LAPORAN JOBSHEET 15

KONSEP DASAR PEMROGRAMAN

Mata Kuliah : Algoritma dan Struktur Data

Dosen : **Mungki Astiningrum, S.T., M.Kom.**

Alfreda Dhaifullah Mahezwara
244107020219
Kelas : 1A
Absen : 04

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG TAHUN 2025

laksd

Class Mahasiswa

```
package Pertemuan_15;

public class Mahasiswa04 {
    String nim;
    String nama;
    String kelas;
    double ipk;

    Mahasiswa04 () {

    }

    Mahasiswa04 (String nm, String nma, String kls, double ip) {
        nim = nm;
        nama = nma;
        kelas = kls;
        ipk = ip;
    }

    public void tampilInformasi () {
        System.out.println("NIM : " + nim +" "+
        "Nama : " +nama+" "+
        "Kelas : " + kelas+ " " +
        "IPK : " + ipk);
    }
}
```

Class Node

```
package Pertemuan_15;
```

```
public class Node04 {
    Mahasiswa04 mahasiswa;
    Node04 left, right;

    public Node04 () {

    }

    public Node04 (Mahasiswa04 mahasiswa) {
        this.mahasiswa = mahasiswa;
        left = right = null;
    }
}
```

Class BinaryTree

```
package Pertemuan_15;

public class BinaryTree04 {
    Node04 root;

    public BinaryTree04() {
        root = null;
    }

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

    // Method untuk menambahkan node baru ke dalam binary search
tree
    public void add(Mahasiswa04 mahasiswa) {
        Node04 newNode = new Node04(mahasiswa);
        if (isEmpty()) {
            root = newNode;
```

```
    } else {
        Node04 current = root;
        Node04 parent = null;
        while (true) {
            parent = current;
            if (mahasiswa.ipk < current.mahasiswa.ipk) {
                current = current.left;
                if (current == null) {
                    parent.left = newNode;
                    return;
                }
            } else {
                current = current.right;
                if (current == null) {
                    parent.right = newNode;
                    return;
                }
            }
        }
    }
}

// method find ynag digunakan untuk mencari nilai
boolean find(double ipk) {
    boolean result = false;
    Node04 current = root;
    while (current != null) {
        if (current.mahasiswa.ipk == ipk) {
            result = true;
            break;
        } else if (ipk > current.mahasiswa.ipk) {
            current = current.right;
```

```
            } else {
                current = current.left;
            }
        }
        return result;
    }

    void traversePreOrder(Node04 node) {
        if (node != null) {
            node.mahasiswa.tampilInformasi();
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    void traverseInOrder(Node04 node) {
        if (node != null) {
            traverseInOrder(node.left);
            node.mahasiswa.tampilInformasi();
            traverseInOrder(node.right);
        }
    }

    void traversePostOrder(Node04 node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            node.mahasiswa.tampilInformasi();
        }
    }

    // method yang digunakan ketika proses penghapusan node yang
memiliki 2 child
```

```
Node04 getSuccessor(Node04 del) {
    Node04 successor = del.right;
    Node04 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(double ipk) {
    if (isEmpty()) {
        System.out.println("Binary tree kosong");
        return;
    }
    // Cari node (current) yang akan dihapus
    Node04 parent = root;
    Node04 current = root;
    boolean isLeftChild = false;

    while (current != null) {
        if (current.mahasiswa.ipk == ipk) {
            break;
        } else if (ipk < current.mahasiswa.ipk) {
            parent = current;
            current = current.left;
            isLeftChild = true;
        } else if (ipk > current.mahasiswa.ipk) {
```

```
                parent = current;
                current = current.right;
                isLeftChild = false;
            }
        }

        // Penghapusan
        if (current == null) {
            System.out.println("Data tidak ditemukan");
            return;
        } else {
            // Jika tidak ada anak (leaf), maka node dihapus
            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) { // Jika hanya punya 1
anak (kanan)
                if (current == root) {
                    root = current.right;
                } else {
                    if (isLeftChild) {
                        parent.left = current.right;
                    } else {
                        parent.right = current.right;
                    }
                }
```

```
            } else if (current.right == null) { // Jika hanya punya
1 anak (kiri)
                if (current == root) {
                    root = current.left;
                } else {
                    if (isLeftChild) {
                        parent.left = current.left;
                    } else {
                        parent.right = current.left;
                    }
                }
            } else { // Jika punya 2 anak
                Node04 successor = getSuccessor(current);
                System.out.println("Jika 2 anak, current = ");
                successor.mahasiswa.tampilInformasi();

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

Class BinaryTree Main

```
package Pertemuan_15;
```

```
public class BinaryTreeMain04 {
    public static void main (String[] args) {
        BinaryTree04 bet = new BinaryTree04();

        bet.add(new Mahasiswa04("24416022","All","A", 3.57));
        bet.add(new Mahasiswa04("24416092","Reader","B", 3.85));
        bet.add(new Mahasiswa04("24416018","Cancer","C", 3.21));
        bet.add(new Mahasiswa04("24416020","Dowd","B", 3.54));

        System.out.println("\nDaftar semua mahasiswa (in order
traversal):");
        bet.traverseInOrder(bet.root);

        System.out.println("\nPencarian data mahasiswa:");
        System.out.print("Cari mahasiswa dengan ipk 3.54 : ");
        String hasilCari = bet.find(3.54) ? "Ditemukan" : "Tidak
ditemukan";
        System.out.println(hasilCari);

        System.out.print("Cari mahasiswa dengan ipk 3.22 : ");
        hasilCari = bet.find(3.22) ? "Ditemukan" : "Tidak
ditemukan";
        System.out.println(hasilCari);

        bet.add(new Mahasiswa04("24416013","Dowd","A",  3.72));
        bet.add(new Mahasiswa04("24416029","Reader","D",  3.27));
        bet.add(new Mahasiswa04("24416017","Fail","B",  3.46));

        System.out.println("\nDaftar semua mahasiswa setelah
penambahan 3 mahasiswa:");
        System.out.println("InOrder Traversal:");
        bet.traverseInOrder(bet.root);
        System.out.println("\nPreOrder Traversal:");
```

```
        bet.traversePreOrder(bet.root);
        System.out.println("\nPostOrder Traversal:");
        bet.traversePostOrder(bet.root);

        System.out.println("\nPenghapusan data mahasiswa:");
        bet.delete(3.57);
        System.out.println("\nDaftar semua mahasiswa setelah
penghapusan 1 mahasiswa (in order traversal):");
        bet.traverseInOrder(bet.root);
    }
}
```

```
Daftar semua mahasiswa (in order traversal):
NIM : 24416018 Nama : Cancer Kelas : C IPK : 3.21
NIM : 24416020 Nama : Dowd Kelas : B IPK : 3.54
NIM : 24416022 Nama : All Kelas : A IPK : 3.57
NIM : 24416092 Nama : Reader Kelas : B IPK : 3.85

Pencarian data mahasiswa:
Cari mahasiswa dengan ipk 3.54 : Ditemukan
Cari mahasiswa dengan ipk 3.22 : Tidak ditemukan

Daftar semua mahasiswa setelah penambahan 3 mahasiswa:
InOrder Traversal:
NIM : 24416018 Nama : Cancer Kelas : C IPK : 3.21
NIM : 24416029 Nama : Reader Kelas : D IPK : 3.27
NIM : 24416017 Nama : Fail Kelas : B IPK : 3.46
NIM : 24416020 Nama : Dowd Kelas : B IPK : 3.54
NIM : 24416022 Nama : All Kelas : A IPK : 3.57
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72
NIM : 24416092 Nama : Reader Kelas : B IPK : 3.85

PreOrder Traversal:
NIM : 24416022 Nama : All Kelas : A IPK : 3.57
NIM : 24416018 Nama : Cancer Kelas : C IPK : 3.21
NIM : 24416020 Nama : Dowd Kelas : B IPK : 3.54
NIM : 24416029 Nama : Reader Kelas : D IPK : 3.27
NIM : 24416017 Nama : Fail Kelas : B IPK : 3.46
NIM : 24416092 Nama : Reader Kelas : B IPK : 3.85
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72

PostOrder Traversal:
NIM : 24416017 Nama : Fail Kelas : B IPK : 3.46
NIM : 24416029 Nama : Reader Kelas : D IPK : 3.27
NIM : 24416020 Nama : Dowd Kelas : B IPK : 3.54
NIM : 24416018 Nama : Cancer Kelas : C IPK : 3.21
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72
NIM : 24416092 Nama : Reader Kelas : B IPK : 3.85
```

```
Penghapusan data mahasiswa:
Jika 2 anak, current =
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72

Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM : 24416018 Nama : Cancer Kelas : C IPK : 3.21
NIM : 24416029 Nama : Reader Kelas : D IPK : 3.27
NIM : 24416017 Nama : Fail Kelas : B IPK : 3.46
NIM : 24416020 Nama : Dowd Kelas : B IPK : 3.54
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72
NIM : 24416092 Nama : Reader Kelas : B IPK : 3.85
PS D:\Kuliahh\kuliahhh\Semester2\PrakAlgoritmaStrukturDT>

Jika 2 anak, current =
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72

Jika 2 anak, current =
Jika 2 anak, current =
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72
Jika 2 anak, current =
Jika 2 anak, current =
Jika 2 anak, current =
Jika 2 anak, current =
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72

Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
Jika 2 anak, current =
NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72

NIM : 24416013 Nama : Dowd Kelas : A IPK : 3.72


Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM : 24416018 Nama : Cancer Kelas : C IPK : 3.21
NIM : 24416029 Nama : Reader Kelas : D IPK : 3.27
NIM : 24416017 Nama : Fail Kelas : B IPK : 3.46
NIM : 24416020 Nama : Dowd Kelas : B IPK : 3.54
```

1. Mengapa dalam binary tree proses pencarian data bisa lebih efektif dilakukan disbanding binary tree biasa?
   - Dalam binary search tree (BST), proses pencarian data lebih efektif dibandingkan binary tree biasa karena setiap node BST memiliki aturan:
     Semua data di subtree kiri bernilai lebih kecil dari node induk.
     Semua data di subtree kanan bernilai lebih besar dari node induk.
     Dengan aturan ini, saat mencari data, kita bisa langsung membandingkan nilai yang dicari dengan node saat ini dan memutuskan untuk bergerak ke kiri atau ke kanan, sehingga mengurangi jumlah node yang harus diperiksa.
     Pada binary tree biasa, data tidak terurut, sehingga pencarian harus dilakukan ke setiap node (traversal), yang jauh lebih lambat.
     Kesimpulan:

BST memungkinkan pencarian lebih cepat (rata-rata O(log n)), sedangkan binary tree biasa membutuhkan waktu O(n) karena tidak ada aturan penempatan data.

2. Untuk apakah di class Node, kegunaan dari atribut left dan right?
   - Atribut left dan right pada class Node04 digunakan untuk menunjuk ke anak kiri dan anak kanan dari node tersebut dalam struktur pohon biner (binary tree).
     left menunjuk ke node anak kiri.
     right menunjuk ke node anak kanan.
     Dengan dua atribut ini, setiap node dapat terhubung ke dua node lain, sehingga membentuk struktur pohon biner yang memungkinkan traversal, pencarian, penambahan, dan penghapusan data secara efisien.
3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?
   - Atribut root di dalam class BinaryTree04 digunakan untuk menyimpan referensi ke node paling atas (node akar) dari pohon biner.
     Semua operasi pada binary tree (seperti penambahan, pencarian, penghapusan, dan traversal) selalu dimulai dari node root ini.
     Tanpa atribut root, pohon biner tidak akan memiliki titik awal untuk mengakses seluruh struktur pohon.

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?
   - Null
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   - Ketika tree masih kosong dan akan ditambahkan sebuah node baru, proses yang terjadi adalah:
     Method add() akan membuat objek node baru.
     Karena tree kosong (root == null), node baru tersebut langsung dijadikan root (akar) dari tree.
     Tidak ada proses pencarian atau perbandingan, node pertama selalu menjadi root.
     Jadi, prosesnya hanya membuat node baru dan mengisi atribut root dengan node tersebut.

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?
6. Jelaskan langkah-langkah pada method delete() saat menghapus sebuah node yang memiliki dua anak. Bagaimana method getSuccessor() membantu dalam proses ini?
   - Berikut langkah-langkah pada method delete() saat menghapus node yang memiliki dua anak pada binary search tree:
     Langkah-langkah:
     Cari Node yang Akan Dihapus

Program mencari node (current) yang IPK-nya sama dengan nilai yang ingin dihapus, serta menyimpan parent-nya (parent).
Deteksi Node dengan Dua Anak
Kondisi else terakhir pada method delete():

```
else { // Jika punya 2 anak
Node04 successor = getSuccessor(current);
...
    successor.left = current.left;
}
```

Menandakan node yang akan dihapus (current) memiliki dua anak (current.left != null && current.right != null).

Praktikum 2

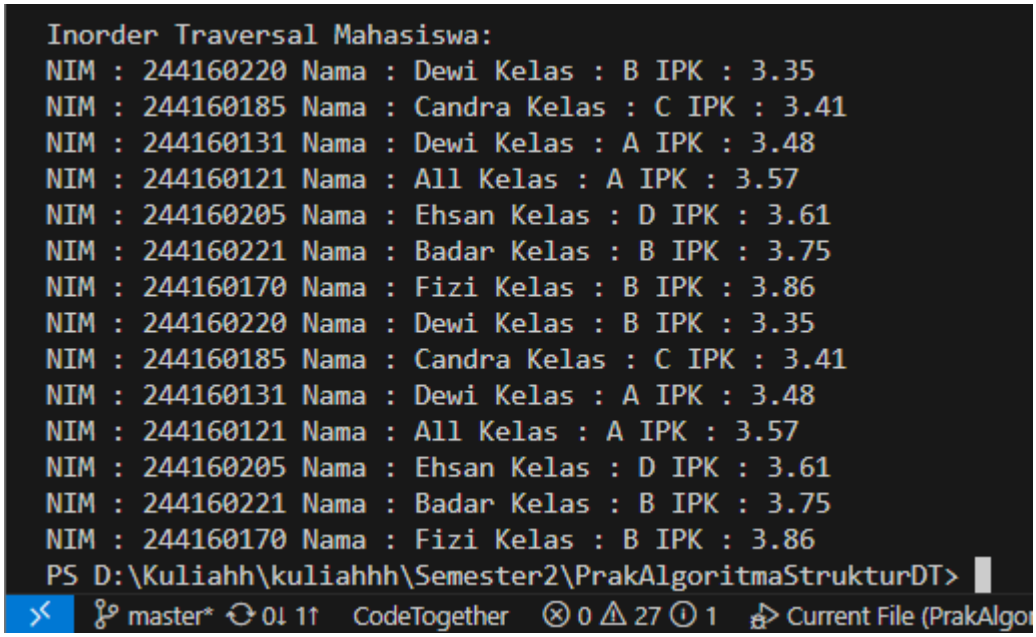
Class binarytree array

```
package Pertemuan_15;


public class BinarytreeArray04 {
    Mahasiswa04 [] dataMahasiswa;
    int idxLast;

    public BinarytreeArray04 () {
        this.dataMahasiswa = new Mahasiswa04[10];
    }

    void PopulateData (Mahasiswa04 dataMhs[], int idxLast) {
        this.dataMahasiswa = dataMhs;
        this.idxLast = idxLast;
    }

    void traverseInOrder (int idxStart) {
        if (idxStart <= idxLast) {
```

```
            if (dataMahasiswa[idxStart] != null) {
                traverseInOrder(2*idxStart+1);
                dataMahasiswa[idxStart].tampilInformasi();
                traverseInOrder(2*idxStart+2);
            }
        }
    }
}
```

Class binarytree array

```
package Pertemuan_15;

public class BinaryTreeArrayMain {
    public static void main(String[] args) {
    BinarytreeArray04 bta = new BinarytreeArray04();

    Mahasiswa04 mhs1 = new Mahasiswa04("244160121", "All", "A", 3.57);
    Mahasiswa04 mhs2 = new Mahasiswa04("244160185", "Candra", "C", 3.41);
    Mahasiswa04 mhs3 = new Mahasiswa04("244160221", "Badar", "B", 3.75);
    Mahasiswa04 mhs4 = new Mahasiswa04("244160220", "Dewi", "B", 3.35);

    Mahasiswa04 mhs5 = new Mahasiswa04("244160131", "Dewi", "A", 3.48);
    Mahasiswa04 mhs6 = new Mahasiswa04("244160205", "Ehsan", "D", 3.61);
    Mahasiswa04 mhs7 = new Mahasiswa04("244160170", "Fizi", "B", 3.86);

    Mahasiswa04[] dataMahasiswas = {mhs1, mhs2, mhs3, mhs4, mhs5, mhs6, mhs7, null,
null, null };
    int idxLast = 6;
    bta.PopulateData(dataMahasiswas,idxLast);
    System.out.println("\nInorder Traversal Mahasiswa: ");bta.traverseInOrder(0);
    bta.traverseInOrder(0);

    }
}
```



Pertanyaan percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
   - Kegunaan atribut pada class BinarytreeArray04:
     dataMahasiswa:
     Merupakan array yang digunakan untuk menyimpan data node-node pohon biner (dalam hal ini objek Mahasiswa04) dengan representasi berbasis array. Setiap indeks array mewakili posisi node pada pohon.
     idxLast:
     Menyimpan indeks terakhir (paling besar) yang berisi data pada array dataMahasiswa. Atribut ini digunakan untuk mengetahui batas akhir data yang valid di dalam array, sehingga traversal atau operasi lain tidak melewati data yang belum diisi.

2. Apakah kegunaan dari method populateData()?
   - Method populateData() pada class BinarytreeArray04 digunakan untuk mengisi (menginisialisasi) array dataMahasiswa dengan data mahasiswa yang diberikan dan menetapkan nilai idxLast sesuai dengan jumlah data yang diisi.
   - Dengan kata lain, method ini menyiapkan struktur pohon biner berbasis array dengan data yang sudah ada, sehingga pohon bisa langsung digunakan untuk traversal atau operasi lainnya.
3. Apakah kegunaan dari method traverseInOrder()?
   - Method traverseInOrder() pada class BinarytreeArray04 digunakan untuk melakukan penelusuran (traversal) pohon biner secara in-order pada struktur pohon yang direpresentasikan dengan array.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masing-masing?
   - Jika suatu node binary tree disimpan pada indeks 2 dalam array, maka:
   - Left child (anak kiri) berada di indeks:
     2 * 2 + 1 = 5
   - Right child (anak kanan) berada di indeks:
     2 * 2 + 2 = 6
   - Jadi, left child di indeks 5 dan right child di indeks 6.

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

6. Mengapa indeks 2*idxStart+1 dan 2*idxStart+2 digunakan dalam pemanggilan rekursif, dan apa kaitannya dengan struktur pohon biner yang disusun dalam array?

   - ndeks 2*idxStart+1 dan 2*idxStart+2 digunakan dalam pemanggilan rekursif karena itulah rumus untuk menentukan posisi anak kiri dan anak kanan pada representasi pohon biner menggunakan array.
     Penjelasan:
     Jika sebuah node berada pada indeks idxStart:
     Anak kiri (left child) berada di indeks 2*idxStart + 1
     Anak kanan (right child) berada di indeks 2*idxStart + 2
     Kaitannya dengan struktur pohon biner dalam array:
     Dengan rumus ini, setiap node dan anak-anaknya dapat diakses langsung melalui indeks array tanpa perlu pointer.
     Struktur ini memudahkan traversal dan operasi pohon secara efisien, karena posisi setiap anak dapat dihitung langsung dari posisi induknya.
     Kesimpulan:
     Rumus tersebut adalah standar representasi pohon biner dalam array, sehingga traversal rekursif dapat berjalan dengan benar sesuai struktur pohon.

Tugas Praktikum

1. Buat method di dalam class BinaryTree00 yang akan menambahkan node dengan cara rekursif (addRekursif()).

```
public void addRekursif(Mahasiswa04 mahasiswa) {
        root = addRekursif(root, mahasiswa);
    }

    private Node04 addRekursif(Node04 current, Mahasiswa04 mahasiswa) {
        if (current == null) {
            return new Node04(mahasiswa);
        }
        if (mahasiswa.ipk < current.mahasiswa.ipk) {
```

```
            current.left = addRekursif(current.left,
    mahasiswa);
            } else {
                current.right = addRekursif(current.right,
    mahasiswa);
            }
            return current;
        }
```

2. Buat method di dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (cariMinIPK() dan cariMaxIPK()) yang ada di dalam binary search tree.

```
public void cariMinIPK() {
        if (isEmpty()) {
            System.out.println("Binary tree kosong");
            return;
        }
        Node04 current = root;
        while (current.left != null) {
            current = current.left;
        }
        System.out.println("Data Mahasiswa dengan IPK terkecil:");
        current.mahasiswa.tampilInformasi();
    }

    // Method untuk menampilkan data mahasiswa dengan IPK paling besar
    public void cariMaxIPK() {
        if (isEmpty()) {
            System.out.println("Binary tree kosong");
            return;
        }
        Node04 current = root;
        while (current.right != null) {
            current = current.right;
        }
        System.out.println("Data Mahasiswa dengan IPK terbesar:");
        current.mahasiswa.tampilInformasi();
    }
```

3. Buat method dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK di atas suatu batas tertentu, misal di atas 3.50 (tampilMahasiswaIPKdiAtas(double ipkBatas)) yang ada di dalam binary search tree.

```
// Method untuk menampilkan data mahasiswa dengan IPK paling kecil
    public void cariMinIPK() {
        if (isEmpty()) {
            System.out.println("Binary tree kosong");
            return;
        }
        Node04 current = root;
        while (current.left != null) {
            current = current.left;
        }
        System.out.println("Data Mahasiswa dengan IPK terkecil:");
        current.mahasiswa.tampilInformasi();
    }

    // Method untuk menampilkan data mahasiswa dengan IPK paling besar
    public void cariMaxIPK() {
        if (isEmpty()) {
            System.out.println("Binary tree kosong");
            return;
        }
        Node04 current = root;
        while (current.right != null) {
            current = current.right;
        }
        System.out.println("Data Mahasiswa dengan IPK terbesar:");
        current.mahasiswa.tampilInformasi();
    }
```

4. Modifikasi class BinaryTreeArray00 di atas, dan tambahkan :
   • method add(Mahasiswa data) untuk memasukan data ke dalam binary tree
   • method traversePreOrder()

sa

https://github.com/AlfredaDhaifullah04/Semester-2/tree/master/Pertemuan_15