GIT Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework 5 Report

Abdurrahman BULUT

1901042258

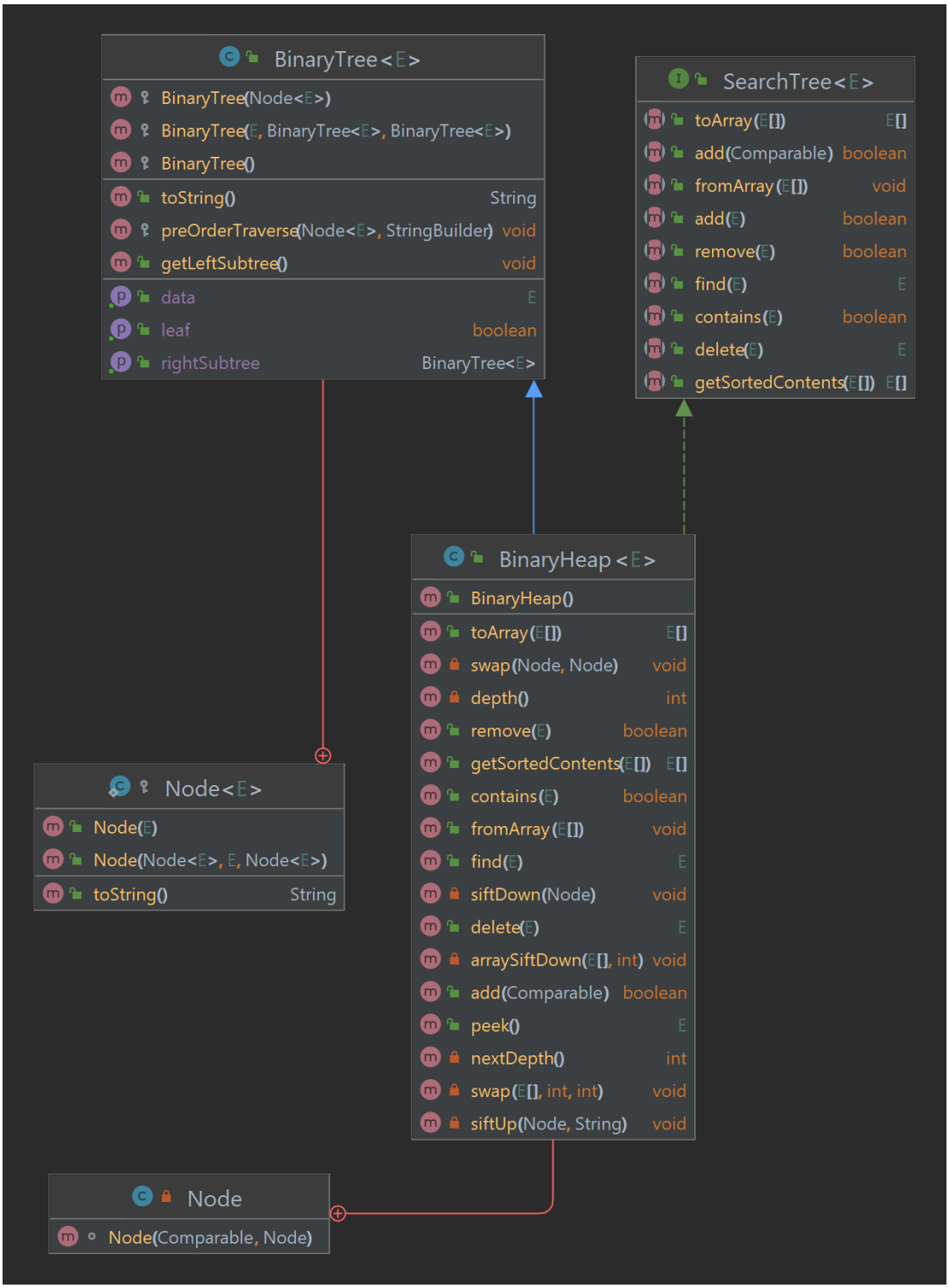
1. SYSTEM REQUIREMENTS

❖ Functional Requirements

➢ System

- openjdk version "11.0.14.1" 2022-02-08 LTS
- OpenJDK Runtime Environment Corretto-11.0.14.10.1 (build 11.0.14.1+10-LTS)
- OpenJDK 64-Bit Server VM Corretto-11.0.14.10.1 (build 11.0.14.1+10-LTS, mixed mode)

2. USE CASE AND CLASS DIAGRAMS



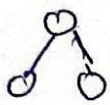
3. PROBLEM SOLUTION APPROACH

Foe question 2, I drew relations with arrows. I have put data values to table and inserted to binary tree. For question 3, I wrote a search Tree java file, a binary Tree file and binary Heap file. Binary heap class implements search Tree class. Most of the time Binary heap can be written with array. But we are asked to write using nodes. So, I created a node class. To go left and right, I used 2n+1 and 2n+2 formulas respectively to traverse. For question 4, I used an array to represent. Again, to move left side of the tree 2n+1 is used. To move right side of the tree 2n+2 is used.

4. Question 1

a) Calculate the total depth of the nodes in a complete binary tree of height h. Note that total depth is 5 if height is 2 where the depth of the root is one and there are two nodes with depth 2.

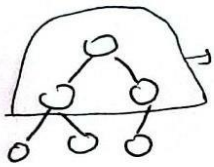
As example

→ level 1  → height 1 , depth 1

→ level 2 → height 2 , depth 2

So, Total depth is $1+2+2=5$ for this example.

A complete binary tree is a perfect binary tree except the last level.

for example :  → this is a perfect BT.

and, $\boxed{2^n-1}$ number of Nodes.

For Perfect BT Part;

$$1.1 + 2.2 + 3.4 + 4.8 \cdots$$

height, number of nodes

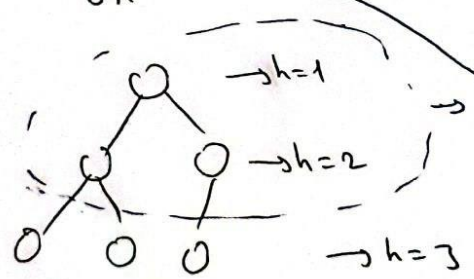
$$\Rightarrow \sum_{n=0}^{n} (n+1).2^{n} \quad , n = h-1$$

$\Rightarrow$ Depends on the last level of tree,
The total Depth will be;

P: Number of Nodes on the last level

$$\underbrace{\left[\sum_{h=1}^{h} (h).2^{h-1}\right]}_{\text{if it is perfect BT.}} - \underbrace{\left[(2^{h-1} - p).(h-1)\right]}_{\text{Number of Nodes on the last level}}$$

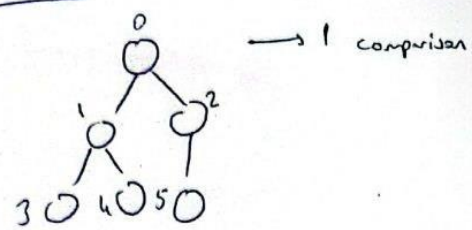
Example



$\rightarrow$ It is always Perfect BT

b) Calculate the average number of comparisons for a successful search operation in a binary search tree which has the structural property of being complete binary tree.

for example



~~For~~ To find Node 0; 1 comparison → height 1

Node 1; 2 comparison
Node 2; 2 comparison } → height 2

Node 3; 3 comparison
Node 4 and 5; 3 comparison. } height 3

so; First we should calculate Perfect BT Part (h-1)

p: Number of Nodes on the last level

Total Comparison result = $\underbrace{\left[\sum_{h=1}^{h} h \cdot 2^{h-1} \cdot \underbrace{h}_{\text{comparison}}\right]}_{\text{if the tree is a perfect BT.}} - \underbrace{\left[(2^{h-1}-p).(h-1).\underbrace{(h-1)}_{\text{missing nodes comparison}}\right]}$

Total Node Number: $2^h - 1$, if it is Perfect BT

Total Node Number of our assumption: $2^h - 1 - p$

So, Average = $\dfrac{\left[\sum_{h=1}^{h} h^2 \cdot 2^{h-1} \cdot \blacksquare\right] - \left[(2^{h-1}-p).(h-1)^2\right]}{2^h - p - 1 + 1}$ → Total

↳ Not lk on the tree

↳ last level Nodes

c) Is there a restriction on the number of nodes in a full binary tree? What is the number of internal nodes and number of leaves in an n node full binary tree?

The number of leaves in a non-empty full binarytree is one more than the number of internal nodes. (By Mathematical Induction)

Induction step: Given tree T with n internal nodes,

Pick internal node I with two leaf children.

Remove I's children, call resulting tree T',

T has I+1 nodes (leaf)

- we start with 1 leaf and each branching step creates 2 new leaf nodes, and one leaf nodes turns into an internal node (for a net of +1 leaf in the tree). So the tree has 2b+1 nodes, b internal nodes and b+1 leaves where b is the number of branching

$n = 2b+1$

$b = (n-1)/2 \Rightarrow$ internal nodes

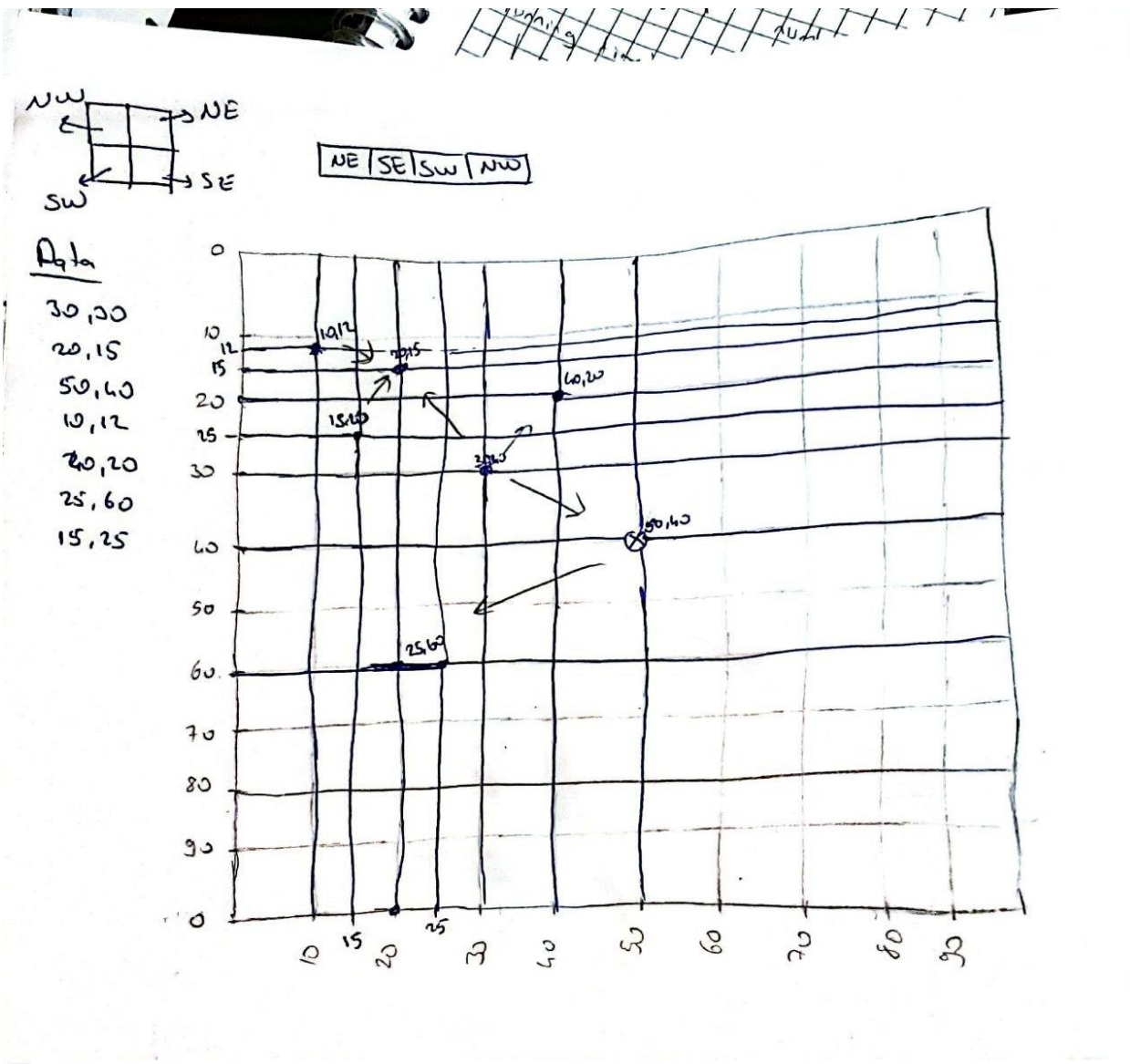
$b+1 = \frac{n-1}{2} + 1 = \frac{n+1}{2} \Rightarrow$ leaves

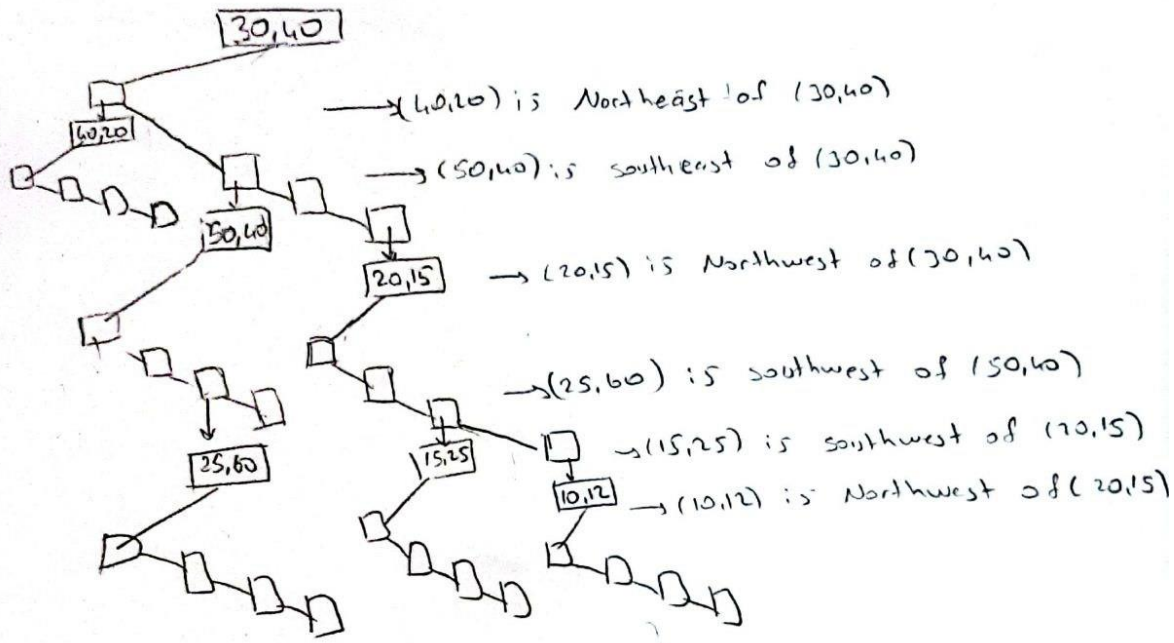
In binary tree each non-leaf node provides two edges. The full tree contains $2*n$ nodes, Each non-leaf node connected to an ancestor consumes one edges, which is tree of all nodes except the root node of tree.

Question 2

Research about the quadtree structure for two-dimensional point data. Consider using the binary tree representation of general trees in our textbook to implement the quadtree structure. Insert the following elements one by one into an empty quadtree. Show the nodes traversed during each insertion and resulting tree after each insertion. Assume that the range is (0, 100) for both dimensions. Note that you are expected to draw the binary tree representation of the quadtree.

(30,30), (20,15), (50,40), (10,12), (40,20), (25,60), (15,25)





Analysis

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```java
public boolean add(Comparable value) {
    int tempSize = size + 1;
    Node currentNode = root;
    if (size == 0 || root == null) {
        root = new Node(value, parent: null);
        currentNode = root;
        //System.out.println(currentNode.value + " is added at root");
    } else {
        boolean left = true;
        for (int i = 0; i < nextDepth() - 1; i++) {

            if ((((Math.pow(2, nextDepth() + 1 - i) - 1)) - (tempSize)) >= Math
                    .pow(2, nextDepth() - i - 1)) {
                currentNode = currentNode.leftChild;
                left = true;
                //  System.out.println("moved left");
            } else {
                currentNode = currentNode.rightChild;
                left = false;
                //  System.out.println("moved right");
            }

            if (left) {
                tempSize = (int) (tempSize - Math.pow(2, nextDepth() - i
                        - 1));
            } else {
                tempSize = (int) (tempSize - Math.pow(2, nextDepth() - i));
            }
        }

        if (currentNode.leftChild == null) {
            currentNode.leftChild = new Node(value, currentNode);
            //System.out.println(currentNode.leftChild.value
            //      + " added to left. parent is " + currentNode.value);
            siftUp(currentNode.leftChild, value: "add");

        } else if (currentNode.rightChild == null) {
            currentNode.rightChild = new Node(value, currentNode);

            //  System.out.println(currentNode.rightChild.value
            //      + " added to right. parent is " + currentNode.value);
```

$\Theta(n)$

$\Theta(1)$

$\Theta(n)$

$\Theta(1)$

$\Theta(1)$

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```java
@Override
public boolean remove(E target) {
    E removedValue = root.value;

    Node currentNode = root;
    int tempSize = size;
    boolean left = true;

    if (size <= 1) {

        root = null;
        if (size > 0) {
            size--;
        }
        return true;
    } else if (size == 2) {
        root.leftChild.parent = null;
        root = root.leftChild;
        size--;
        return true;
    }
    for (int i = 0; i < depth(); i++) {

        if ((((Math.pow(2, depth() + 1 - i) - 1)) - (tempSize)) >= Math
                .pow(2, depth() - i - 1)) {

            currentNode = currentNode.leftChild;

            left = true;
            //  System.out.println("moved left");
        } else {

            currentNode = currentNode.rightChild;
            left = false;
            //System.out.println("moved right");
        }

        if (left) {
            tempSize = (int) (tempSize - Math.pow(2, depth() - i - 1));
        } else {
            tempSize = (int) (tempSize - Math.pow(2, depth() - i));
```

$\Theta(n)$

$\Theta(1)$

$\Theta(n)$

$\Theta(1)$

Lightshot
Screenshot is saved to Screenshot_50.png. Click here to open in the folder.

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```java
        } else {
            tempSize = (int) (tempSize - Math.pow(2, depth() - i));
        }

    }
    root.value = currentNode.value;
    if (left && size > 1) {
        currentNode.parent.leftChild = null;
    } else if (!left && size > 2) {
        currentNode.parent.rightChild = null;
    }
    //  System.out.println("root is now " + root.value);
    siftDown(root);

    if (size > 0) {

        size--;
    }

    return false;
}


/*
 * here lies the array methods of the heap.
 */
public void fromArray(E[] array) {
    E[] valueArray = array;
    for (int i = 0; i < valueArray.length; i++) {
        add(valueArray[i]);
    }

}
```

$\Theta(1)$

$\Theta(n)$

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```
1 usage
public E[] toArray(E[] array) {
    int counter = 0;
    Node tempNode;
    E[] valuesArray = (E[]) java.lang.reflect.Array.newInstance(array.getClass().getComponentType(), size);
    ArrayList<Node> queue = new ArrayList<~>();
    queue.add(root);

    while (!queue.isEmpty()) {
        tempNode = (Node) queue.remove( index: 0);
        valuesArray[counter] = tempNode.value;
        if (tempNode.leftChild != null) {
            queue.add(tempNode.leftChild);
        }
        if (tempNode.rightChild != null) {
            queue.add(tempNode.rightChild);
        }
        counter++;
    }

    return valuesArray;
}


/*
 * These are methods used to shift nodes around.
 */
2 usages
private void siftUp(Node node, String value) {
    Node n = node;
    if (n.parent == null) {
        return;
    }

    for (int i = 0; i < nextDepth(); i++) {

        if (n.value.compareTo(n.parent.value) > 0) {
            swap(n, n.parent);
            n = n.parent;
        } else {
```

$O(1)$

$O(n)$

$O(n)$

$O(1)$

$O(n)$

Services | Build

Lightshot
Screenshot is saved to Screenshot_52.png. Click here to open in the folder.

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```java
     */
1 usage
    public Node( Node<E> left , E data , Node<E> right ) {
        this.left = left;
        this.data = data;
        this.right = right;
    }

    /** Construct a node with given data and no children.
     @param data The data to store in this node
     */
1 usage
    public Node( E data ) { this( left: null , data , right: null ); }

    /** Return a string representation of the node.
     @return A string representation of the data fields
     */
    @Override
    public String toString() { return data.toString(); }
}

/** The root of the binary tree */
21 usages
protected Node<E> root;

/**
 * No parameter constructor.
 */
1 usage
protected BinaryTree() { root = null; }

/**
 *  Construct a node with given node as root.
 * @param root node that will be root of BinaryTree.
 */
2 usages
protected BinaryTree(Node<E> root ) { this.root = root; }

/**
 * Constructs a new binary tree with data in its root leftTree
```

$O(1)$

$O(1)$

$O(1)$

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```java
 * Return the left subtree.
 */
1 usage
public void getLeftSubtree() {

    if( root != null && root.left != null ) {
        new BinaryTree<>(root.left);
    }
    else {
        return;
    }

}

/** Return the right subtree.
 @return The right subtree or null if either the root or
 the right subtree is null
 */
1 usage
public BinaryTree<E> getRightSubtree() {

    if( root != null && root.right != null ) {
        return new BinaryTree<>( root.right );
    }
    else {
        return null;
    }

}

/**
 * Return the data of the root node.
 * @return data which is hold in tree
 */
public E getData() {
    if( root != null ) {
        return root.data;
    }

    System.out.println("Error : getData() root is null");
    return null;
}
```

$O(1)$

$O(1)$

$O(1)$

Services | Build

Lightshot
Screenshot is saved to Screenshot_54.png. Click here open in the folder

package | q3\SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java | q4\SearchTree.java | BinaryHeap.java

```java
        if( root != null ) {
            return ( root.left == null && root.right == null );
        }

        System.out.println("Error : root null in isLeaf ! ");
        return false;
    }

    /** Converts a sub-tree to a string.
     Performs a preorder traversal.
     @param node The local root
     @param sb The StringBuilder to save the output
     */
    3 usages
    protected void preOrderTraverse( Node<E> node , StringBuilder sb ) {

        if( node != null ) {
            sb.append( node.toString() ).append(" ");
            preOrderTraverse( node.left , sb );
            preOrderTraverse( node.right , sb );
        }
    }


    /**
     * Return prefix constructed string representation of the tree.
     * @return String representation of current tree.
     */
    @Override
    public String toString() {

        StringBuilder sb = new StringBuilder();
        preOrderTraverse( root , sb );
        return sb.toString();

    }
}
```

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

Services | Build

Lightshot
Screenshot is saved to Screenshot_55.png. Click here to

package | SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java

```java
package q4;

public class BinarySearchTree<E extends Comparable<E>> extends BinaryTree<E> implements SearchTree<E>{

    58 usages
    E[] tree;
    1 usage
    public BinarySearchTree(int size) {
        tree = (E[]) new Comparable[size];
    }

    @Override
    public boolean remove(E target) { return false; }

    @Override
    public boolean add(Comparable d){
        int i = 0; //Counter to traverse the array.
        while (i < tree.length && tree[i] != null) { //End of Array or null location found.
            if (d.compareTo(tree[i]) < 0) //Incoming data is smaller than data present in the array.
                i = (2 * i) + 1; //Move to left side of the tree.
            else
                i = (2 * i) + 2; //Move to right side of the tree
        }
        tree[i] = (E) d;
        return false;
    }

    @Override
    public boolean contains(E target) { return false; }
```

$\Theta(1)$

$\Theta(\log n)$

$\Theta(\log n)$

$T_b \rightarrow \Theta(1)$

$T_w \rightarrow \Theta(\log n)$

$T_A = \Theta(\log n)$

package | SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java

1 usage

```java
public E find(E d) {
        int i = 0; //Counter to traverse the array.
        boolean found = false;
        while (i < tree.length) { //End of Array or 'd' found.
            if (d.compareTo(tree[i]) < 0) //Data to be searched is smaller than data present in the array.
                i = (2 * i) + 1; //Move to left side of the tree.
            else if (d.compareTo(tree[i]) > 0)
                i = (2 * i) + 2; //Move to right side of the tree
            else {//'d' is same as tree[i] i.e. d found.
                found = true;
                break;
            }
        }
        if (found) //'d' found.
            System.out.println(d + " found at " + i + ".");
        else //'d' not found.
            System.out.println(d + " not found.");
    return d;
}
```

$O(n)$

2 usages

```java
public int[] findDel(E d) {
    int[] indices = new int[2];
    boolean found = false;
    int c = 0; //Child
    int p = 0; //Parent
    while (c < tree.length && tree[c] != null) { //End of Array or 'd' found.
        if (d.compareTo(tree[c]) < 0) { //Data to be searched is smaller than data present in the array.
            p = c; //Parent index
            c = (2 * c) + 1; //Move to left side of the tree.
        }
        else if (d.compareTo(tree[c]) > 0) {
            p = c; //Parent index
            c = (2 * c) + 2; //Move to right side of the tree
        }
        else { //'d' is same as tree[i] i.e. d found.
            found = true;
            break;
        }
    }
```

$O(\log n)$

package | SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java

```java
}

1 usage
public E delete(E d) {

    if (findDel(d) == null) {
        System.out.printf("\nKey not found, can't delete.\nThe list remains unchanged\n");
        return null;
    }
    else {
        int[] ref = findDel(d);
        int c = ref[1];
        int right = (2 * c) + 2;
        int left = (2 * c) + 1;
        System.out.printf("\nDeleting " + tree[c] + "\n");
        if ((right > tree.length || left > tree.length) || (tree[left] == null && tree[right] == null)) {//No child case
            delNoChild(c);
        } else if ((tree[right] == null && tree[left] != null) || (tree[right] != null && tree[left] == null)) { //One child case
            delOneChild(c, left, right);
        } else { //Two Child Case
            int temp = right; //Move to the right of the node to be deleted
            int temp2 = 0;
            while (tree[temp] != null) { //Until the right is empty
                temp2 = temp; //Because temp will be null at the end of the loop, a variable to store the previous value
                temp = (2 * temp) + 1; //Move to the left
            }
            tree[c] = tree[temp2]; //The value at temp2 copied at the node to be deleted
            c = temp2; //Child updated to carry out deletion of no child/one child at temp2
            //right and left updated to carry out deletion at temp2
            right = (2 * c) + 2;
            left = (2 * c) + 1;
            if ((right > tree.length || left > tree.length) || (tree[left] == null && tree[right] == null)) { //No child case
                delNoChild(c);
            } else if ((tree[right] == null && tree[left] != null) || (tree[right] != null && tree[left] == null)) { //One child case
                delOneChild(c, left, right);
            }
        }
    }
    return d;
}
```

$O(\log n)$

$O(1)$

$O(1)$

$O(\log n)$

$O(1)$

Lightshot

package | SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java

```java
     * as its left subtree and rightTree as its right subtree.
     * @param data data
     * @param left left reference
     * @param right right reference
     */
@    protected BinaryTree( E data , BinaryTree<E> left , BinaryTree<E> right ) {

         root = new Node<>( data );

         if( left.root != null ) {
             root.left = left.root;
         }

         if( right.root != null ) {
             root.right = right.root;
         }

     }

     /** Return the left subtree.
      @return The left subtree or null if either the root or
      the left subtree is null
      */
     1 usage
     public BinaryTree<E> getLeftSubtree() {

         if( root != null && root.left != null ) {
             return new BinaryTree<>( root.left );
         }
         else {
             return null;
         }

     }

     /** Return the right subtree.
      @return The right subtree or null if either the root or
      the right subtree is null
      */
```

O(1)

O(1)

package | SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java

```java
 @return The right subtree or null if either the root or
 the right subtree is null
 */
1 usage
public BinaryTree<E> getRightSubtree() {

    if( root != null && root.right != null ) {
        return new BinaryTree<>( root.right );
    }
    else {
        return null;
    }

}

/**
 * Return the data of the root node.
 * @return data which is hold in tree
 */
public E getData() {
    if( root != null ) {
        return root.data;
    }

    System.out.println("Error : getData() root is null");
    return null;
}

/** Determine whether this tree is a leaf.
 @return true if the root has no children
 */
1 usage
public boolean isLeaf() {

    if( root != null ) {
        return ( root.left == null && root.right == null );
    }

    System.out.println("Error : root null in isLeaf ! ");
    return false;
}
```

$O(1)$

$O(1)$

$O(1)$

Services | Build

Lightshot
Screenshot is saved to Screenshot_47.png. Click he
open in the folder.

package | SearchTree.java | q3\BinaryTree.java | Driver.java | BinarySearchTree.java | q4\BinaryTree.java

```java
        if( root != null ) {
            return ( root.left == null && root.right == null );
        }

        System.out.println("Error : root null in isLeaf ! ");
        return false;
    }

    /** Converts a sub-tree to a string.
     Performs a preorder traversal.
     @param node The local root
     @param sb The StringBuilder to save the output
     */
    3 usages
    protected void preOrderTraverse( Node<E> node , StringBuilder sb ) {

        if( node != null ) {
            sb.append( node.toString() ).append(" ");
            preOrderTraverse( node.left , sb );
            preOrderTraverse( node.right , sb );
        }
    }


    /**
     * Return prefix constructed string representation of the tree.
     * @return String representation of current tree.
     */
    @Override
    public String toString() {

        StringBuilder sb = new StringBuilder();
        preOrderTraverse( root , sb );
        return sb.toString();

    }
}
```

O(1)

O(1)

O(1)

5. TEST CASES

Heap constructor worked

12 added to heap

44 added to heap

12 removed from heap

Binary Search Tree constructor worked with size 6

15 added to BST

13 added to BST

13 deleted

15 found at 0

Error: root null in is Leaf !

6. RUNNING AND RESULTS