Data Structure and Algorithm Practicum
Binary Tree

Name
Dicha Zelianivan Arkana

NIM
2241720002

Class
1i

Department
Information Technology

Study Program
D4 Informatics Engineering

1 Questions

1. Why the data searching process is more efficient in the binary search tree than in ordinary binary tree?

   Because in binary search tree, the data is already sorted, which means we can apply the binary search algorithm throughout the binary tree. Unlike an ordinary binary tree which may not be sorted so we can't apply the binary search algorithm to the data.

2. Why do we need the **Node** class? What are the **left** and **right** attributes?

   The `Node` class represents the node inside the tree that contains the data. The `left` and `right` attribute represents the left and right child of the node. Since this is a binary tree, there would only be 2 child nodes.

3. a.) What are the uses of the **root** attribute in `BinaryTree` class?

   It is used so that we can keep track of the root of the tree.

   b.) When the tree object was first created, what is the value of **root**?

   The root of the tree is set to be `null` at first because it doesn't contain anything

4. When the tree is still empty, and a new node is added, what process will happen?

   The newly added node will be assigned as a root because the tree is empty which means that it has no root yet.

5. Pay attention to the `add()` method, in which there are program lines as below. Explain in detail what the program line is for?

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

   It checks if the data that we want to insert is less than the current node that we're in. If so, then we'll go to the left by assigning the current left node into current itself. This will make it so that we traverse to the left of the current node. When we reach a condition where the current data is greater than the current node data, then we'll assign it as its left child node.

6. What is the difference between pre-order, in-order, and post-order traverse modes?

   - **pre-order** pre-order traversal of the tree starts from the root node and then traverses the left subtree, then traverses the right subtree. It goes from root -¿ left -¿ right
   - **in-order** in-order traversal of the tree starts from the root node and then traverses the left subtree, then traverses the root node, then traverses the right subtree. It goes from left -¿ root -¿ right
   - **post-order** post-order traversal of the tree starts from the root node and then traverses the left subtree, then traverses the right subtree, then traverses the root node. It goes from left -¿ right -¿ root

7. Look at the `delete()` method. Before the node removal process, it is preceeded by the process of finding the node to be deleted. Besides intended to find the node to be deleted (current), the search process will also look for the parent of the node to be deleted (parent). In your opinion, why is it necessary to know the parent of the left node to be deleted?

   We need to know the parent of the left node to be deleted because we want to re-connect the left child into its parent. If we don't connect the node after removing it, then the tree starting from the one we remove will be broken / disconnected from the rest of the tree.

8. For what is a variable named `isLeftChild` created in the `delete()` method?

   It is used to check if the current node is a left child.

9. What is the `getSuccessor()` method for?

   It is used to find the successor of the current node in the binary tree.

10. In a theoritical view, it is stated that when a node has 2 children is deleted, the node is replaced by the successor node, where the successor node can be obtained in 2 ways, namely:

    (a) Looking for the largest value of the subtree to the left, or

    (b) Looking for the smallest value of the subtree on the right.

    Which 1 of 2 methods is implemented in the `getSuccessor()` method in the above program?

    It looks for the largest value of the subtree to the left. This is determined by the `successfor.left != null` which means it will try to traverse the left path until there is no left child node left, effectively finding the largest value of the subtree to the left.

11. What are the uses of the data and `idxLast` attributes in the `BinaryTreeArray` class?

    It is used so that we can keep track of the last index of the array that contains the tree.

12. What are the uses of the `populateData()` and `traverseInOrder()` methods?

    - `populateData`: used to populate the data inside the `BinaryTreeArray` class
    - `traverseInOrder`: used to traverse the data inside the `BinaryTreeArray` class

13. If a binary tree node is stored in index array 2, when in what index are the left-child and right-child position respectively?

    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. In this case it's 2, so that means the left child will be positioned at index 5 and the right child will be positioned at index 6.

2 Assignments

1. Create a method inside the `BinaryTree` class that will add nodes with recursive approach!

```
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. Create a method inside the `BinaryTree` class to display the smallest and largest value in the tree!

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

    // traverse the left part to find the smallest value
    Node current = root;
    while (current != null) {
        if (current.data < smallest) {
            smallest = current.data;
        }
        current = current.left;
    }

    // traverse the right part to find the largest value
    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. Create a method in the `BinaryTree` class to display the data in the leaf!

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

4. Create a method in the `BinaryTree` class to display the number of leaves in the tree!

```
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. Modify the `BinaryTreeMain` class, so that it has a menu option:

   a.) `add`

   b.) `delete`

   c.) `find`

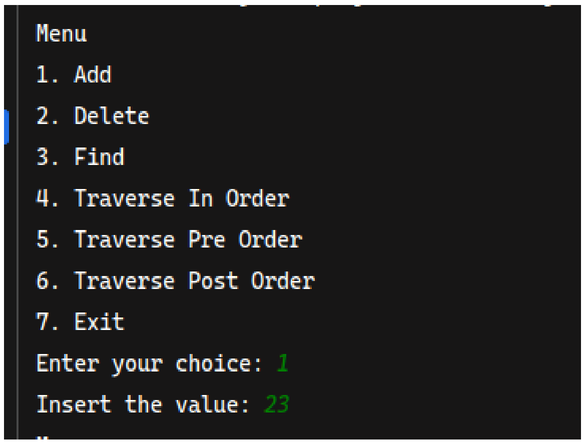
   d.) `traverseInOrder`

   e.) `traversePreOrder`

   f.) `traversePostOrder`

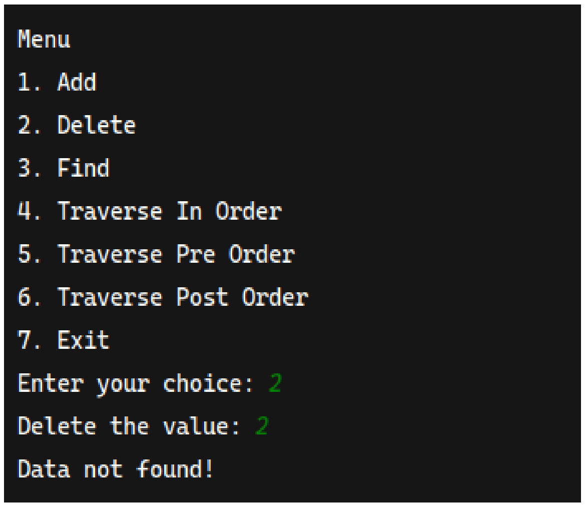
   g.) `exit`

```
public class BinaryTreeMain {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BinaryTree binaryTree = new BinaryTree();

        while (true) {
            showMenu();
            System.out.print("Enter your choice: ");
            int chosenMenu = scanner.nextInt();
            switch (chosenMenu) {
                case 1:
                    System.out.print("Insert the value: ");
                    int valueAdded = scanner.nextInt();
                    binaryTree.add(valueAdded);
                    break;
                case 2:
                    System.out.print("Delete the value: ");
                    int valueDeleted = scanner.nextInt();
                    binaryTree.delete(valueDeleted);
                    break;
                case 3:
                    System.out.print("Find the value: ");
                    int valueSearch = scanner.nextInt();
                    boolean found= binaryTree.find(valueSearch);
                    System.out.println(found ? "Found" : "Not Found");
                    break;
                case 4:
                    binaryTree.traverseInOrder(binaryTree.root);
                    System.out.println();
                    break;
                case 5:
                    binaryTree.traversePreOrder(binaryTree.root);
                    System.out.println();
                    break;
                case 6:
                    binaryTree.traversePostOrder(binaryTree.root);
                    System.out.println();
                    break;
                case 7:
                    System.out.println("Bye!");
                    return;
            }
        }
    }

    static void showMenu() {
        System.out.println("Menu");
        System.out.println("1. Add");
        System.out.println("2. Delete");
        System.out.println("3. Find");
        System.out.println("4. Traverse In Order");
        System.out.println("5. Traverse Pre Order");
        System.out.println("6. Traverse Post Order");
        System.out.println("7. Exit");
    }
}
```



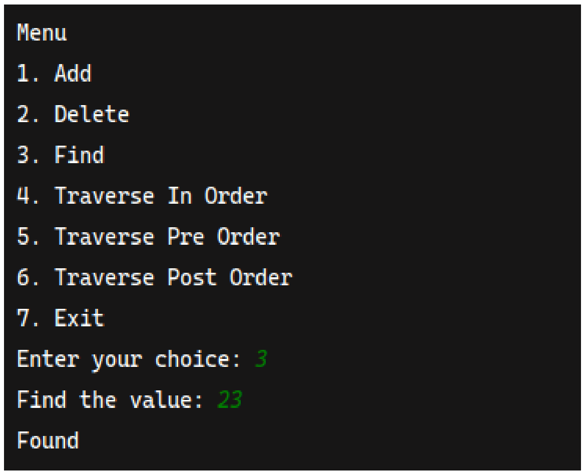
```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 1
Insert the value: 23
```

Figure 1: Add



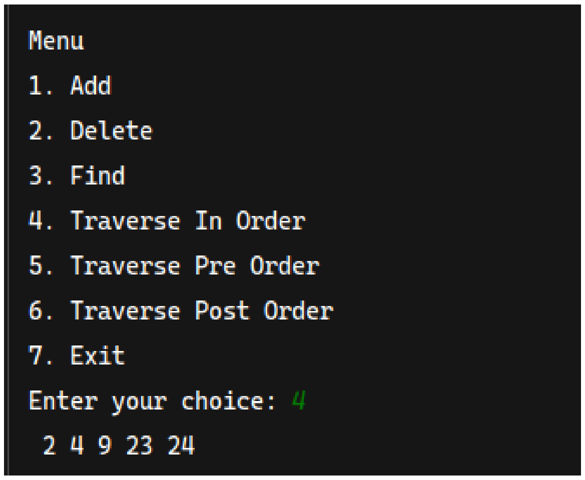
```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 2
Delete the value: 2
Data not found!
```

Figure 2: Delete



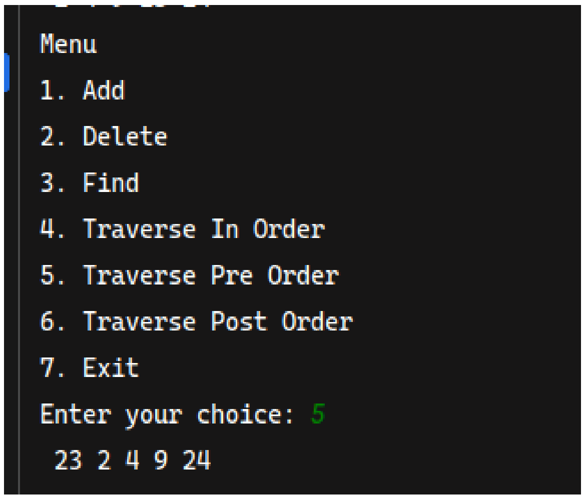
```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 3
Find the value: 23
Found
```

Figure 3: Find



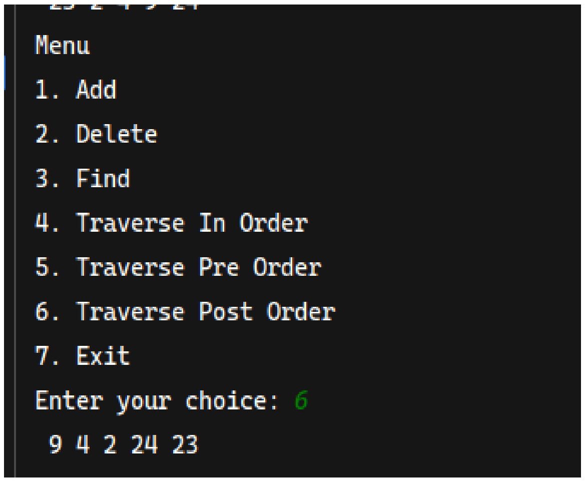
```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 4
 2 4 9 23 24
```

Figure 4: Traverse In Order



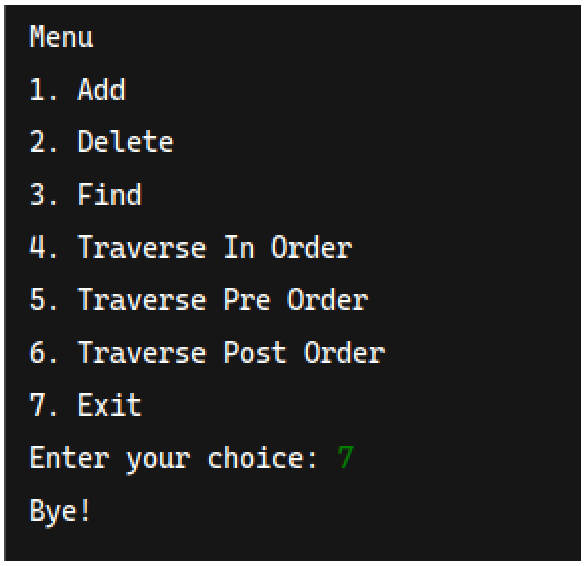
```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 5
 23 2 4 9 24
```

Figure 5: Traverse Pre Order



```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 6
 9 4 2 24 23
```

Figure 6: Traverse Post Order



```
Menu
1. Add
2. Delete
3. Find
4. Traverse In Order
5. Traverse Pre Order
6. Traverse Post Order
7. Exit
Enter your choice: 7
Bye!
```

Figure 7: Exit

6. Modify the `BinaryTreeArray` class and add:

   a.) Add a method `add(int data)` to enter data into the tree

```java
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.) `traversePreOrder()` and `traversePostOrder` methods

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

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