LAB 06 - Relatório de Desempenho do Quicksort com Diferentes Estratégias de Escolha do Pivô

Aluno: Arthur Costa Serra Negra

1. Funcionamento de cada estratégia de escolha do pivô:

- **FirstPivot:** O pivô é o primeiro elemento do array. Durante a execução do Quicksort, o array é dividido em duas partes, sendo os elementos menores que o pivô movidos para a esquerda e os maiores para a direita. Após a divisão, o algoritmo recursivamente ordena as subpartes.

```java
public static void FirstPivotQuicksort(int esq, int dir, int[] array) {
    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, array);
            i++;
            j--;
        }
    }
    if (esq < j)  FirstPivotQuicksort(esq, j, array);
    if (i < dir)  FirstPivotQuicksort(i, dir, array);
}
```

- **LastPivot:** O pivô é o último elemento do array. O funcionamento é semelhante ao do FirstPivot, mas o pivô é o elemento da última posição do array.

```java
public static void LastPivotQuicksort(int esq, int dir, int[] array) {
    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, array);
            i++;
            j--;
        }
    }
    if (esq < j)  LastPivotQuicksort(esq, j, array);
    if (i < dir)  LastPivotQuicksort(i, dir, array);
}
```

- **RandomPivot:** O pivô é escolhido aleatoriamente dentro do intervalo atual do array. Esta estratégia visa evitar o pior caso.

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

- **MedianOfThree**: O pivô é escolhido como a mediana de três elementos: o primeiro, o meio e o último. Esta abordagem visa melhorar a escolha do pivô em arrays, evitando o caso em que o pivô escolhido é o menor ou maior elemento.

```java
public static void MedianOfThreQuicksort(int esq, int dir, int[] array) {
    int i = esq, j = dir;
    int pivo = array[ MedianOfThre ( (esq), ((esq+dir)/2), (dir),  array ) ];
    while (i <= j) {
        while (array[i] < pivo) i++;
        while (array[j] > pivo) j--;
        if (i <= j) {
            swap(i, j, array);
            i++;
            j--;
        }
    }
    if (esq < j)  MedianOfThreQuicksort(esq, j, array);
    if (i < dir)  MedianOfThreQuicksort(i, dir, array);
}
public static int MedianOfThre(int a, int b, int c, int[] array) {
    int A = array[a];
    int B = array[b];
    int C = array[c];

    if ((A > B && A < C) || (A > C && A < B)) {
        return a;
    } else if ((B > A && B < C) || (B > C && B < A)) {
        return b;
    } else {
        return c;
    }
}
```

2. Desempenho observado em cada cenário:

Abaixo estão as tabelas que mostram os tempos de execução observados em nanosegundos para cada estratégia e tipo de array:

Tamanho do Array: 100	Ordered	NearlyOrdered	Random
FirstPivot	23000 ns	23900 ns	18800 ns
LastPivot	52300 ns	25500 ns	19900 ns
RandomPivot	219100 ns	98900 ns	70300 ns
MedianOfThree	24700 ns	27800 ns	23700 ns

Tamanho do Array: 1000	Ordered	NearlyOrdered	Random
FirstPivot	335600 ns	80300 ns	62200 ns
LastPivot	303100 ns	67300 ns	65500 ns
RandomPivot	385600 ns	340400 ns	320500 ns
MedianOfThree	175200 ns	198900 ns	252600 ns

Tamanho do Array: 10000	Ordered	NearlyOrdered	Random
FirstPivot	560800 ns	614900 ns	968400 ns
LastPivot	567200 ns	991800 ns	1132200 ns
RandomPivot	2326500 ns	885700 ns	1687500 ns
MedianOfThree	1845200 ns	579400 ns	987300 ns

3. Discussão sobre a eficiência:

Com base nos resultados acima, podemos observar que a estratégia **FirstPivot** foi geralmente eficiente para arrays de tamanho pequeno (100 elementos), mas o desempenho piorou conforme o tamanho do array aumentou, especialmente para o caso "Ordered".

A estratégia **MedianOfThree** foi a mais eficiente no geral, especialmente para arrays de tamanho médio e grande (1000 e 10000), em particular nos casos de arrays ordenados e quase ordenados. Isso se deve à sua capacidade de escolher um pivô que reflete melhor a distribuição dos dados, evitando cenários em que o pivô é sempre o maior ou menor valor, o que degrada o desempenho do Quicksort.

Por outro lado, **RandomPivot** mostrou-se mais ineficiente no geral, com o maior tempo de execução para arrays aleatórios, especialmente quando o tamanho do array aumentou.

Conclusão:

Para arrays pequenos, qualquer estratégia funciona razoavelmente bem, mas para arrays maiores, a escolha do pivô se torna crítica. A estratégia de **MedianOfThree** foi a mais eficiente para a maioria dos cenários, enquanto **RandomPivot** foi a mais lenta. Assim, para otimizar o desempenho do Quicksort em uma ampla gama de casos, a escolha de um pivô com base na mediana de três elementos é recomendada.

Código:

```java
import java.util.*;



public class Main {

    static public void main(String[] args) {



        int[] sizes = {100, 1000, 10000};

        String[] methods = {"FirstPivot", "LastPivot", "RandomPivot", "MedianOfThree"};

        String[] arrayTypes = {"Ordered", "NearlyOrdered", "Random"};



        for (int size : sizes) {

            int[] array = null;

            for(String type : arrayTypes) {
```

```
                if(type.equals("Ordered")){

                    array = generateNearlyOrderedArray(size);

                } else if (type.equals("NearlyOrdered")){

                    array = generateNearlyOrderedArray(size);

                } else {

                    array = generateRandomArray(size);

                }

                for (String method : methods) {

                    int[] arrayToSort = Arrays.copyOf(array, array.length);

                    long time = SortTime(arrayToSort, method);

                    System.out.printf("    %s: %d ns \t \t (Array Size - %s) (Array Type - %s)\n", method, time,
size, type);

                }

            }

            System.out.println();

        }

    }



    public static void FirstPivotQuicksort(int esq, int dir, int[] array) {

        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, array);

                i++;

                j--;

            }

        }

        if (esq < j)  FirstPivotQuicksort(esq, j, array);

        if (i < dir)  FirstPivotQuicksort(i, dir, array);
```

```java
}

public static void LastPivotQuicksort(int esq, int dir, int[] array) {
    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, array);
            i++;
            j--;
        }
    }
    if (esq < j)  LastPivotQuicksort(esq, j, array);
    if (i < dir)  LastPivotQuicksort(i, dir, array);
}

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

```
}

public static void MedianOfThreQuicksort(int esq, int dir, int[] array) {
    int i = esq, j = dir;
    int pivo = array[ MedianOfThre ( (esq), ((esq+dir)/2), (dir),  array ) ];
    while (i <= j) {
        while (array[i] < pivo) i++;
        while (array[j] > pivo) j--;
        if (i <= j) {
            swap(i, j, array);
            i++;
            j--;
        }
    }
    if (esq < j)  MedianOfThreQuicksort(esq, j, array);
    if (i < dir)  MedianOfThreQuicksort(i, dir, array);
}
public static int MedianOfThre(int a, int b, int c, int[] array) {
    int A = array[a];
    int B = array[b];
    int C = array[c];

    if ((A > B && A < C) || (A > C && A < B)) {
        return a;
    } else if ((B > A && B < C) || (B > C && B < A)) {
        return b;
    } else {
        return c;
    }
}

public static void swap (int A, int B, int[] array) {
```

```
        int aux = array[A];
        array[A] = array[B];
        array[B] = aux;
    }

    public static int[] generateOrderedArray(int size) {
        int[] array = new int[size];
        for (int i = 0; i < size; i++) {
            array[i] = i;
        }
        return array;
    }

    public static int[] generateNearlyOrderedArray(int size) {
        int[] array = generateOrderedArray(size);
        Random rand = new Random();
        for (int i = 0; i < size/10; i++) {
            int index1 = rand.nextInt(size);
            int index2 = rand.nextInt(size);
            swap(index1, index2, array);
        }
        return array;
    }

    public static int[] generateRandomArray(int size) {
        int[] array = new int[size];
        Random rand = new Random();
        for (int i = 0; i < size; i++)
            array[i] = rand.nextInt(size * 10);
        return array;
    }
```

```java
    public static long SortTime(int[] array, String method) {

        long startTime = System.nanoTime();

        if(method.equals("FirstPivot")){

            FirstPivotQuicksort(0, array.length - 1, array);

        }else if(method.equals("LastPivot")){

            LastPivotQuicksort(0, array.length - 1, array);

        }else if(method.equals("RandomPivot")){

            RandomPivotQuicksort(0, array.length - 1, array);

        }else{

            MedianOfThreQuicksort(0, array.length - 1, array);

        }

        long endTime = System.nanoTime();

        return endTime - startTime;

    }

}
```

Saída:

```
LABS > LAB06 > a.out
    FirstPivot: 21300 ns         (Array Size - 100) (Array Type - Ordered)
    LastPivot: 33600 ns          (Array Size - 100) (Array Type - Ordered)
    RandomPivot: 350600 ns       (Array Size - 100) (Array Type - Ordered)
    MedianOfThree: 22800 ns          (Array Size - 100) (Array Type - Ordered)
    FirstPivot: 75300 ns         (Array Size - 100) (Array Type - NearlyOrdered)
    LastPivot: 37000 ns          (Array Size - 100) (Array Type - NearlyOrdered)
    RandomPivot: 33800 ns        (Array Size - 100) (Array Type - NearlyOrdered)
    MedianOfThree: 15000 ns          (Array Size - 100) (Array Type - NearlyOrdered)
    FirstPivot: 17200 ns         (Array Size - 100) (Array Type - Random)
    LastPivot: 15700 ns          (Array Size - 100) (Array Type - Random)
    RandomPivot: 34400 ns        (Array Size - 100) (Array Type - Random)
    MedianOfThree: 80000 ns          (Array Size - 100) (Array Type - Random)

    FirstPivot: 63000 ns         (Array Size - 1000) (Array Type - Ordered)
    LastPivot: 106500 ns         (Array Size - 1000) (Array Type - Ordered)
    RandomPivot: 183600 ns       (Array Size - 1000) (Array Type - Ordered)
    MedianOfThree: 172700 ns         (Array Size - 1000) (Array Type - Ordered)
    FirstPivot: 70800 ns         (Array Size - 1000) (Array Type - NearlyOrdered)
    LastPivot: 68400 ns          (Array Size - 1000) (Array Type - NearlyOrdered)
    RandomPivot: 67800 ns        (Array Size - 1000) (Array Type - NearlyOrdered)
    MedianOfThree: 59200 ns          (Array Size - 1000) (Array Type - NearlyOrdered)
    FirstPivot: 79200 ns         (Array Size - 1000) (Array Type - Random)
    LastPivot: 81500 ns          (Array Size - 1000) (Array Type - Random)
    RandomPivot: 110300 ns       (Array Size - 1000) (Array Type - Random)
    MedianOfThree: 86100 ns          (Array Size - 1000) (Array Type - Random)

    FirstPivot: 905200 ns        (Array Size - 10000) (Array Type - Ordered)
    LastPivot: 1040300 ns        (Array Size - 10000) (Array Type - Ordered)
    RandomPivot: 779100 ns       (Array Size - 10000) (Array Type - Ordered)
    MedianOfThree: 527800 ns         (Array Size - 10000) (Array Type - Ordered)
    FirstPivot: 437000 ns        (Array Size - 10000) (Array Type - NearlyOrdered)
    LastPivot: 607600 ns         (Array Size - 10000) (Array Type - NearlyOrdered)
    RandomPivot: 779400 ns       (Array Size - 10000) (Array Type - NearlyOrdered)
    MedianOfThree: 541300 ns         (Array Size - 10000) (Array Type - NearlyOrdered)
    FirstPivot: 731600 ns        (Array Size - 10000) (Array Type - Random)
    LastPivot: 714100 ns         (Array Size - 10000) (Array Type - Random)
    RandomPivot: 2509000 ns          (Array Size - 10000) (Array Type - Random)
    MedianOfThree: 1238300 ns        (Array Size - 10000) (Array Type - Random)

```