: O(n log n) (quando o array está balanceado)
- Caso Médio : O(n log n)
- Pior Caso : O(n²) (ocorre em arrays já ordenados ou inversamente ordenados)

2. Último elemento como Pivô

- Melhor Caso: O(n log n)
- Caso Médio: O(n log n)
- Pior Caso: O(n²) (semelhante ao primeiro pivô, em arrays ordenados)

3. Pivô Aleatório

- Melhor Caso: O(n log n)
- Caso Médio: O(n log n) (o desempenho é equilibrado devido à escolha aleatória do pivô)
- Pior Caso: O(n²) (ocorre em situações muito desfavoráveis, mas é raro)

4. Mediana de Três como Pivô

- Melhor Caso: O(n log n)
- Caso Médio: O(n log n)
- Pior Caso: O(n log n) (tende a se comportar melhor em arrays ordenados ou quase ordenados)

Conclusão:

Funcionamento de Cada Estratégia de Escolha do Pivô

O Quicksort é um algoritmo de ordenação que baseia-se na técnica de "dividir para conquistar". Dessa forma, o pivô influencia de forma direta no tempo de execução e eficácia do algoritmo:

- Primeiro Pivô: O primeiro elemento é o pivô. Pode ser ineficiente para arrays já ordenados ou quase ordenados, possui pior caso de $O(n^2)$.
- Último Pivô: O último elemento é o pivô. Pode ser ineficiente para arrays ordenados.
- Pivô Aleatório: Um elemento aleatório é o pivô. Tende a melhorar o desempenho já que diminui as chances de cair no pior caso, possui complexidade média de O(n log n).
- Mediana de Três: O pivô é uma mediana de três elementos (primeiro, último e central). Tende a manter o pivô como centro do array, tornando-se mais eficiente mesmo em arrays quase ordenados por criar subarrays mais balanceados.

Análise de cada caso:

Array Aleatório:

- A estratégia mais eficiente parece ser Mediana de Três (linha vermelha), com o tempo de execução consistentemente mais baixo para tamanhos crescentes de array.
- Essa estratégia seleciona o pivô com base em três elementos (primeiro, meio e último), o que ajuda a evitar o pior caso (quando o pivô é muito distante da mediana), reduzindo o número de partições desbalanceadas e, consequentemente, o tempo total.

Array Ordenado:

- A Mediana de Três (linha amarela) também é a melhor estratégia para arrays ordenados. O tempo de execução aumenta à medida que o tamanho cresce, mas é significativamente menor do que as outras estratégias.
- Nos arrays ordenados, escolher o primeiro ou último elemento como pivô pode levar ao pior caso de desempenho O(n2)O(n^2)O(n2), enquanto a Mediana de Três continua balanceada.

Array Quase Ordenado:

- Para o caso do array quase ordenado, a estratégia da Mediana de Três (linha laranja) ainda é a mais eficiente, com o menor tempo de execução.
- Como o array é quase ordenado, a mediana de três ajuda a evitar cenários onde o pivô escolhido é o maior ou menor elemento do subarray, mantendo o balanceamento das partições.