Nama : Siti Mutmainah

NIM : 244107020143

Percobaan 1

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan linked list.

1. Pada Project yang sudah dibuat pada pertemuan sebelumnya, buat package dengan nama Pertemuan14.
2. Tambahkan class-class berikut:
   a. Mahasiswa00.java
   b. Node00.java
   c. BinaryTree00.java
   d. BinaryTreeMain00.java

   Ganti 00 dengan nomer absen Anda.
3. Implementasikan Class Mahasiswa00, Node00, BinaryTree00 sesuai dengan diagram class berikut ini:

BinaryTree
root: Node
BinaryTree() add(data: Mahasiswa): void

Mahasiswa
nim: String nama: String kelas: String ipk: double
Mahasiswa() Mahasiswa(nm: String, name: String, kls: String, ipk: double) tampilInformasi(): void

Node
data: Mahasiswa left: Node right: Node
Node (left: Node, data:Mahasiswa, right:Node)

find(ipk: double) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) delete(ipk: double): void

1. Di dalam class **Mahasiswa00**, deklarasikan atribut sesuai dengan diagram class Mahasiswa di atas. Tambahkan juga konstruktor dan method sesuai diagram di atas.

```java
public class Mahasiswa21 {
    String nim;
    String nama;
    String kelas;
    double ipk;

    public Mahasiswa21() {
    }

    public Mahasiswa21(String nim, String nama, String kelas, double ipk) {
        this.nim = nim;
        this.nama = nama;
        this.kelas = kelas;
        this.ipk = ipk;
    }

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

2. Di dalam class **Node**00, tambahkan atribut **data**, **left** dan **right**, serta konstruktor default dan berparameter.

```java
public class Node21 {
    Mahasiswa21 mahasiswa;
    Node21 left, right;

    public Node21() {
    }

    public Node21(Mahasiswa21 mahasiswa) {
        this.mahasiswa = mahasiswa;
        left = right = null;
    }
}
```

3. Di dalam class **BinaryTree**00, tambahkan atribut **root**.

```java
public class BinaryTree21 {
    Node21 root;
```

4. Tambahkan konstruktor dan method **isEmpty()** di dalam class **BinaryTree**00**.**

```
public BinaryTree21() {
    root = null;
}

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

}
```

5. Tambahkan method **add()** di dalam class **BinaryTree**00. Node ditambahkan di binary search tree pada posisi sesuai dengan besar nilai IPK mahasiswa. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya dengan proses rekursif penulisan kode akan lebih efisien.

```
public void add(Mahasiswa21 mahasiswa) {
    Node21 newNode = new Node21(mahasiswa);
    if (isEmpty()) {
        root = newNode;
    } else {
        Node21 current = root;
        Node21 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;
                }
            }
        }
    }
}
```

6. Tambahkan method **find()**

```
boolean find(double ipk) {
    boolean result = false;
    Node21 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;
}
```

7. Tambahkan method **traversePreOrder()**, **traverseInOrder()** dan **traversePostOrder()**. Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam binary tree, baik dalam mode pre-order, in-order maupun post-order.

```
void traversePreOrder(Node21 node) {
    if (node != null) {
        node.mahasiswa.tampilInformasi();
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traverseInOrder(Node21 node) {
    if (node != null) {
        traverseInOrder(node.left);
        node.mahasiswa.tampilInformasi();
        traverseInOrder(node.right);
    }
}

void traversePostOrder(Node21 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        node.mahasiswa.tampilInformasi();
    }
}
```

8. Tambahkan method **getSuccessor()**. Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
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;
}
```

9. Tambahkan method **delete()**. Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak, cari posisi node yang akan dihapus.

```
void delete(double ipk) {
    if (isEmpty()) {
        System.out.println(x:"Binary tree kosong");
        return;
    }

    Node21 parent = root;
    Node21 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;
        }
    }
```

10. Kemudian tambahkan proses penghapusan di dalam method **delete()** terhadap node current yang telah ditemukan.

```
// penghapusan
if (current == null) {
    System.out.println(x:"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;
            }
        }

        // jika hanya punya 1 anak (kanan)
    } else if (current.left == null) {
        if (current == root) {
            root = current.right;
        } else {
            if (isLeftChild) {
                parent.left = current.right;
            } else {
                parent.right = current.right;
            }
        }
```

```
        // jika hanya punya 1 anak (kiri)
    } else if (current.right == null) {
        if (current == root) {
            root = current.left;
        } else {
            if (isLeftChild) {
                parent.left = current.left;
            } else {
                parent.right = current.left;
            }
        }

        // jika punya 2 anak
    } else {
        Node21 successor = getSuccessor(current);
        System.out.println(x:"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;
    }
}
```

11. Buka class **BinaryTreeMain**00 dan tambahkan method **main()** kemudian tambahkan kode berikut ini:

```
BinaryTreeMain21.java > BinaryTreeMain21 > main(String[])
1   public class BinaryTreeMain21 {
        Run | Debug
2       public static void main(String[] args) {
3           BinaryTree21 bst = new BinaryTree21();
4
5           bst.add(new Mahasiswa21(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57));
6           bst.add(new Mahasiswa21(nim:"244160221", nama:"Badar", kelas:"B", ipk:3.85));
7           bst.add(new Mahasiswa21(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.21));
8           bst.add(new Mahasiswa21(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.54));
9
10          System.out.println(x:"\nDaftar semua mahasiswa (in order traversal):");
11          bst.traverseInOrder(bst.root);
12
13          System.out.println(x:"\nPencarian data mahasiswa:");
14          System.out.print(s:"Cari mahasiswa dengan ipk: 3.54 : ");
15          String hasilCari = bst.find(ipk:3.54) ? "Ditemukan" : "Tidak ditemukan";
16          System.out.println(hasilCari);
17
18          System.out.print(s:"Cari mahasiswa dengan ipk: 3.22 : ");
19          hasilCari = bst.find(ipk:3.22) ? "Ditemukan" : "Tidak ditemukan";
20          System.out.println(hasilCari);
21
22          bst.add(new Mahasiswa21(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.72));
23          bst.add(new Mahasiswa21(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.37));
24          bst.add(new Mahasiswa21(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.46));

    System.out.println(x:"\nDaftar semua mahasiswa setelah penambahan 3 mahasiswa:");
    System.out.println(x:"InOrder Traversal:");
    bst.traverseInOrder(bst.root);
    System.out.println(x:"\nPreOrder Traversal:");
    bst.traversePreOrder(bst.root);
    System.out.println(x:"\nPostOrder Traversal:");
    bst.traversePostOrder(bst.root);

    System.out.println(x:"\nPenghapusan data mahasiswa");
    bst.delete(ipk:3.57);

    System.out.println(x:"\nDaftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):");
    bst.traverseInOrder(bst.root);
}
```

12. Compile dan jalankan class **BinaryTreeMain**00 untuk mendapatkan simulasi jalannya program binary tree yang telah dibuat.
13. Amati hasil running tersebut.

```
PS D:\Semester 2\Praktikum ASD\Jobsheet\Jobsheet14>  & 'C:\Program Fil
\User\workspaceStorage\672faa6c64c71f49b1182b9dfbd38ef5\redhat.java\jd

Daftar semua mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160221 Nama: Badar 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: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85

PreOrder Traversal:
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

PreOrder Traversal:
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

PostOrder Traversal:
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57

Penghapusan data mahasiswa
Jika 2 anak, current =
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
PS D:\Semester 2\Praktikum ASD\Jobsheet\Jobsheet14>
```

Nama : Siti Mutmainah
NIM : 244107020143

Pertanyaan Percobaan

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

   Jawab: Karena pada binary search tree data disusun secara terurut.
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
   Jawab: Atribut left dan right digunakan untuk menunjuk ke **anak kiri** dan **anak kanan** dari sebuah node. Ini yang membuat struktur tree terbentuk, karena setiap node bisa punya cabang ke kiri dan kanan.
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 berfungsi sebagai titik awal atau akar dari tree. Semua proses seperti penambahan, pencarian, dan penghapusan node dimulai dari node root.
   b. Nilai root adalah null, karena tree belum memiliki node sama sekali.
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   Jawab: Saat tree masih kosong (artinya root masih null), node pertama yang ditambahkan akan menjadi **root**. Node inilah yang menjadi pusat awal dari tree.
5. Perhatikan method **add()**, di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
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;
    }
}
```

   Jawab: Kode ini digunakan untuk menentukan posisi dimana node baru akan ditempatkan. Pertama parent = current menyimpan posisi node saat ini, sebelum pindah ke kiri atau kanan. Jika ipk mahasiswa lebih kecil, maka pindah ke anak kiri. Jika ternyata anak kiri kosong, berarti disitulah tempat menambahkan node baru. Jika ipk lebih besar pindah ke kanan, jika kosong tambahkan node di kanan.
6. Jelaskan langkah-langkah pada method **delete()** saat menghapus sebuah node yang memiliki dua anak. Bagaimana method **getSuccessor()** membantu dalam proses ini?

   Jawab: Cari pengganti yaitu node terkecil di subtree kanan. Ini dilakukan method getSuccessor(). Gantikan isi node yang akan dihapus dengan data dari successor. Hapus node successor karena datanya sudah dipindah ke node yang akan dihapus tadi.

Nama : Siti Mutmainah

NIM : 244107020143

Kegiatan Praktikum 2

Implementasi Binary Tree dengan Array (45 Menit)

Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method **main(),** dan selanjutnya akan disimulasikan proses traversal secara **inOrder**.
2. Buatlah class **BinaryTreeArray**00 dan **BinaryTreeArrayMain**00**.** Ganti **00** dengan nomer absen Anda.
3. Buat atribut **data** dan **idxLast** di dalam class **BinaryTreeArray**00. Buat juga method
   populateData() dan **traverseInOrder().**

BinaryTreeArray21.java > ...

```java
public class BinaryTreeArray21 {
    Mahasiswa21[] dataMahasiswa;
    int idxLast;

    public BinaryTreeArray21() {
        this.dataMahasiswa = new Mahasiswa21[10];
    }

    void populateData(Mahasiswa21 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);
            }
        }
    }
}

```

4. Kemudian dalam class **BinaryTreeArrayMain**00 buat method **main()** dan tambahkan kode seperti gambar berikut ini di dalam method main()**.**

```
public class BinaryTreeArrayMain21 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray21 bta = new BinaryTreeArray21();

        Mahasiswa21 mhs1 = new Mahasiswa21(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57);
        Mahasiswa21 mhs2 = new Mahasiswa21(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.41);
        Mahasiswa21 mhs3 = new Mahasiswa21(nim:"244160221", nama:"Badar", kelas:"B", ipk:3.75);
        Mahasiswa21 mhs4 = new Mahasiswa21(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.35);
        Mahasiswa21 mhs5 = new Mahasiswa21(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.48);
        Mahasiswa21 mhs6 = new Mahasiswa21(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.61);
        Mahasiswa21 mhs7 = new Mahasiswa21(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.86);

        Mahasiswa21[] dataMahasiswas = {mhs1, mhs2, mhs3, mhs4, mhs5, mhs6, mhs7, null, null, null};
        int idxLast = 6;

        bta.populateData(dataMahasiswas, idxLast);

        System.out.println(x:"\nInorder Traversal Mahasiswa: ");
        bta.traverseInOrder(idxStart:0);
    }
}
```

5. Jalankan class **BinaryTreeArrayMain00** dan amati hasilnya!

```
PS D:\Semester 2\Praktikum ASD\Jobsheet\Jobsheet14>  & 'C:\Program Files\Jav
ata\Roaming\Code\User\workspaceStorage\672faa6c64c71f49b1182b9dfbd38ef5\redh

Inorder Traversal Mahasiswa:
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.35
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.41
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.48
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.61
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.75
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.86
PS D:\Semester 2\Praktikum ASD\Jobsheet\Jobsheet14>
```

Pertanyaan Percobaan

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

   Jawab: data merupakan array yang menyimpan elemen pohon biner. idxLast menandakan indeks terakhir dalam array yang berisi data valid. Ini membantu membatasi operasi hanya pada elemen yang berisi data, bukan seluruh array.
5. Apakah kegunaan dari method **populateData()**?
   Jawab: Mengisi data pohon biner dari array yang diberikan.
6. Apakah kegunaan dari method **traverseInOrder()**?
   Jawab: Menelusuri pohon biner secara in-order
7. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masing-masing?
   Jawab: left indeks 5, right indeks 6
8. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Jawab: menunjukan bahwa data valid hanya ada sampai indeks 6 dalam array. Elemen setelah indeks 6 diabaikan.
9. Mengapa indeks 2*idxStart+1 dan 2*idxStart+2 digunakan dalam pemanggilan rekursif, dan apa kaitannya dengan struktur pohon biner yang disusun dalam array?
   Jawab: binary tree dalam array disimpan level by level. Rumus ini menghitung posisi child berdasarkan indeks parent.

Nama : Siti Mutmainah

NIM : 244107020143

Tugas Praktikum

Waktu pengerjaan: 150 menit

1. Buat method di dalam class **BinaryTree**00 yang akan menambahkan node dengan cara rekursif (**addRekursif**()).
   Jawab:

```
// Method add rekursif
public void addRekursif(Mahasiswa21 data) {
    root = addRekursif(root, data);
}

private Node21 addRekursif(Node21 current, Mahasiswa21 data) {
    if (current == null) {
        return new Node21(data);
    }

    if (data.ipk < current.mahasiswa.ipk) {
        current.left = addRekursif(current.left, data);
    } else if (data.ipk > current.mahasiswa.ipk) {
        current.right = addRekursif(current.right, data);
    }

    return current;
}
```

2. Buat method di dalam class **BinaryTree**00 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (**cariMinIPK**() dan **cariMaxIPK**()) yang ada di dalam binary search tree.
   Jawab:

```
// Method untuk mencari IPK terkecil
public Mahasiswa21 cariMinIPK() {
    if (root == null) {
        return null;
    }
    return cariMinIPK(root);
}

private Mahasiswa21 cariMinIPK(Node21 node) {
    return node.left == null ? node.mahasiswa : cariMinIPK(node.left);
}

// Method untuk mencari IPK terbesar
public Mahasiswa21 cariMaxIPK() {
    if (root == null) {
        return null;
    }
    return cariMaxIPK(root);
}

private Mahasiswa21 cariMaxIPK(Node21 node) {
    return node.right == null ? node.mahasiswa : cariMaxIPK(node.right);
}
```

3. Buat method dalam class **BinaryTree**00 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.
   Jawab:

```
public void tampilMahasiswaIPKdiAtas(double ipkBatas) {
    tampilMahasiswaIPKdiAtas(root, ipkBatas);
}

private void tampilMahasiswaIPKdiAtas(Node21 node, double ipkBatas) {
    if (node != null) {
        tampilMahasiswaIPKdiAtas(node.left, ipkBatas);
        if (node.mahasiswa.ipk > ipkBatas) {
            node.mahasiswa.tampilInformasi();
        }
        tampilMahasiswaIPKdiAtas(node.right, ipkBatas);
    }
}
```

4. Modifikasi class **BinaryTreeArray**00 di atas, dan tambahkan :
   - method **add(**Mahasiswa data**)** untuk memasukan data ke dalam binary tree
   - method **traversePreOrder()**

Jawab:

```
public void add(Mahasiswa21 mahasiswa) {
    if (size == 0) {
        data[0] = mahasiswa;
        size++;
        idxLast = 0;
    } else {
        add(idx:0, mahasiswa);
    }
}

private void add(int idx, Mahasiswa21 mahasiswa) {
    if (data[idx] == null) {
        data[idx] = mahasiswa;
        size++;
        if (idx > idxLast) {
            idxLast = idx;
        }
    } else {
        if (mahasiswa.ipk < data[idx].ipk) {
            add(2 * idx + 1, mahasiswa);
        } else {
            add(2 * idx + 2, mahasiswa);
        }
    }
}
```

```
public void traversePreOrder() {
    traversePreOrder(idx:0);
}

private void traversePreOrder(int idx) {
    if (idx >= data.length || data[idx] == null) {
        return;
    }
    data[idx].tampilInformasi();
    traversePreOrder(2 * idx + 1);
    traversePreOrder(2 * idx + 2);
}

public void traverseInOrder() {
    traverseInOrder(idx:0);
}

private void traverseInOrder(int idx) {
    if (idx >= data.length || data[idx] == null) {
        return;
    }
    traverseInOrder(2 * idx + 1);
    data[idx].tampilInformasi();
    traverseInOrder(2 * idx + 2);
}
```