NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

Percobaan 1

```
Preorder Traversal :  6 4 3 5 8 7 9 10 15
Inorder Traversal :  3 4 5 6 7 8 9 10 15
PostOrder Traversal :  3 5 4 7 15 10 9 8 6
Find Node : true
Delete Node 8

PreOrder Traversal :  6 4 3 5 9 7 10 15
```

```java
package P13;

public class BinaryTreeMain27 {
    public static void main(String[] args) {
        BinaryTree27 bt = new BinaryTree27();
        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("Find 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("");
    }

}
```

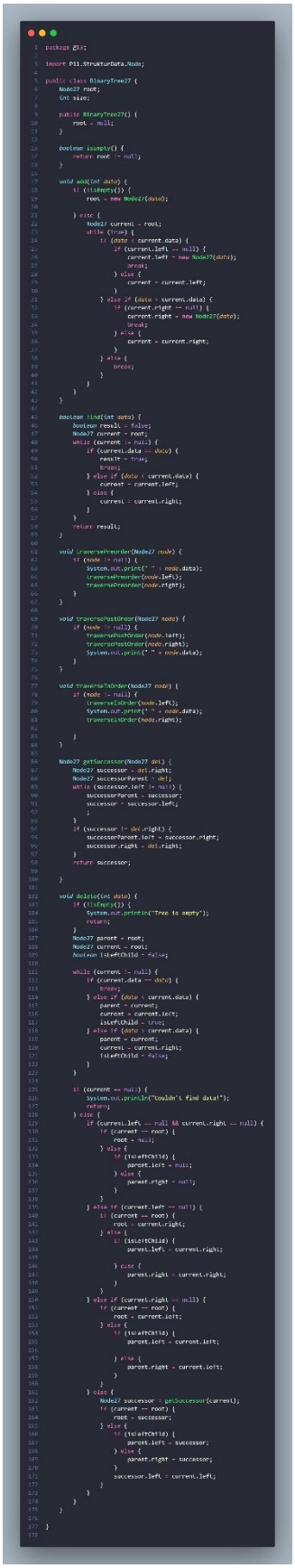
```java
package P13;

public class Node27 {
    int data;
    Node27 left;
    Node27 right;

    Node27(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

```

NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)



```java
package P11;

import P11.StrukturData.Node;

public class BinaryTree27 {
    Node27 root;
    int size;

    public BinaryTree27() {
        root = null;
    }

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

    void add(int data) {
        if (!isEmpty()) {
            root = new Node27(data);

        } else {
            Node27 current = root;
            while (true) {
                if (data < current.data) {
                    if (current.left == null) {
                        current.left = new Node27(data);
                        break;
                    } else {
                        current = current.left;
                    }
                } else if (data > current.data) {
                    if (current.right == null) {
                        current.right = new Node27(data);
                        break;
                    } else {
                        current = current.right;
                    }
                } else {
                    break;
                }
            }
        }
    }

    boolean find(int data) {
        boolean result = false;
        Node27 current = root;
        while (current != null) {
            if (current.data == data) {
                result = true;
                break;
            } else if (data < current.data) {
                current = current.left;
            } else {
                current = current.right;
            }
        }
        return result;
    }

    void traversePreorder(Node27 node) {
        if (node != null) {
            System.out.print(" " + node.data);
            traversePreorder(node.left);
            traversePreorder(node.right);
        }
    }

    void traversePostOrder(Node27 node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" " + node.data);
        }
    }

    void traverseInOrder(Node27 node) {
        if (node != null) {
            traverseInOrder(node.left);
            System.out.print(" " + node.data);
            traverseInOrder(node.right);

        }
    }

    Node27 getSuccessor(Node27 del) {
        Node27 successor = del.right;
        Node27 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;
        }
        Node27 parent = root;
        Node27 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 (data > current.data) {
                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 {
                Node27 successor = getSuccessor(current);
                if (current == root) {
                    root = successor;
                } else {
                    if (isLeftChild) {
                        parent.left = successor;
                    } else {
                        parent.right = successor;
                    }
                    successor.left = current.left;
                }
            }
        }
    }
}
```

NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

PERTANYAAN

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

   Jawab:

   Metode traverse pada binary tree preorder, postorder, dan inorder mengecek setiap node dalam urutan tertentu, tetapi tidak memberikan keuntungan efisiensi dalam pencarian karena method tersebut mungkin harus mengecek semua node.

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

   Jawab:

   Left dan right dalam class node digunakan untuk mendefinisikan child dari node yang sekarang, jika left maka untuk left child jika right untuk right child

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

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

   Jawab:

   a. atribut root digunakan untuk mendefinisikan sebuah induk/parent dalam suatu tree
   b. ketika object tree pertama kali dibuat, nilai root adalah null

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

   Jawab:

   Jika tree masih kosong node baru akan ditambahkan dan node baru tersebut akan menjadi root atau parrent.

NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

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;
        }
 }
```

Jawab:

Kode pada soal adalah salah, kode yang benar adalah seperti kode dibawah.

```
if (data < current.data) {
                    if (current.left == null) {
                        current.left = new Node27(data);
                        break;
                    } else {
                        current = current.left;
                    }
                }
```

Pada baris pertama terdapat pengecekan apakah data yang akan diinputkan lebih kecil dari data yang dibandingkan saat ini, jika benar akan dicek kembali apakah left child dari current tersebut bernilai null, jika benar maka data baru akan menempati posisi tersebut, jika nilai left dari current tidak null maka nilai current digantikan dengan nilai left dari current.

NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

Percobaan 2

```
InOrder Traversal : 3 4 5 6 7 8 9
```

```java
package P14;

public class BinaryTreeArrayMain27 {
    public static void main(String[] args) {
        BinaryTreeArray27 bta = new BinaryTreeArray27();
        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.traverseInOrder(0);
        System.out.println("\n");

    }
}

```

```java
package P14;

public class BinaryTreeArray27 {
    int[] data;
    int idxLast;

    public BinaryTreeArray27() {
        data = new int[10];

    }

    void populateData(int data[], int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }

    void traverseInOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traverseInOrder(2 * idxStart + 1);
            System.out.print(data[idxStart] + " ");
            traverseInOrder(2 * idxStart + 2);
        }
    }
}

```

NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

Pertanyaan

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

   Jawab:

   Atribut data digunakan untuk menyimpan data dalam array. Atribut idxLast digunakan untuk menentukan batas indeks array tersebut

2. Apakah kegunaan dari method populateData()?

   Jawab:

   Method tersebut digunakan untuk mengisi data pada objek yang telah dibuat yaitu bta dan batas indeks tersebut juga ditentukan dengan parameter method tersebut.

3. Apakah kegunaan dari method traverseInOrder()?

   Jawab:

   traverseInOrder adalah method penelusuran seluruh node pada binary tree dengan pengecekan dimulai pada indeks yang dikirimkan dengan argumen pada parameter method tersebut

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

   Jawab:

   Dengan i adalah 2 maka:

   Anak kiri diperoleh dengan 2*i+1 = 5 => pada indeks ke-5

   Anak kanan diperoleh dengan 2*i+2 = 6 => pada indeks ke-6

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

   Jawab:

   Statement tersebut digunakan untuk menunjukkan indeks valid terakhir yang terisi dalam array data.

NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

LINK GITHUB:
https://github.com/RoyW12/AlgoritmaStrukturData_1G_28/tree/main/src/P14