Relatório Projeto 2 V1.1 AED 2023/2024

Nome: João António Faustino Vaz Nº Estudante: 2022231087
PL (inscrição): 8 Email: joao.vaz1810@gmail.com

IMPORTANTE:

- Os textos das conclusões devem ser manuscritos... o texto deve obedecer a este requisito para não ser penalizado.
- Texto para além das linhas reservadas, ou que não seja legível para um leitor comum, não será tido em conta.
- O relatório deve ser submetido num único PDF que deve incluir os anexos. A não observância deste formato é penalizada.

1. Planeamento

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5
Árvore Binária	**feito**				
Árvore Binária Pesquisa		**feito**			
Árvore AVL			**incompleta**	**feito**	
Árvore VP				**feito**	
Finalização Relatório					**feito**

2. Recolha de Resultados *(tabelas)*

		binaria		
tam	A	B	C	D
200000	334,33	360,9	359,52	19,5
400000	2563,62	2193,7	2819,02	70,09
600000	6848,86	6187,4	6345,3	189,9
800000	9948,23	9872,4	10023,5	350,55
1000000	12242,5	12456,3	14234,2	521,03

		avl		
tam	A	B	C	D
200000	0,034	0,032	0,14	0,07
400000	0,064	0,061	0,27	0,11
600000	0,098	0,11	0,48	0,17
800000	0,11	0,13	0,56	0,2
1000000	0,14	0,14	0,68	0,27

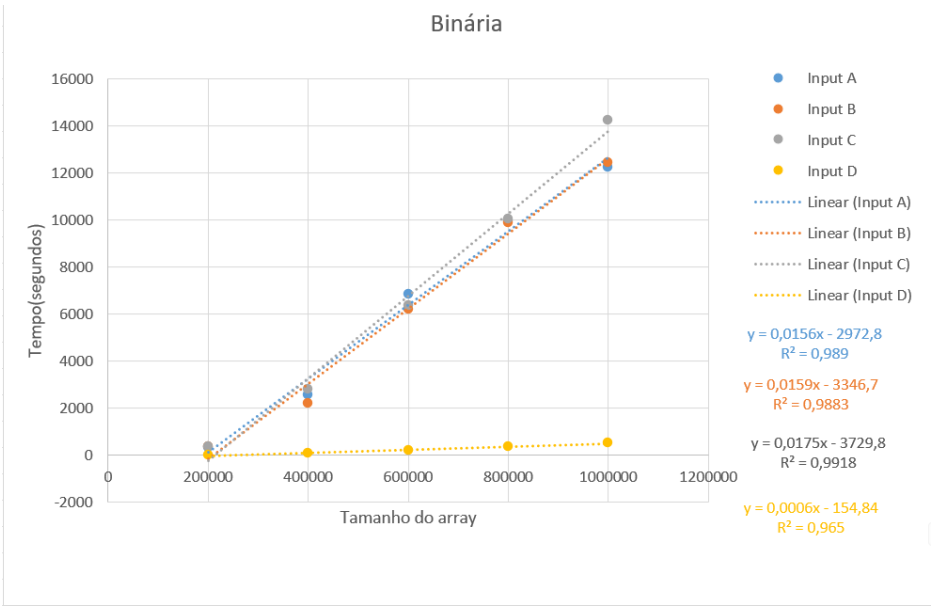
		rotaçoes		
200000	181913	181891	84787	8506
400000	363830	363797	169547	17079
600000	545673	545819	254244	25456
800000	727433	727718	339300	33652
1000000	909592	909405	423149	42340

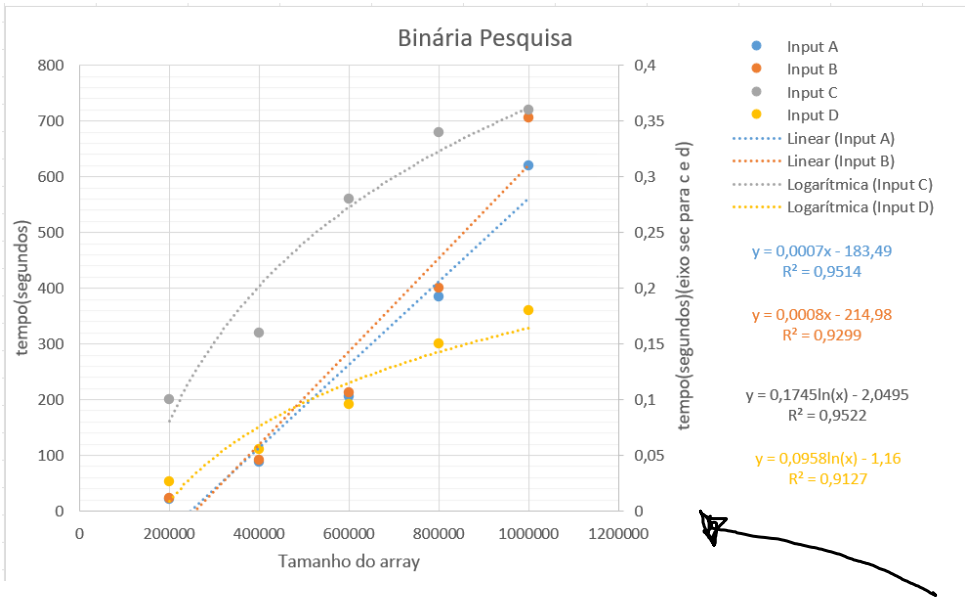
		Binária Pesquisa		
tam	A	B	C	D
200000	21,9	23,01	0,1	0,026
400000	87,83	92,17	0,16	0,055
600000	206,1	212,72	0,28	0,096
800000	383,88	400,7	0,34	0,15
1000000	619,39	704,86	0,36	0,18

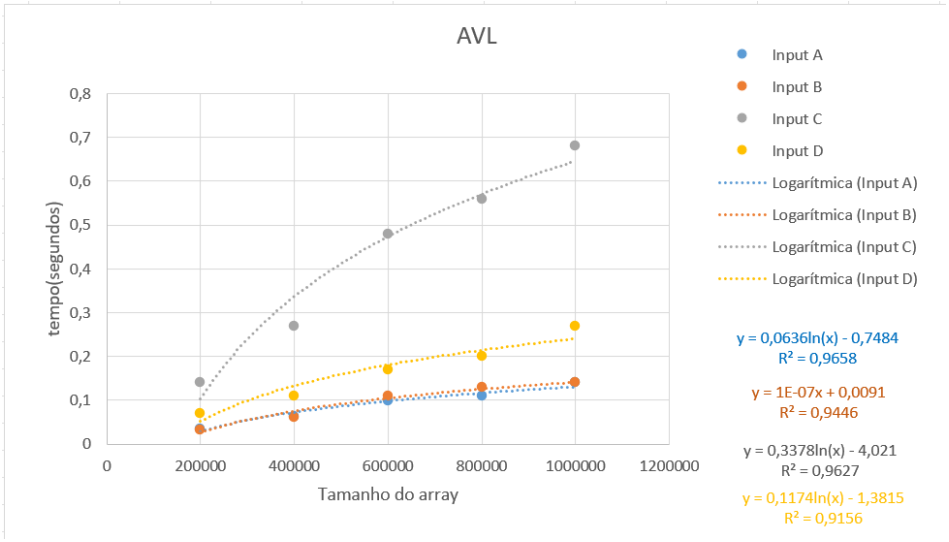
		vp		
tam	A	B	C	D
200000	0,055	0,05	0,11	0,045
400000	0,11	0,11	0,29	0,12
600000	0,17	0,17	0,48	0,17
800000	0,21	0,19	0,66	0,23
1000000	0,25	0,27	0,79	0,3

		rotaçoes		
200000	181867	181836	70585	7106
400000	363782	363789	141262	14062
600000	545633	545678	212292	21073
800000	727548	727575	282395	28020
1000000	909321	909542	353327	35238

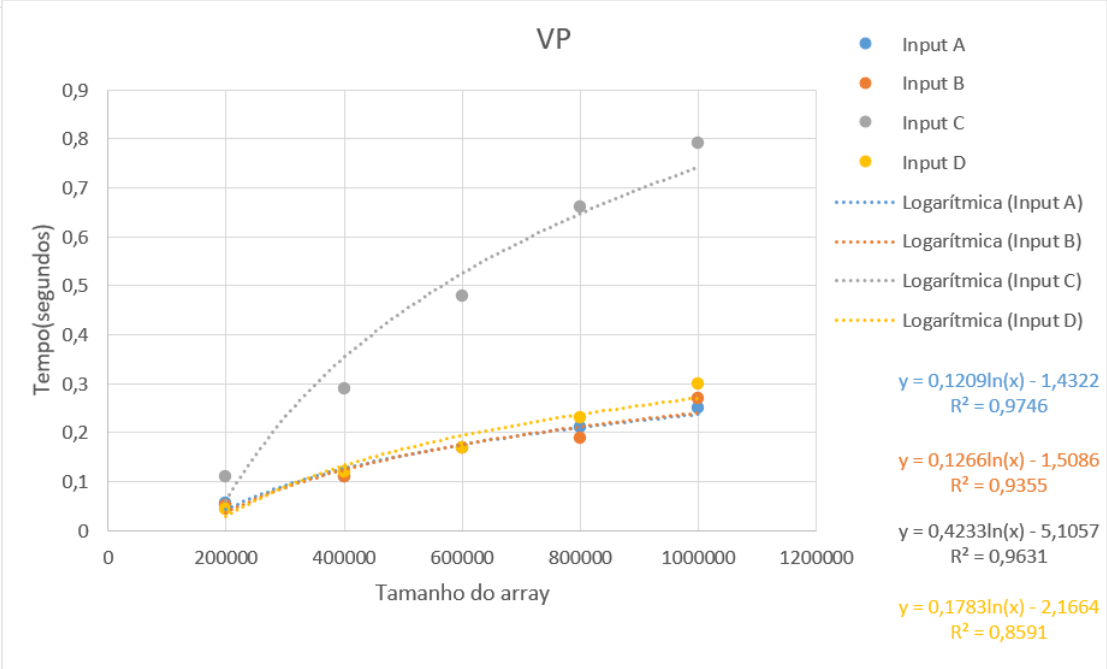
3. Visualização de Resultados *(gráficos)*







Eixo secundário para o C e D



4. Conclusões *(as linhas desenhadas representam a extensão máxima de texto manuscrito)*

4.1 Tarefa 1

Relativamente à árvore binária simples podemos ver através do gráfico que os tempos de execução para os três primeiros inputs (A, B e C) são muito parecidos. No input D já temos uma diminuição dos tempos de execução muito significativa devido ao facto de o input ter 90% dos números repetidos. Em relação à complexidade é O(n), ou seja, percorremos no pior caso a árvore toda e isso observa-se no gráfico.

4.2 Tarefa 2

Relativamente à árvore binária de pesquisa os tempos são muito inferiores. Optei por usar a regressão linear nos inputs A e B visto que a árvore será toda percorrida nesses inputs e tendera só para um lado caso se faça uma lista. Nos inputs C e D isso já não acontece

e teremos uma arvore mais equilibrada sendo que a complexidade é O(log n) e portanto utilizei regressão logaritmicas na sua representação.

4.3 Tarefa 3

Na AVL, vemos claramente uma redução do tempo. A complexidade é O(log n) e consiguimos tirar varias conclusões:
O tempo de execução para o input A e B é muito similar, no C o tempo aumenta o que nos indica que as Arvores AVL vão se comportar de forma muito rapida quando os inputs vão estão ordenados. Para além disso, podemos ver que o número de rotações é significativamente baixo no conjunto D, o que faz sentido devido aos tais 90% repetidos.

4.4 Tarefa 4

Em relação às VP, a complexidade é igualmente O(log n) mas podemos ver uma melhoria de desempenho principalmente em relação aos conjuntos C e D.
É interessante ver que em termos de tempo as AVL e VP são muito similares, no entanto as rotações nas VP são menores o que demonstra uma vantagem sobre as AVL.

Anexo B - Código de Autor

Todo o código feito na arvore binária simples, exceto a função de dar print à árvore e do funcionamento da stack em java.

Todo o código feito Binária de pesquisa à exceção da função de dar print à arvore.

Na AVL, as funções insert, balancefactor , height, updateheigh e rebalance.

Na VP, a função getcolor.

Anexo C - Referências

https://www.geeksforgeeks.org/stack-class-in-java/

Função de dar print à arvore – chatgpt

Função rotateleft e rotateright na AVL – chatgpt

Função insert, rotateleft e rotateright na VP - chatgpt

Código
Binaria

```java
import java.util.*;

public class Binaria {
    public class Node{
        int data;
        Node left;
        Node right;
        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    Node root=null;
    public void insert(int data) {
        if (root == null) {
           root=new Node(data);
           return;
        }
        Random random = new Random();
        Node atual = root;
        while (true) {
            if (random.nextBoolean()) {
                if (atual.left == null) {
                    atual.left = new Node(data);
                    return;
                } else {
                    atual = atual.left;
                }
            } else {
                if (atual.right == null) {
                    atual.right = new Node(data);
                    return;
                } else {
                    atual = atual.right;
                }
            }
        }
    }

    public boolean find(int data) {
        if (root == null) {
            return false;
        }
        Stack<Node> pilha = new Stack<>();
        pilha.push(root);
        while (!pilha.isEmpty()) {
            Node atual = pilha.pop();
            if (atual.data == data) {
                return true;
            }
            if (atual.right != null) {
                pilha.push(atual.right);
            }
            if (atual.left != null) {
                pilha.push(atual.left);
            }
        }
        return false;
    }

    public void printTree(Node node) {

        printTreeHelper(node, 0);
    }

    private void printTreeHelper(Node node, int level) {
        if (node != null) {
            printTreeHelper(node.right, level + 1);
            for (int i = 0; i < level; i++) {
                System.out.print("   ");
            }
            System.out.println(node.data);
            printTreeHelper(node.left, level + 1);
        }
    }

    public static void main(String[] args) {
        int size = 1000000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<20;i++){
            teste.set(i,i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        for (int j = (int)(0.9*size); j < size; j++) {
            int num = random.nextInt(size);
            input.set(j, num);
        }
        Collections.sort(input);
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        ArrayList<Integer> inputd = new ArrayList<>(input);
        int numDistinct = size / 10;
        for (int k = numDistinct; k < size; k++) {
            inputd.set(k, inputd.get(k % numDistinct));
        }
        Collections.shuffle(inputd);
        Binaria arvoreBinaria=new Binaria();
        long ti=System.nanoTime();
        for(int i=0;i<size;i++){
            if(!arvoreBinaria.find(inputd.get(i))){
                arvoreBinaria.insert(inputd.get(i));
            }
        }
        long tf=System.nanoTime();
        System.out.println(tf-ti);
    }
}
```

Binária Pesquisa

```java
import java.util.*;

public class BinariaPesquisa {
    public class Node {
        int data;
        Node left;
        Node right;

        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    Node root = null;
    public boolean insert(int data) {
        if (root == null) {
            root = new Node(data);
            return false;
        }
        Node atual = root;
        while (true) {
            if (data < atual.data) {
                if (atual.left == null) {
                    atual.left = new Node(data);
                    return false;
                }
                atual = atual.left;
            } else if (data > atual.data) {
                if (atual.right == null) {
                    atual.right = new Node(data);
                    return false;
                }
                atual = atual.right;
            } else {
                return true;
            }
        }
    }
    public void printTree(Node node) {
        printTreeHelper(node, 0);
    }

    private void printTreeHelper(Node node, int level) {
        if (node != null) {
            printTreeHelper(node.right, level + 1);
            for (int i = 0; i < level; i++) {
                System.out.print("    ");
            }
            System.out.println(node.data);
            printTreeHelper(node.left, level + 1);
        }
    }

    public static void main(String[] args) {
        int size = 1000000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<20;i++){
            teste.set(i,i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        for (int j = (int)(0.9*size); j < size; j++) {
            int num = random.nextInt(size);
            input.set(j, num);
        }
        Collections.sort(input);
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        ArrayList<Integer> inputd = new ArrayList<>(input);
        int numDistinct = size / 10;
        for (int k = numDistinct; k < size; k++) {
            inputd.set(k, inputd.get(k % numDistinct));
        }
        Collections.shuffle(inputd);
        BinariaPesquisa arvoreBinariaPesquisa = new BinariaPesquisa();
        long ti=System.nanoTime();
        for (int i = 0; i < size; i++) {
            arvoreBinariaPesquisa.insert(inputc.get(i));
        }
        long tf=System.nanoTime();
        System.out.println((tf-ti));
```

AVL

```java
import java.util.*;

public class AVL {
    public class Node {
        int data;
        Node left;
        Node right;
        int height;

        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
            this.height = 1;
        }
    }
    Node root = null;
    int rotationCount = 0;
    public int height(Node node) {
        if (node != null) {
            return node.height;
        } else {
            return 0;
        }
    }
    public void updateHeight(Node node) {
        if (node != null) {
            int leftChildHeight = height(node.left);
            int rightChildHeight = height(node.right);
            node.height = Math.max(leftChildHeight, rightChildHeight) + 1;
        }
    }
    public int balanceFactor(Node node) {
        if (node != null) {
            return height(node.left) - height(node.right);
        } else {
            return 0;
        }
    }
    public Node insert(Node node, int data) {
        if (node == null) {
            return new Node(data);
        }
        if (data < node.data) {
            node.left = insert(node.left, data);
        } else if (data > node.data) {
            node.right = insert(node.right, data);
        } else {
            return node;
        }
        updateHeight(node);
        return rebalance(node);
    }
    private Node rotateRight(Node node) {
        Node leftChild = node.left;
        node.left = leftChild.right;
        leftChild.right = node;
        updateHeight(node);
        updateHeight(leftChild);
        return leftChild;
    }
    private Node rotateLeft(Node node) {
        Node rightChild = node.right;
        node.right = rightChild.left;
        rightChild.left = node;
        updateHeight(node);
        updateHeight(rightChild);
        return rightChild;
    }
    public Node rebalance(Node node) {
        updateHeight(node);
        int balanceFactor = balanceFactor(node);

        if (balanceFactor > 1) {
            if (height(node.left.left) >= height(node.left.right)) {
                node = rotateRight(node);
            } else {
                node.left = rotateLeft(node.left);
                node = rotateRight(node);
            }
            rotationCount++;
        }
        else if (balanceFactor < -1) {
            if (height(node.right.right) >= height(node.right.left)) {
                node = rotateLeft(node);
            } else {
                node.right = rotateRight(node.right);
                node = rotateLeft(node);
            }
            rotationCount++;
        }

        return node;
    }
    public void printTree(Node node) {
        printTreeHelper(node, 0);
    }
    private void printTreeHelper(Node node, int level) {
        if (node != null) {
            printTreeHelper(node.right, level + 1);
            for (int i = 0; i < level; i++) {
                System.out.print("    ");
            }
            System.out.println(node.data);
            printTreeHelper(node.left, level + 1);
        }
    }
    public static void main(String[] args) {
        int size = 1000000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<20;i++){
            teste.set(i,i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        for (int j = (int)(0.9*size); j < size; j++) {
            int num = random.nextInt(size);
            input.set(j, num);
        }
        Collections.sort(input);
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        ArrayList<Integer> inputd = new ArrayList<>(input);
        int numDistinct = size / 10;
        for (int k = numDistinct; k < size; k++) {
            inputd.set(k, inputd.get(k % numDistinct));
        }
        Collections.shuffle(inputd);
        AVL avl = new AVL();
        long ti = System.nanoTime();
        for(int i = 0; i < size; i++) {
            avl.root = avl.insert(avl.root, inputc.get(i));
        }
        long tf = System.nanoTime();
        System.out.println("Tempo de inserção: " + (tf - ti) + " nanossegundos");
        System.out.println("Número total de rotações durante a inserção: " + avl.rotationCount);
    }
}
```

VP

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class VP {
    public class Node {
        int data;
        Node left;
        Node right;
        char color;

        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
            this.color = 'r';
        }
    }

    Node root = null;
    int rotationCount = 0;

    char getColor(Node node) {
        if (node != null) {
            return node.color;
        } else {
            return 'b';
        }
    }

    public Node insert(Node node, int data) {
        if (node == null) {
            Node newNode = new Node(data);
            if (root == null) {
                newNode.color = 'b';
            }
            return newNode;
        }

        if (data < node.data) {
            node.left = insert(node.left, data);
        } else if (data > node.data) {
            node.right = insert(node.right, data);
        } else {
            return node;
        }

        if (getColor(node.left) == 'r' && getColor(node.left.left) == 'r') {
            if (getColor(node.right) == 'b' || getColor(node.right) == ' ') {
                node = rotateRight(node);
                rotationCount++;
            } else {
                node.color = 'r';
                node.left.color = 'b';
                node.right.color = 'b';
            }
        } else if (getColor(node.left) == 'r' && getColor(node.left.right) == 'r') {
            if (getColor(node.right) == 'b' || getColor(node.right) == ' ') {
                node.left = rotateLeft(node.left);
                node = rotateRight(node);
                rotationCount++;
            } else {
                node.color = 'r';
                node.left.color = 'b';
                node.right.color = 'b';
            }
        } else if (getColor(node.right) == 'r' && getColor(node.right.left) == 'r') {
            if (getColor(node.left) == 'b' || getColor(node.left) == ' ') {
                node.right = rotateRight(node.right);
                node = rotateLeft(node);
                rotationCount++;
            } else {
                node.color = 'r';
                node.left.color = 'b';
                node.right.color = 'b';
            }
        } else if (getColor(node.right) == 'r' && getColor(node.right.right) == 'r') {
            if (getColor(node.left) == 'b' || getColor(node.left) == ' ') {
                node = rotateLeft(node);
                rotationCount++;
            } else {
                node.color = 'r';
                node.left.color = 'b';
                node.right.color = 'b';
            }

        }
        if (root == node) {
            node.color = 'b';
        }

        return node;
    }


    private Node rotateRight(Node node) {
        Node leftChild = node.left;
        node.left = leftChild.right;
        leftChild.right = node;
        leftChild.color = 'b';
        node.color = 'r';
        return leftChild;
    }

    private Node rotateLeft(Node node) {
        Node rightChild = node.right;
        node.right = rightChild.left;
        rightChild.left = node;
        rightChild.color = 'b';
        node.color = 'r';
        return rightChild;
    }

    public void printTree(Node node) {
        printTreeHelper(node, 0);
    }

    private void printTreeHelper(Node node, int level) {
        if (node != null) {
            printTreeHelper(node.right, level + 1);
            for (int i = 0; i < level; i++) {
                System.out.print("    ");
            }
            System.out.println(node.data + "(" + node.color + ")");
            printTreeHelper(node.left, level + 1);
        }
    }

    public static void main(String[] args) {
        int size = 1000000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<20;i++){
            teste.set(i,i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        for (int j = (int)(0.9*size); j < size; j++) {
            int num = random.nextInt(size);
            input.set(j, num);
        }
        Collections.sort(input);
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        ArrayList<Integer> inputd = new ArrayList<>(input);
        int numDistinct = size / 10;
        for (int k = numDistinct; k < size; k++) {
            inputd.set(k, inputd.get(k % numDistinct));
        }
        Collections.shuffle(inputd);
        VP arvoreVP = new VP();
        long ti = System.nanoTime();
        for(int i = 0; i < size; i++) {
            arvoreVP.root = arvoreVP.insert(arvoreVP.root, inputc.get(i));
        }
        long tf = System.nanoTime();
        System.out.println("Tempo de inserção: " + (tf - ti) + " nanossegundos");
        System.out.println("Número total de rotações durante a inserção: " + arvoreVP.rotationCount);
    }
}
```