

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

In 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 with descending order. Also insert, iterator, headSet and tailSet method were written for AVL Tree with using , navigableset interface and my Iterator interface. Insert 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 expression is true included the considerations or not. tailSet method return the bigger elements for the given element in the AVL, if boolean expression is true included the considerations or not.

In the second part,

For determine a Binary Search Tree is a AVL Tree : If 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.

In 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

```
////////////////////////////////////// FMS1.1 ////////////////////////////////////////
//
//      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+tailSet 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
1 10 10 10 10
2 10 10 10 10 System.out.println();
3 10 10 10 10 System.out.println("-----");
4 10 10 10 10 System.out.println("-----");
5 10 10 10 10 System.out.println("-----");
6 10 10 10 10 System.out.println();
7 10 10 10 10 System.out.println("-----> After the inserting The AVL Tree headSet Method :");
8 10 10 10 10 System.out.println();
9 10 10 10 10 System.out.println("-----> smaller than 12(12 not included) in the AVL Tree ");
10 10 10 10 10 System.out.println();

11 10 10 10 10 NavigableSet<Integer> avl3=avl.headSet(12, false);
12 10 10 10 10
13 10 10 10 10 MyIterator itr4= ((AVLTree)avl3).iterator();
14 10 10 10 10 while(itr4.hasNext()){
15 10 10 10 10 10 try{
16 10 10 10 10 10 10 System.out.println(itr4.next());
17 10 10 10 10 10 }
18 10 10 10 10 10 catch(NoSuchElementException e){
19 10 10 10 10 10 System.out.println(e);
20 10 10 10 10 }
21 10 10 10 10 }

//////////////////////////////////// headSet(TRUE) Method
22 10 10 10 10
23 10 10 10 10 System.out.println();
24 10 10 10 10 System.out.println("-----");
25 10 10 10 10 System.out.println("-----");
26 10 10 10 10 System.out.println("-----");
27 10 10 10 10 System.out.println();
28 10 10 10 10 System.out.println("-----> After the inserting The AVL Tree headSet Method :");
29 10 10 10 10 System.out.println();
30 10 10 10 10 System.out.println("-----> smaller than 12(12 included) in the AVL Tree ");
31 10 10 10 10 System.out.println();

32 10 10 10 10 NavigableSet<Integer> avl5=avl.headSet(12, true);
33 10 10 10 10
34 10 10 10 10 MyIterator itr6= ((AVLTree)avl5).iterator();
35 10 10 10 10 while(itr6.hasNext()){
36 10 10 10 10 10 try{
37 10 10 10 10 10 10 System.out.println(itr6.next());
38 10 10 10 10 10 }
39 10 10 10 10 10 catch(NoSuchElementException e){
40 10 10 10 10 10 System.out.println(e);
41 10 10 10 10 }
42 10 10 10 10 }
43 10 10 10 10 }
44 10 10 10 10 }
45 10 10 10 10 }
```

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 -----");

```

```

//////////////////////////////////// Deleting some numbers //////////////////////////////////////

10 10 10 10 tree.remove(28);
10 10 10 10
10 10 10 10 System.out.println();
10 10 10 10 System.out.println("-----");
10 10 10 10 System.out.println("-----");
10 10 10 10 System.out.println("-----");
10 10 10 10 System.out.println();

10 10 10 10 System.out.println("---- Deleting 28 number from Binary Search Tree ----");
10 10 10 10 System.out.println();
10 10 10 10 System.out.println("---- After Deleting Operations ----");
10 10 10 10 System.out.println();

10 10 10 10
10 10 10 10 System.out.println("---- The Binary Search Tree ----");
10 10 10 10 System.out.println();

10 10 10 10 System.out.println("-----> Binary Search Tree PreOrder View ");
10 10 10 10 tree.preOrder();
10 10 10 10 System.out.println();
10 10 10 10 System.out.println();

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

10 10 10 10 System.out.println("This Binary Search Tree is an AVL Tree : "+tree.isAvl());
10 10 10 10 System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree.is_Red());

10 10 10 10 System.out.println();
10 10 10 10 System.out.println("----After the deleting The Binary Search Tree is not a Red-Black Tree or an AVL Tree ----");

.....

1 10 10 10
1 10 10 10 System.out.println();
1 10 10 10 System.out.println("-----");
1 10 10 10 System.out.println("-----");
1 10 10 10 System.out.println("-----");
1 10 10 10 System.out.println();

1 10 10 10 BinarySearchTree tree2 = new BinarySearchTree();
1 10 10 10 tree2.add(1);
1 10 10 10 tree2.add(2);
1 10 10 10 tree2.add(3);
1 10 10 10 tree2.add(4);

1 10 10 10 System.out.println();
1 10 10 10 System.out.println("---- Other testing operation ----");
1 10 10 10 System.out.println();
1 10 10 10 System.out.println("---- The Binary Search Tree ----");
1 10 10 10 System.out.println();

1 10 10 10 System.out.println("-----> Binary Search Tree PreOrder View ");
1 10 10 10 tree2.preOrder();
1 10 10 10 System.out.println();
1 10 10 10 System.out.println();

1 10 10 10 System.out.println("----- 1 ");
1 10 10 10 System.out.println("----- 2 ");
1 10 10 10 System.out.println("----- 3");
1 10 10 10 System.out.println("----- 4");
1 10 10 10 System.out.println();

1 10 10 10 System.out.println("This Binary Search Tree is an AVL Tree : "+tree2.isAvl());
1 10 10 10 System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree2.is_Red());

1 10 10 10 System.out.println();
1 10 10 10 System.out.println("---- The Binary Search Tree is not a Red-Black Tree or an AVL Tree ----");

```



```

20 20 20 20 System.out.println();
20 20 20 20 System.out.println("-----");
20 20 20 20 System.out.println("-----");
20 20 20 20 System.out.println("-----");
20 20 20 20 System.out.println();
20 20 20 20 BinarySearchTree tree3 = new BinarySearchTree();
20 20 20 20 tree3.add(11);
20 20 20 20 tree3.add(3);
20 20 20 20 tree3.add(115);
20 20 20 20 tree3.add(60);
20 20 20 20 tree3.add(150);
20 20 20 20 tree3.add(35);
20 20 20 20 System.out.println();
20 20 20 20 System.out.println("----- Other testing operation -----");
20 20 20 20 System.out.println();
20 20 20 20 System.out.println("----- The Binary Search Tree -----");
20 20 20 20 System.out.println();
20 20 20 20 System.out.println("-----> Binary Search Tree PreOrder View ");
20 20 20 20 tree3.preOrder();
20 20 20 20 System.out.println();
20 20 20 20 System.out.println();
20 20 20 20 System.out.println("----- 11 ");
20 20 20 20 System.out.println("----- 3 ----- 115 ");
20 20 20 20 System.out.println("----- 60 ----- 150");
20 20 20 20 System.out.println("----- 35 -----");
20 20 20 20 System.out.println();
20 20 20 20 System.out.println("This Binary Search Tree is an AVL Tree : "+tree3.isAvl());
20 20 20 20 System.out.println("This Binary Search Tree is an Red-Black Tree : "+tree3.is_Red());
20 20 20 20 System.out.println();
20 20 20 20 System.out.println("----- The Binary Search Tree is a Red-Black Tree but not an AVL Tree -----");
20 20 20 20 }

```

Part3

Inserting 100 extra random numbers to 10.000 size of Data Structures

```

20 20 20 System.out.println();
20 20 20 System.out.println("////////////////////////////////////////");
20 20 20 System.out.println("////////////////////////////////////////");
20 20 20 System.out.println("////////////////////////////////////////");
20 20 20 System.out.println();
20 20 20 System.out.println("----- Part 3 -----");
20 20 20 System.out.println();
20 20 20 System.out.println("-----");
20 20 20 System.out.println(" !!!!! It will take about 2 minutes !!!!!");
20 20 20 System.out.println("-----");
20 20 20 long startTime;
20 20 20 long endTime;
20 20 20 long estimatedTime;
20 20 20 long bst_time_sum=0;
20 20 20 long rbt_time_sum=0;
20 20 20 long skiplist_time_sum=0;
20 20 20 long bTree_time_sum=0;
20 20 20 long two_three_Tree_time_sum=0;
20 20 20 Random randNum = new Random();
20 20 20 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 Insert 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 Insert 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));
}

```

```

1  20  20  20  20  ////////////////////////////////////// 2-3 Tree Insert Time Operations
2  20  20  20  20  for(int i=0;i<10;i++){
3  20  20  20  20  TwoThreeTree two_three_tree = new TwoThreeTree(3);
4  20  20  20  20  .....
5  20  20  20  20  Set<Integer>set = new LinkedHashSet<Integer>();
6  20  20  20  20  Set<Integer>set2 = new LinkedHashSet<Integer>();
7  20  20  20  20  .....
8  20  20  20  20  while (set.size() < 10000*2) {
9  20  20  20  20  20  set.add(randNum.nextInt(10000*2)+1);
10 20  20  20  20  }
11 20  20  20  20  .....
12 20  20  20  20  Iterator value = set.iterator();
13 20  20  20  20  while (value.hasNext()){
14 20  20  20  20  20  two_three_tree.insert((Integer)value.next());
15 20  20  20  20  }
16 20  20  20  20  .....
17 20  20  20  20  while (set2.size() < 100*2) {
18 20  20  20  20  20  set2.add(randNum.nextInt(100*2)+1);
19 20  20  20  20  }
20 20  20  20  20  .....
21 20  20  20  20  Iterator value2 = set2.iterator();
22 20  20  20  20  .....
23 20  20  20  20  startTime = System.nanoTime();
24 20  20  20  20  while (value2.hasNext()){
25 20  20  20  20  20  two_three_tree.insert((Integer)value2.next());
26 20  20  20  20  }
27 20  20  20  20  endTime=System.nanoTime();
28 20  20  20  20  estimatedTime=endTime-startTime;
29 20  20  20  20  .....
30 20  20  20  20  two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
31 20  20  20  20  }
32 20  20  20  20  .....
33 20  20  20  20  System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
34 20  20  20  20  ////////////////////////////////////// B Tree Insert Time Operations
35 20  20  20  20  .....
36 20  20  20  20  ////////////////////////////////////// B Tree Insert Time Operations
37 20  20  20  20  for(int i=0;i<10;i++){
38 20  20  20  20  BTree btree = new BTree(2);
39 20  20  20  20  .....
40 20  20  20  20  Set<Integer>set = new LinkedHashSet<Integer>();
41 20  20  20  20  Set<Integer>set2 = new LinkedHashSet<Integer>();
42 20  20  20  20  .....
43 20  20  20  20  while (set.size() < 10000*2) {
44 20  20  20  20  20  set.add(randNum.nextInt(10000*2)+1);
45 20  20  20  20  }
46 20  20  20  20  .....
47 20  20  20  20  Iterator value = set.iterator();
48 20  20  20  20  while (value.hasNext()){
49 20  20  20  20  20  btree.insert((Integer)value.next());
50 20  20  20  20  }
51 20  20  20  20  .....
52 20  20  20  20  while (set2.size() < 100*2) {
53 20  20  20  20  20  set2.add(randNum.nextInt(100*2)+1);
54 20  20  20  20  }
55 20  20  20  20  .....
56 20  20  20  20  Iterator value2 = set2.iterator();
57 20  20  20  20  .....
58 20  20  20  20  startTime = System.nanoTime();
59 20  20  20  20  while (value2.hasNext()){
60 20  20  20  20  20  btree.insert((Integer)value2.next());
61 20  20  20  20  }
62 20  20  20  20  endTime=System.nanoTime();
63 20  20  20  20  estimatedTime=endTime-startTime;
64 20  20  20  20  .....
65 20  20  20  20  bTree_time_sum=bTree_time_sum+estimatedTime;
66 20  20  20  20  }
67 20  20  20  20  System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
68 20  20  20  20  .....

```

```

//////////////////////////////////// Skip-List Insert Time Operations
10 10 10 10 10 for(int i=0;i<10;i++){
10 10 10 10 10     SkipList<Integer> skip_list = new SkipList<Integer>();
10 10 10 10 10     Set<Integer>set = new LinkedHashSet<Integer>();
10 10 10 10 10     Set<Integer>set2 = new LinkedHashSet<Integer>();
10 10 10 10 10     while (set.size() < 10000*2) {
10 10 10 10 10         set.add(randNum.nextInt(10000*2)+1);
10 10 10 10 10     }
10 10 10 10 10     Iterator value = set.iterator();
10 10 10 10 10     while (value.hasNext()){
10 10 10 10 10         skip_list.insert((Integer)value.next());
10 10 10 10 10     }
10 10 10 10 10     while (set2.size() < 100*2) {
10 10 10 10 10         set2.add(randNum.nextInt(100*2)+1);
10 10 10 10 10     }
10 10 10 10 10     Iterator value2 = set2.iterator();
10 10 10 10 10     startTime = System.nanoTime();
10 10 10 10 10     while (value2.hasNext()){
10 10 10 10 10         skip_list.insert((Integer)value2.next());
10 10 10 10 10     }
10 10 10 10 10     endTime=System.nanoTime();
10 10 10 10 10     estimatedTime=endTime-startTime;
10 10 10 10 10     skipList_time_sum=skipList_time_sum+estimatedTime;
10 10 10 10 10 }
10 10 10 10 10 System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));

```

Inserting 100 extra random numbers to 20.000 size of Data Structures

```

10 10 10 10 10 System.out.println();
10 10 10 10 10 System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 20.000 Size Of Data Structures");
10 10 10 10 10 ////////////////////////////////////// Binary Search Tree Insert Time Operations
10 10 10 10 10 for(int i=0;i<10;i++){
10 10 10 10 10     BinarySearchTree bst= new BinarySearchTree();
10 10 10 10 10     Set<Integer>set = new LinkedHashSet<Integer>();
10 10 10 10 10     Set<Integer>set2 = new LinkedHashSet<Integer>();
10 10 10 10 10     while (set.size() < 20000*2) {
10 10 10 10 10         set.add(randNum.nextInt(20000*2)+1);
10 10 10 10 10     }
10 10 10 10 10     Iterator value = set.iterator();
10 10 10 10 10     while (value.hasNext()){
10 10 10 10 10         bst.add((Integer)value.next());
10 10 10 10 10     }
10 10 10 10 10     while (set2.size() < 100*2) {
10 10 10 10 10         set2.add(randNum.nextInt(100*2)+1);
10 10 10 10 10     }
10 10 10 10 10     Iterator value2 = set2.iterator();
10 10 10 10 10     startTime = System.nanoTime();
10 10 10 10 10     while (value2.hasNext()){
10 10 10 10 10         bst.add((Integer)value2.next());
10 10 10 10 10     }
10 10 10 10 10     endTime=System.nanoTime();
10 10 10 10 10     estimatedTime=endTime-startTime;
10 10 10 10 10     bst_time_sum=bst_time_sum+estimatedTime;
10 10 10 10 10 }
10 10 10 10 10 System.out.println("Avarage time of Binary Search Tree :"+(bst_time_sum/10));

```

//////////////////////////////////// Red-Black Tree Insert Time Operations

```
0 0 0 0 0 for(int i=0;i<10;i++){
0 0 0 0 0 RedBlackTree rbt = new RedBlackTree();
0 0 0 0 0
0 0 0 0 0 Set<Integer>set = new LinkedHashSet<Integer>();
0 0 0 0 0 Set<Integer>set2 = new LinkedHashSet<Integer>();
0 0 0 0 0
0 0 0 0 0 while (set.size() < 20000*2) {
0 0 0 0 0 0 set.add(randNum.nextInt(20000*2)+1);
0 0 0 0 0 }
0 0 0 0 0
0 0 0 0 0 Iterator value = set.iterator();
0 0 0 0 0 while (value.hasNext()){
0 0 0 0 0 0 rbt.insert((Integer)value.next());
0 0 0 0 0 }
0 0 0 0 0
0 0 0 0 0 while (set2.size() < 100*2) {
0 0 0 0 0 0 set2.add(randNum.nextInt(100*2)+1);
0 0 0 0 0 }
0 0 0 0 0
0 0 0 0 0 Iterator value2 = set2.iterator();
0 0 0 0 0
0 0 0 0 0 startTime = System.nanoTime();
0 0 0 0 0 while (value2.hasNext()){
0 0 0 0 0 0 rbt.insert((Integer)value2.next());
0 0 0 0 0 }
0 0 0 0 0 endTime=System.nanoTime();
0 0 0 0 0 estimatedTime=endTime-startTime;
0 0 0 0 0
0 0 0 0 0 rbt_time_sum=rbt_time_sum+(estimatedTime);
0 0 0 0 0 }
0 0 0 0 0 System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));
.....
```

//////////////////////////////////// 2-3 Tree Insert Time Operations

```
0 0 0 0 0 for(int i=0;i<10;i++){
0 0 0 0 0 TwoThreeTree two_three_tree = new TwoThreeTree(3);
0 0 0 0 0
0 0 0 0 0 Set<Integer>set = new LinkedHashSet<Integer>();
0 0 0 0 0 Set<Integer>set2 = new LinkedHashSet<Integer>();
0 0 0 0 0
0 0 0 0 0 while (set.size() < 20000*2) {
0 0 0 0 0 0 set.add(randNum.nextInt(20000*2)+1);
0 0 0 0 0 }
0 0 0 0 0
0 0 0 0 0 Iterator value = set.iterator();
0 0 0 0 0 while (value.hasNext()){
0 0 0 0 0 0 two_three_tree.insert((Integer)value.next());
0 0 0 0 0 }
0 0 0 0 0
0 0 0 0 0 while (set2.size() < 100*2) {
0 0 0 0 0 0 set2.add(randNum.nextInt(100*2)+1);
0 0 0 0 0 }
0 0 0 0 0
0 0 0 0 0 Iterator value2 = set2.iterator();
0 0 0 0 0
0 0 0 0 0 startTime = System.nanoTime();
0 0 0 0 0 while (value2.hasNext()){
0 0 0 0 0 0 two_three_tree.insert((Integer)value2.next());
0 0 0 0 0 }
0 0 0 0 0 endTime=System.nanoTime();
0 0 0 0 0 estimatedTime=endTime-startTime;
0 0 0 0 0
0 0 0 0 0 two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
0 0 0 0 0 }
0 0 0 0 0 System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
.....
```

```

//////////////////////////////////// B Tree Insert Time Operations
5 10 10 10 10 for(int i=0;i<10;i++){
5 10 10 10 10 BTree btree = new BTree(2);
5 10 10 10 10
5 10 10 10 10 Set<Integer>set = new LinkedHashSet<Integer>();
5 10 10 10 10 Set<Integer>set2 = new LinkedHashSet<Integer>();
5 10 10 10 10
5 10 10 10 10 while (set.size() < 20000*2) {
5 10 10 10 10 10 set.add(randNum.nextInt(20000*2)+1);
5 10 10 10 10 10 }
5 10 10 10 10
5 10 10 10 10 Iterator value = set.iterator();
5 10 10 10 10 while (value.hasNext()){
5 10 10 10 10 10 btree.insert((Integer)value.next());
5 10 10 10 10 10 }
5 10 10 10 10
5 10 10 10 10 while (set2.size() < 100*2) {
5 10 10 10 10 10 set2.add(randNum.nextInt(100*2)+1);
5 10 10 10 10 10 }
5 10 10 10 10
5 10 10 10 10 Iterator value2 = set2.iterator();
5 10 10 10 10
5 10 10 10 10 startTime = System.nanoTime();
5 10 10 10 10 while (value2.hasNext()){
5 10 10 10 10 10 btree.insert((Integer)value2.next());
5 10 10 10 10 10 }
5 10 10 10 10 10 endTime=System.nanoTime();
5 10 10 10 10 10 estimatedTime=endTime-startTime;
5 10 10 10 10 10
5 10 10 10 10 10 bTree_time_sum=bTree_time_sum+estimatedTime;
5 10 10 10 10 10 }
5 10 10 10 10 System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
//////////////////////////////////// Skip-List Insert Time Operations

//////////////////////////////////// Skip-List Insert Time Operations
10 10 10 10 10 for(int i=0;i<10;i++){
10 10 10 10 10 10 SkipList<Integer> skip_list = new SkipList<Integer>();
10 10 10 10 10 10
10 10 10 10 10 10 Set<Integer>set = new LinkedHashSet<Integer>();
10 10 10 10 10 10 Set<Integer>set2 = new LinkedHashSet<Integer>();
10 10 10 10 10 10
10 10 10 10 10 10 while (set.size() < 20000*2) {
10 10 10 10 10 10 10 set.add(randNum.nextInt(20000*2)+1);
10 10 10 10 10 10 10 }
10 10 10 10 10 10
10 10 10 10 10 10 Iterator value = set.iterator();
10 10 10 10 10 10 while (value.hasNext()){
10 10 10 10 10 10 10 skip_list.insert((Integer)value.next());
10 10 10 10 10 10 10 }
10 10 10 10 10 10
10 10 10 10 10 10 while (set2.size() < 100*2) {
10 10 10 10 10 10 10 set2.add(randNum.nextInt(100*2)+1);
10 10 10 10 10 10 10 }
10 10 10 10 10 10
10 10 10 10 10 10 Iterator value2 = set2.iterator();
10 10 10 10 10 10
10 10 10 10 10 10 startTime = System.nanoTime();
10 10 10 10 10 10 while (value2.hasNext()){
10 10 10 10 10 10 10 skip_list.insert((Integer)value2.next());
10 10 10 10 10 10 10 }
10 10 10 10 10 10 10 endTime=System.nanoTime();
10 10 10 10 10 10 10 estimatedTime=endTime-startTime;
10 10 10 10 10 10 10
10 10 10 10 10 10 10 skipList_time_sum=skipList_time_sum+estimatedTime;
10 10 10 10 10 10 10 }
10 10 10 10 10 10 System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));

```

```

10 // System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 40.000 Size Of Data Structures");
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////// Binary Search Tree Insert Time Operations
12
13 10 10 10 10 for(int i=0;i<10;i++){
14 10 10 10 10     BinarySearchTree bst= new BinarySearchTree();
15
16 10 10 10 10 Set<Integer>set = new LinkedHashSet<Integer>();
17 10 10 10 10 Set<Integer>set2 = new LinkedHashSet<Integer>();
18
19 10 10 10 10 while (set.size() < 40000*2) {
20 10 10 10 10     set.add(randNum.nextInt(40000*2)+1);
21 10 10 10 10 }
22
23 10 10 10 10 Iterator value = set.iterator();
24 10 10 10 10 while (value.hasNext()){
25 10 10 10 10     bst.add((Integer)value.next());
26 10 10 10 10 }
27
28 10 10 10 10 while (set2.size() < 100*2) {
29 10 10 10 10     set2.add(randNum.nextInt(100*2)+1);
30 10 10 10 10 }
31
32 10 10 10 10 Iterator value2 = set2.iterator();
33
34 10 10 10 10 startTime = System.nanoTime();
35 10 10 10 10 while (value2.hasNext()){
36 10 10 10 10     bst.add((Integer)value2.next());
37 10 10 10 10 }
38 10 10 10 10 endTime=System.nanoTime();
39 10 10 10 10 estimatedTime=endTime-startTime;
40
41 10 10 10 10 bst_time_sum=bst_time_sum+estimatedTime;
42 10 10 10 10 }
43
44 10 10 10 10 System.out.println("Avarage time of Binary Search Tree :"+(bst_time_sum/10));
45
46 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////// Red-Black Tree Insert Time Operations
47
48 10 10 10 10 for(int i=0;i<10;i++){
49 10 10 10 10     RedBlackTree rbt = new RedBlackTree();
50
51 10 10 10 10 Set<Integer>set = new LinkedHashSet<Integer>();
52 10 10 10 10 Set<Integer>set2 = new LinkedHashSet<Integer>();
53
54 10 10 10 10 while (set.size() < 40000*2) {
55 10 10 10 10     set.add(randNum.nextInt(40000*2)+1);
56 10 10 10 10 }
57
58 10 10 10 10 Iterator value = set.iterator();
59 10 10 10 10 while (value.hasNext()){
60 10 10 10 10     rbt.insert((Integer)value.next());
61 10 10 10 10 }
62
63 10 10 10 10 while (set2.size() < 100*2) {
64 10 10 10 10     set2.add(randNum.nextInt(100*2)+1);
65 10 10 10 10 }
66
67 10 10 10 10 Iterator value2 = set2.iterator();
68
69 10 10 10 10 startTime = System.nanoTime();
70 10 10 10 10 while (value2.hasNext()){
71 10 10 10 10     rbt.insert((Integer)value2.next());
72 10 10 10 10 }
73 10 10 10 10 endTime=System.nanoTime();
74 10 10 10 10 estimatedTime=endTime-startTime;
75
76 10 10 10 10 rbt_time_sum=rbt_time_sum+(estimatedTime);
77 10 10 10 10 }
78
79 10 10 10 10 System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));

```

```

//////////////////////////////////// 2-3 Tree Insert Time Operations
0 10 10 10 10 for(int i=0;i<10;i++){
0 10 10 10 10 TwoThreeTree two_three_tree = new TwoThreeTree(3);
0 10 10 -----10 Set<Integer>set = new LinkedHashSet<Integer>();
0 10 10 -----10 Set<Integer>set2 = new LinkedHashSet<Integer>();
0 10 10 10 10
0 10 10 -----10 while (set.size() < 40000*2) {
0 10 -----10 10 set.add(randNum.nextInt(40000*2)+1);
0 10 -----10 10 }
0 10 10 -----10
0 10 10 -----10 Iterator value = set.iterator();
0 10 -----10 10 while (value.hasNext()){
0 10 -----10 10 two_three_tree.insert((Integer)value.next());
0 10 -----10 10 }
0 10 10 10 10
0 10 10 10 10 while (set2.size() < 100*2) {
0 10 10 -----10 10 set2.add(randNum.nextInt(100*2)+1);
0 10 10 -----10 10 }
0 10 10 -----10
0 10 10 -----10 Iterator value2 = set2.iterator();
0 10 10 -----10
0 10 10 -----10 startTime = System.nanoTime();
0 10 -----10 10 while (value2.hasNext()){
0 10 -----10 10 two_three_tree.insert((Integer)value2.next());
0 10 -----10 10 }
0 10 -----10 10 endTime=System.nanoTime();
0 10 10 10 10 estimatedTime=endTime-startTime;» »
0 10 10 10 10
0 10 10 10 10 two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
0 10 10 10 10 }

0 10 10 10 System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
//////////////////////////////////// B Tree Insert Time Operations

//////////////////////////////////// B Tree Insert Time Operations
10 10 10 10 for(int i=0;i<10;i++){
10 10 10 10 BTree btree = new BTree(2);
10 10 10 -----10
10 10 10 -----10 Set<Integer>set = new LinkedHashSet<Integer>();
10 10 10 -----10 Set<Integer>set2 = new LinkedHashSet<Integer>();
10 10 10 10 10
10 10 10 -----10 while (set.size() < 40000*2) {
10 10 -----10 10 set.add(randNum.nextInt(40000*2)+1);
10 10 -----10 10 }
10 10 10 -----10
10 10 10 -----10 Iterator value = set.iterator();
10 10 -----10 10 while (value.hasNext()){
10 10 -----10 10 btree.insert((Integer)value.next());
10 10 -----10 10 }
10 10 10 10 10
10 10 10 10 10 while (set2.size() < 100*2) {
10 10 10 -----10 10 set2.add(randNum.nextInt(100*2)+1);
10 10 10 -----10 10 }
10 10 10 -----10
10 10 10 -----10 Iterator value2 = set2.iterator();
10 10 10 -----10
10 10 10 -----10 startTime = System.nanoTime();
10 10 -----10 10 while (value2.hasNext()){
10 10 -----10 10 btree.insert((Integer)value2.next());
10 10 -----10 10 }
10 10 -----10 10 endTime=System.nanoTime();
10 10 10 10 10 estimatedTime=endTime-startTime;» »
10 10 10 10 10
10 10 10 10 10 bTree_time_sum=bTree_time_sum+estimatedTime;
10 10 10 10 10 }
10 10 10 10 System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));

```



```

//////////////////////////////////// Skip-List Insert Time Operations
10 10 10 10 10 for(int i=0;i<10;i++){
10 10 10 10 10 SkipList<Integer> skip_list = new SkipList<Integer>();
10 10 10 10 10
10 10 10 10 10 Set<Integer>set = new LinkedHashSet<Integer>();
10 10 10 10 10 Set<Integer>set2 = new LinkedHashSet<Integer>();
10 10 10 10 10
10 10 10 10 10 while (set.size() < 40000*2) {
10 10 10 10 10 10 set.add(randNum.nextInt(40000*2)+1);
10 10 10 10 10 10 }
10 10 10 10 10
10 10 10 10 10 Iterator value = set.iterator();
10 10 10 10 10 while (value.hasNext()){
10 10 10 10 10 10 skip_list.insert((Integer)value.next());
10 10 10 10 10 10 }
10 10 10 10 10
10 10 10 10 10 while (set2.size() < 100*2) {
10 10 10 10 10 10 set2.add(randNum.nextInt(100*2)+1);
10 10 10 10 10 10 }
10 10 10 10 10
10 10 10 10 10 Iterator value2 = set2.iterator();
10 10 10 10 10
10 10 10 10 10 startTime = System.nanoTime();
10 10 10 10 10 while (value2.hasNext()){
10 10 10 10 10 10 skip_list.insert((Integer)value2.next());
10 10 10 10 10 10 }
10 10 10 10 10 10 endTime=System.nanoTime();
10 10 10 10 10 10 estimatedTime=endTime-startTime;
10 10 10 10 10
10 10 10 10 10 skipList_time_sum=skipList_time_sum+estimatedTime;
10 10 10 10 10 }
10 10 10 10 10
10 10 10 10 10 System.out.println("Average time of Skip-List :"+(skipList_time_sum/10));
10 10 10 10 10

```

Inserting 100 extra random numbers to 80.000 size of Data Structures

```

10 10 10 10 10 System.out.println();
10 10 10 10 10 System.out.println("Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 80.000 Size Of Data Structures");
10 10 10 10 10 ////////////////////////////////////// Binary Search Tree Insert Time Operations
10 10 10 10 10
10 10 10 10 10 for(int i=0;i<10;i++){
10 10 10 10 10 10 BinarySearchTree bst= new BinarySearchTree();
10 10 10 10 10 10
10 10 10 10 10 10 Set<Integer>set = new LinkedHashSet<Integer>();
10 10 10 10 10 10 Set<Integer>set2 = new LinkedHashSet<Integer>();
10 10 10 10 10 10
10 10 10 10 10 10 while (set.size() < 80000*2) {
10 10 10 10 10 10 10 set.add(randNum.nextInt(80000*2)+1);
10 10 10 10 10 10 10 }
10 10 10 10 10 10
10 10 10 10 10 10 Iterator value = set.iterator();
10 10 10 10 10 10 while (value.hasNext()){
10 10 10 10 10 10 10 bst.add((Integer)value.next());
10 10 10 10 10 10 10 }
10 10 10 10 10 10
10 10 10 10 10 10 while (set2.size() < 100*2) {
10 10 10 10 10 10 10 set2.add(randNum.nextInt(100*2)+1);
10 10 10 10 10 10 10 }
10 10 10 10 10 10
10 10 10 10 10 10 Iterator value2 = set2.iterator();
10 10 10 10 10 10
10 10 10 10 10 10 startTime = System.nanoTime();
10 10 10 10 10 10 while (value2.hasNext()){
10 10 10 10 10 10 10 bst.add((Integer)value2.next());
10 10 10 10 10 10 10 }
10 10 10 10 10 10 10 endTime=System.nanoTime();
10 10 10 10 10 10 10 estimatedTime=endTime-startTime;
10 10 10 10 10 10 10
10 10 10 10 10 10 10 bst_time_sum=bst_time_sum+estimatedTime;
10 10 10 10 10 10 10 }
10 10 10 10 10
10 10 10 10 10 System.out.println("Average time of Binary Search Tree :"+(bst_time_sum/10));

```

//////////////////////////////////// Red-Black Tree Insert Time Operations

```
0 10 10 10 10 for(int i=0;i<10;i++){
0 10 10 10 10     RedBlackTree rbt = new RedBlackTree();
0 10 10 10 10     Set<Integer>set = new LinkedHashSet<Integer>();
0 10 10 10 10     Set<Integer>set2 = new LinkedHashSet<Integer>();
0 10 10 10 10     while (set.size() < 80000*2) {
0 10 10 10 10         set.add(randNum.nextInt(80000*2)+1);
0 10 10 10 10     }
0 10 10 10 10     Iterator value = set.iterator();
0 10 10 10 10     while (value.hasNext()){
0 10 10 10 10         rbt.insert((Integer)value.next());
0 10 10 10 10     }
0 10 10 10 10     while (set2.size() < 100*2) {
0 10 10 10 10         set2.add(randNum.nextInt(100*2)+1);
0 10 10 10 10     }
0 10 10 10 10     Iterator value2 = set2.iterator();
0 10 10 10 10     startTime = System.nanoTime();
0 10 10 10 10     while (value2.hasNext()){
0 10 10 10 10         rbt.insert((Integer)value2.next());
0 10 10 10 10     }
0 10 10 10 10     endTime=System.nanoTime();
0 10 10 10 10     estimatedTime=endTime-startTime;
0 10 10 10 10     rbt_time_sum=rbt_time_sum+(estimatedTime);
0 10 10 10 10 }
0 10 10 10 10 System.out.println("Avarage time of Red-Black Tree :"+(rbt_time_sum/10));
.....
```

//////////////////////////////////// 2-3 Tree Insert Time Operations

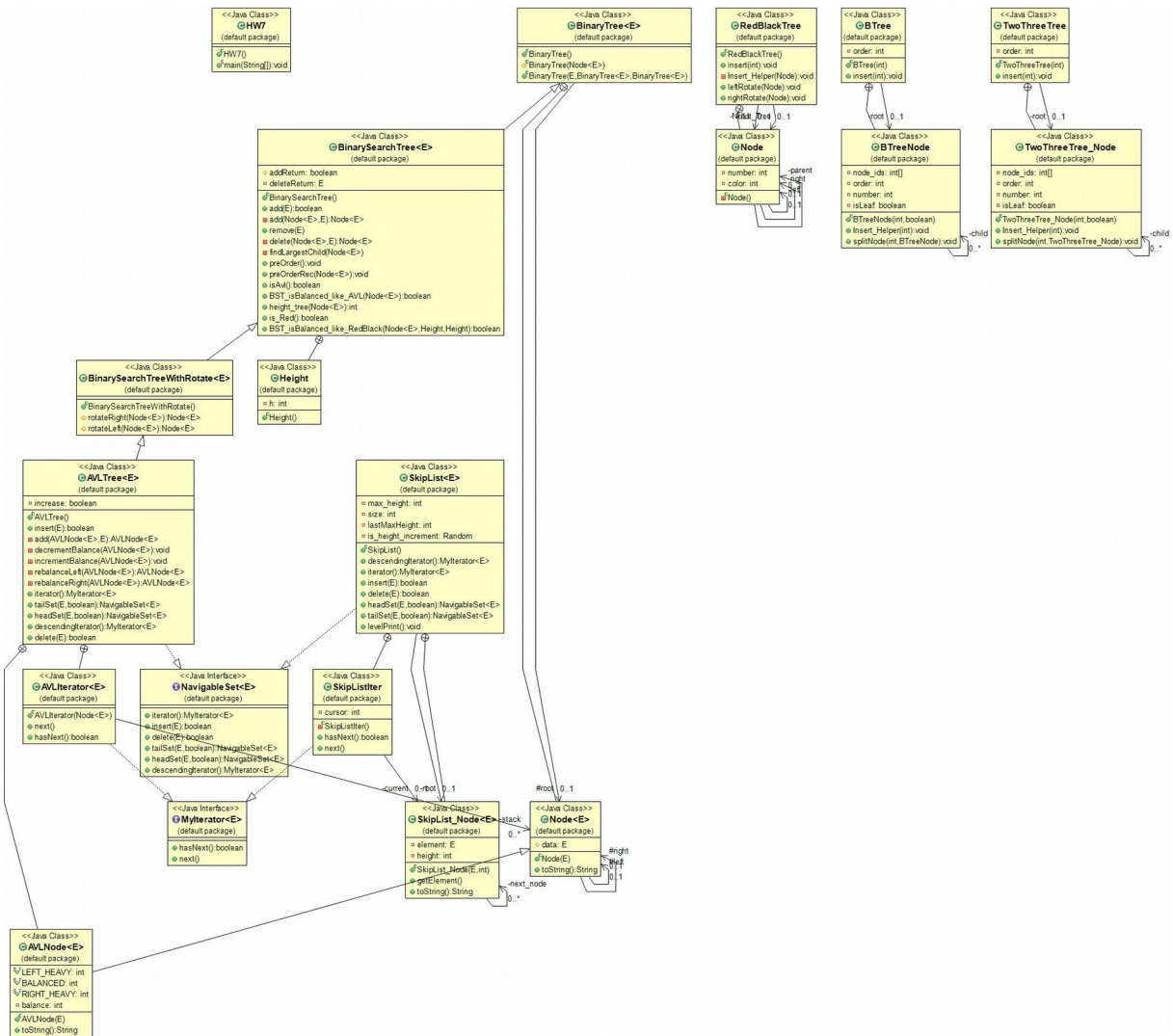
```
0 10 10 10 10 for(int i=0;i<10;i++){
0 10 10 10 10     TwoThreeTree two_three_tree = new TwoThreeTree(3);
0 10 10 10 10     Set<Integer>set = new LinkedHashSet<Integer>();
0 10 10 10 10     Set<Integer>set2 = new LinkedHashSet<Integer>();
0 10 10 10 10     while (set.size() < 80000*2) {
0 10 10 10 10         set.add(randNum.nextInt(80000*2)+1);
0 10 10 10 10     }
0 10 10 10 10     Iterator value = set.iterator();
0 10 10 10 10     while (value.hasNext()){
0 10 10 10 10         two_three_tree.insert((Integer)value.next());
0 10 10 10 10     }
0 10 10 10 10     while (set2.size() < 100*2) {
0 10 10 10 10         set2.add(randNum.nextInt(100*2)+1);
0 10 10 10 10     }
0 10 10 10 10     Iterator value2 = set2.iterator();
0 10 10 10 10     startTime = System.nanoTime();
0 10 10 10 10     while (value2.hasNext()){
0 10 10 10 10         two_three_tree.insert((Integer)value2.next());
0 10 10 10 10     }
0 10 10 10 10     endTime=System.nanoTime();
0 10 10 10 10     estimatedTime=endTime-startTime;
0 10 10 10 10     two_three_Tree_time_sum=two_three_Tree_time_sum+estimatedTime;
0 10 10 10 10 }
0 10 10 10 10 System.out.println("Avarage time of 2-3 Tree :"+(two_three_Tree_time_sum/10));
.....
```

```

////////////////////////////////////// B Tree Insert Time Operations
1 30 30 30 30 for(int i=0;i<10;i++){
2 30 30 30 30 BTree btree = new BTree(2);
3 30 30 30 30 30
4 30 30 30 30 30 Set<Integer>set = new LinkedHashSet<Integer>();
5 30 30 30 30 30 Set<Integer>set2 = new LinkedHashSet<Integer>();
6 30 30 30 30 30
7 30 30 30 30 30 while (set.size() < 80000*2) {
8 30 30 30 30 30 30 set.add(randNum.nextInt(80000*2)+1);
9 30 30 30 30 30 30 }
10 30 30 30 30 30
11 30 30 30 30 30 Iterator value = set.iterator();
12 30 30 30 30 30 while (value.hasNext()){
13 30 30 30 30 30 30 btree.insert((Integer)value.next());
14 30 30 30 30 30 30 }
15 30 30 30 30 30
16 30 30 30 30 30 while (set2.size() < 100*2) {
17 30 30 30 30 30 30 set2.add(randNum.nextInt(100*2)+1);
18 30 30 30 30 30 30 }
19 30 30 30 30 30
20 30 30 30 30 30 Iterator value2 = set2.iterator();
21 30 30 30 30 30
22 30 30 30 30 30 startTime = System.nanoTime();
23 30 30 30 30 30 while (value2.hasNext()){
24 30 30 30 30 30 30 btree.insert((Integer)value2.next());
25 30 30 30 30 30 30 }
26 30 30 30 30 30 30 endTime=System.nanoTime();
27 30 30 30 30 30 30 estimatedTime=endTime-startTime;
28 30 30 30 30 30
29 30 30 30 30 30 bTree_time_sum=bTree_time_sum+estimatedTime;
30 30 30 30 30 30 }
31 30 30 30 30 30 System.out.println("Avarage time of B Tree :"+(bTree_time_sum/10));
32 30 30 30 30 30
33 //////////////////////////////////////// Skip-List Insert Time Operations
34 30 30 30 30
35 30 30 30 30 for(int i=0;i<10;i++){
36 30 30 30 30 30 SkipList<Integer> skip_list = new SkipList<Integer>();
37 30 30 30 30 30
38 30 30 30 30 30 Set<Integer>set = new LinkedHashSet<Integer>();
39 30 30 30 30 30 Set<Integer>set2 = new LinkedHashSet<Integer>();
40 30 30 30 30 30
41 30 30 30 30 30 while (set.size() < 80000*2) {
42 30 30 30 30 30 30 set.add(randNum.nextInt(80000*2)+1);
43 30 30 30 30 30 30 }
44 30 30 30 30 30
45 30 30 30 30 30 Iterator value = set.iterator();
46 30 30 30 30 30 while (value.hasNext()){
47 30 30 30 30 30 30 skip_list.insert((Integer)value.next());
48 30 30 30 30 30 30 }
49 30 30 30 30 30
50 30 30 30 30 30 while (set2.size() < 100*2) {
51 30 30 30 30 30 30 set2.add(randNum.nextInt(100*2)+1);
52 30 30 30 30 30 30 }
53 30 30 30 30 30
54 30 30 30 30 30 Iterator value2 = set2.iterator();
55 30 30 30 30 30
56 30 30 30 30 30 startTime = System.nanoTime();
57 30 30 30 30 30 while (value2.hasNext()){
58 30 30 30 30 30 30 skip_list.insert((Integer)value2.next());
59 30 30 30 30 30 30 }
60 30 30 30 30 30 30 endTime=System.nanoTime();
61 30 30 30 30 30 30 estimatedTime=endTime-startTime;
62 30 30 30 30 30
63 30 30 30 30 30 skipList_time_sum=skipList_time_sum+estimatedTime;
64 30 30 30 30 30 }
65 30 30 30 30 30 System.out.println("Avarage time of Skip-List :"+(skipList_time_sum/10));
66 30 30 30 30 30

```

Class Diagrams



Running command and results

PART1 (AVL Tree and Skip-List Methods)

```
File Edit View Search Terminal Help
Results View Log
Operation: 1

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
----- Part 1 -----
----- Skip-List Testing -----

-----> After the Inserting The Skip-List :
10, 18, 35, 77, 115, 50 and 30 Inserting to Skip-List
Layer 0 | height 4 | root |----> [ height 1 | element 10 ]----> [ height 0 | element 35 ]----> [ height 0 | element 30 ]----> [ height 0 | element 77 ]---->
[ height 0 | element 115 ]
Layer 1 | height 4 | root |----> [ height 1 | element 10 ]----> [ height 2 | element 50 ]----> [ height 2 | element 50 ]
Layer 2 | height 4 | root |----> [ height 2 | element 50 ]

-----> After the Inserting The Skip-List DescendingIterator :
115
77
50
35
30
18
10

-----> After the deleting The Skip-List :
35, 10 and 115 deleting to Skip-List
Layer 0 | height 4 | root |----> [ height 0 | element 30 ]----> [ height 2 | element 50 ]----> [ height 0 | element 77 ]
Layer 1 | height 4 | root |----> [ height 2 | element 50 ]
Layer 2 | height 4 | root |----> [ height 2 | element 50 ]
=====
=====
```

```
File Edit View Search Terminal Help
Results View Log
Operation: 2

----- AVL Tree Testing -----

-----> After the Inserting The AVL Tree :
10, 15, 8, 16, 19 and 12 Inserting to AVL Tree
-----> AVL Tree PreOrder View
15 10 8 12 16 19

-----> After the Inserting The AVL Tree Iterator :
8
10
12
15
16
19

-----> After the Inserting The AVL Tree tailSet Method :
-----> bigger than 10(10 not included) in the AVL Tree
12
15
16
19

-----> After the Inserting The AVL Tree tailSet Method :
-----> bigger than 10(10 included) in the AVL Tree
10
12
15
16
19
```

```
File Edit View Search Terminal Help
-----> After the inserting The AVL Tree Iterator :
8
10
12
15
16
19
-----
-----> After the inserting The AVL Tree tailSet Method :
-----> bigger than 10(10 not included) in the AVL Tree
12
15
16
19
-----
-----> After the inserting The AVL Tree tailSet Method :
-----> bigger than 10(10 included) in the AVL Tree
10
12
15
16
19
-----
-----> After the inserting The AVL Tree headSet Method :
-----> smaller than 12(12 not included) in the AVL Tree
8
10
-----
-----> After the inserting The AVL Tree headSet Method :
-----> smaller than 12(12 included) in the AVL Tree
8
10
12
```

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

```
File Edit View Search Terminal Help
=====
----- Part 2 -----
---- Checking Binary Search Tree is be a AVL Tree or Red-Black tree ----
-----
----- The Binary Search Tree -----
-----> Binary Search Tree PreOrder View
25 18 10 20 30 28 40 35
      25
     /  \
    18   30
   /  \ /  \
  10 20 28 40
         \
         50
This Binary Search Tree is an AVL Tree : true
This Binary Search Tree is an Red-Black Tree : true
----- The Binary Search Tree is a Red-Black Tree and an AVL Tree -----
-----
----- Deleting 28 number from Binary Search Tree -----
----- After Deleting Operations -----
----- The Binary Search Tree -----
-----> Binary Search Tree PreOrder View
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 -----
```



```

File Edit View Search Terminal Help
----- The Binary Search Tree -----
-----> Binary Search Tree PreOrder Wlew
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 Wlew
1 2 3 4
      1
     / \
    2   3
   / \
  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 Wlew
11 3 115 60 35 150
      11
     /  \
    3   115
   / \  / \
  60 35 60 150

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
Average time of Binary Search Tree :157577
Average time of Red-Black Tree :64136
Average time of 2-3 Tree :85269
Average time of B Tree :180432
Average time of Skip-List :45799
-----
-----

Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 20.000 Size Of Data Structures
Average time of Binary Search Tree :186650
Average time of Red-Black Tree :91406
Average time of 2-3 Tree :113942
Average time of B Tree :149236
Average time of Skip-List :87696
-----
-----

Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 40.000 Size Of Data Structures
Average time of Binary Search Tree :215997
Average time of Red-Black Tree :118607
Average time of 2-3 Tree :146984
Average time of B Tree :198119
Average time of Skip-List :131186
-----
-----

Average Running Time(nanoSecond) After Inserting 100 Extra Random Numbers to 80.000 Size Of Data Structures
Average time of Binary Search Tree :268546
Average time of Red-Black Tree :152373
Average time of 2-3 Tree :187363
Average time of B Tree :277137
Average time of Skip-List :181389
-----
-----
Which Part Do You Want to Play ?

```