Week 14

Subject	Basic Programming Practicum
Lecturer	Imam Fahrur Rozi ST. MT.
Type	Assignment
Semester	Semester 2
Time	@June 8, 2023
Files & Media	

Jobsheet 14

questions

1. because the data is already sorted, so it will be more efficient to use binary search algorithm
2. because in order to connect on each data, we still need the node. the `left` and `right` attributes is used to determine whether the child data is in the right or left from parent
3. answer
    a. `root` is used for the root (main parent) of the tree
    b. the value is `null` because it doesn't have any data
4. the `node` will assigned as the root, because the tree is still empty
5. the code is used to check whether the `data` is less than the `current.data`, if so then it will check again whether `current.left` is not `null`, then it will assign `current.left` to `current`, if it isn't, then it will make `current.left` as a new `data` to assign as left child node

6. `pre-order` is traversal of the tree, from the root node to subtree, then to the right subtree, after that to the right subtree, or can be like root → left → right

   `in-order` the same as `pre-order` but goes left → root → right

   `post-order` also same, but goes left → right → root

7. in order to delete, we need to re-connect the left child into its parent, if we don't connect after we remove it, the tree will be broken because it's disconnected from the rest of the tree
8. used to check whether the current node is a left child or not
9. used to find the successor of the current node in the tree
10. looks for the largest value of the subtree to the left, because it's determined with `successor.left != null` so it will try to traverse the left path until it doesn't have any left child node
11. used to see the last index of the array that contains the tree
12. `populateData()` is used to populate the data inside the `BinaryTreeArray` class

    `traverseInOrder()` is used to traverse the data inside the `BinaryTreeArray` class
13. the left-child will be positioned in $n * 2 + 1$ and the right child will be positioned in $n * 2 + 2$ where n is the index of the node

assignment

1. code

```
void recursiveAdd(int data)
    {
        recursiveAdd(data, root);
    }

    void recursiveAdd(int data, Node parent)
    {
        if (isEmpty())
        {
            root = new Node(data);
            return;
        }
        if (data < parent.data) {
            if (parent.left != null) recursiveAdd(data, parent.left);
            else parent.left = new Node(data);
```

```
        }
        else if (data > parent.data) {
            if (parent.right != null) recursiveAdd(data, parent.right);
            else parent.right = new Node(data);
        }
    }
```

2. code

```
void displayLargestAndSmallest()
    {
        int smallest = Integer.MAX_VALUE;
        int largest = Integer.MIN_VALUE;

        Node current = root;
        while (current != null)
        {
            if (current.data < smallest) smallest = current.data;
            current = current.left;
        }

        current = root;
        while (current != null)
        {
            if (current.data > largest) largest = current.data;
            current = current.right;
        }

        System.out.println("Smallest: " + smallest);
        System.out.println("Largest: " + largest);
    }
```

3. code

```
void displayLeafData(Node node)
    {
        if (node == null) return;
        if (node.left == null && node.right == null)
        {
            System.out.println(node.data + " ");
            return;
        }
        displayLeafData(node.left);
        displayLeafData(node.right);
    }
```

4. code

```
int countLeaves(int count, Node node)
    {
        if (node == null) return 0;
        if (node.left == null && node.right == null) return count + 1;
        return  countLeaves(count, node.left) + countLeaves(count, node.right);
    }
```

5. code

```
//add this
int menu;
        do
        {
            System.out.println("1. Add");
            System.out.println("2. Delete");
            System.out.println("3. Find");
            System.out.println("4. traverseInOrder");
            System.out.println("5. traversePreOrder");
            System.out.println("6. traversePostOrder");
            System.out.println("7. Exit");
            menu = sc.nextInt();
            switch (menu) {
                case 1:
                    System.out.println("Insert Data");
                    System.out.print("Data node : ");
                    int data = sc.nextInt();
                    bt.add(data);
                    break;
                case 2:
                    System.out.println("Delete Data");
                    System.out.print("Data node : ");
                    int deleteData = sc.nextInt();
                    bt.delete(deleteData);
                    break;
                case 3:
                    System.out.println("Find Data");
                    System.out.print("Data node : ");
                    int searchData = sc.nextInt();
                    System.out.println(bt.find(searchData) ? "Found" : "Not Found");
                    break;
                case 4:
                    bt.traverseInOrder(bt.root);
                    System.out.println();
                    break;
                case 5:
                    bt.traversePreOrder(bt.root);
                    System.out.println();
```

```
                break;
            case 6:
                bt.traversePostOrder(bt.root);
                System.out.println();
                break;
            case 7:
                break;
            default:
                System.out.println("Invalid menu");
                break;
        }
    }
    while (menu != 7);
```

6. code
    a. code

```
void add(int data)
    {
        int currentIdx = 0;
        while (true)
        {
            if (currentIdx >= idxLast) break;
            if (data > this.data[currentIdx]) currentIdx = currentIdx * 2 + 2;
            else if (data < this.data[currentIdx]) currentIdx = currentIdx * 2 + 1;
            else break;
        }
        this.data[currentIdx] = data;
    }
```

    b. code

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

    void traversePostOrder(int idxStart)
    {
        if (idxStart <= idxLast)
        {
            traversePostOrder(2 * idxStart + 1);
```

```
            traversePostOrder(2 * idxStart + 2);
            System.out.print(data[idxStart] + " ");
        }
    }
```