


```

22         }else{
23             current.left = new Node(data);
24             break;
25         }
26     }else if(data>current.data){
27         if(current.right!=null){
28             current = current.right;
29         }else{
30             current.right = new Node(data);
31             break;
32         }
33     }else{
34         break;
35     }
36 }
37 }
38
39 boolean find(int data){
40     boolean hasil = false;
41     Node current = root;
42     while(current!=null){
43         if(current.data==data){
44             hasil = true;
45             break;
46         }else if(data<current.data){
47             current = current.left;
48         }else{
49             current = current.right;
50         }
51     }
52     return hasil;
53 }
54
55 void traversePreOrder(Node node){
56     if(node != null){
57         System.out.print(" "+node.data);
58         traversePreOrder(node.left);
59         traversePreOrder(node.right);
60     }
61 }
62
63 void traversePostOrder(Node node){
64     if(node != null){
65         traversePostOrder(node.left);
66         traversePostOrder(node.right);

```

```

66         System.out.print(" "+node.data);
67     }
68 }
69
70 void traverseInOrder(Node node){
71     if(node != null){
72         traverseInOrder(node.left);
73         System.out.print(" "+node.data);
74         traverseInOrder(node.right);
75     }
76 }
77
78 Node getSuccessor(Node del){
79     Node successor = del.right;
80     Node successorParent = del;
81     while(successor.left!=null){
82         successorParent = successor;
83         successor = successor.left;
84     }
85     if(successor != del.right){
86         successorParent.left = successor.right;
87         successor.right = del.right;
88     }
89     return successor;
90 }
91
92 void delet(int data){
93     if(isEmpty()){
94         System.out.println("Tree is empty!");
95         return;
96     }
97
98     Node parent = root;
99     Node current = root;
100    boolean isLeftChild = false;
101    while(current!=null){
102        if(current.data==data){
103            break;
104        }else if(data<current.data){
105            parent = current;
106            current = current.left;
107            isLeftChild = true;
108        }else if(data>current.data){
109            parent = current;
110            current = current.right;
111            isLeftChild = false;

```

110	}
111	}
112	
113	if(current==null){
114	System.out.println("Couldn't find data!");
115	return;
116	}else{
117	if(current.left==null&¤t.right==null){
118	if(current==root){
119	root = null;
120	}else{
121	if(isLeftChild){
122	parent.left = null;
123	}else{
124	parent.right = null;
125	}
126	}
127	}else if(current.left==null){
128	if(current==root){
129	root = current.right;
130	}else{
131	if(isLeftChild){
132	parent.left = current.right;
133	}else{
134	parent.right = current.right;
135	}
136	}
137	}else if(current.right==null){
138	if(current==root){
139	root = current.left;
140	}else{
141	if(isLeftChild){
142	parent.left = current.left;
143	}else{
144	parent.right = current.left;
145	}
146	}
147	}else{
148	Node successor = getSuccessor(current);
149	if(current==root){
150	root = successor;
151	}else{
152	if(isLeftChild){
153	parent.left = successor;
	}else{
	parent.right = successor;

154	}
155	successor.left = current.left;
156	}
157	}
158	}
159	}
160	}

No	BinaryTreeMain.java
1	package jobsheet12;
2	
3	public class BinaryTreeMain {
4	public static void main(String[] args){
5	BinaryTree bt = new BinaryTree();
6	
7	bt.add(6);
8	bt.add(4);
9	bt.add(8);
10	bt.add(3);
11	bt.add(5);
12	bt.add(7);
13	bt.add(9);
14	bt.add(10);
15	bt.add(15);
16	
17	bt.traversePreOrder(bt.root);
18	System.out.println("");
19	bt.traverseInOrder(bt.root);
20	System.out.println("");
21	bt.traversePostOrder(bt.root);
22	System.out.println("");
23	System.out.println("Find "+bt.find(5));
24	bt.delet(8);
25	bt.traversePreOrder(bt.root);
26	System.out.println("");
27	}
28	}

Output :

```

run:
6 4 3 5 8 7 9 10 15
3 4 5 6 7 8 9 10 15
3 5 4 7 15 10 9 8 6
Find true
6 4 3 5 9 7 10 15
BUILD SUCCESSFUL (total time: 0 seconds)

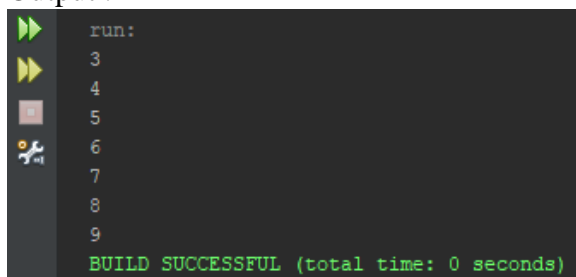
```

2. Praktikum2

No	BinaryTreeArray.java
1	package jobsheet12;
2	
3	public class BinaryTreeArray {
4	int[] data;
5	int idxLast;
6	
7	public BinaryTreeArray(){
8	data = new int[10];
9	}
10	
11	void populateData(int data[], int idxLast){
12	this.data = data;
13	this.idxLast = idxLast;
14	}
15	
16	void tranverseInOrder(int idxStart){
17	if(idxStart<=idxLast){
18	tranverseInOrder(2*idxStart+1);
19	System.out.println(data[idxStart]+" ");
20	tranverseInOrder(2*idxStart+2);
21	}
22	}
23	}

No	BinaryTreeArrayMain.java
1	package jobsheet12;
2	
3	public class BinaryTreeArrayMain {
4	public static void main(String[] args){
5	BinaryTreeArray bt = new BinaryTreeArray();
6	int[] data = {6,4,8,3,5,7,9,0,0,0};
7	int idxlast = 6;
8	bt.populateData(data, idxlast);
9	bt.tranverseInOrder(0);
10	}
11	}

Output :



```

run:
3
4
5
6
7
8
9
BUILD SUCCESSFUL (total time: 0 seconds)

```

B. Pertanyaan

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

Jawab :

karena pada Binary search tree data terletak pada pola yang terstruktur dimana data yang lebih kecil akan terletak pada sebelah kiri sedangkan data yang besar di sebelah kanan.

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

Jawab :

untuk menyimpan data dan pointer. Atribut left untuk menaruh data ke sebelah kiri dan right untuk menaruh data sebelah 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. untuk menandai Node paling awal pada program binarytree
- b. Node pertama bernilai null.

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

Jawab :

Node yang baru saja dimasukkan akan menempati posisi root.

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 :

apabila data yang dimasukkan lebih kecil dari current.data maka data akan menuju ke posisi kiri lalu jika current left tidak sama dengan null maka Node akan bertambah dan menuju ke kiri , jika sama dengan null maka node yang ditambahkan berada di tempat yang belum terisi.

6. Apa perbedaan antara traverse mode pre-order, in-order dan postorder?

Jawab :

- a. PreOrder: Parent diprint terlebih dahulu lalu child kiri lalu child kanan.
- b. InOrder: Child kiri diprint terlebih dahulu lalu parent lalu child kanan.
- c. PostOrder: Child kiri diprint terlebih dahulu lalu child kanan lalu parent.

7. Perhatikan method delete(). Sebelum proses penghapusan node, didahului dengan proses pencarian node yang akan dihapus. Selain ditujukan untuk mencari node yang akan dihapus (current), proses pencarian tersebut juga akan mencari parent dari node yang akan dihapus (parent). Menurut Anda, mengapa diperlukan juga untuk mengetahui parent dari node yang akan dihapus?

Jawab :

8. Untuk apakah dibuat variabel dengan nama isLeftChild di dalam method delete()?

Jawab :

untuk mengetahui apakah delete tersebut child left atau bukan.

9. Untuk apakah method getSuccessor()?

Jawab :

untuk mencari successor yang nantinya akan menggantikan Node yang dihapus.

10. Di ulasan teori, disebutkan bahwa ketika suatu node yang memiliki 2 child dihapus, node tersebut digantikan oleh node successor, dimana node successor bisa didapat dengan 2 cara, yaitu 1) mencari nilai terbesar dari subtree di sebelah kirinya, atau 2) mencari nilai terkecil dari subtree di sebelah kanannya. Manakah 1 dari 2 cara tersebut yang diimplementasikan dalam method getSuccessor() di program di atas?

Jawab :

mencari nilai terkecil dari subtree pada sebelah kanan

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

Jawab :

Karena Binary Search tree Array memiliki panjang batas untuk tree, jadi untuk menentukan indeks nya.

12. Apakah kegunaan dari method populateData() dan traverseInOrder()?

Jawab :

mengisi atribut yang ada pada objek

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

Jawab :

pada 5 dan 6

C. Tugas

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

No	Tugas 1
1	Node addrecursive(Node current, int data){
2	if(current == null){
3	return new Node(data);
4	}
5	if(data < current.data){
6	current.left = addrecursive(current.left, data);
7	}
8	else if(data > current.data){
9	current.right = addrecursive(current.right, data);
10	}
11	else{
12	return current;
13	}
14	return current;
15	}
16	void addRecursive(int data){
17	root = addrecursive(root, data);
18	}

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

1	int kecil() {
2	Node current = root;
3	if (current.left == null) {
4	return current.data;
5	} else {
6	while (current.left != null) {
7	current = current.left;
8	}
9	return current.data;
10	}
11	}
12	int besar() {
13	Node current = root;
14	if(current.right == null) {
15	return current.data;
16	} else {
17	while (current.right != null) {
18	current = current.right;
19	}
20	}
21	return current.data;

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

1	void dataleaf(Node node){
2	if(node!=null){
3	dataleaf(node.left);
4	if(node.left == null && node.right == null);
5	System.out.print(" " +node.data);
6	dataleaf(node.right);
7	}
8	}

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

1	int jumlah;
2	int jumlahleaf(Node node){
3	if(node !=null){
4	jumlahleaf(node.left);
5	if(node.left == null && node.right == null)
6	jumlah++;
7	jumlahleaf(node.right);
8	}
9	return jumlah;
10	}

5. Modifikasi class BinaryTreeMain, sehingga di dalam terdapat pilihan menu :

- Add
- Delete
- find
- traverse inOrder
- traverse preOrder
- traverse postOrder
- Keluar

1	public class BinaryTreeMain {
2	public static void main(String[] args) {
3	Scanner sc = new Scanner(System.in);
4	BinaryTree bt = new BinaryTree();
5	while(true){
6	System.out.println("1. Add");
7	System.out.println("2. Delete");
8	System.out.println("3. Find");
9	System.out.println("4. Traverse Inorder");
10	System.out.println("5. Traverse Preorder");
11	System.out.println("6. Traverse Postorder");
12	System.out.println("7. Exit");
13	System.out.println(" Menu = ");
14	int menu = sc.nextInt();

15	switch(menu){
16	case 1:
17	System.out.println(" > ");
18	int c1=sc.nextInt();
19	bt.add(c1);
20	break;
21	case 2:
22	System.out.println(" > ");
23	int c2=sc.nextInt();
24	bt.add(c2);
25	break;
26	case 3:
27	System.out.println(" > ");
28	int c3=sc.nextInt();
29	bt.add(c3);
30	break;
31	case 4:
32	bt.traverseInOrder(bt.root);
33	System.out.println(" ");
34	break;
35	case 5:
36	bt.traversePreOrder(bt.root);
37	System.out.println(" ");
38	break;
39	case 6:
40	bt.traversePostOrder(bt.root);
41	System.out.println(" ");
42	break;
43	default:
44	break;
45	}
46	}
47	}
48	}