Exercício – Lab 06 - Quicksort e seu pivô

Aluno: Gabriel Chaves Mendes

Código de Aluno: 1453522

Professor: Felipe

1) Funcionamento de cada estratégia de escolha do pivô

- void QuickSortFirstPivot (int [] array , int left , int right) ;

  Com o pivô no início do array, os elementos menores que o pivô serão movidos para esquerda, enquanto os maiores para a direita. No entanto, para arrays ordenados e quase ordenados, é uma desvantagem pelas chamadas recursivas desnecessárias. Θ(n²)

- void QuickSortLastPivot (int [] array , int left , int right) ;

  De forma similar ao pivô no início, a escolha do pivô no final terá os elementos menores que ele movidos para a esquerda, enquanto os elementos maiores para direita. Sua implementação é fácil, mas enfrenta a mesma desvantagem do pivô no início. Θ(n²)

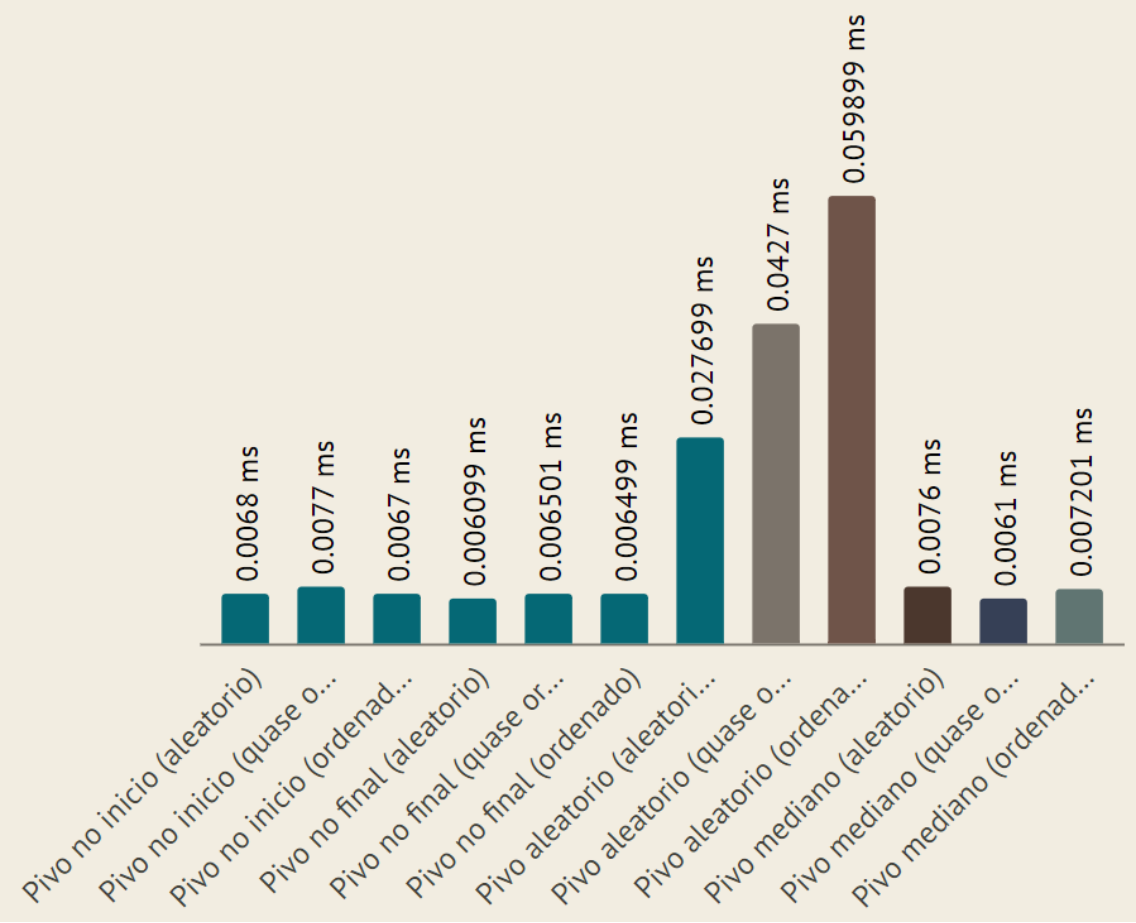
- void QuickSortRandomPivot (int [] array , int left , int right) ;

  De forma aleatória, é escolhido um pivô a cada iteração do código. Durante a iteração, o pivô é trocado com o ultimo elemento do array, para evitar comparações desnecessárias e simplificar a ordenação. A escolha do pivô aleatório é vantajosa para grandes arrays, pelo fato de reduzir a quantidade de sub-arrays desbalanceados. Θ(n log^n)

- void QuickSortMedianOfThree (int [] array , int left , int right) ;

  Um pivô é escolhido por meio da mediana de três elementos (primeiro elemento do array, ultimo elemento e elemento central). A partir da escolha do pivô, é iniciada a ordenação do QuickSort, colocando os valores menores na esquerda e os maiores na direita. Essa escolha é vantajosa para evitar piores casos e tentar produzir ordenações mais balanceadas, principalmente em arrays menore. Θ(n log^n)

2) Gráfico com o tempo de execução



3) Qual estratégia foi mais eficiente em cada caso e por quê:

Array desordenado: Em um array desordenado, é mais vantajoso a escolha das estratégias de pivô aleatório e pivô mediano de três, pois evita uma divisão desiguais que aumentam o número de chamadas recursivas desnecessárias.

Array quase ordenado: Em um array quase ordenado, ainda é mais vantajoso a escolha do pivô aleatório e pivô mediano de três, já que eles proporcionam divisões mais balanceadas e menos desiguais, diferente do pivô no inicio ou no final, que também fazem com que ocorram mais chamadas recursivas desnecessárias.

Array ordenado: Ainda assim, o pivô aleatório e o pivô mediano de três, continuam sendo os mais vantajosos para esse caso. Pois eles evitam desbalanceamento na divisão do array, tendo uma análise de Θ(n log^n) enquanto o pivô no inicio ou no fim possuem uma análise de Θ(n²).

4) Código:

```
import java.util.Scanner;
import java.util.Random;

public class LAB06 {

    static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    static void QuickSortFirstPivot (int [] array , int left , int
right ){ //Pivo no incio
        int i = left, j = right, pivo = array[left];

        while (i <= j) {

            while (array[i] < pivo){
                i++;
```

```
            }

            while (array[j] > pivo){
                j--;
            }

            if (i <= j){
                swap(array, i, j);
                i++;
                j--;
            }
        }

        if (left < j){
            QuickSortFirstPivot(array, left, j);
        }

        if (i < right){
            QuickSortFirstPivot(array, i, right);
        }
     }

    static void QuickSortLastPivot (int [] array , int left , int
right ){ //Pivo no final
        int i = left, j = right, pivo = array[right];

        while (i <= j) {

            while (array[i] < pivo){
                i++;
            }

            while (array[j] > pivo){
                j--;
            }

            if (i <= j){
                swap(array, i, j);
                i++;
                j--;
            }
        }

        if (left < j){
            QuickSortLastPivot(array, left, j);
        }

        if (i < right){
            QuickSortLastPivot(array, i, right);
```

```
        }
    }

    static void QuickSortRandomPivot(int[] array, int left, int
right) { //Pivo aleatorio
        if (left < right) {

            Random rand = new Random();
            int index_Aleatorio = left + rand.nextInt(right - left
+ 1);
            int pivo = array[index_Aleatorio];

            // Troca o pivô aleatório com o último elemento
            swap(array, index_Aleatorio, right);

            int i = left, j = right - 1;

            while (i <= j) {
                while (i <= j && array[i] < pivo) {
                    i++;
                }

                while (i <= j && array[j] > pivo) {
                    j--;
                }

                if (i <= j) {
                    swap(array, i, j);
                    i++;
                    j--;
                }
            }

            swap(array, i, right);

            QuickSortRandomPivot(array, left, i - 1);
            QuickSortRandomPivot(array, i + 1, right);
        }
    }

    static void QuickSortMedianOfThree(int [] array , int left ,
int right ){ //Pivo media de tres
        int meio = (left + right) / 2;

        if (array[left] > array[meio]) {
            swap(array, left, meio);
        }
        if (array[left] > array[right]) {
            swap(array, left, right);
```

```
        }
        if (array[meio] > array[right]) {
            swap(array, meio, right);
        }

        int pivo = array[meio];


        int i = left, j = right;

        while (i <= j) {

            while (array[i] < pivo){
                i++;
            }

            while (array[j] > pivo){
                j--;
            }

            if (i <= j){
                swap(array, i, j);
                i++;
                j--;
            }
        }

        if (left < j){
            QuickSortMedianOfThree(array, left, j);
        }

        if (i < right){
            QuickSortMedianOfThree(array, i, right);
        }
    }

    public static void main(String[] args) throws Exception {

        Scanner sc = new Scanner(System.in);
        Random rand = new Random();

        int escolha;
        int left, right;
        int array1[] = new int[10]; //aleatorio
        int array2[] = {1, 2, 5, 4, 3, 6, 10, 9, 8, 7, 11, 12, 14, 13,
15}; //quase ordenado
        int array3[] = new int[20]; //ordenado

        for(int i = 0; i < 10; i++){
```

```
            array1[i] = rand.nextInt(100);
        }

        for(int i = 0; i < 10; i++){
            array3[i] = i;
        }

        System.out.println("Escolha entre as opcoes de implementacao
de QuickSort:");
        System.out.println("1  - Pivo no inicio
(aleatorio)          4  - Pivo no final (aleatorio)          7  - Pivo
aleatorio (aleatorio)          10 - Pivo mediano (aleatorio)");
        System.out.println("2  - Pivo no inicio (quase
ordenado)     5  - Pivo no final (quase ordenado)     8  - Pivo
aleatorio (quase ordenado)     11 - Pivo mediano (quase ordenado)");
        System.out.println("3  - Pivo no inicio
(ordenado)           6  - Pivo no final (ordenado)           9  - Pivo
aleatorio (ordenado)           12 - Pivo mediano (ordenado)");
        System.out.println("Digite 0 para sair.");

        escolha = sc.nextInt();

        left = 0;
        right = array1.length - 1;
        long startTime, endTime;
        double duration;

        switch(escolha){
            case 1:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();  // Medir em
nanosegundos
                QuickSortFirstPivot(array1, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();
```

```
                System.out.println("Tempo de execucao: " + duration  +
" ms"); // 0.0068 ms
                break;

            case 2:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortFirstPivot(array2, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.0077 ms
                break;

            case 3:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortFirstPivot(array3, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
```

```
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.0067 ms
                break;

            case 4:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortLastPivot(array1, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.006099 ms
                break;

            case 5:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortLastPivot(array2, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
```

```
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.006501 ms
                break;

            case 6:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortLastPivot(array3, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.006499 ms
                break;

            case 7:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortRandomPivot(array1, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
```

```
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.027699 ms
                break;

            case 8:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortRandomPivot(array2, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration + "
ms"); //0.0427 ms
                break;

            case 9:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortRandomPivot(array3, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
```

```
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.059899 ms
                break;

            case 10:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortMedianOfThree(array1, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array1[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.0076 ms
                break;

            case 11:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortMedianOfThree(array2, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
```

```
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array2[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.0061 ms
                break;

            case 12:
                //Vetor desordenado
                System.out.println("Array antes:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                //Medir o tempo de execução
                startTime = System.nanoTime();
                QuickSortMedianOfThree(array3, left, right);
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1_000_000.0;

                //Vetor ordenado
                System.out.println("Array depois:");
                for (int i = 0; i < 10; i++) {
                    System.out.print(array3[i] + " ");
                }
                System.out.println();

                System.out.println("Tempo de execucao: " + duration +
" ms"); //0.007201 ms
                break;

            default:
                System.out.println("Invalido\n");;

            }

        sc.close();
    }
}
```