GIT

DEPARTMENT OF COMPUTER ENGINEERING

CSE 222/505 – Spring 2021

REPORT FOR HOMEWORK 7

SÜLEYMAN GÖLBOL

1801042656

1. SYSTEM REQUIREMENTS

FUNCTIONAL REQUIREMENTS

This application can be run by makefile file. In a bash or make entegreted PowerShell/cmd screen, “make” command will run the all tests.

Another functional requirement is having Java in computer. At least should *have Java 7 because* Java Comparable methods are used.

NON-FUNCTIONAL REQUIREMENTS

Because of the application is going to be used by a tester, a test method was prepared.

The program must be able to access all the book classes’ methods.

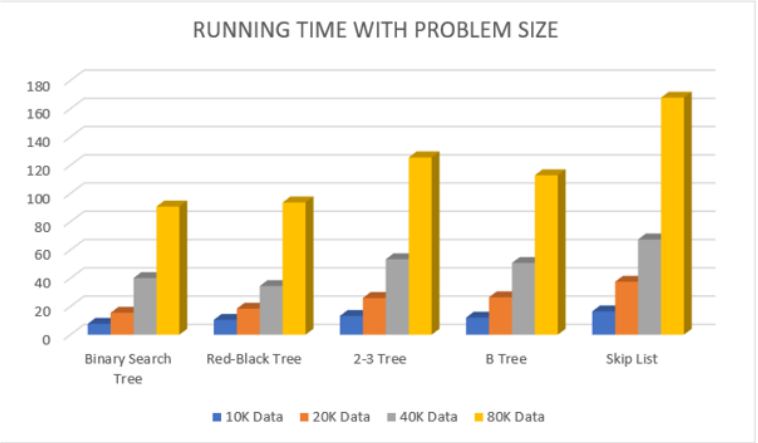
The program should be portable because it just making test.

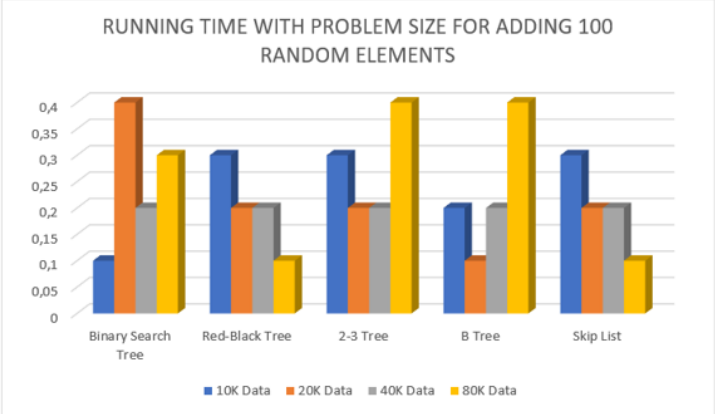
The program uses ram to store values in the memory.

So the computer must have a little bit ram to hold the data.

The program doesn’t create temporary text files to store values inside of text.

2. PART3 RESULT IN A GRAPH (BOTH ADDING ALL AND ADDING 100 ELEMENTS)

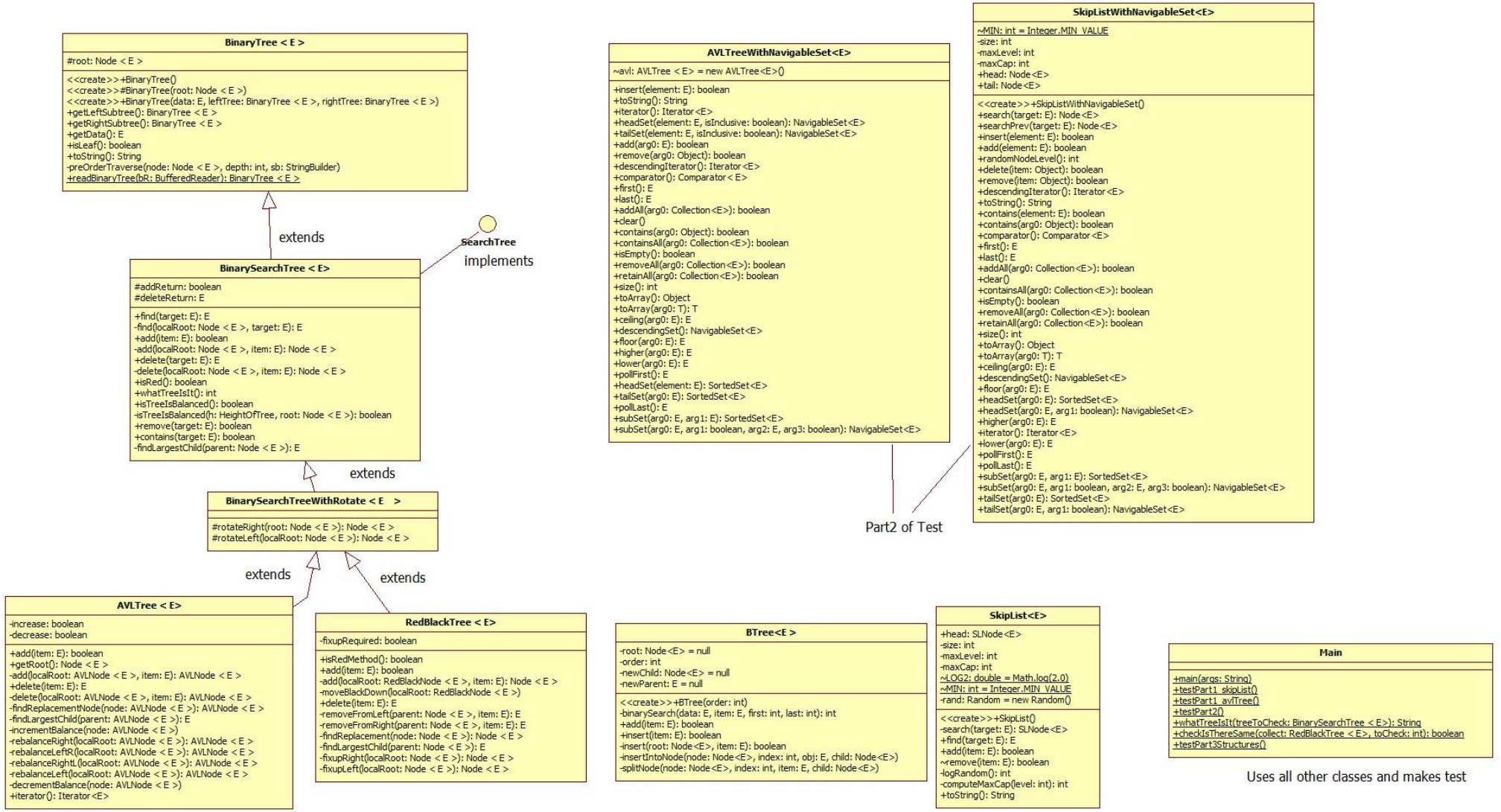




For adding 10k,20k,40k and 80k data; adding more data takes more time like it seen in first graph. But when we add elements skip list takes more time than binary trees. I think it is because skip list is not that good in worst case. In average case it is $\Theta(\log(n))$ but in worst case it is $O(n)$.

When we check for adding 100 random elements after all creating etc. done; because of we add less data it can be more obvious which is good. Skip list and red-black tree easily adds 100 random numbers, but B-Tree and 2-3 Tree takes more time for adding 100 elements.

3. CLASS DIAGRAMS



4. PROBLEM SOLUTION APPROACH

My solution approach for Part1 was after creating 2 new empty classes and extend them from NavigableSet to add insert/delete method, I used add/remove method from SkipList. I called it inside of insert method. Also for descending iterator, I created a nameless iterator class that implements Iterator<E>. I used search method to begin from end of skip list. Because of skip list search method returns predator, I used this tactique.
For AVL with navigable class, I used TailSet and HeadSet with iterator. So I iterated all data and if data is with I want I added to tailSet/headset.
For part2, I created isTreeIsBalanced() method inside of Bst class. Also to hold height of subtrees to find if tree is balanced, I created a private inner class called HeightOfTree class. So when I found height I changed the data inside of this class. I had to because java doesn't have call by reference.
For part3, to find out the running time results I checked in a for loop 10 different times and before the loop I created variables to hold both adding 100 element times + adding all elements times for every data structure and every problem size. In loop I summed up and after the loop I divided by 10 to find average values.

5. TEST CASES

5.1) Creating skip list, adding elements, checking an element if successfully added, removing elements, and checking if successfully removed, Then iterating through skip list in descending order to see if successfully iterates.

```
public static void testPart1_skipList(){
    SkipListWithNavigableSet<Integer> skipList = new SkipListWithNavigableSet<Integer>();
    System.out.println("---Adding elements---");
    for(int i = 0; i < 100; i++){
        skipList.insert(i);
    }

    System.out.println("---Checking if an element is added---");
    System.out.println( skipList.contains(3) );

    System.out.println("---Removing some elements---");
    for(int i = 99; i > 5; i--){
        skipList.delete(i);
    }

    System.out.println("---Checking two elements if successfully removed---");
    System.out.println( skipList.contains(55) );
    System.out.println( skipList.contains(25) );

    System.out.println("---Iterating through skip list in descending order and printing: ---");
    Iterator<Integer> iter = skipList.descendingIterator();
    while(iter.hasNext())
        System.out.print( iter.next() + ", " );
    System.out.println();
}
```

5.2) Creating avl tree, adding elements, checking an element if successfully added, then iterating through avl tree in normal order to see if successfully iterates.

```
public static void testPart1_avlTree() {
    AVLTreeWithNavigableSet<Integer> avlTree = new AVLTreeWithNavigableSet<Integer>();
    for(int i = 0; i < 100; i++){
        avlTree.insert(i);
    }

    System.out.println("---Checking if an element is added successfully---");
    System.out.println( avlTree.contains(3) );

    System.out.println("---Checking an element that doesn't exist in avl tree.---");
    System.out.println( avlTree.contains(59142) );

    System.out.println("---Iterating through AVLTREE and printing: ---");
    Iterator<Integer> iter = avlTree.iterator();
    while(iter.hasNext())
        System.out.print(iter.next() + ",");
```

5.3) Using headSet and tailSet methods and printing if successfully partitioned.

```
System.out.println("---Using HeadSet and Printing---");
SortedSet<Integer> avlTreeWithHeadSet = new AVLTreeWithNavigableSet<Integer>();
avlTreeWithHeadSet = avlTree.headSet(4, true);
System.out.println( avlTreeWithHeadSet.toString() );

System.out.println("---Using TailSet and Printing---");
SortedSet<Integer> avlTreeWithTailSet = new AVLTreeWithNavigableSet<Integer>();
avlTreeWithTailSet = avlTree.tailSet(96, true);
System.out.println( avlTreeWithTailSet.toString() );
```

5.4) Testing part2 with creating avl tree and red black tree. Adding some elements to them and checking which tree type are they.

```
public static void testPart2(){
    AVLTree<Integer> avlTree = new AVLTree<Integer>();
    avlTree.add(15); avlTree.add(45); avlTree.add(12);

    RedBlackTree<Integer> redBlack = new RedBlackTree<Integer>();
    redBlack.add(15); redBlack.add(45); redBlack.add(12);
    redBlack.add(65); redBlack.add(96); redBlack.add(98);
    //System.out.println(redBlack);
    System.out.println("---TESTING RED-BLACK TREE---");
    System.out.println( whatTreeIsIt(redBlack) );

    //System.out.println(avlTree);
    System.out.println("---TESTING AVL TREE---");
    System.out.println( whatTreeIsIt(avlTree) );
```

5.5) TESTING BST, RBT, 2-3 TREE, B TREE AND SKIP LIST with Creating average variables for every data structure and for every adding elements size(like 20k, 40k) also creating variables for insertion 100 elements.

```
System.out.println("---TESTING BST, RBT, 2-3 TREE, B TREE AND SKIP LIST---");

int randomNum, randomNum2;
long nano_start = System.currentTimeMillis();
long nano_end = System.currentTimeMillis();
// To calculate average running time.
float bstTimeAverage=0, rbtTimeAverage=0, ttTreeTimeAverage=0, bTreeTimeAverage=0, skipListTimeAverage=0;
float bstTimeAverage2=0, rbtTimeAverage2=0, ttTreeTimeAverage2=0, bTreeTimeAverage2=0, skipListTimeAverage2=0;
float bstTimeAverage3=0, rbtTimeAverage3=0, ttTreeTimeAverage3=0, bTreeTimeAverage3=0, skipListTimeAverage3=0;
float bstTimeAverage4=0, rbtTimeAverage4=0, ttTreeTimeAverage4=0, bTreeTimeAverage4=0, skipListTimeAverage4=0;
// To calculate average running time for inserting 100 extra.
float bstTimeInsertAverage=0, rbtTimeInsertAverage=0, ttTreeTimeInsertAverage=0, bTreeTimeInsertAverage=0, skipListTimeInsertAverage=0;
float bstTimeInsertAverage2=0, rbtTimeInsertAverage2=0, ttTreeTimeInsertAverage2=0, bTreeTimeInsertAverage2=0, skipListTimeInsertAverage2=0;
float bstTimeInsertAverage3=0, rbtTimeInsertAverage3=0, ttTreeTimeInsertAverage3=0, bTreeTimeInsertAverage3=0, skipListTimeInsertAverage3=0;
float bstTimeInsertAverage4=0, rbtTimeInsertAverage4=0, ttTreeTimeInsertAverage4=0, bTreeTimeInsertAverage4=0, skipListTimeInsertAverage4=0;
```

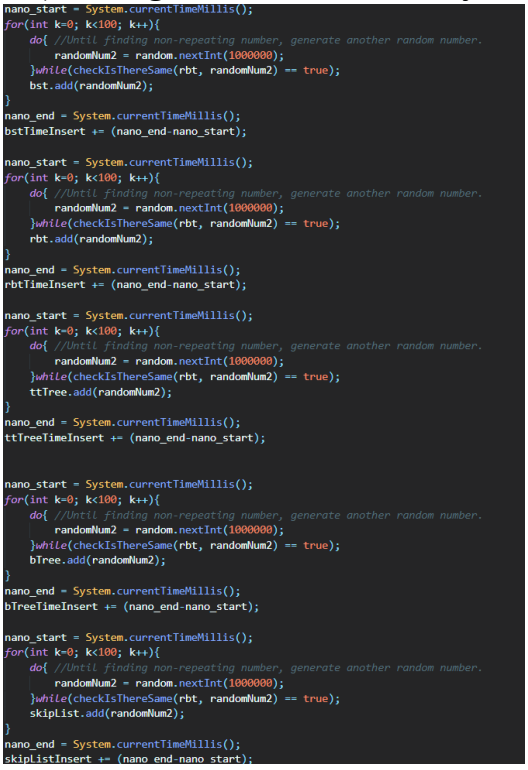
5.6) In for loop repeating 10 times, creating variables and class objects using constructors.

```
for(int i=0; i<10; i++){
    // To calculate running time for each
    int bstTime=0, rbtTime=0, ttTreeTime=0, bTreeTime=0, skipListTime=0;
    int bstTime2=0, rbtTime2=0, ttTreeTime2=0, bTreeTime2=0, skipListTime2=0;
    int bstTime3=0, rbtTime3=0, ttTreeTime3=0, bTreeTime3=0, skipListTime3=0;
    int bstTime4=0, rbtTime4=0, ttTreeTime4=0, bTreeTime4=0, skipListTime4=0;
    // To calculate running time for inserting 100 extra number
    int bstTimeInsert=0, rbtTimeInsert=0, ttTreeTimeInsert=0, bTreeTimeInsert=0, skipListInsert=0;
    int bstTimeInsert2=0, rbtTimeInsert2=0, ttTreeTimeInsert2=0, bTreeTimeInsert2=0, skipListInsert2=0;
    int bstTimeInsert3=0, rbtTimeInsert3=0, ttTreeTimeInsert3=0, bTreeTimeInsert3=0, skipListInsert3=0;
    int bstTimeInsert4=0, rbtTimeInsert4=0, ttTreeTimeInsert4=0, bTreeTimeInsert4=0, skipListInsert4=0;
    /*************************** FOR 10K ITEMS  ***************************************/
    BinarySearchTree<Integer> bst = new BinarySearchTree<Integer>();
    RedBlackTree<Integer> rbt = new RedBlackTree<Integer>();
    BTree<Integer> ttTree = new BTree<Integer>(3); //Using btree with 3 order makes it 2-3 tree.
    BTree<Integer> bTree = new BTree<Integer>(4);
    SkipList<Integer> skipList = new SkipList<Integer>();
```

5.7) Adding 10k elements for every data structure and checking times.

```
int j;
for(j=0; j<10000; j++){

    do{ //Until finding non-repeating number, generate another random number.
        randomNum = random.nextInt(1000000);
    }while(checkIsThereSame(rbt, randomNum) == true);

    nano_start = System.currentTimeMillis();
    bst.add(randomNum);
    nano_end = System.currentTimeMillis();
    bstTime += (nano_end-nano_start);

    nano_start = System.currentTimeMillis();
    rbt.add(randomNum);
    nano_end = System.currentTimeMillis();
    rbtTime += (nano_end-nano_start);

    nano_start = System.currentTimeMillis();
    ttTree.add(randomNum);
    nano_end = System.currentTimeMillis();
    ttTreeTime += (nano_end-nano_start);

    nano_start = System.currentTimeMillis();
    bTree.add(randomNum);
    nano_end = System.currentTimeMillis();
    bTreeTime += (nano_end-nano_start);

    nano_start = System.currentTimeMillis();
    skipList.add(randomNum);
    nano_end = System.currentTimeMillis();
    skipListTime += (nano_end-nano_start);

}
```

5.8) Adding 100 elements for every data structure.



```
nano_start = System.currentTimeMillis();
for(int k=0; k<100; k++){
    do{ //Until finding non-repeating number, generate another random number.
        randomNum2 = random.nextInt(1000000);
    }while(checkIsThereSame(rbt, randomNum2) == true);
    bst.add(randomNum2);
}
nano_end = System.currentTimeMillis();
bstTimeInsert += (nano_end-nano_start);

nano_start = System.currentTimeMillis();
for(int k=0; k<100; k++){
    do{ //Until finding non-repeating number, generate another random number.
        randomNum2 = random.nextInt(1000000);
    }while(checkIsThereSame(rbt, randomNum2) == true);
    rbt.add(randomNum2);
}
nano_end = System.currentTimeMillis();
rbtTimeInsert += (nano_end-nano_start);

nano_start = System.currentTimeMillis();
for(int k=0; k<100; k++){
    do{ //Until finding non-repeating number, generate another random number.
        randomNum2 = random.nextInt(1000000);
    }while(checkIsThereSame(rbt, randomNum2) == true);
    ttTree.add(randomNum2);
}
nano_end = System.currentTimeMillis();
ttTreeTimeInsert += (nano_end-nano_start);


nano_start = System.currentTimeMillis();
for(int k=0; k<100; k++){
    do{ //Until finding non-repeating number, generate another random number.
        randomNum2 = random.nextInt(1000000);
    }while(checkIsThereSame(rbt, randomNum2) == true);
    bTree.add(randomNum2);
}
nano_end = System.currentTimeMillis();
bTreeTimeInsert += (nano_end-nano_start);

nano_start = System.currentTimeMillis();
for(int k=0; k<100; k++){
    do{ //Until finding non-repeating number, generate another random number.
        randomNum2 = random.nextInt(1000000);
    }while(checkIsThereSame(rbt, randomNum2) == true);
    skipList.add(randomNum2);
}
nano_end = System.currentTimeMillis();
skipListInsert += (nano_end-nano_start);
```

5.9) Repeating 5.7 and 5.8 for 20k, 40k and 80k data.

5.10) Printing results in loop 10 times

```
System.out.println("\nTest" + (i+1) + " with milliSeconds:" );
System.out.println( "For 10k element | Bst : " + bstTime  + ", Rbt : " + rbtTime  + ", ttTree : " + ttTreeTime +  ", bTree : " + bTreeTime  + ", skiplist : " + skipListTime);
System.out.println( "Insert 100 elements to Bst : " + bstTimeInsert  + ", Insert to Rbt : " + rbtTimeInsert  + ", Insert to ttTree : " + ttTreeTimeInsert +  ", Insert to bTree : " + bTreeTi
System.out.println( "For 20k element | Bst2: " + bstTime2 + ", Rbt2: " + rbtTime2 + ", ttTree2: " + ttTreeTime2 + ", bTree2: " + bTreeTime2 + ", skiplist2: " + skipListTime2);
System.out.println( "Insert 100 elements to Bst : " + bstTimeInsert2  + ", Insert to Rbt : " + rbtTimeInsert2  + ", Insert to ttTree : " + ttTreeTimeInsert2 +  ", Insert to bTree : " + bTre
System.out.println( "For 40k element | Bst3: " + bstTime3 + ", Rbt3: " + rbtTime3 + ", ttTree3: " + ttTreeTime3 + ", bTree3: " + bTreeTime3 + ", skiplist3: " + skipListTime3);
System.out.println( "Insert 100 elements to Bst : " + bstTimeInsert3  + ", Insert to Rbt : " + rbtTimeInsert3  + ", Insert to ttTree : " + ttTreeTimeInsert3 +  ", Insert to bTree : " + bTre
System.out.println( "For 80k element | Bst4: " + bstTime4 + ", Rbt4: " + rbtTime4 + ", ttTree4: " + ttTreeTime4 + ", bTree4: " + bTreeTime4 + ", skiplist4: " + skipListTime4);
System.out.println( "Insert 100 elements to Bst : " + bstTimeInsert4  + ", Insert to Rbt : " + rbtTimeInsert4  + ", Insert to ttTree : " + ttTreeTimeInsert4 +  ", Insert to bTree : " + bTre
```

5.11) Adding to average variable to sum all.

```
bstTimeAverage += bstTime;              bstTimeAverage2 += bstTime2;             bstTimeAverage3 += bstTime3;              bstTimeAverage4 += bstTime4;
bstTimeInsertAverage += bstTimeInsert;  bstTimeInsertAverage2 += bstTimeInsert2; bstTimeInsertAverage3 += bstTimeInsert3;   bstTimeInsertAverage4 += bstTimeInsert4;

rbtTimeAverage += rbtTime;              rbtTimeAverage2 += rbtTime2;             rbtTimeAverage3 += rbtTime3;              rbtTimeAverage4 += rbtTime4;
rbtTimeInsertAverage += rbtTimeInsert;  rbtTimeInsertAverage2 += rbtTimeInsert2; rbtTimeInsertAverage3 += bstTimeInsert3;   rbtTimeInsertAverage4 += rbtTimeInsert4;

ttTreeTimeAverage += ttTreeTime;              ttTreeTimeAverage2 += ttTreeTime2;             ttTreeTimeAverage3 += ttTreeTime3;           ttTreeTimeAverage4 += ttTreeTime4;
ttTreeTimeInsertAverage += ttTreeTimeInsert;  ttTreeTimeInsertAverage2 += ttTreeTimeInsert2; ttTreeTimeInsertAverage3 += bstTimeInsert3;   ttTreeTimeInsertAverage4 += ttTreeTimeInsert4;

bTreeTimeAverage += bTreeTime;              bTreeTimeAverage2 += bTreeTime2;             bTreeTimeAverage3 += bTreeTime3;           bTreeTimeAverage4 += bTreeTime4;
bTreeTimeInsertAverage += bTreeTimeInsert;  bTreeTimeInsertAverage2 += bTreeTimeInsert2; bTreeTimeInsertAverage3 += bstTimeInsert3;   bTreeTimeInsertAverage4 += bTreeTimeInsert4;

skipListTimeAverage += skipListTime;          skipListTimeAverage2 += skipListTime2;         skipListTimeAverage3 += skipListTime3;       skipListTimeAverage4 += skipListTime4;
skipListTimeInsertAverage += skipListInsert;  skipListTimeInsertAverage2 += skipListInsert2; skipListTimeInsertAverage3 += bstTimeInsert3;   skipListTimeInsertAverage4 += skipListInsert4;
```

5.12) To find average dividing by 10 outside of loop.

```
//Finding average after ten times for everything.
bstTimeAverage /= 10;              bstTimeAverage2 /= 10;             bstTimeAverage3 /= 10;              bstTimeAverage4 /= 10;
bstTimeInsertAverage /= 10;  bstTimeInsertAverage2 /= 10; bstTimeInsertAverage3 /= 10;   bstTimeInsertAverage4 /= 10;

rbtTimeAverage /= 10;              rbtTimeAverage2 /= 10;             rbtTimeAverage3 /= 10;              rbtTimeAverage4 /= 10;
rbtTimeInsertAverage /= 10;  rbtTimeInsertAverage2 /= 10; rbtTimeInsertAverage3 /= 10;   rbtTimeInsertAverage4 /= 10;

ttTreeTimeAverage /= 10;            ttTreeTimeAverage2 /= 10;       ttTreeTimeAverage3 /= 10;          ttTreeTimeAverage4 /= 10;
ttTreeTimeInsertAverage /= 10;      ttTreeTimeInsertAverage2 /= 10; ttTreeTimeInsertAverage3 /= 10;    ttTreeTimeInsertAverage4 /= 10;

bTreeTimeAverage /= 10;              bTreeTimeAverage2 /= 10; bTreeTimeAverage3 /= 10;         bTreeTimeAverage4 /= 10;
bTreeTimeInsertAverage /= 10;  bTreeTimeInsertAverage2 /= 10; bTreeTimeInsertAverage3 /= 10;   bTreeTimeInsertAverage4 /= 10;

skipListTimeAverage /= 10;       skipListTimeAverage2 /= 10;        skipListTimeAverage3 /= 10;          skipListTimeAverage4 /= 10;
skipListTimeInsertAverage /= 10; skipListTimeInsertAverage2 /= 10;  skipListTimeInsertAverage3 /= 10;    skipListTimeInsertAverage4 /= 10;
```

5.13) Printing all average results.

```
//Acerage results.
System.out.println("\n---AVERAGE RESULTS:---");
System.out.println( "For 10k element | Bst : " + bstTimeAverage  + ", Rbt : " + rbtTimeAverage  + ", ttTree : " + ttTreeTimeAverage +  ", bTree : " + bTreeTimeAverage  + ", skiplist : " + skipListTimeAverage);
System.out.println( "Insert to Bst : " + bstTimeInsertAverage  + ", Insert to Rbt : " + rbtTimeInsertAverage  + ", Insert to ttTree : " + ttTreeTimeInsertAverage +  ", Insert to bTree : " + bTreeTimeInsertAverage  + ", Ins
System.out.println( "For 20k element | Bst2: " + bstTimeAverage2 + ", Rbt2: " + rbtTimeAverage2 + ", ttTree2: " + ttTreeTimeAverage2 + ", bTree2: " + bTreeTimeAverage2 + ", skiplist2: " + skipListTimeAverage2);
System.out.println( "Insert to Bst : " + bstTimeInsertAverage2  + ", Insert to Rbt : " + rbtTimeInsertAverage2  + ", Insert to ttTree : " + ttTreeTimeInsertAverage2 +  ", Insert to bTree : " + bTreeTimeInsertAverage2  + ",
System.out.println( "For 40k element | Bst3: " + bstTimeAverage3 + ", Rbt3: " + rbtTimeAverage3 + ", ttTree3: " + ttTreeTimeAverage3 + ", bTree3: " + bTreeTimeAverage3 + ", skiplist3: " + skipListTimeAverage3);
System.out.println( "Insert to Bst : " + bstTimeInsertAverage3  + ", Insert to Rbt : " + rbtTimeInsertAverage3  + ", Insert to ttTree : " + ttTreeTimeInsertAverage3 +  ", Insert to bTree : " + bTreeTimeInsertAverage3  + ",
System.out.println( "For 80k element | Bst4: " + bstTimeAverage4 + ", Rbt4: " + rbtTimeAverage4 + ", ttTree4: " + ttTreeTimeAverage4 + ", bTree4: " + bTreeTimeAverage4 + ", skiplist4: " + skipListTimeAverage4);
System.out.println( "Insert to Bst : " + bstTimeInsertAverage4  + ", Insert to Rbt : " + rbtTimeInsertAverage4  + ", Insert to ttTree : " + ttTreeTimeInsertAverage4 +  ", Insert to bTree : " + bTreeTimeInsertAverage4  + ",
```

6) RUNNING COMMAND AND RESULTS

Running make command to run the program.

```
sglbl@SglblPC:/mnt/c/Araclar/GTU/2.Sınıf/2.Dönem/Cse222/hw7$ make
javac -d classfiles *.java
java -cp classfiles Main
```

Adding elements, removing elements, cheking if successfully added

```
---Adding elements---
---Checking if an element is added---
true
---Removing some elements---
---Checking two elements if successfully removed---
false
false
---Iterating through skip list in descending order and printing: ---

---Checking if an element is added successfully---
false
---Checking an element that doesn't exist in avl tree.---
false
---Iterating through AVLTREE and printing: ---
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,-
```

Using headSet and tailSet and printing if successfully partitioned

```
---Using HeadSet and Printing---
1: 1
  0: 0
    null
    null
  0: 3
    0: 2
      null
      null
    0: 4
      null
      null

---Using TailSet and Printing---
1: 97
  0: 96
    null
    null
  1: 98
    null
    0: 99
      null
      null
```

Testing red-black tree and avl tree from binary search tree.

```
---TESTING RED-BLACK TREE---
It is a red-black tree.
---TESTING AVL TREE---
It is a AVL tree.
```

Testing part3 for 10k, 20k,40k and 80k data (1th from 10 test) Also testing for inserting 100 elements for all data sizes and for all the data structures(binary search tree, red black tree, two-three tree, B-tree and skip list.)

```
Test1 with milliSeconds:
For 10k element | Bst : 28, Rbt : 51, ttTree : 65, bTree : 33, skiplist : 65
Insert 100 elements to Bst : 0, Insert to Rbt : 1, Insert to ttTree : 0, Insert to bTree : 1, Insert to skiplist : 0
For 20k element | Bst2: 36, Rbt2: 81, ttTree2: 65, bTree2: 68, skiplist2: 75
Insert 100 elements to Bst : 1, Insert to Rbt : 1, Insert to ttTree : 0, Insert to bTree : 1, Insert to skiplist : 1
For 40k element | Bst3: 111, Rbt3: 156, ttTree3: 130, bTree3: 142, skiplist3: 252
Insert 100 elements to Bst : 1, Insert to Rbt : 1, Insert to ttTree : 0, Insert to bTree : 0, Insert to skiplist : 1
For 80k element | Bst4: 183, Rbt4: 146, ttTree4: 341, bTree4: 309, skiplist4: 504
Insert 100 elements to Bst : 0, Insert to Rbt : 0, Insert to ttTree : 1, Insert to bTree : 1, Insert to skiplist : 0
```

Testing and printing for 10 times and printing for 10th time.

```
Test10 with milliSeconds:
For 10k element | Bst : 14, Rbt : 19, ttTree : 27, bTree : 18, skiplist : 30
Insert 100 elements to Bst : 0, Insert to Rbt : 0, Insert to ttTree : 1, Insert to bTree : 0, Insert to skiplist : 0
For 20k element | Bst2: 28, Rbt2: 21, ttTree2: 57, bTree2: 48, skiplist2: 44
Insert 100 elements to Bst : 0, Insert to Rbt : 0, Insert to ttTree : 0, Insert to bTree : 1, Insert to skiplist : 0
For 40k element | Bst3: 60, Rbt3: 54, ttTree3: 94, bTree3: 89, skiplist3: 98
Insert 100 elements to Bst : 0, Insert to Rbt : 0, Insert to ttTree : 0, Insert to bTree : 1, Insert to skiplist : 0
For 80k element | Bst4: 155, Rbt4: 109, ttTree4: 251, bTree4: 227, skiplist4: 270
Insert 100 elements to Bst : 0, Insert to Rbt : 0, Insert to ttTree : 1, Insert to bTree : 1, Insert to skiplist : 0
```

Printing average result for adding 10k,20k,40k,80k elements and adding 100 elements for every data structure.

```
---AVERAGE RESULTS:---
For 10k element | Bst : 19.1, Rbt : 21.4, ttTree : 31.9, bTree : 25.3, skiplist : 32.6
Insert to Bst : 0.4, Insert to Rbt : 0.5, Insert to ttTree : 0.6, Insert to bTree : 0.4, Insert to skiplist : 0.9
For 20k element | Bst2: 44.3, Rbt2: 38.6, ttTree2: 60.1, bTree2: 55.0, skiplist2: 73.2
Insert to Bst : 0.4, Insert to Rbt : 0.3, Insert to ttTree : 0.4, Insert to bTree : 0.6, Insert to skiplist : 0.7
For 40k element | Bst3: 82.7, Rbt3: 75.1, ttTree3: 112.6, bTree3: 104.3, skiplist3: 148.3
Insert to Bst : 0.3, Insert to Rbt : 0.3, Insert to ttTree : 0.3, Insert to bTree : 0.3, Insert to skiplist : 0.3
For 80k element | Bst4: 186.1, Rbt4: 144.5, ttTree4: 276.0, bTree4: 242.4, skiplist4: 367.2
Insert to Bst : 0.2, Insert to Rbt : 0.6, Insert to ttTree : 0.4, Insert to bTree : 0.7, Insert to skiplist : 1.0
sglbl@SglblPC:/mnt/c/Araclar/GTU/2.Sınıf/2.Dönem/Cse222/hw7$
```