# **LAPORAN** PRAKTIKUM **ALGORITMA DAN STRUKTUR DATA**

## **PERTEMUAN 14 TREE**

Dosen Pengampu : Triana Fatmawati, S.T, M.T


Muhammad Afiq Firdaus

2341760189 / 21

SIB-1E


**PROGRAM STUDI D-IV SISTEM INFORMASI BISNIS**

**JURUSAN TEKNOLOGI INFORMASI**

**POLITEKNIK NEGERI MALANG**

**2024**

## Implementasi Binary Search Tree menggunakan Linked List

### Percobaan 1

- **Kode Program Node21.java**

```
public class Node21 {

    int data;
    Node21 left;
    Node21 right;

    public Node21(){

    }
    public Node21(int data){
        this.left = null;
        this.right = null;
        this.data = data;
    }
}
```

- **Kode Program BinaryTree21.java**

```
public class BinaryTree21 {
    Node21 root;

    public BinaryTree21() {
        root = null;
    }

    boolean isEmpty() {
        return root == null;
    }
```

```
void add(int data) {
       if (isEmpty()) {
           root = new Node21(data);
       } else {
           Node21 current = root;
           while (true) {
               if (data < current.data) {
                   if (current.left == null) {
                       current.left = new Node21(data);
                       break;
                   } else {
                       current = current.left;
                   }
               } else if (data > current.data) {
                   if (current.right == null) {
                       current.right = new Node21(data);
                       break;
                   } else {
                       current = current.right;
                   }
               } else {
                   break; // Data already exists in the tree, do nothing
               }
           }
       }
   }

   boolean find(int data) {
       Node21 current = root;
       while (current != null) {
           if (current.data == data) {
               return true;
           } else if (data < current.data) {
               current = current.left;
           } else {
               current = current.right;
           }
       }
       return false;
   }

   void traversePreOrder(Node21 node) {
       if (node != null) {
           System.out.print(" " + node.data);
           traversePreOrder(node.left);
           traversePreOrder(node.right);
       }
   }
```

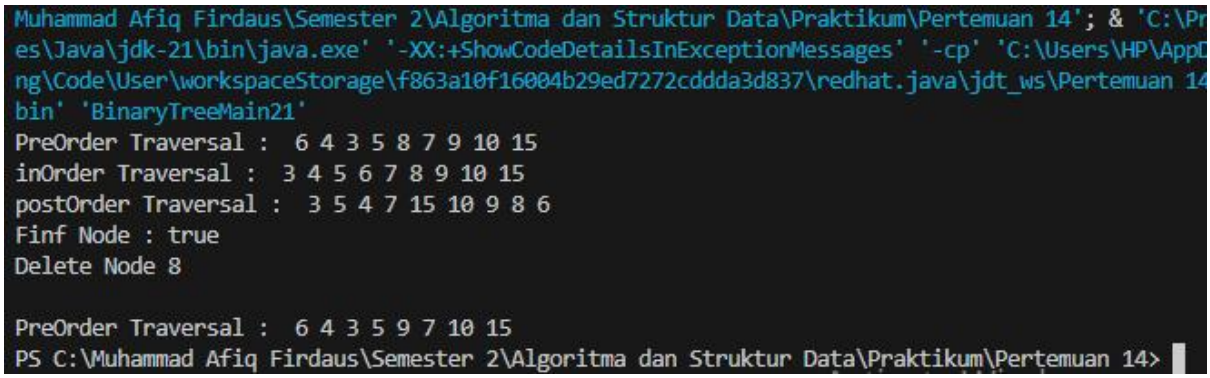
```
void traverseInOrder(Node21 node) {
       if (node != null) {
           traverseInOrder(node.left);
           System.out.print(" " + node.data);
           traverseInOrder(node.right);
       }
   }

   Node21 getSuccessor(Node21 del) {
       Node21 successor = del.right;
       Node21 successorParent = del;
       while (successor.left != null) {
           successorParent = successor;
           successor = successor.left;
       }
       if (successor != del.right) {
           successorParent.left = successor.right;
           successor.right = del.right;
       }
       return successor;
   }

   void delete(int data) {
       if (isEmpty()) {
           System.out.println("Tree is empty!");
           return;
       }
       Node21 parent = root;
       Node21 current = root;
       boolean isLeftChild = false;
       while (current != null) {
           if (current.data == data) {
               break;
           } else if (data < current.data) {
               parent = current;
               current = current.left;
               isLeftChild = true;
           } else {
               parent = current;
               current = current.right;
               isLeftChild = false;
           }
       }
```

```
if (current == null) {
            System.out.println("Couldn't find data!");
            return;
        } else {
            if (current.left == null && current.right == null) {
                if (current == root) {
                    root = null;
                } else {
                    if (isLeftChild) {
                        parent.left = null;
                    } else {
                        parent.right = null;
                    }
                }
            } else if (current.left == null) {
                if (current == root) {
                    root = current.right;
                } else {
                    if (isLeftChild) {
                        parent.left = current.right;
                    } else {
                        parent.right = current.right;
                    }
                }
            } else if (current.right == null) {
                if (current == root) {
                    root = current.left;
                } else {
                    if (isLeftChild) {
                        parent.left = current.left;
                    } else {
                        parent.right = current.left;
                    }
                }
            } else {
                Node21 successor = getSuccessor(current);
                if (current == root) {
                    root = successor;
                } else {
                    if (isLeftChild) {
                        parent.left = successor;
                    } else {
                        parent.right = successor;
                    }
                }
                successor.left = current.left;
            }
        }
    }
```

- **Code Program BinaryTreeMain21.java**

```
public class BinaryTreeMain21 {
    public static void main(String[] args) {
        BinaryTree21 bt = new BinaryTree21();
        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);
        System.out.print("PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("inOrder Traversal : ");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.print("postOrder Traversal : ");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Finf Node : " + bt.find(5));
        System.out.println("Delete Node 8 ");
        bt.delete(8);
        System.out.println("");
        System.out.print("PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}
```

- **Berikut adalah hasil run dari code diatas**




**Pertanyaan Percobaan**

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?

2. Untuk apakah di class Node, kegunaan dari atribut left dan right?

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
if(data<current.data){
      if(current.left!=null)
            { current = current.left;
      }else{
      current.left = new Node(data); break;
      }
}
```

**Jawaban**

1. Binary Search Tree (BST) lebih efektif untuk pencarian data dibandingkan dengan binary tree biasa karena struktur dan aturan yang diterapkan pada BST. Dalam BST, setiap node kiri memiliki nilai lebih kecil dan setiap node kanan memiliki nilai lebih besar dari node induknya. Hal ini memungkinkan pencarian yang terarah dan terstruktur, yang secara signifikan mengurangi jumlah node yang perlu diperiksa.

2. a) Atribut left dan right pada class Node09 digunakan untuk menunjuk ke node anak kiri dan kanan, memungkinkan struktur untuk menyimpan data, dan mendukung operasi seperti penambahan, pencarian, dan traversal secara efisien.

b) Ketika objek tree pertama kali dibuat, nilai dari root adalah null, yang menunjukkan bahwa tree masih kosong dan belum memiliki node apa pun. Pada saat node pertama ditambahkan ke tree, node tersebut akan menjadi root.

3. Atribut root di dalam class BinaryTree09 berfungsi sebagai titik awal untuk semua operasi tree dan memastikan bahwa tree dapat diakses, dimodifikasi, dan ditelusuri dengan benar. Tanpa atribut root, tidak akan memiliki cara efektif untuk mengakses dan mengelola node-node dalam tree.

4. Ketika tree masih kosong dan akan ditambahkan sebuah node baru, proses yang terjadi yaitu memeriksa apakah tree kosong. Jika kosong, node baru tersebut akan dijadikan root dari tree. Hal tersebut memastikan bahwa node pertama yang ditambahkan memiliki posisi yang tepat sebagai root dan struktur tree dapat tumbuh dari sana dengan menambahkan node tambahan sesuai dengan aturan binary search tree.

5.

- If(data<current.data) { = bagian tersebut digunakan untuk memeriksa apakah nilai data yang akan ditambahkan lebih kecil dari nilai data di node saat ini (current.data)

- If(current.left != null) { = bagian tersebut digunakan untuk memeriksa subtree bagian kiri ada (tidak null). Jika subtree kiri dari node saat ini (current.left) tidak null, berarti ada node di subtree kiri dan perlu melanjutkan pencarian ke bawah pohon di subtree kiri.
- current = current.left; = bagian tersebut digunakan untuk menggeser pointer current ke subtree kiri dengan mempertimbangkan node ini dengan node saat ini, dan perlu melanjutkan perbandingan data dari node ini dalam iterasi berikutnya dari loop.
- } else {

Current.left = new Node(data);

Bagian tersebut digunakan untuk memeriksa subtree bagian kiri kosong (null). Jika subtree kiri dari node saat ini (current.left) kosong (null), maka posisi yang tepat untuk menambahkan node baru.

- break = bagian tersebut digunakan untuk menghentikan proses iterasi.

## Percobaan 2

- **Code Program BinaryTreeArray21.java**

```
public class BinaryTreeArray21 {
    int[] data;
    int idxLast;

    public BinaryTreeArray21() {
        data = new int[10];
    }
    void populateData(int data[], int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }
    void traversalInOrder(int idxStart) {
        if (idxStart <= idxLast && data[idxStart] != 0) {
            traversalInOrder(2 * idxStart + 1);
            System.out.print(data[idxStart] + " ");
            traversalInOrder(2 * idxStart + 2);
        }
    }
}
```

- **Code Program BinaryTreeArrayMain21.java**

```
public class BinaryTreeArrayMain21 {

    public static void main(String[] args) {

        BinaryTreeArray21 bta = new BinaryTreeArray21();

        int[] data = { 6, 4, 8, 3, 5, 7, 9, 0, 0, 0 };

        int idxLast = 6;

        bta.populateData(data, idxLast);

        System.out.print("\nInOrder Traversal : ");

        bta.traversalInOrder(0);

        System.out.println("\n");

    }

}
```

- **Berikut adalah hasil run dari code diatas**

```
PS C:\Muhammad Afiq Firdaus\Semester 2\Algoritma dan Struktur Data\Praktikum\Pertemuan 1
4>  & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessag
es' '-cp' 'C:\Users\HP\AppData\Roaming\Code\User\workspaceStorage\f863a10f16004b29ed7272
cddda3d837\redhat.java\jdt_ws\Pertemuan 14_fa28d1e\bin' 'BinaryTreeArrayMain21'

InOrder Traversal : 3 4 5 6 7 8 9

PS C:\Muhammad Afiq Firdaus\Semester 2\Algoritma dan Struktur Data\Praktikum\Pertemuan 1
```

Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?

2. Apakah kegunaan dari method populateData()?

3. Apakah kegunaan dari method traverseInOrder()?

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

**Jawaban**

1. Atribut data digunakan untuk menyimpan nilai-nilai dari setiap node dalam pohon biner. Sedangkan atribut idxLast digunakan untuk menunjukkan indeks dari elemen terakhir yang valid dalam array data. Dengan menggunakan kedua atribut, dapat mengimplementasikan pohon biner menggunakan struktur array, yang memiliki keuntungan dalam penggunaan memori dan akses data secara efisien.

2. Method populateData() digunakan untuk menginisialisasi atau mengisi data dalam struktur pohon biner yang direpresentasikan dalam bentuk array. Dengan menggunakan method tersebut, dapat menginisialisasi struktur pohon biner dalam bentuk array dengan nilai-nilai yang sesuai.

3. Method traverseInOrder() digunakan untuk melakukan pencarian inorder pada pohon biner yang direpresentasikan dalam bentuk array. Dengan menggunakan method traverseInOrder(), dapat melakukan traversal inorder yang merepresentasikan dalam bentuk array dan mencetak nilai-nilai simpul dalam urutan yang sesuai dengan aturan traversal inorder.

4. jika suatu node disimpan dalam array pada indeks 2, maka :

- Left child akan berada pada indeks 2*2 + 1 = 5
- Right child akan berada pada indeks 2*2 + 2 = 6

5. Statement int idxLast = 6 digunakan untuk menandai indeks terakhir dari data yang valid dalam array yang digunakan untuk merepresentasikan pohon biner. Indeks terakhir ini menunjukkan batas dari data yang sebenarnya digunakan dalam array. Hal ini berguna untuk berbagai operasi seperti traversal, penambahan data, atau penghapusan data. Statement tersebut menunjukkan bahwa elemen elemen setelah indeks 6 mungkin tidak diisi atau dianggap sebagai elemen kosong.

## Tugas

1. Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.

2. Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

3. Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.

4. Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

- **Code Program Node21.java**

```
public class Node21 {
    int data;
    Node21 left;
    Node21 right;
    public Node21(){
    }
    public Node21(int data){
        this.left = null;
        this.right = null;
        this.data = data;
    }
}
```

```
// TUGAS 2
public class BinaryTree21 {
    Node21 root;
    public BinaryTree21() {
        root = null;
    }
    boolean isEmpty() {
        return root == null;
    }
    void add(int data) {
        root = addRecursive(root, data);
    }
```

- **Code Program BinaryTree21.java**

```
public Node21 addRecursive(Node21 current, int data) {
        if (current == null) {
            return new Node21(data);
        }
        if (data < current.data) {
            current.left = addRecursive(current.left, data);
        } else if (data > current.data) {
            current.right = addRecursive(current.right, data);
        }
        return current;
    }
    boolean find(int data) {
        Node21 current = root;
        while (current != null) {
            if (current.data == data) {
                return true;
            } else if (data < current.data) {
                current = current.left;
            } else {
                current = current.right;
            }
        }
        return false;
    }
    void traversePreOrder(Node21 node) {
        if (node != null) {
            System.out.print(" " + node.data);
            traversePreOrder(node.left);
```

```
}
    }
    Node21 getSuccessor(Node21 del) {
        Node21 successor = del.right;
        Node21 successorParent = del;
        while (successor.left != null) {
            successorParent = successor;
            successor = successor.left;
        }
        if (successor != del.right) {
            successorParent.left = successor.right;
            successor.right = del.right;
        }
        return successor;
    }
    void delete(int data) {
        if (isEmpty()) {
            System.out.println("Tree is empty!");
            return;
        }
        Node21 parent = root;
        Node21 current = root;
        boolean isLeftChild = false;
        while (current != null) {
            if (current.data == data) {
                break;
            } else if (data < current.data) {
                parent = current;
                current = current.left;
                isLeftChild = true;
```

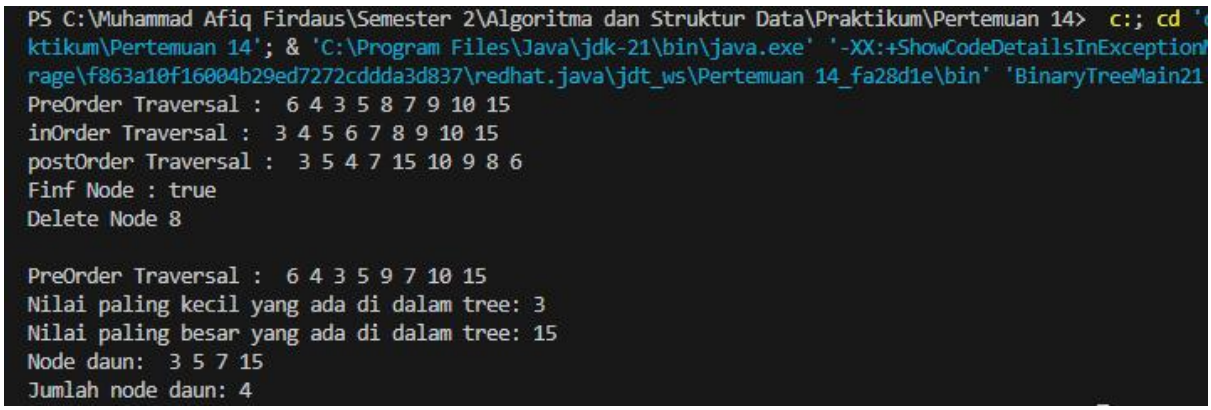
```
} else if (current.left == null) {
                if (current == root) {
                    root = current.right;
                } else {
                    if (isLeftChild) {
                        parent.left = current.right;
                    } else {
                        parent.right = current.right;
                    }
                }
            } else if (current.right == null) {
                if (current == root) {
                    root = current.left;
                } else {
                    if (isLeftChild) {
                        parent.left = current.left;
                    } else {
                        parent.right = current.left;
                    }
                }
            } else {
                Node21 successor = getSuccessor(current);
                if (current == root) {
                    root = successor;
                } else {
                    if (isLeftChild) {
                        parent.left = successor;
                    } else {
                        parent.right = successor;
                    }
                }
                successor.left = current.left;
            }
        }
    }
```

```
 int findMin() {
        if (isEmpty()) {
            throw new IllegalStateException("Tree is empty!");
        }
        Node21 current = root;
        while (current.left != null) {
            current = current.left;
        }
        return current.data;
    }
    int findMax() {
        if (isEmpty()) {
            throw new IllegalStateException("Tree is empty!");
        }
        Node21 current = root;
        while (current.right != null) {
            current = current.right;
        }
        return current.data;
    }
    void printLeafNodes(Node21 node) {
        if (node != null) {
            if (node.left == null && node.right == null) {
                System.out.print(" " + node.data);
            }
            printLeafNodes(node.left);
            printLeafNodes(node.right);
        }
    }
    int countLeafNodes(Node21 node) {
        if (node == null) {
            return 0;
        }
        if (node.left == null && node.right == null) {
            return 1;
        }
        return countLeafNodes(node.left) +
                countLeafNodes(node.right);
    }
}
```

- Code Program BinaryTreeMain21.java

```
// TUGAS
public class BinaryTreeMain21 {
    public static void main(String[] args) {
        BinaryTree21 bt = new BinaryTree21();
        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);
        System.out.print("PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("inOrder Traversal : ");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.print("postOrder Traversal : ");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Finf Node : " + bt.find(5));
        System.out.println("Delete Node 8 ");
        bt.delete(8);
        System.out.println("");
        System.out.print("PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.println("Nilai paling kecil yang ada di dalam tree: " +
            bt.findMin());
        System.out.println("Nilai paling besar yang ada di dalam tree: " +
            bt.findMax());
        System.out.print("Node daun: ");
        bt.printLeafNodes(bt.root);
        System.out.println("");
        System.out.println("Jumlah node daun: " +
                bt.countLeafNodes(bt.root));
    }
}
```

- **Berikut adalah hasil run dari code tersebut**




Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.

**Jawab :**

```
   void add(int data) {
       root = addRecursive(root, data);
   }
public Node21 addRecursive(Node21 current, int data) {
       if (current == null) {
           return new Node21(data);
       }
       if (data < current.data) {
           current.left = addRecursive(current.left, data);
       } else if (data > current.data) {
           current.right = addRecursive(current.right, data);
       }
       return current;
   }
```

Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

**Jawab :**

```
int findMin() {
        if (isEmpty()) {
            throw new IllegalStateException("Tree is empty!");
        }
        Node21 current = root;
        while (current.left != null) {
            current = current.left;
        }
        return current.data;
    }
    int findMax() {
        if (isEmpty()) {
            throw new IllegalStateException("Tree is empty!");
        }
        Node21 current = root;
        while (current.right != null) {
            current = current.right;
        }
        return current.data;
    }
--------------------------------------------------------------------------------
       System.out.println("Nilai paling kecil yang ada di dalam tree: " +
       bt.findMin());
      System.out.println("Nilai paling besar yang ada di dalam tree: " +
       bt.findMax());
```

```
Nilai paling kecil yang ada di dalam tree: 3
Nilai paling besar yang ada di dalam tree: 15
```

Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.

**Jawab :**

```
 void printLeafNodes(Node21 node) {
        if (node != null) {
            if (node.left == null && node.right == null) {
                System.out.print(" " + node.data);
            }
            printLeafNodes(node.left);
            printLeafNodes(node.right);
        }
    }
-------------------------------------------------------------------------

 System.out.print("Node daun: ");
        bt.printLeafNodes(bt.root);
        System.out.println("");
```

```
Node daun:  3 5 7 15
```

Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

***Jawab*** :

```
  int countLeafNodes(Node21 node) {
        if (node == null) {
            return 0;
        }
        if (node.left == null && node.right == null) {
            return 1;
        }
        return countLeafNodes(node.left) +
                countLeafNodes(node.right);
    }
-------------------------------------------------------------------------
System.out.print("Node daun: ");
        bt.printLeafNodes(bt.root);
```

```
Jumlah node daun: 4
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :

a. method add(int data) untuk memasukan data ke dalam tree

b. method traversePreOrder() dan traversePostOrder()

**Jawab :**

- **Code Program BinaryTreeArray21.java_modifikasi**

```
public class BinaryTreeArray21 {
    int[] data;
    int idxLast;

    public BinaryTreeArray21(int maxSize) {
        data = new int[maxSize];
        idxLast = -1;
    }

    void populateData(int[] sourceData, int sourceIdxLast) {
        for (int i = 0; i <= sourceIdxLast; i++) {
            add(sourceData[i]);
        }
    }

    void add(int newData) {
    if (idxLast + 1 >= data.length) {
    System.out.println("Binary tree array is full, cannot add new element.");
    return;
    }
    idxLast++;
    data[idxLast] = newData;
    }

    void traverseInOrder() {
        traverseInOrder(0);
    }
```

```
    public void traverseInOrder(int idxStart) {
        if (idxStart < data.length && data[idxStart] != 0) {
            traverseInOrder(2 * idxStart + 1);
            System.out.print(data[idxStart] + " ");
            traverseInOrder(2 * idxStart + 2);
        }
    }

    void traversePreOrder() {
        traversePreOrder(0);
    }

    public void traversePreOrder(int idxStart) {
        if (idxStart < data.length && data[idxStart] != 0) {
            System.out.print(data[idxStart] + " ");
            traversePreOrder(2 * idxStart + 1);
            traversePreOrder(2 * idxStart + 2);
        }
    }

    void traversePostOrder() {
        traversePostOrder(0);
    }

    public void traversePostOrder(int idxStart) {
        if (idxStart < data.length && data[idxStart] != 0) {
            traversePostOrder(2 * idxStart + 1);
            traversePostOrder(2 * idxStart + 2);
            System.out.print(data[idxStart] + " ");
        }
    }
}
```

- ***Code Program BinaryTreeArrayMain21.java_modifikasi***

```
// TUGAS MODIFIKASI
import java.util.Scanner;

public class BinaryTreeArrayMain21 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Masukkan size Binary Tree Array : ");
        int size = sc.nextInt();
        BinaryTreeArray21 bta = new BinaryTreeArray21(size);
        System.out.print("Masukkan data (pisahkan dengan spasi) : ");
        sc.nextLine();
        for (int i = 0; i < size; i++) {
            int num = sc.nextInt();
            bta.add(num);
        }
        System.out.print("InOrder Traversal : ");
        bta.traverseInOrder();
        System.out.println();
        System.out.print("PreOrder Traversal : ");
        bta.traversePreOrder();
        System.out.println();
        System.out.print("PostOrder Traversal : ");
        bta.traversePostOrder();
        System.out.println();
        sc.close();
    }
}
```

- **Berikut adalah hasil run dari code tersebut**

```
Masukkan size Binary Tree Array : 5
Masukkan data (pisahkan dengan spasi) : 2 5 8 6 4
InOrder Traversal : 6 5 4 2 8
PreOrder Traversal : 2 5 6 4 8
PostOrder Traversal : 6 4 5 8 2
PS C:\Muhammad Afiq Firdaus\Semester 2\Algoritma dan Struktur Data\Praktikum\Pertemuan 14>
```