Detailed System Requirements

1.Functional Requirments

PART 1:

- Insert() : insert operation

- Delete(): delete operation

-descending iterator(): return a iterator with descending order.

-iterator(): returns iterator

-headSet(): used to return elements are less than (or equal to, if inclusive is true) toElement.

-tailSet(): (): used to return elements are greater than (or equal to, if inclusive is true) fromElement.

PART 2:

-is_AVL():This function check the tree is AVL tree or not.

-is_Red():This function check the tree is Red-Black tree or not.

PART 3:

-Testing some data structers insert operation and compare them.

2.Unfunctional Requirments

-Hardware should be able to run at least JAVA SE13.

Problem Solutions Approach

İn the first part,

navigableset interface and my Iterator interface was implemented.Insert, delete and descendingiterator methods were written. Skip-List insert method insert the element with creating random height,descendingIterator method return a iterator class from created in skipList class and this iterator iterate the Skip-List wit descending order.Also insert, iterator, headSet and tailSet method were written for AVL Tree with using , navigableset interface and my Iterator interface. İnsert method a Avl Tree is find a node like a ordinary binary search tree but after then arrange the AVL Tree Balance . iterator method is as like usual iterator method. headSet method return the smaller elements for the given element in the AVL, if boolean experssion is true included the considerations or not.tailSet method return the bigger elements for the given element in the AVL, if boolean experssion is true included the considerations or not.

İn the second part,

For determine a Binary Search Tree is a AVL Tree : İf the Binary Search Tree’s any node’s left subTree and right subTree height difference smaller or equal 1 then it can be a AVL Tree. Or not.

For determine a Binary Search Tree is a Red-Black Tree : every path through a leaf has the same number of black nodes, and at least every second node on the path will be black, since a red node can't have a red child. Therefore, the one with the most red nodes's the longest path to a leaf in the tree is at most twice as long as the shortest path to a leaf.

İn the third part,

the insert functions of some data structures were tested incrementally. Also, which one was faster was compared.

Test Cases

Part1

A.Skip-List

->Skip-List insert method

```
////////////////////////////////////////////////// PART 1 ////////////////////////////////////////////////////////
        System.out.println();
        System.out.println("-------------------- Part 1 --------------------");
        System.out.println();

        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("----- Skip-List Testing -----");
        System.out.println();

        SkipList<Integer> skip_List = new SkipList<Integer>();

        skip_List.insert(10);
        skip_List.insert(18);
        skip_List.insert(35);
        skip_List.insert(77);
        skip_List.insert(115);
        skip_List.insert(50);
        skip_List.insert(30);

//////////////////////////////////////////////////////////////////////////////////////////////////////////////// insert Method
        System.out.println();
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The Skip-List :");
        System.out.println();
        System.out.println(" 10, 18, 35, 77, 115, 50 and 30 inserting to Skip-List ");
        System.out.println();
        skip_List.levelPrint();
```

->Skip-List descendingIterator

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////// DescendingIterator Method
        System.out.println();
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println("------------------------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The Skip-List DescendingIterator :");
        System.out.println();

        MyIterator itr=skip_List.descendingIterator();

        while(itr.hasNext()){
            try{
                System.out.println(itr.next());
            }
            catch(NoSuchElementException e){
                System.out.println(e);
            }
        }
```

->Skip-List delete method

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// delete Method
            System.out.println();
            System.out.println("-----------------------------------------------------------------------------------------------------------");
            System.out.println("-----------------------------------------------------------------------------------------------------------");
            System.out.println("-----------------------------------------------------------------------------------------------------------");
            System.out.println();
            System.out.println("--------------------> After the deleting The Skip-List :");
            System.out.println();
            System.out.println(" 35, 10 and 115 deleting to Skip-List ");
            System.out.println();

            skip_List.delete(10);
            skip_List.delete(115);
            skip_List.delete(35);

            skip_List.levelPrint();
```

B.AVL Tree

->AVL Tree insert method

```
            System.out.println();
            System.out.println("***********************************************************************************************************");
            System.out.println("***********************************************************************************************************");
            System.out.println("***********************************************************************************************************");
            System.out.println();
            System.out.println("----- AVL Tree Testing -----");
            System.out.println();
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// AVL Tree insert Method

            AVLTree<Integer> avl=new AVLTree<Integer>();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// insert Method
            System.out.println();
            System.out.println("-----------------------------------------------------------------------------------------------------------");
            System.out.println("-----------------------------------------------------------------------------------------------------------");
            System.out.println("-----------------------------------------------------------------------------------------------------------");
            System.out.println();
            System.out.println("--------------------> After the inserting The AVL Tree :");
            System.out.println();
            System.out.println(" 10, 15, 8, 16, 19 and 12 inserting to AVL Tree ");
            System.out.println();

            avl.insert(10);
            avl.insert(15);
            avl.insert(8);
            avl.insert(16);
            avl.insert(19);
            avl.insert(12);

            System.out.println("--------------------> AVL Tree PreORder Wiew ");
            avl.preOrder();
```

->AVL Tree iterator method

```
//////////////////////////////////////////////////////////////////////////////////////////// Iterator Method

        System.out.println();
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The AVL Tree Iterator :");
        System.out.println();

        MyIterator itr2=avl.iterator();

        while(itr2.hasNext()){
            try{
                System.out.println(itr2.next());
            }
            catch(NoSuchElementException e){
                System.out.println(e);
            }
        }
//////////////////////////////////////////////////////////////////////////////////////////// tailSet(FALSE) Method
```

->AVL Tree tailSet method

```
//////////////////////////////////////////////////////////////////////////////////////////// tailSet(FALSE) Method

        System.out.println();
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The AVL Tree tailSet Method :");
        System.out.println();
        System.out.println("-------> bigger than 10(10 not included) in the AVL Tree ");

        NavigableSet<Integer> avl2=avl.tailSet(10, false);

        MyIterator itr3= ((AVLTree)avl2).iterator();
        while(itr3.hasNext()){
            try{

                System.out.println(itr3.next());
            }
            catch(NoSuchElementException e){
                System.out.println(e);
            }
        }
```

```
//////////////////////////////////////////////////////////////////////////////////////////// tailSet(TRUE) Method

        System.out.println();
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The AVL Tree tailSet Method :");
        System.out.println();
        System.out.println("-------> bigger than 10(10 included) in the AVL Tree ");

        NavigableSet<Integer> avl4=avl.tailSet(10, true);

        MyIterator itr5= ((AVLTree)avl4).iterator();
        while(itr5.hasNext()){
            try{
                System.out.println(itr5.next());
            }
            catch(NoSuchElementException e){
                System.out.println(e);
            }
        }
```

->AVL Tree headSet method

```
////////////////////////////////////////////////////////////////////////////////////////////////////// headSet(FALSE) Method
        System.out.println();
        System.out.println("-----------------------------------------------------------------------------------------------------------");
        System.out.println("-----------------------------------------------------------------------------------------------------------");
        System.out.println("-----------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The AVL Tree headSet Method :");
        System.out.println();
        System.out.println("-------> smaller than 12(12 not included) in the AVL Tree ");
        System.out.println();

        NavigableSet<Integer> avl3=avl.headSet(12, false);

        MyIterator itr4= ((AVLTree)avl3).iterator();
        while(itr4.hasNext()){
            try{

                System.out.println(itr4.next());
            }
            catch(NoSuchElementException e){
                System.out.println(e);
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////// headSet(TRUE) Method

        System.out.println();
        System.out.println("-----------------------------------------------------------------------------------------------------------");
        System.out.println("-----------------------------------------------------------------------------------------------------------");
        System.out.println("-----------------------------------------------------------------------------------------------------------");
        System.out.println();
        System.out.println("--------------------> After the inserting The AVL Tree headSet Method :");
        System.out.println();
        System.out.println("-------> smaller than 12(12 included) in the AVL Tree ");
        System.out.println();

        NavigableSet<Integer> avl5=avl.headSet(12, true);

        MyIterator itr6= ((AVLTree)avl5).iterator();
        while(itr6.hasNext()){
            try{

                System.out.println(itr6.next());
            }
            catch(NoSuchElementException e){
                System.out.println(e);
            }
        }
    }
```

Part2

Binary Search Tree is an AVL tree or Red-Black Tree

```
        BinarySearchTree tree = new BinarySearchTree();

        System.out.println();
        System.out.println("////////////////////////////////////////////////////////////////////////////////////////////////");
        System.out.println("////////////////////////////////////////////////////////////////////////////////////////////////");
        System.out.println("////////////////////////////////////////////////////////////////////////////////////////////////");
        System.out.println();
        System.out.println("-------------------- Part 2 --------------------");
        System.out.println();

        System.out.println("----- Checking  Binary Search Tree is be a AVL Tree or Red-Black tree -----");

        System.out.println();
        System.out.println("-------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------");


////////////////////////////////////////// Inserting some numbers ////////////////////////////////////////////////////////
            tree.add(25);
            tree.add(18);
            tree.add(30);
            tree.add(10);
            tree.add(20);
            tree.add(40);
            tree.add(35);
            tree.add(28);

            System.out.println();
            System.out.println("----- The Binary Search Tree -----");
            System.out.println();

            System.out.println("--------------------> Binary Search Tree PreORder Wiew ");
            tree.preOrder();
            System.out.println();
            System.out.println();

            System.out.println("            25 ");
            System.out.println("        18      30 ");
            System.out.println("    10    20   28   40");
            System.out.println("                      50");
            System.out.println();

            System.out.println("This Binary Search Tree is an AVL Tree : "+tree.isAvl());
            System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree.is_Red());

            System.out.println();
            System.out.println("----- The Binary Search Tree is  a Red-Black Tree and an AVL Tree -----");
```

```java
/////////////////////////////////////// Deleting some numbers ////////////////////////////////////////////////////////

                tree.remove(28);

                System.out.println();
                System.out.println("-----------------------------------------------------------------------");
                System.out.println("-----------------------------------------------------------------------");
                System.out.println("-----------------------------------------------------------------------");
                System.out.println();

                System.out.println("----- Deleting 28 number from Binary Search Tree -----");
                System.out.println();
                System.out.println("----- After Deleting Operations -----");
                System.out.println();

                System.out.println("----- The Binary Search Tree -----");
                System.out.println();

                System.out.println("-------------------> Binary Search Tree PreORder Wiew ");
                tree.preOrder();
                System.out.println();
                System.out.println();

                System.out.println("              25 ");
                System.out.println("          18      30 ");
                System.out.println("       10    20       40");
                System.out.println("                        50");
                System.out.println();

                System.out.println("This Binary Search Tree is an AVL Tree : "+tree.isAvl());
                System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree.is_Red());

                System.out.println();
                System.out.println("-----After the deleting The Binary Search Tree is not a Red-Black Tree or an AVL Tree -----");

                System.out.println();
                System.out.println("-----------------------------------------------------------------------");
                System.out.println("-----------------------------------------------------------------------");
                System.out.println("-----------------------------------------------------------------------");
                System.out.println();

                BinarySearchTree tree2 = new BinarySearchTree();
                tree2.add(1);
                tree2.add(2);
                tree2.add(3);
                tree2.add(4);

                System.out.println();
                System.out.println("----- Other testing operation -----");
                System.out.println();
                System.out.println("----- The Binary Search Tree -----");
                System.out.println();

                System.out.println("-------------------> Binary Search Tree PreORder Wiew ");
                tree2.preOrder();
                System.out.println();
                System.out.println();

                System.out.println("            1 ");
                System.out.println("               2 ");
                System.out.println("                 3");
                System.out.println("                   4");
                System.out.println();

                System.out.println("This Binary Search Tree is an AVL Tree : "+tree2.isAvl());
                System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree2.is_Red());

                System.out.println();
                System.out.println("----- The Binary Search Tree is not a Red-Black Tree or an AVL Tree -----");
```

```
        System.out.println();
        System.out.println("-------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------");
        System.out.println("-------------------------------------------------------------------------");
        System.out.println();

        BinarySearchTree tree3 = new BinarySearchTree();
        tree3.add(11);
        tree3.add(3);
        tree3.add(115);
        tree3.add(60);
        tree3.add(150);
        tree3.add(35);

        System.out.println();
        System.out.println("----- Other testing operation -----");
        System.out.println();
        System.out.println("----- The Binary Search Tree -----");
        System.out.println();

        System.out.println("--------------------> Binary Search Tree PreORder Wiew ");
        tree3.preOrder();
        System.out.println();
        System.out.println();

        System.out.println("              11 ");
        System.out.println("           3      115 ");
        System.out.println("                60    150");
        System.out.println("              35       ");
        System.out.println();

        System.out.println("This Binary Search Tree is an AVL Tree : "+tree3.isAvl());
        System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree3.is_Red());

        System.out.println();
        System.out.println("----- The Binary Search Tree is a Red-Black Tree but not an AVL Tree -----");
    }
```

Part3

Inserting 100 extra random numbers to 10.000 size of Data Structures

```
        System.out.println();
        System.out.println("////////////////////////////////////////////////////////////////////////////////////////");
        System.out.println("////////////////////////////////////////////////////////////////////////////////////////");
        System.out.println("////////////////////////////////////////////////////////////////////////////////////////");
        System.out.println();
        System.out.println("-------------------- Part 3 --------------------");
        System.out.println();


        System.out.println("-------------------------------------------------------------------------");
        System.out.println(" !!!!! It will take about 2 minutes !!!!!");
        System.out.println("-------------------------------------------------------------------------");

        long startTime;
        long endTime;
        long estimatedTime;

        long bst_time_sum=0;
        long rbt_time_sum=0;
        long skipList_time_sum=0;
        long bTree_time_sum=0;
        long two_three_Tree_time_sum=0;

        Random randNum = new Random();
        int num;
```

```
        System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 10.000 Size Of Data Structures");
////////////////////////////////////////////////////////////////////////////////////////////// Binary Search Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BinarySearchTree bst= new BinarySearchTree();

            Set set = new LinkedHashSet<>();
            Set set2 = new LinkedHashSet<>();

            while (set.size() < 10000*2) {
                set.add(randNum.nextInt(10000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                bst.add((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                bst.add((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;


            bst_time_sum=bst_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Binary Search Tree :"+(bst_time_sum/10));
```

```
////////////////////////////////////////////////////////////////////////////////////////////// Red-Black Tree İnsert Time Operations

        for(int i=0;i<10;i++){
            RedBlackTree rbt = new RedBlackTree();

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 10000*2) {
                set.add(randNum.nextInt(10000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                rbt.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                rbt.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            rbt_time_sum=rbt_time_sum+(estimatedTime);
        }
        System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////// 2-3 Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            TwoThreeTree two_three_tree = new TwoThreeTree(3);

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 10000*2) {
                set.add(randNum.nextInt(10000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                two_three_tree.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                two_three_tree.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
////////////////////////////////////////////////////////////////////////////////////////////////////// B Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BTree btree = new BTree(2);

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 10000*2) {
                set.add(randNum.nextInt(10000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                btree.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                btree.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            bTree_time_sum=bTree_time_sum+estimatedTime;
        }
        System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
```

```
////////////////////////////////////////////////////////////////////////////////////////// Skip-List Insert Time Operations
        for(int i=0;i<10;i++){
            SkipList<Integer> skip_list = new SkipList<Integer>();

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 10000*2) {
                set.add(randNum.nextInt(10000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                skip_list.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                skip_list.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            skipList_time_sum=skipList_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));
```

Inserting 100 extra random numbers to 20.000 size of Data Structures

```
        System.out.println();
        System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 20.000 Size Of Data Structures");
////////////////////////////////////////////////////////////////////////////////////////// Binary Search Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BinarySearchTree bst= new BinarySearchTree();


            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 20000*2) {
                set.add(randNum.nextInt(20000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                bst.add((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                bst.add((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            bst_time_sum=bst_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Binary Search Tree :"+(bst_time_sum/10));
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////// Red-Black Tree İnsert Time Operations

            for(int i=0;i<10;i++){
                RedBlackTree rbt = new RedBlackTree();

                Set<Integer>set = new LinkedHashSet<Integer>();
                Set<Integer>set2 = new LinkedHashSet<Integer>();

                while (set.size() < 20000*2) {
                    set.add(randNum.nextInt(20000*2)+1);
                }

                Iterator value = set.iterator();
                while (value.hasNext()){
                    rbt.insert((Integer)value.next());
                }

                while (set2.size() < 100*2) {
                    set2.add(randNum.nextInt(100*2)+1);
                }

                Iterator value2 = set2.iterator();

                startTime = System.nanoTime();
                while (value2.hasNext()){
                    rbt.insert((Integer)value2.next());
                }
                endTime=System.nanoTime();
                estimatedTime=endTime-startTime;

                rbt_time_sum=rbt_time_sum+(estimatedTime);
            }
            System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));

//////////////////////////////////////////////////////////////////////////////////////////////////// 2-3 Tree İnsert Time Operations
             for(int i=0;i<10;i++){
                TwoThreeTree two_three_tree = new TwoThreeTree(3);

                Set<Integer>set = new LinkedHashSet<Integer>();
                Set<Integer>set2 = new LinkedHashSet<Integer>();

                while (set.size() < 20000*2) {
                    set.add(randNum.nextInt(20000*2)+1);
                }

                Iterator value = set.iterator();
                while (value.hasNext()){
                    two_three_tree.insert((Integer)value.next());
                }

                while (set2.size() < 100*2) {
                    set2.add(randNum.nextInt(100*2)+1);
                }

                Iterator value2 = set2.iterator();

                startTime = System.nanoTime();
                while (value2.hasNext()){
                    two_three_tree.insert((Integer)value2.next());
                }
                endTime=System.nanoTime();
                estimatedTime=endTime-startTime;

                two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
            }

            System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////// B Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BTree btree = new BTree(2);

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 20000*2) {
                set.add(randNum.nextInt(20000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                btree.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                btree.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            bTree_time_sum=bTree_time_sum+estimatedTime;
        }
        System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////// Skip-List İnsert Time Operations
        for(int i=0;i<10;i++){
            SkipList<Integer> skip_list = new SkipList<Integer>();

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 20000*2) {
                set.add(randNum.nextInt(20000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                skip_list.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                skip_list.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            skipList_time_sum=skipList_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));
```

Inserting 100 extra random numbers to 40.000 size of Data Structures

```
        System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 40.000 Size Of Data Structures");
////////////////////////////////////////////////////////////////////////////////////////////////////// Binary Search Tree İnsert Time Operations
    for(int i=0;i<10;i++){
        BinarySearchTree bst= new BinarySearchTree();


        Set<Integer>set = new LinkedHashSet<Integer>();
        Set<Integer>set2 = new LinkedHashSet<Integer>();

        while (set.size() < 40000*2) {
            set.add(randNum.nextInt(40000*2)+1);
        }

        Iterator value = set.iterator();
        while (value.hasNext()){
            bst.add((Integer)value.next());
        }

        while (set2.size() < 100*2) {
            set2.add(randNum.nextInt(100*2)+1);
        }

        Iterator value2 = set2.iterator();

        startTime = System.nanoTime();
        while (value2.hasNext()){
            bst.add((Integer)value2.next());
        }
        endTime=System.nanoTime();
        estimatedTime=endTime-startTime;

        bst_time_sum=bst_time_sum+estimatedTime;
    }

        System.out.println("Avarage time of Binary Search Tree :"+(bst_time_sum/10));


////////////////////////////////////////////////////////////////////////////////////////////////////// Red-Black Tree İnsert Time Operations

    for(int i=0;i<10;i++){
        RedBlackTree rbt = new RedBlackTree();

        Set<Integer>set = new LinkedHashSet<Integer>();
        Set<Integer>set2 = new LinkedHashSet<Integer>();

        while (set.size() < 40000*2) {
            set.add(randNum.nextInt(40000*2)+1);
        }

        Iterator value = set.iterator();
        while (value.hasNext()){
            rbt.insert((Integer)value.next());
        }

        while (set2.size() < 100*2) {
            set2.add(randNum.nextInt(100*2)+1);
        }

        Iterator value2 = set2.iterator();

        startTime = System.nanoTime();
        while (value2.hasNext()){
            rbt.insert((Integer)value2.next());
        }
        endTime=System.nanoTime();
        estimatedTime=endTime-startTime;

        rbt_time_sum=rbt_time_sum+(estimatedTime);
    }
    System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////// 2-3 Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            TwoThreeTree two_three_tree = new TwoThreeTree(3);

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 40000*2) {
                set.add(randNum.nextInt(40000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                two_three_tree.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                two_three_tree.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
//////////////////////////////////////////////////////////////////////////////////////////////////////////////// B Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BTree btree = new BTree(2);

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 40000*2) {
                set.add(randNum.nextInt(40000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                btree.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                btree.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            bTree_time_sum=bTree_time_sum+estimatedTime;
        }
        System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
```

```
//////////////////////////////////////////////////////////////////////////////////////////// Skip-List İnsert Time Operations
        for(int i=0;i<10;i++){
            SkipList<Integer> skip_list = new SkipList<Integer>();

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 40000*2) {
                set.add(randNum.nextInt(40000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                skip_list.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                skip_list.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            skipList_time_sum=skipList_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));
```

Inserting 100 extra random numbers to 80.000 size of Data Structures

```
        System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 80.000 Size Of Data Structures");
//////////////////////////////////////////////////////////////////////////////////////////// Binary Search Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BinarySearchTree bst= new BinarySearchTree();

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 80000*2) {
                set.add(randNum.nextInt(80000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                bst.add((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                bst.add((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            bst_time_sum=bst_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Binary Search Tree :"+(bst_time_sum/10));
```

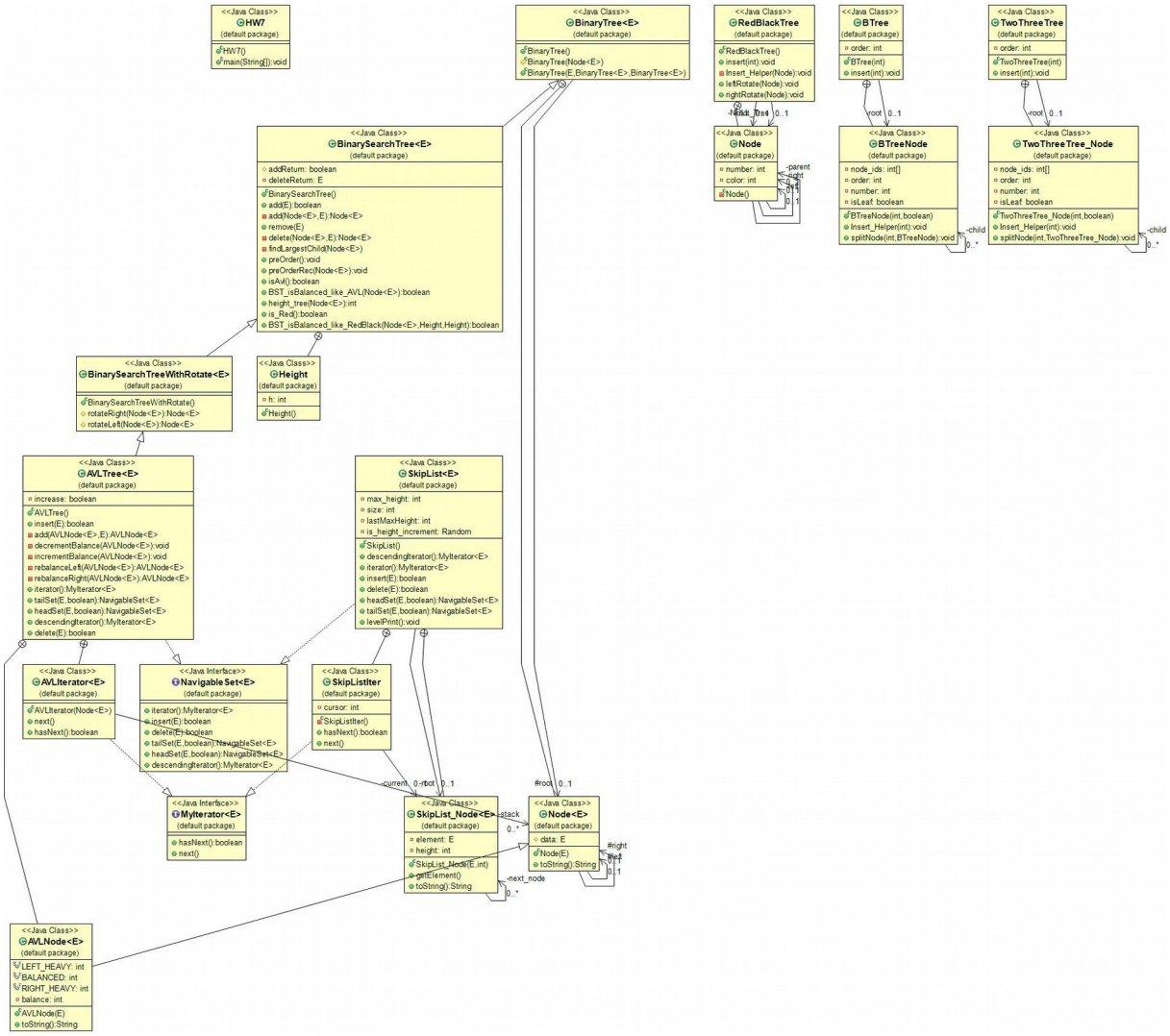
```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////// Red-Black Tree İnsert Time Operations

            for(int i=0;i<10;i++){
                RedBlackTree rbt = new RedBlackTree();

                Set<Integer>set = new LinkedHashSet<Integer>();
                Set<Integer>set2 = new LinkedHashSet<Integer>();

                while (set.size() < 80000*2) {
                    set.add(randNum.nextInt(80000*2)+1);
                }

                Iterator value = set.iterator();
                while (value.hasNext()){
                    rbt.insert((Integer)value.next());
                }

                while (set2.size() < 100*2) {
                    set2.add(randNum.nextInt(100*2)+1);
                }

                Iterator value2 = set2.iterator();

                startTime = System.nanoTime();
                while (value2.hasNext()){
                    rbt.insert((Integer)value2.next());
                }
                endTime=System.nanoTime();
                estimatedTime=endTime-startTime;

                rbt_time_sum=rbt_time_sum+(estimatedTime);
            }
            System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////// 2-3 Tree İnsert Time Operations
             for(int i=0;i<10;i++){
                TwoThreeTree two_three_tree = new TwoThreeTree(3);

                Set<Integer>set = new LinkedHashSet<Integer>();
                Set<Integer>set2 = new LinkedHashSet<Integer>();

                while (set.size() < 80000*2) {
                    set.add(randNum.nextInt(80000*2)+1);
                }

                Iterator value = set.iterator();
                while (value.hasNext()){
                    two_three_tree.insert((Integer)value.next());
                }

                while (set2.size() < 100*2) {
                    set2.add(randNum.nextInt(100*2)+1);
                }

                Iterator value2 = set2.iterator();

                startTime = System.nanoTime();
                while (value2.hasNext()){
                    two_three_tree.insert((Integer)value2.next());
                }
                endTime=System.nanoTime();
                estimatedTime=endTime-startTime;

                two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
             }

             System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
```

```
//////////////////////////////////////////////////////////////////////////////////////////// B Tree İnsert Time Operations
        for(int i=0;i<10;i++){
            BTree btree = new BTree(2);

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 80000*2) {
                set.add(randNum.nextInt(80000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                btree.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                btree.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            bTree_time_sum=bTree_time_sum+estimatedTime;
        }
        System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
```

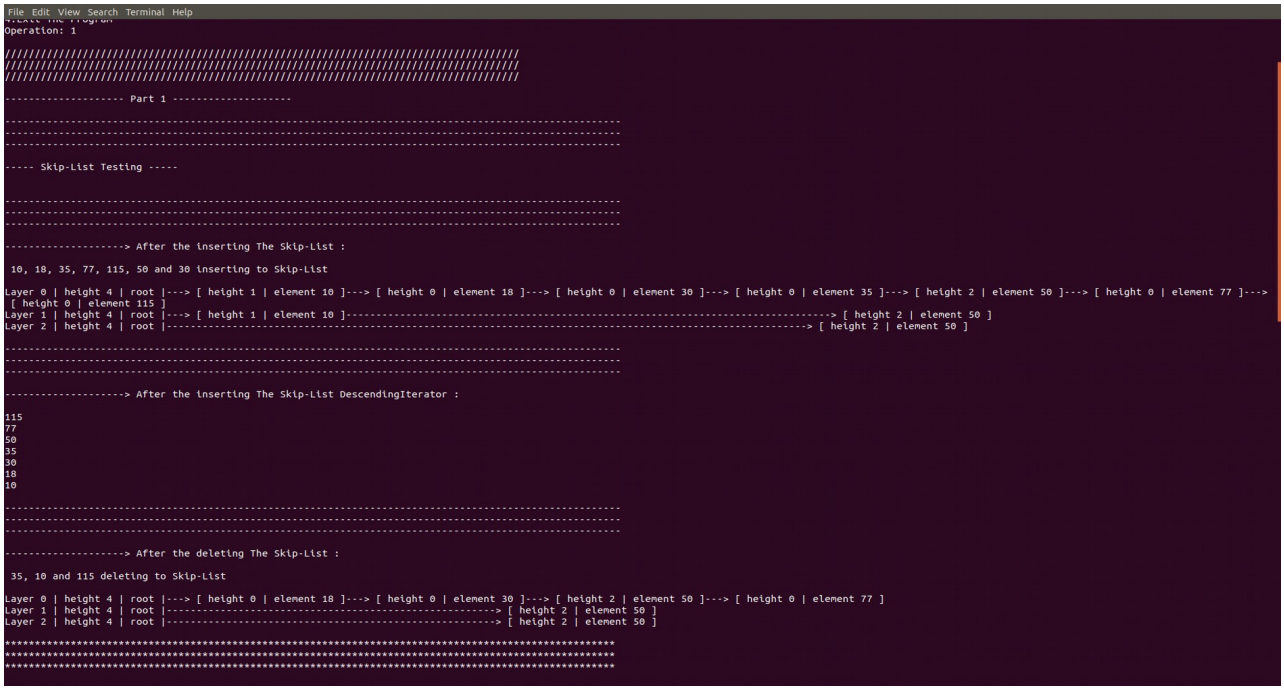
```
//////////////////////////////////////////////////////////////////////////////////////////// Skip-List İnsert Time Operations


        for(int i=0;i<10;i++){
            SkipList<Integer> skip_list = new SkipList<Integer>();

            Set<Integer>set = new LinkedHashSet<Integer>();
            Set<Integer>set2 = new LinkedHashSet<Integer>();

            while (set.size() < 80000*2) {
                set.add(randNum.nextInt(80000*2)+1);
            }

            Iterator value = set.iterator();
            while (value.hasNext()){
                skip_list.insert((Integer)value.next());
            }

            while (set2.size() < 100*2) {
                set2.add(randNum.nextInt(100*2)+1);
            }

            Iterator value2 = set2.iterator();

            startTime = System.nanoTime();
            while (value2.hasNext()){
                skip_list.insert((Integer)value2.next());
            }
            endTime=System.nanoTime();
            estimatedTime=endTime-startTime;

            skipList_time_sum=skipList_time_sum+estimatedTime;
        }

        System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));
```

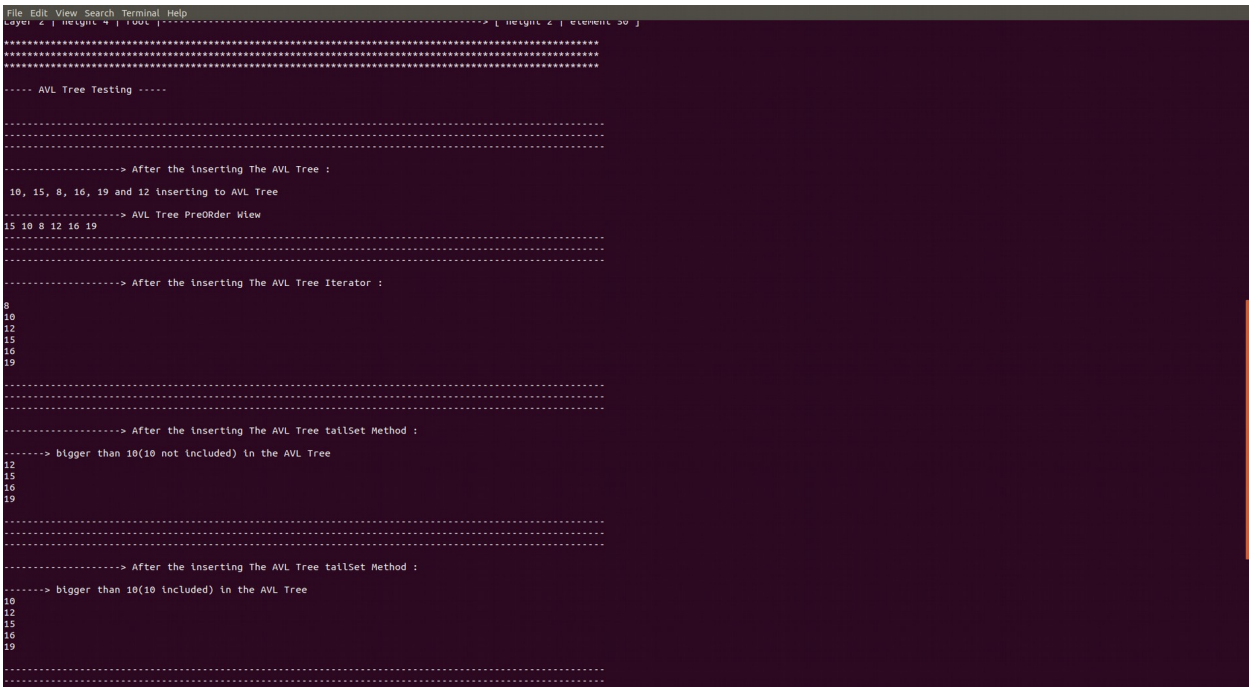
Class Diagrams

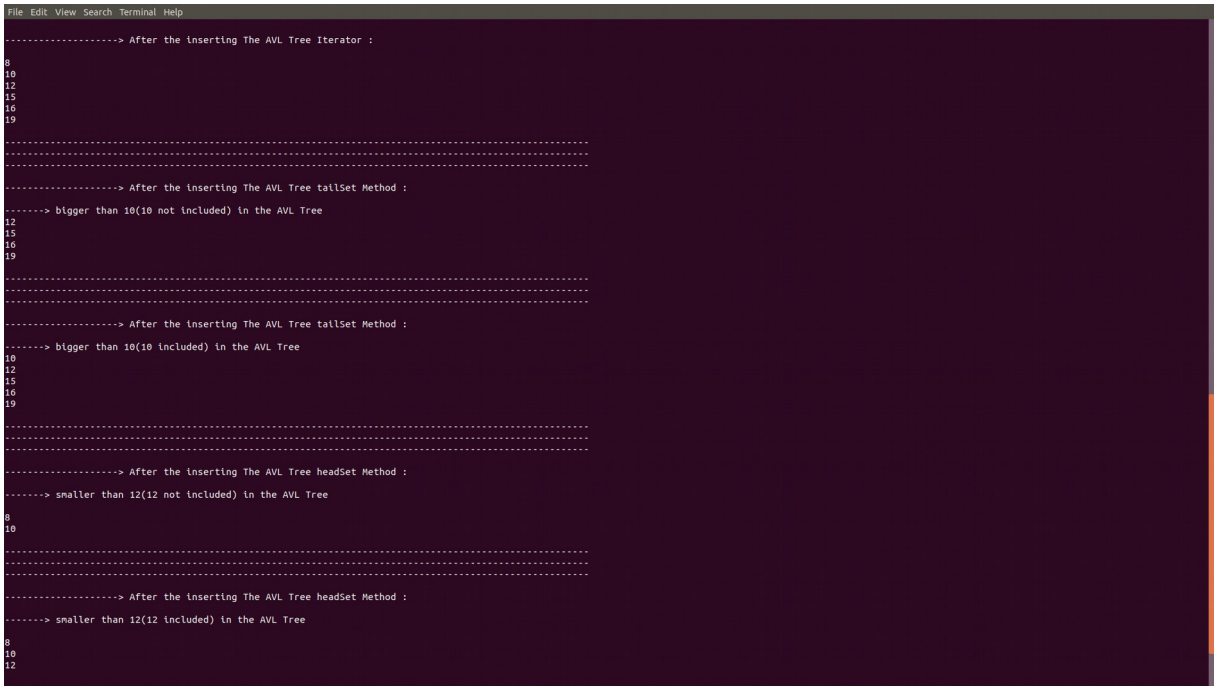


Running command and results

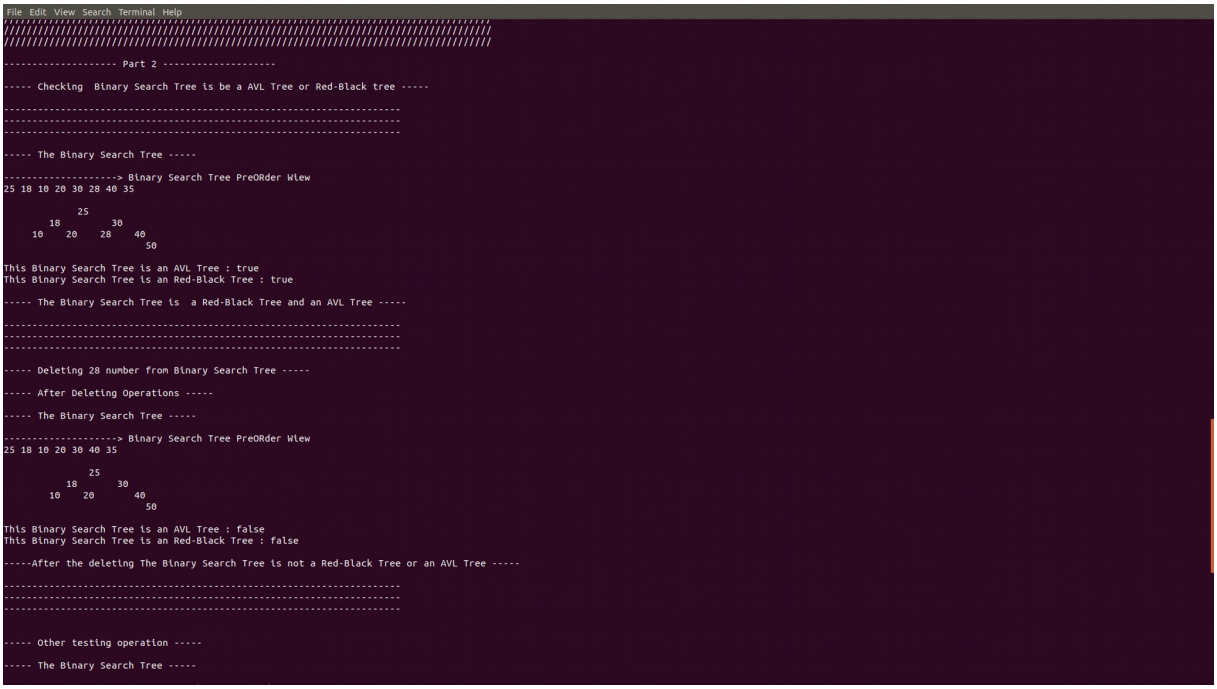
PART1 (AVL Tree and Skip-List Methods)







PART2 (Binary Search Tree is AVL Tree or Red-Black Tree)



```
File  Edit  View  Search  Terminal  Help
----- The Binary Search Tree -----

--------------------> Binary Search Tree PreORder Wiew
25 18 10 20 30 40 35

              25
          18      30
       10    20      40
                       50

This Binary Search Tree is an AVL Tree : false
This Binary Search Tree is an Red-Black Tree : false

-----After the deleting The Binary Search Tree is not a Red-Black Tree or an AVL Tree -----

------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------


----- Other testing operation -----

----- The Binary Search Tree -----

--------------------> Binary Search Tree PreORder Wiew
1 2 3 4

            1
               2
                  3
                     4

This Binary Search Tree is an AVL Tree : false
This Binary Search Tree is an Red-Black Tree : false

----- The Binary Search Tree is not a Red-Black Tree or an AVL Tree -----

------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------


----- Other testing operation -----

----- The Binary Search Tree -----

--------------------> Binary Search Tree PreORder Wiew
11 3 115 60 35 150

           11
        3      115
            60     150
          35

This Binary Search Tree is an AVL Tree : false
This Binary Search Tree is an Red-Black Tree : true

----- The Binary Search Tree is a Red-Black Tree but not an AVL Tree -----
```

PART3 (Test and Compare Data Structure)

```
File  Edit  View  Search  Terminal  Help
------------------------------------------------------------------
Which Part Do You Want to Play ?
1.PART 1
2.PART 2
3.PART 3
4.Exit The Program
Operation: 3

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////

------------------- Part 3 -------------------

------------------------------------------------------------------
!!!!! It will take about 2 minutes !!!!!
------------------------------------------------------------------
Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 10.000 Size Of Data Structures
Avarage time of Binary Search Tree :157577
Avarage time of Red-Black Tree :64136
Avarage time of 2-3 Tree :85269
Avarage time of B Tree :100432
Avarage time of Skip-List :45799

------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------


Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 20.000 Size Of Data Structures
Avarage time of Binary Search Tree :186650
Avarage time of Red-Black Tree :91406
Avarage time of 2-3 Tree :113942
Avarage time of B Tree :149236
Avarage time of Skip-List :87096

------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------


Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 40.000 Size Of Data Structures
Avarage time of Binary Search Tree :215997
Avarage time of Red-Black Tree :118607
Avarage time of 2-3 Tree :146984
Avarage time of B Tree :198119
Avarage time of Skip-List :131106

------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------


Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 80.000 Size Of Data Structures
Avarage time of Binary Search Tree :268546
Avarage time of Red-Black Tree :152373
Avarage time of 2-3 Tree :187363
Avarage time of B Tree :277137
Avarage time of Skip-List :181309

------------------------------------------------------------------
Which Part Do You Want to Play ?
```