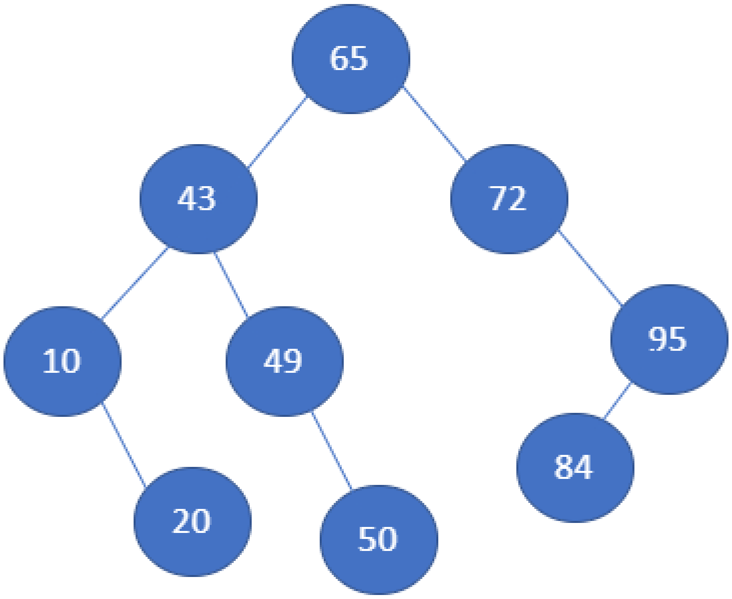
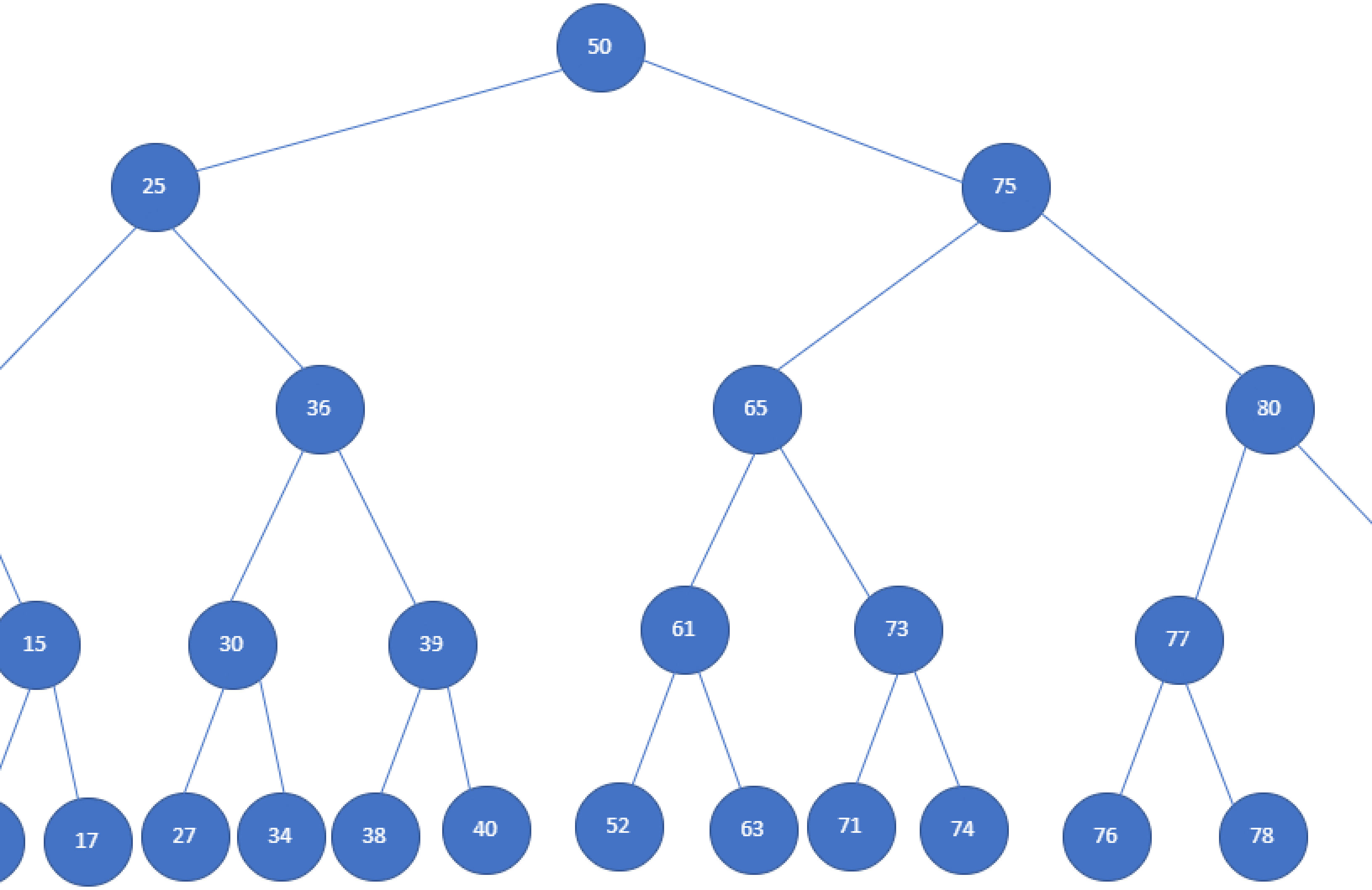
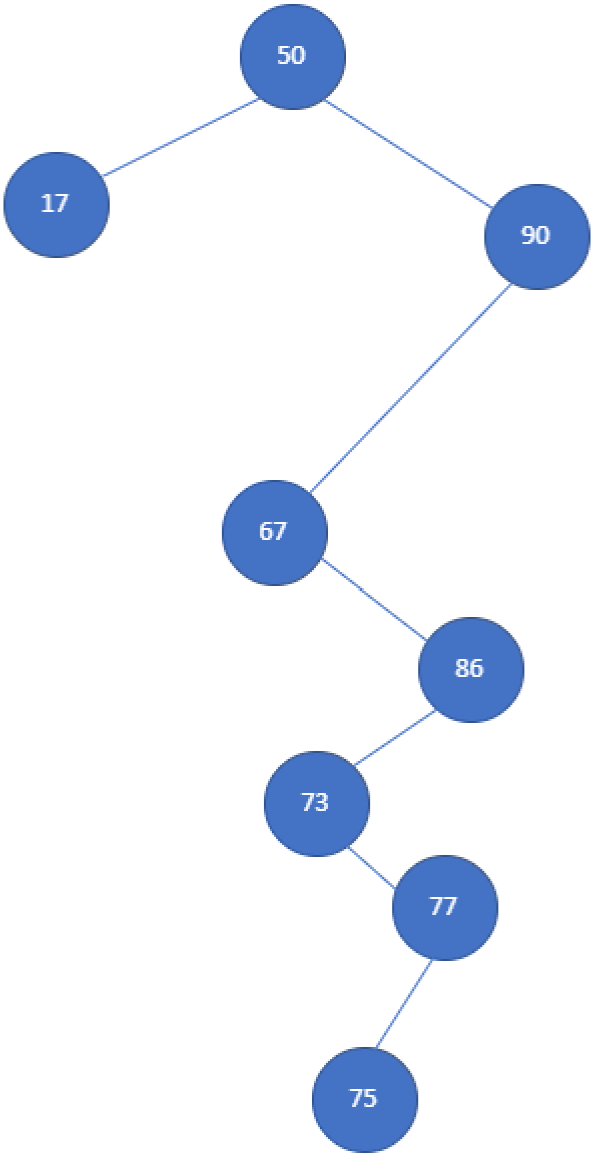
/*******************************
Name: Noah Buchanan
Username: ua203
Problem Set: PS9
Due Date: July 29, 2020
*******************************/
Data Structures

July 29, 2020




50
25
75
36
65
80
15
30
39
61
73
77
17
27
34
38
40
52
63
71
74
76
78


50
17
90
67
86
73
77
75

# 1 UABinarySearchTree

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * Implementation of Binary Search Tree
 * @author noah
 *
 */

public class UABinarySearchTree {

        UANode root;
        UABinarySearchTree Tree;
        int size;

        /**
         * Loads nodes into Binary Search Tree, this method runs in O(n) time
         * @param file File being read from into Binary Tree
         * @throws IOException
         */
        public void load(String file) throws IOException {

                BufferedReader br = new BufferedReader(new FileReader(file));

                String line = "";

                while((line = br.readLine()) != null) {

                        String[] x = line.split(",");

                        treeInsert(Tree, new Node(x));

                }
                br.close();
        }
        /**
         * Finds the minimum key from given node, this method runs in O(n) time
         * @param x node given
         * @return The minimum key from given node x
         */
        public UANode treeMinimum(UANode x) {
                while(x.getLeft() != null) {
```

```
                x = x.getLeft();
        }
        return x;
}
/**
 * Finds the maximum key from given node, this method runs in O(n) time
 * @param x node given
 * @return The maximum key from given node x
 */
public UANode treeMaximum(UANode x) {
        while(x.getRight() != null) {
                x = x.getRight();
        }
        return x;
}
/**
 * Finds the direct successor of given node x, this method runs in O(lgn)?
 * @param x node given
 * @return Direct successor of x
 */
public UANode treeSuccessor(UANode x) {
        if(x.getRight() != null) {
                return treeMinimum(x.getRight());
        }

        UANode y = x.getParent();
        while(y != null && x == y.getRight()) {
                x = y;
                y = y.getParent();
        }
        return y;
}
/**
 * Finds the direct predecessor of given node x, this method runs in O(lgn)?
 * @param x node given
 * @return Direct predecessor of x
 */
public UANode treePredecessor(UANode x) {
        if(x.getLeft() != null) {
                return treeMaximum(x.getLeft());
        }

        UANode y = x.getParent();
        while(y != null && x == y.getLeft()) {
                x = y;
                y = y.getParent();
        }
        return y;
}
/**
 * Iteratively searches the tree for a node containing the key given in the paramete
 * @param x Node to start search from
 * @param key Key we are looking for
 * @return Node containing key that we were looking for
 */
public UANode treeSearch(UANode x, int key) {
        while(x != null && key != x.getKey()) {
                if(key < x.getKey()) {
                        x = x.getLeft();
                } else {
                        x = x.getRight();
                }
        }
        return x;
}
/**
 * Inserts a value into the Binary Search Tree, this method runs in O(lgn)
 * @param T
 * @param z
 */
public void treeInsert(UABinarySearchTree T, UANode z) {

        UANode y = null;
        UANode x = root;
        while(x != null) {

                y = x;

                if(z.getKey() < x.getKey()) {

                        x = x.getLeft();
                } else {

                        x = x.getRight();
                }
        }
        z.setParent(y);
        if(y == null) {

                root = z;
        } else if(z.getKey() < y.getKey()) {
                y.setLeft(z);
        } else {

                y.setRight(z);
        }
        size++;
}
/**
 * Transplants two nodes getting rid of pointers to param u, this method runs in O(1
 * @param T The tree we want to transplant in
 * @param u Node to be transplanted
 * @param v Node to be transplanted
 */
public void transplant(UABinarySearchTree T, UANode u, UANode v) {

        if(u.getParent() == null) {

                T.root = v;
        } else if(u == u.getParent().getLeft()) {

                u.getParent().setLeft(v);
        } else {

                u.getParent().setRight(v);
        }

        if(v != null) {

                v.setParent(u.getParent());
        }
}
/**
 * Deletes a node from the tree and correctly fixes location after a node is removed
 * @param T Tree we are deleting from
 * @param z Node to be deleted
 */
public void treeDelete(UABinarySearchTree T, UANode z) {
        if(z.getLeft() == null) {
                transplant(T,z,z.getRight());
        } else if(z.getRight() == null) {
                transplant(T,z,z.getLeft());
        } else {
                UANode y = treeMinimum(z.getRight());
                if(y.getParent() != z) {
                        transplant(T,y,y.getRight());
                        y.setRight(z.getRight());
                        y.getRight().setParent(y);
                }

                transplant(T,z,y);
                y.setLeft(z.getLeft());
                y.getLeft().setParent(y);
        }
        size--;
}
/**
 * Prints out the Binary Search Tree in reverse sorted order, this method runs in O(
 * @param x Node to start function from
 */
public void print(UANode x) {
        if(x != null) {
                print(x.getRight());
                System.out.println(x.getKey());
                print(x.getLeft());
        }
}
/**
 * Returns the amount of nodes currently in the Binary Search Tree, this method runs
 * @return size of Binary Search Tree
 */
public int size() {
        return this.size;
}

/**
 * Node class for Binary Search Tree that implements UANode interface
 * @author noah
 *
 */
public static class Node implements UANode{

        /**
         * Constructor to make load method more streamline when inserting from file
         * @param x
         */
        public Node(String[] x) {
                studentID = Integer.parseInt(x[0]);
                studentName = x[1];
                studentEmail = x[2];
        }

        private UANode left;
        private UANode right;
        private UANode parent;
        private int studentID;
        private String studentName;
        private String studentEmail;

        public int getKey() {
                return studentID;
        }

        public String getName() {
                return studentName;
        }

        public String getEmail() {
                return studentEmail;
        }

        public UANode getLeft() {
                return left;
        }

        public UANode getRight() {
                return right;
        }

        public UANode getParent() {
                return parent;
        }

        public void setKey(int key) {
                this.studentID = key;
        }

        public void setName(String name) {
                this.studentName = name;
        }

        public void setEmail(String email) {
                this.studentEmail = email;
        }

        public void setLeft(UANode left) {
                this.left = left;
        }
```

```
                public void setRight(UANode right) {
                        this.right = right;
                }

                public void setParent(UANode parent) {
                        this.parent = parent;
                }
                /**
                 * Overridden toString for Node fields
                 */
                public String toString() {
                        return "Student: | " + studentID + " | " + studentName + " | " + stu
                }


        }

        /**
         * Class to override print method of parent class
         * @author noah
         *
         */
        public static class UABinarySearchTree2 extends UABinarySearchTree{

                /**
                 * Prints the Binary Search Tree in sorted order, this method runs in O(n)
                 */
                public void print(UANode x) {
                        if(x != null) {
                                print(x.getLeft());
                                System.out.println(x.getKey());
                                print(x.getRight());
                        }
                }

        }


}
```