(***Create a table of contents before your solution***)

Contents

[Problem 1. 3](#_Toc70682486)

[Problem 2. 11](#_Toc70682487)

[Problem 3. 11](#_Toc70682488)

[Problem 4. 11](#_Toc70682489)

[Problem 5. 12](#_Toc70682490)

[Problem 6. 12](#_Toc70682491)

[Problem 7. 12](#_Toc70682492)

[Problem 8. 12](#_Toc70682493)

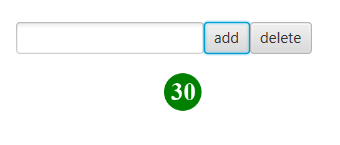
[Problem 9. 13](#_Toc70682494)

# Problem 1.

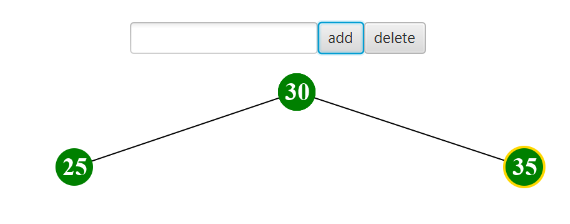
1. - No run-time or compilation errors.

2. - InsertNode function implemented.

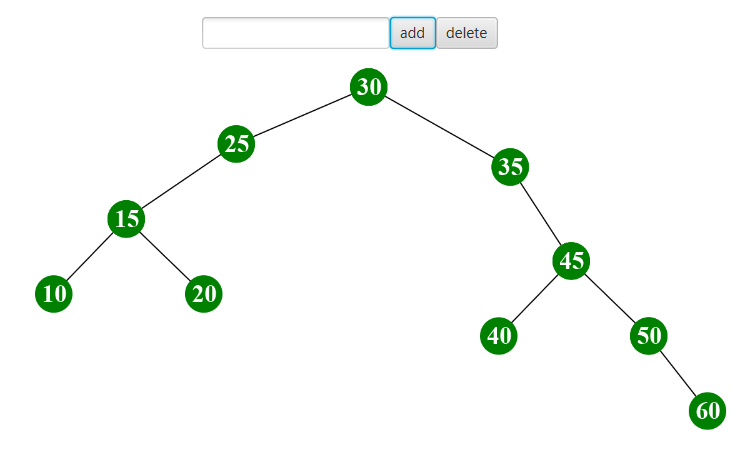
2a - Add the root node successfully



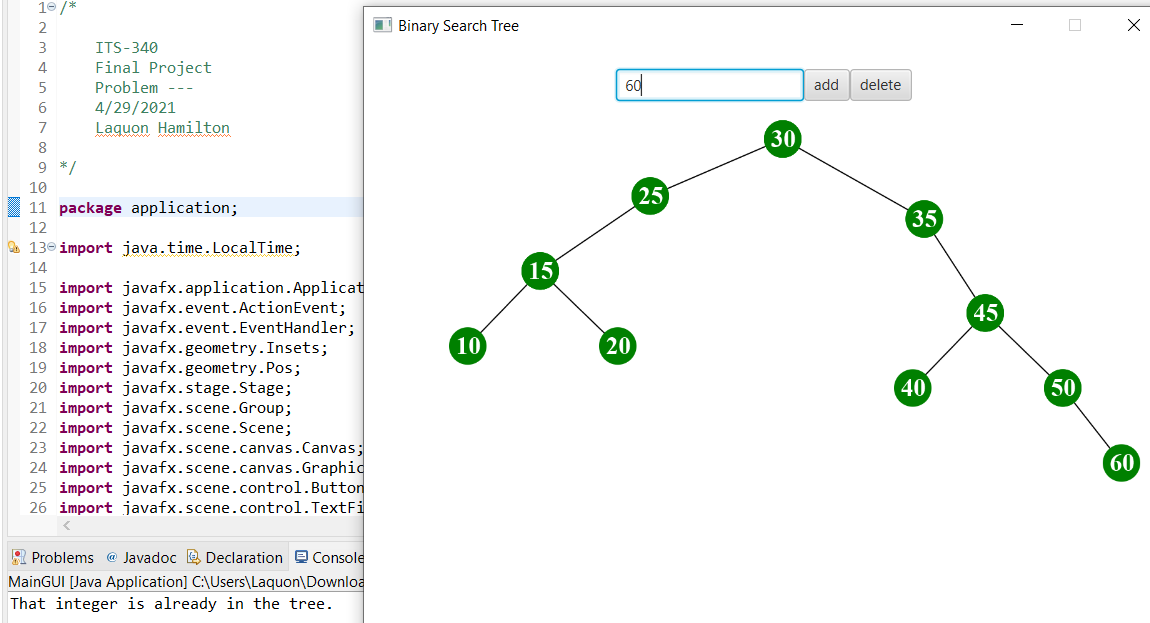
**2b.** Add left and right child nodes



2c. Unlimited amount of nodes

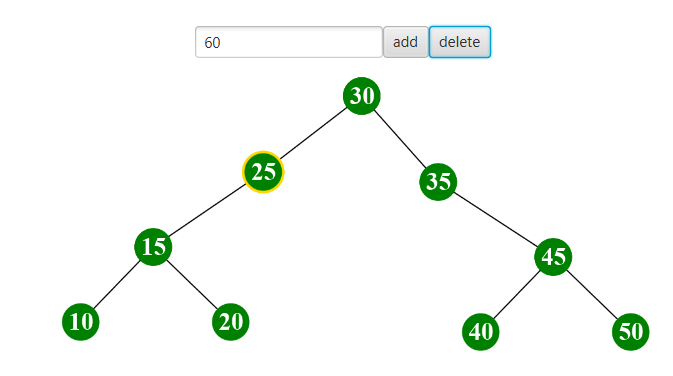


2d. No duplicate nodes

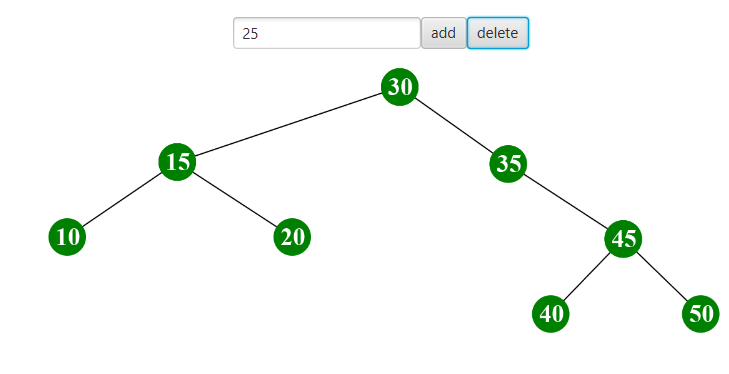


3. DeleteNode function implemented

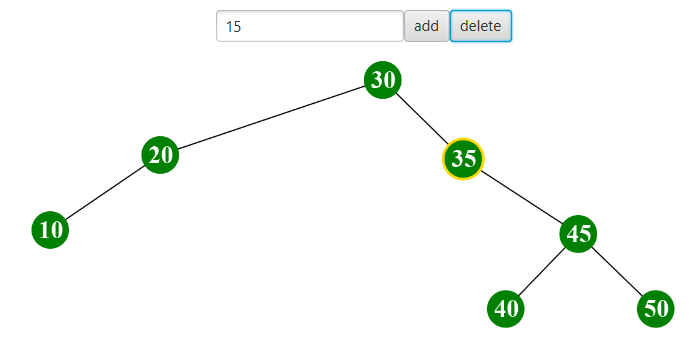
3a. Delete node without children



3b. Delete node with one child.



3c. Delete a node with two children.



Code:

Part 1 (Node.java)

  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Implementation Here (Optional) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

  //Feel free to add any additional data or member functions here and the corresponding "setter" and "getter"

  //It is really not necessary to write any additional code for this Node.java class. But just in case, if you want to

  //modify this class for your specific implementation, you can put your code here

  public Node deleteRec(Node root, int key) //receives the new node and key from the deleteNode method

  { //this method constantly recurs down the tree until it finds the value equal to the key

    if (root == null)

    {

      return null; //exits the method if the root is not yet set

    }

    if (key < root.value) //if key less than current value, go down the left side of the tree

    {

      root.left = deleteRec(root.left, key);

    }

    else if (key > root.value) //if key greater than current value, go down the right side of the tree

    {

      root.right = deleteRec(root.right, key);

    }

    else //if the key equals the current value, then this value will be deleted

    {

      //next two if statements will delete node if it has 1 child or none

      if (root.left == null)

      {

        return root.right;

      }

      else if (root.right == null)

      {

        return root.left;

      }

      root.value = minValue(root.right); //if the node has two children, send the value to the minVal method which

                        //will get the inorder successor (smallest node in the right subtree)

      root.right = deleteRec(root.right, root.value); //delete the successor

    }

    return root; //returns root to the calling method

  }

int minValue(Node root) //method for determining the inorder successor

{

int minVal = root.value; //set minimum value to the current value

while (root.left != null) //recur down the left side of the tree until it reaches the end

{

minVal = root.left.value;

root = root.left;

}

return minVal; //return the minimum value to the calling method which will then be deleted

}

  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  End of Implementation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Part 2 (Tree.java)

  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Implement Here: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

  /\*Finish the two basic functions below "insertNode(Node node)" and "deleteNode(Node node)\*/

  /\*

   \* The function below is to insert a new node into the binary search tree. You need to

   \* implement the code: if a node is successfully

   \* inserted, it returns "true"; if the node to be inserted has the value

   \* already exist in the tree, it should not be inserted and return "false".

   \* So you need to modify the code to forbid nodes with repeating values to

   \* be inserted.

   \*/

  ArrayList<Integer> checkTree = new ArrayList<>(); //list used to check for duplicate integers

  public boolean insertNode(Node node, int key) //New node / key will be sent by MainGUI.createNode

  {//this method mainly checks for duplicate keys and calls the recursion method to do all the actual work.

//Implementation Here...

    if (checkTree.contains(key) == true) //checks if the integer exists within the tree already

    {

      System.out.println("That integer is already in the tree.");

      return false; //end the method to prevent further code from being executed

    }

    checkTree.add(key); //add the key to the arraylist

    root = recurInsert(root, node); //root will equal the output of the first recursion and sends the root value

                    //and new node to the recursion method

    return true; //return true to createNode() for further execution

  }

  public Node recurInsert(Node current, Node newNode) //receives the current node (root if null) and new node from

                            //insertNode()

  {//This method will constantly recur new nodes down the tree where necessary

    if (current == null) //set current node to the new node entered

    {

      current = newNode;

    }

    else if (newNode.getValue() < current.getValue()) //if new node less than current node, apply node to left of

                            //parent

    {

      current.setLeft(recurInsert(current.getLeft(), newNode)); //continue to recur down the tree for each new node

    }

    else if (newNode.getValue() > current.getValue())//if new node greater than current node, apply node to right

                            //of parent

    {

      current.setRight(recurInsert(current.getRight(), newNode)); //continue to recur down the tree for each new node

    }

    return current; //returns the current node to the calling function

  }

  /\*

   \* The function below is to delete a selected node from the tree. You need

   \* to finish the implementation.

   \*/

  public boolean deleteNode(Node node, int key) //receives the new node / key from MainGUI.deleteNode()

  {//this method is mainly for checking if node exists in the tree already and sending the new node to the

    //recursion function

    if (checkTree.contains(key) == false) //checks if key exists in tree before continuing

    {

      System.out.println("That integer does not exist in the tree");

      return false;

    }

    checkTree.removeIf(checkTree -> checkTree.equals(key)); //removes key from the arraylist if new key equals any

                                //key inside arraylist

    node = root; //start at the root

    node.deleteRec(node, key); //calls the deleteRec method located in the Node class and sends the new node

                 //and key to it.

    //The two variables below are for GUI use only. You don't need to use them.

    select\_node\_value = -1;

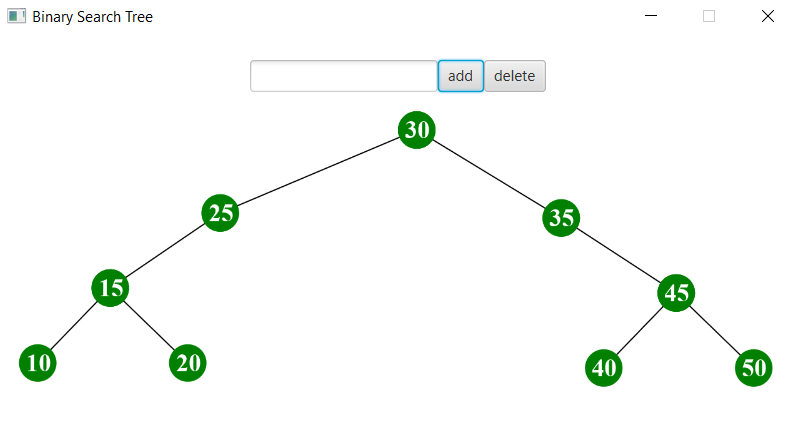
    selected\_node = null;

    return true;

  }

  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Implementation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Screenshot



# Problem 2.

Code

Screenshot

# Problem 3.

Code:

Screenshot

# Problem 4.

Code

Screenshot

# Problem 5.

Code

Screenshot

# Problem 6.

Code

Screenshot

# Problem 7.

Code

Screenshot

# Problem 8.

Code

Screenshot

# Problem 9.

Code

Screenshot