GTU Department of Computer Engineering

CSE 222/505 – SPRING 2021

Homework 7 Report

Oğulcan Kalafatoğlu

1801042613

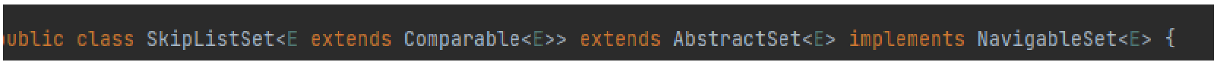
1 – Detailed System Requirements

2 – Problem Solution Approach

For part1,

BinarySearchTree, BinarySearch, BinaryTree, SearchTree and some methods of AVLTree and skipList were taken from textbook.

part1- first part:



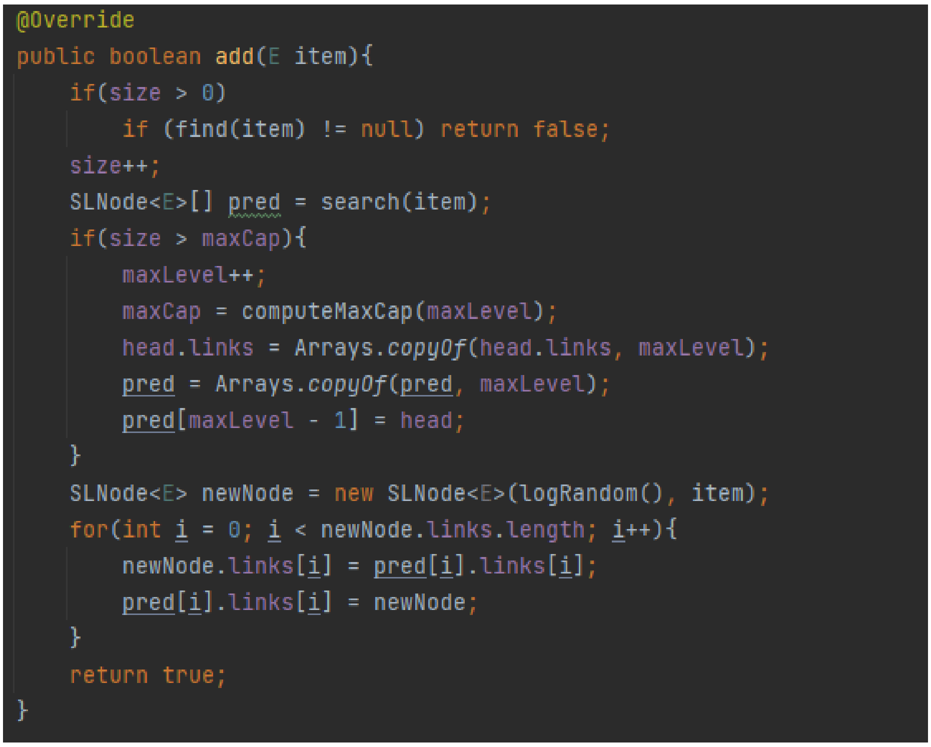
```
ublic class SkipListSet<E extends Comparable<E>> extends AbstractSet<E> implements NavigableSet<E> {
```

added implements NavigableSet<E> and extended AbstractSet<E> since most of the implementatin of navigableSet extends AbstractSet, but I did not use any method from abstractSet class

Only add, delete, descendingIterator methods from NavigableSet interface was implemented and other methods were left empty.

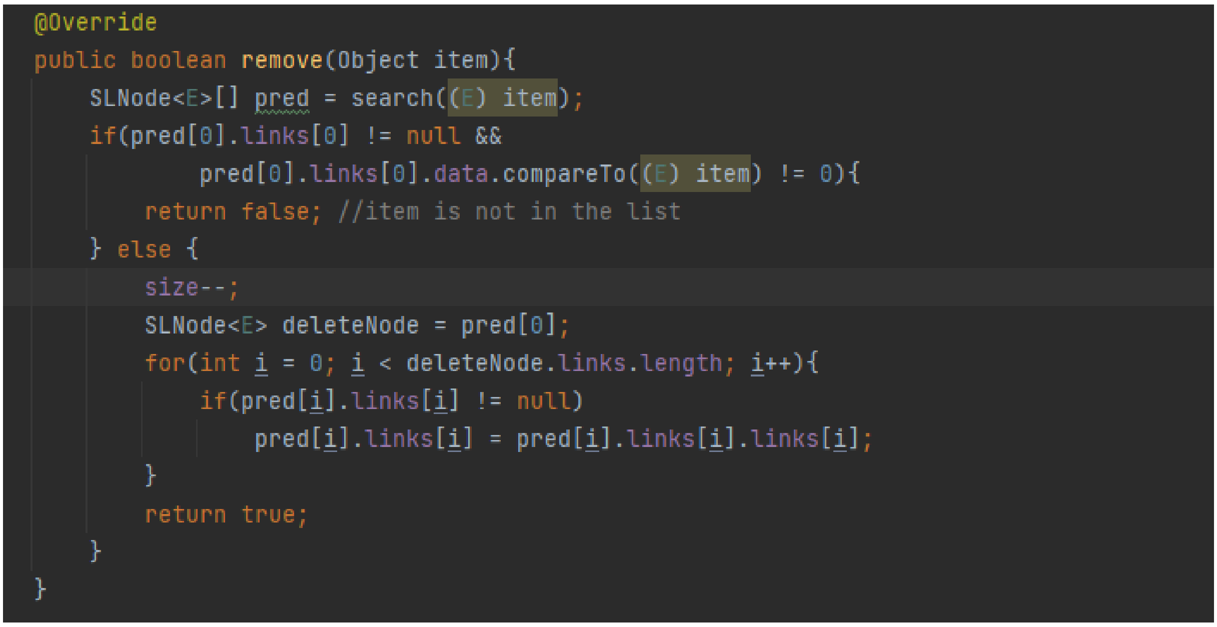
for implementation of add method

I assumed the method wanted from us was add method of NavigableSet, since insert method does not exist in the interface.



```
@Override
public boolean add(E item){
    if(size > 0)
        if (find(item) != null) return false;
    size++;
    SLNode<E>[] pred = search(item);
    if(size > maxCap){
        maxLevel++;
        maxCap = computeMaxCap(maxLevel);
        head.links = Arrays.copyOf(head.links, maxLevel);
        pred = Arrays.copyOf(pred, maxLevel);
        pred[maxLevel - 1] = head;
    }
    SLNode<E> newNode = new SLNode<E>(logRandom(), item);
    for(int i = 0; i < newNode.links.length; i++){
        newNode.links[i] = pred[i].links[i];
        pred[i].links[i] = newNode;
    }
    return true;
}
```

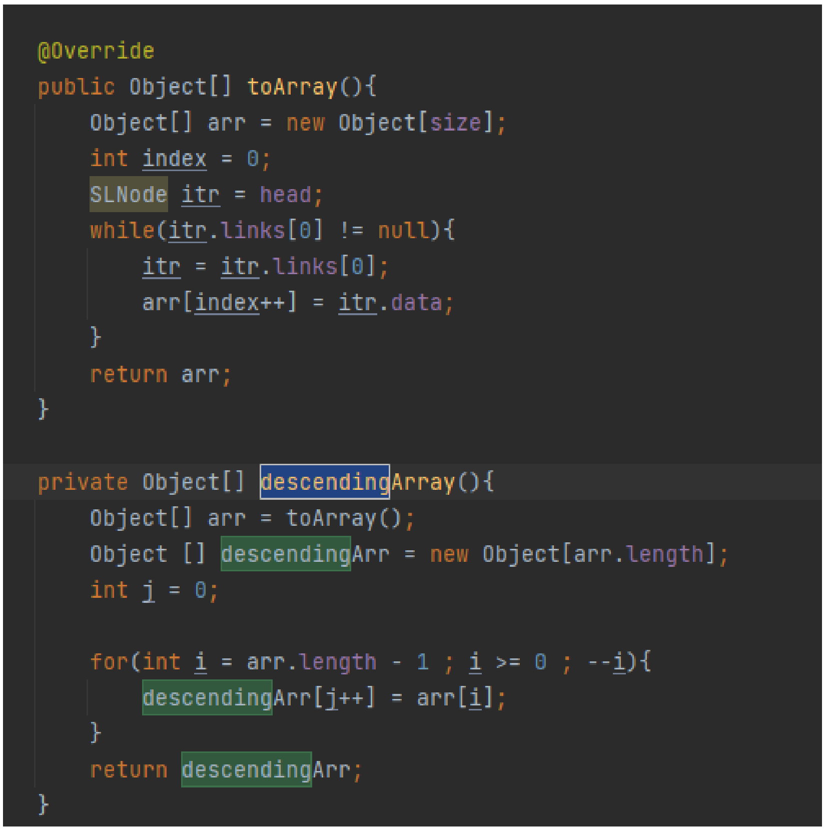
Before adding to the skipList, first i checked if item was already in sets, if it was exited remove (delete) method



```
@Override
public boolean remove(Object item){
    SLNode<E>[] pred = search((E) item);
    if(pred[0].links[0] != null &&
            pred[0].links[0].data.compareTo((E) item) != 0){
        return false; //item is not in the list
    } else {
        size--;
        SLNode<E> deleteNode = pred[0];
        for(int i = 0; i < deleteNode.links.length; i++){
            if(pred[i].links[i] != null)
                pred[i].links[i] = pred[i].links[i].links[i];
        }
        return true;
    }
}
```

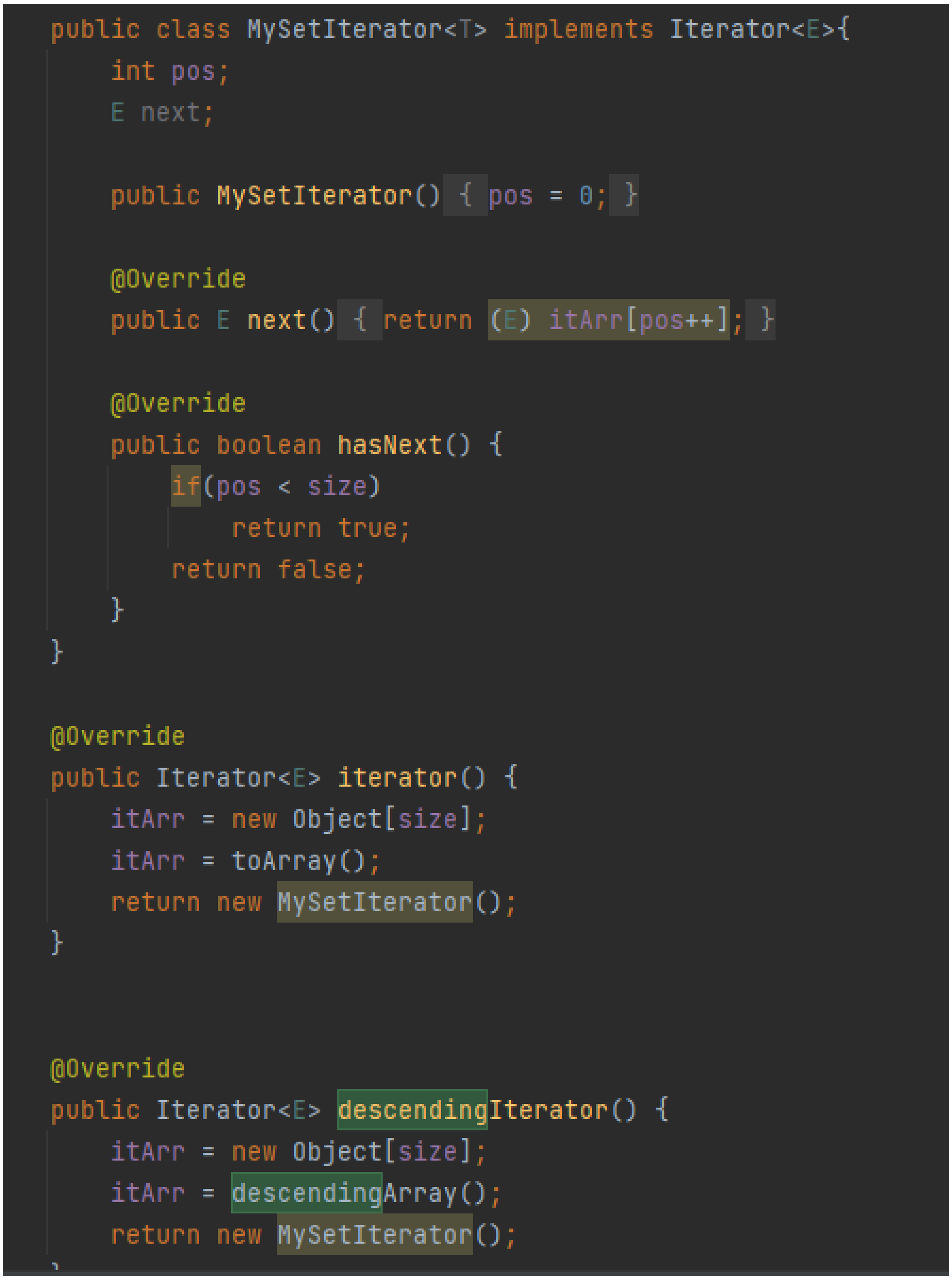
We do search according to levels and if wanted item is found, remove it from the skip-list

for descendingIterator method



```
@Override
public Object[] toArray(){
    Object[] arr = new Object[size];
    int index = 0;
    SLNode itr = head;
    while(itr.links[0] != null){
        itr = itr.links[0];
        arr[index++] = itr.data;
    }
    return arr;
}

private Object[] descendingArray(){
    Object[] arr = toArray();
    Object [] descendingArr = new Object[arr.length];
    int j = 0;

    for(int i = arr.length - 1 ; i >= 0 ; --i){
        descendingArr[j++] = arr[i];
    }
    return descendingArr;
}
```

one helper method is added, and toArray method is overrided.

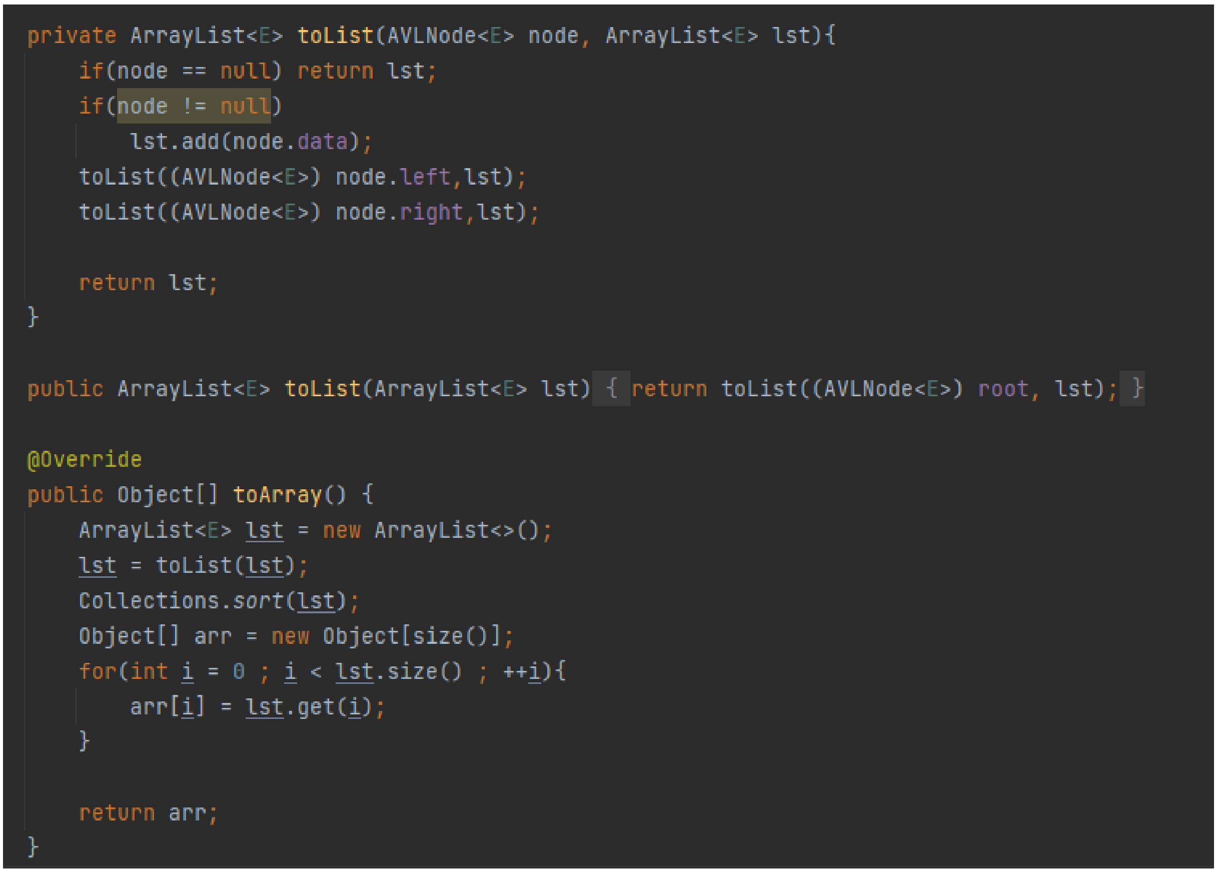


```
public class MySetIterator<T> implements Iterator<E>{
    int pos;
    E next;

    public MySetIterator() { pos = 0; }

    @Override
    public E next() { return (E) itArr[pos++]; }

    @Override
    public boolean hasNext() {
        if(pos < size)
            return true;
        return false;
    }
}

@Override
public Iterator<E> iterator() {
    itArr = new Object[size];
    itArr = toArray();
    return new MySetIterator();
}


@Override
public Iterator<E> descendingIterator() {
    itArr = new Object[size];
    itArr = descendingArray();
    return new MySetIterator();
}
```

Iterator is implemented as inner class, and toArray or descendingArray is called to iterate on from iterator(); and descendingIterator();

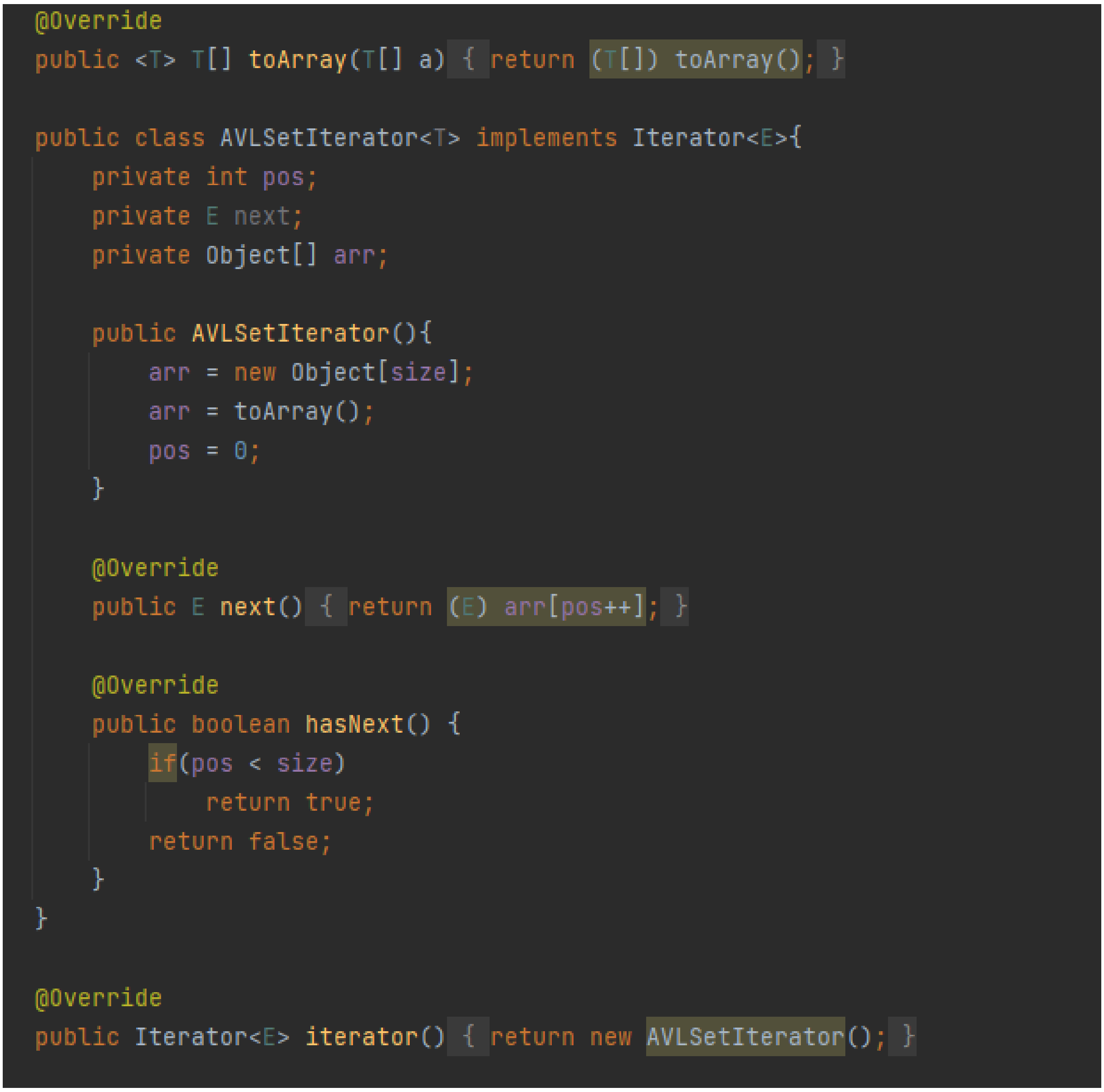
part 1-2

most of the methods were taken from textbook, and again navigableSet is implemented, and unneeded methods are left empty.



```java
private ArrayList<E> toList(AVLNode<E> node, ArrayList<E> lst){
    if(node == null) return lst;
    if(node != null)
        lst.add(node.data);
    toList((AVLNode<E>) node.left,lst);
    toList((AVLNode<E>) node.right,lst);

    return lst;
}

public ArrayList<E> toList(ArrayList<E> lst) { return toList((AVLNode<E>) root, lst); }

@Override
public Object[] toArray() {
    ArrayList<E> lst = new ArrayList<>();
    lst = toList(lst);
    Collections.sort(lst);
    Object[] arr = new Object[size()];
    for(int i = 0 ; i < lst.size() ; ++i){
        arr[i] = lst.get(i);
    }

    return arr;
}
```

toList helper method is implemented, and toArray is overrided for iterator.

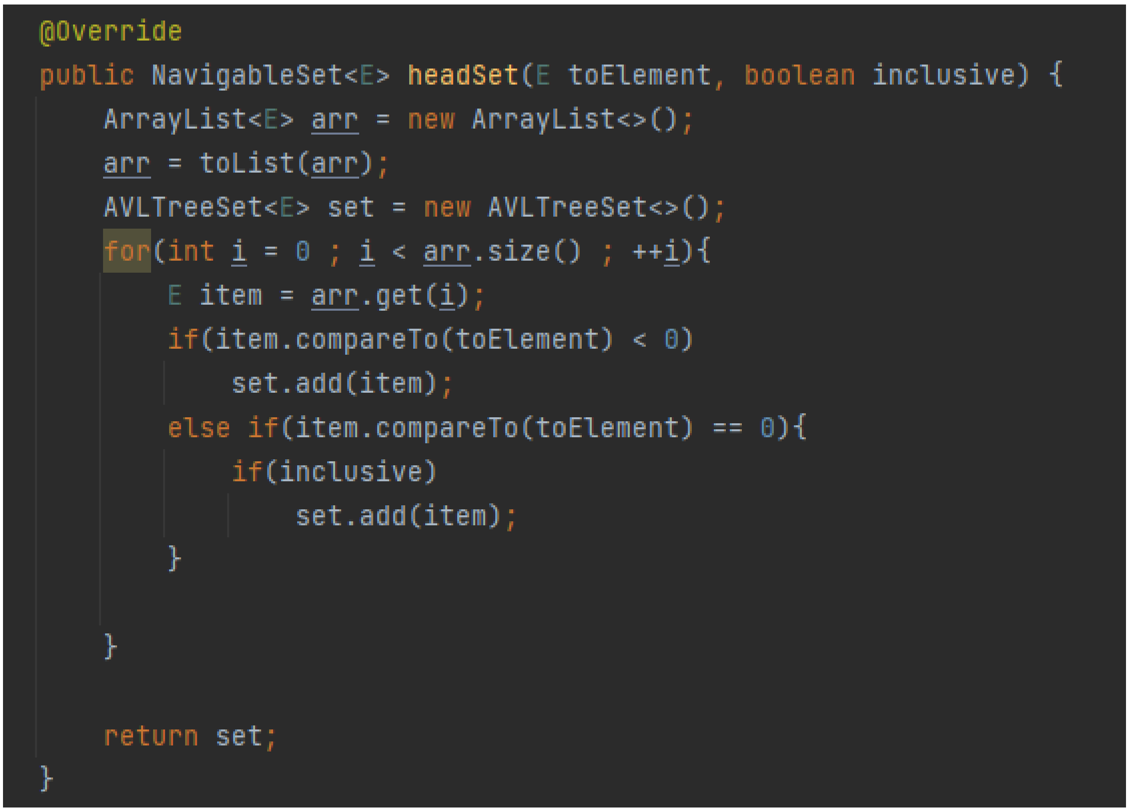


```
@Override
public <T> T[] toArray(T[] a) { return (T[]) toArray(); }

public class AVLSetIterator<T> implements Iterator<E>{
    private int pos;
    private E next;
    private Object[] arr;

    public AVLSetIterator(){
        arr = new Object[size];
        arr = toArray();
        pos = 0;
    }

    @Override
    public E next() { return (E) arr[pos++]; }

    @Override
    public boolean hasNext() {
        if(pos < size)
            return true;
        return false;
    }
}

@Override
public Iterator<E> iterator() { return new AVLSetIterator(); }
```

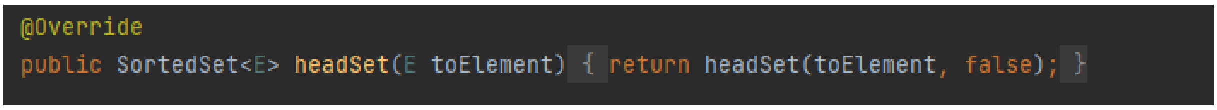
Iterator is implemented as inner class, this time object array is holded inside for iteration.

Iterator is view-only iterator and next and hasNext methods are implemented.

for headSet



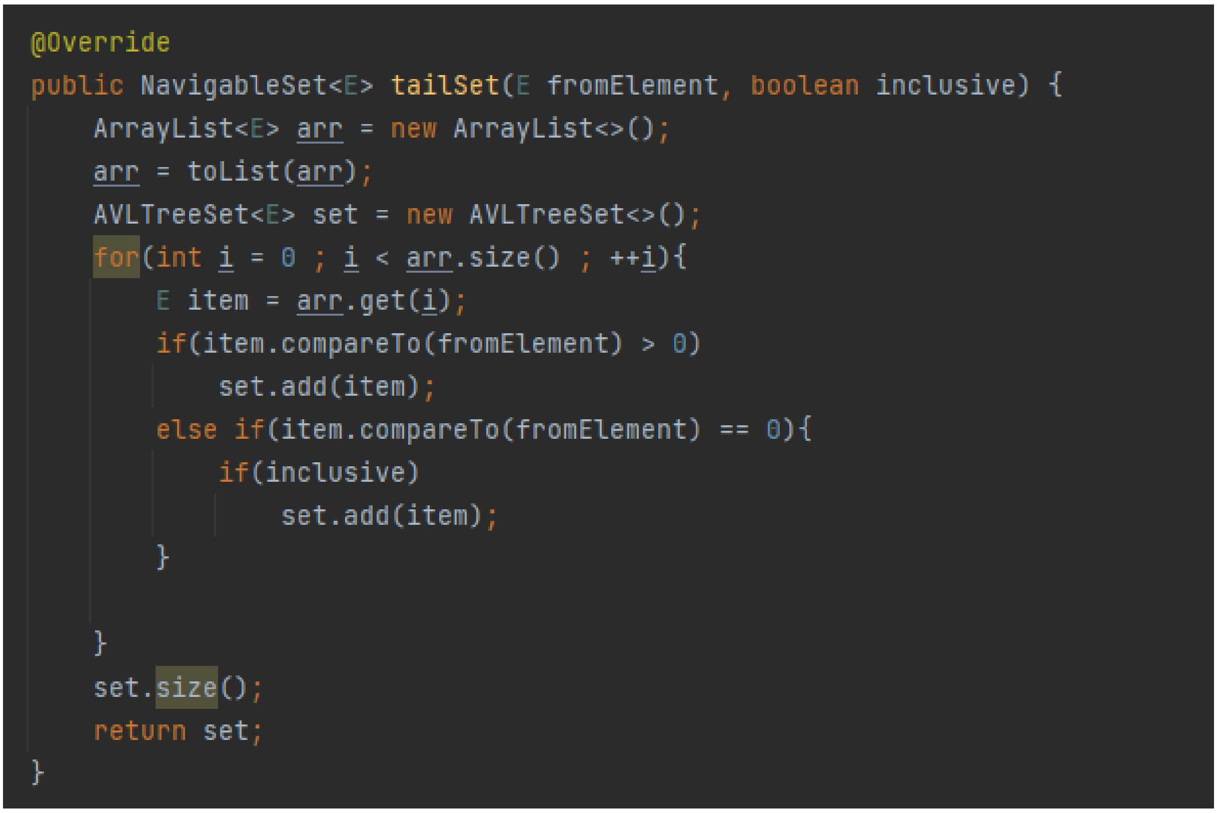
```java
@Override
public NavigableSet<E> headSet(E toElement, boolean inclusive) {
    ArrayList<E> arr = new ArrayList<>();
    arr = toList(arr);
    AVLTreeSet<E> set = new AVLTreeSet<>();
    for(int i = 0 ; i < arr.size() ; ++i){
        E item = arr.get(i);
        if(item.compareTo(toElement) < 0)
            set.add(item);
        else if(item.compareTo(toElement) == 0){
            if(inclusive)
                set.add(item);
        }

    }

    return set;
}
```



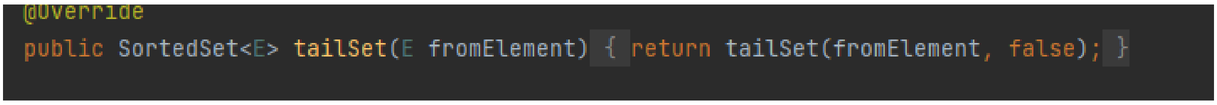
```java
@Override
public SortedSet<E> headSet(E toElement) { return headSet(toElement, false); }
```

new Set is built for wanted values.

for tailSet

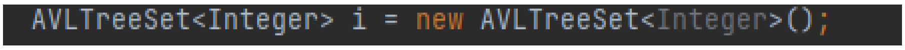


```
@Override
public NavigableSet<E> tailSet(E fromElement, boolean inclusive) {
    ArrayList<E> arr = new ArrayList<>();
    arr = toList(arr);
    AVLTreeSet<E> set = new AVLTreeSet<>();
    for(int i = 0 ; i < arr.size() ; ++i){
        E item = arr.get(i);
        if(item.compareTo(fromElement) > 0)
            set.add(item);
        else if(item.compareTo(fromElement) == 0){
            if(inclusive)
                set.add(item);
        }

    }
    set.size();
    return set;
}
```

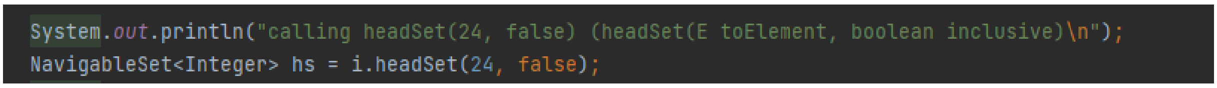


```
@Override
public SortedSet<E> tailSet(E fromElement) { return tailSet(fromElement, false); }
```

for these two methods, new arrayList is constructed with toList method, and new set is built and returned.

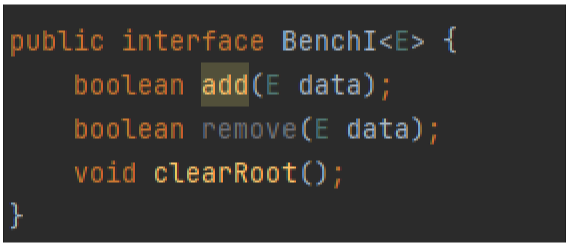


```
AVLTreeSet<Integer> i = new AVLTreeSet<Integer>();
```



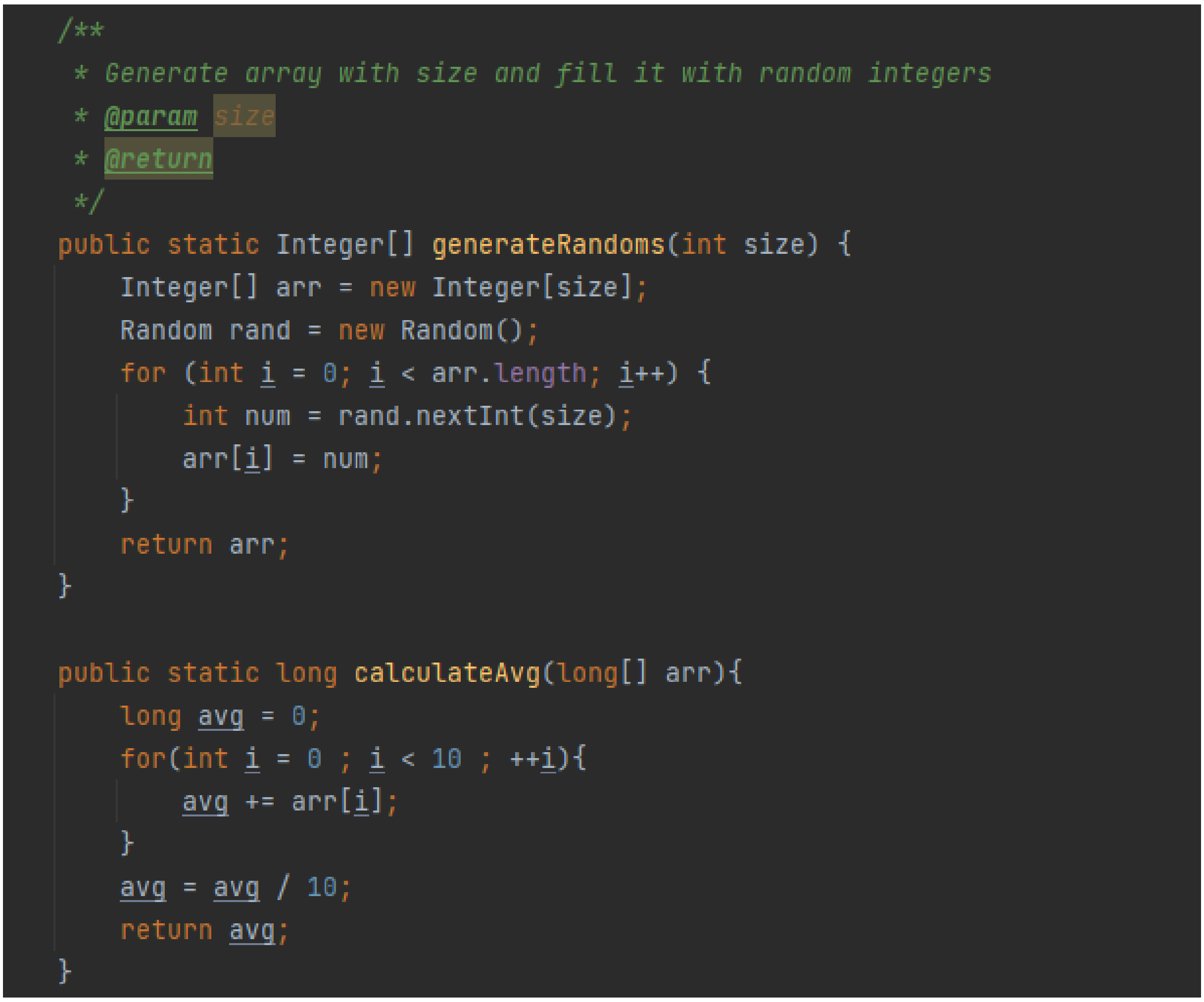
```
System.out.println("calling headSet(24, false) (headSet(E toElement, boolean inclusive)\n");
NavigableSet<Integer> hs = i.headSet(24, false);
```

for part3:



```java
public interface BenchI<E> {
    boolean add(E data);
    boolean remove(E data);
    void clearRoot();
}
```

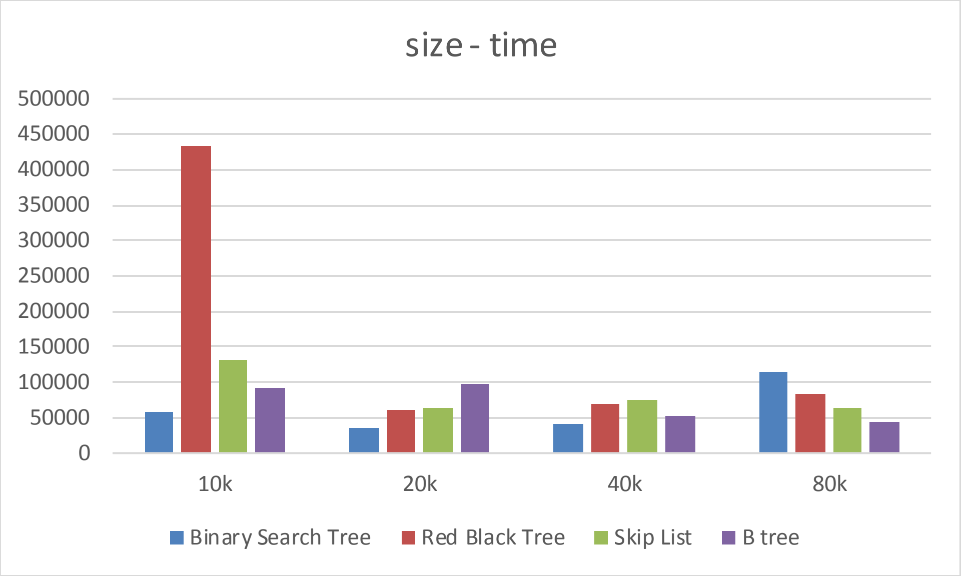
A new interface is created for wrapping data structures to make testing easier.



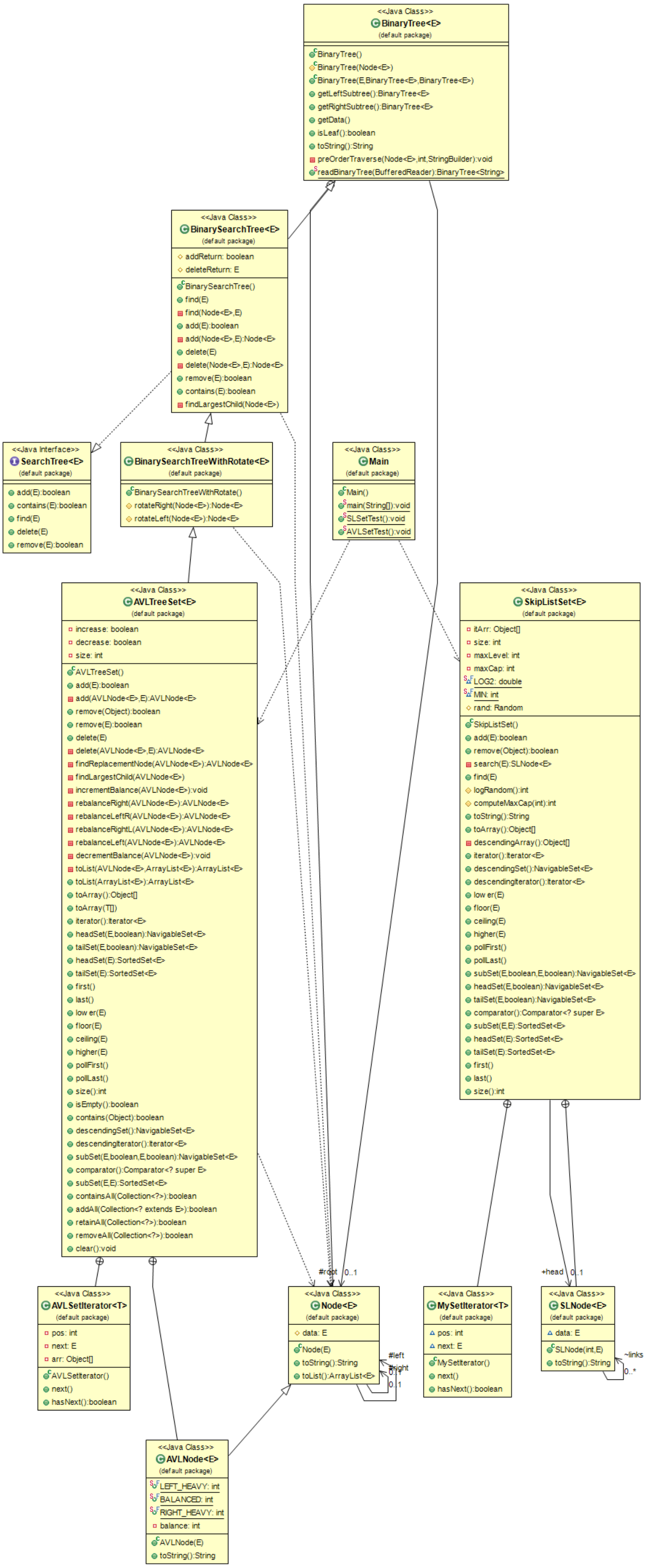
```java
    /**
     * Generate array with size and fill it with random integers
     * @param size
     * @return
     */
    public static Integer[] generateRandoms(int size) {
        Integer[] arr = new Integer[size];
        Random rand = new Random();
        for (int i = 0; i < arr.length; i++) {
            int num = rand.nextInt(size);
            arr[i] = num;
        }
        return arr;
    }

    public static long calculateAvg(long[] arr){
        long avg = 0;
        for(int i = 0 ; i < 10 ; ++i){
            avg += arr[i];
        }
        avg = avg / 10;
        return avg;
    }
```

Helpers for part3, generic test method is included in test case section.

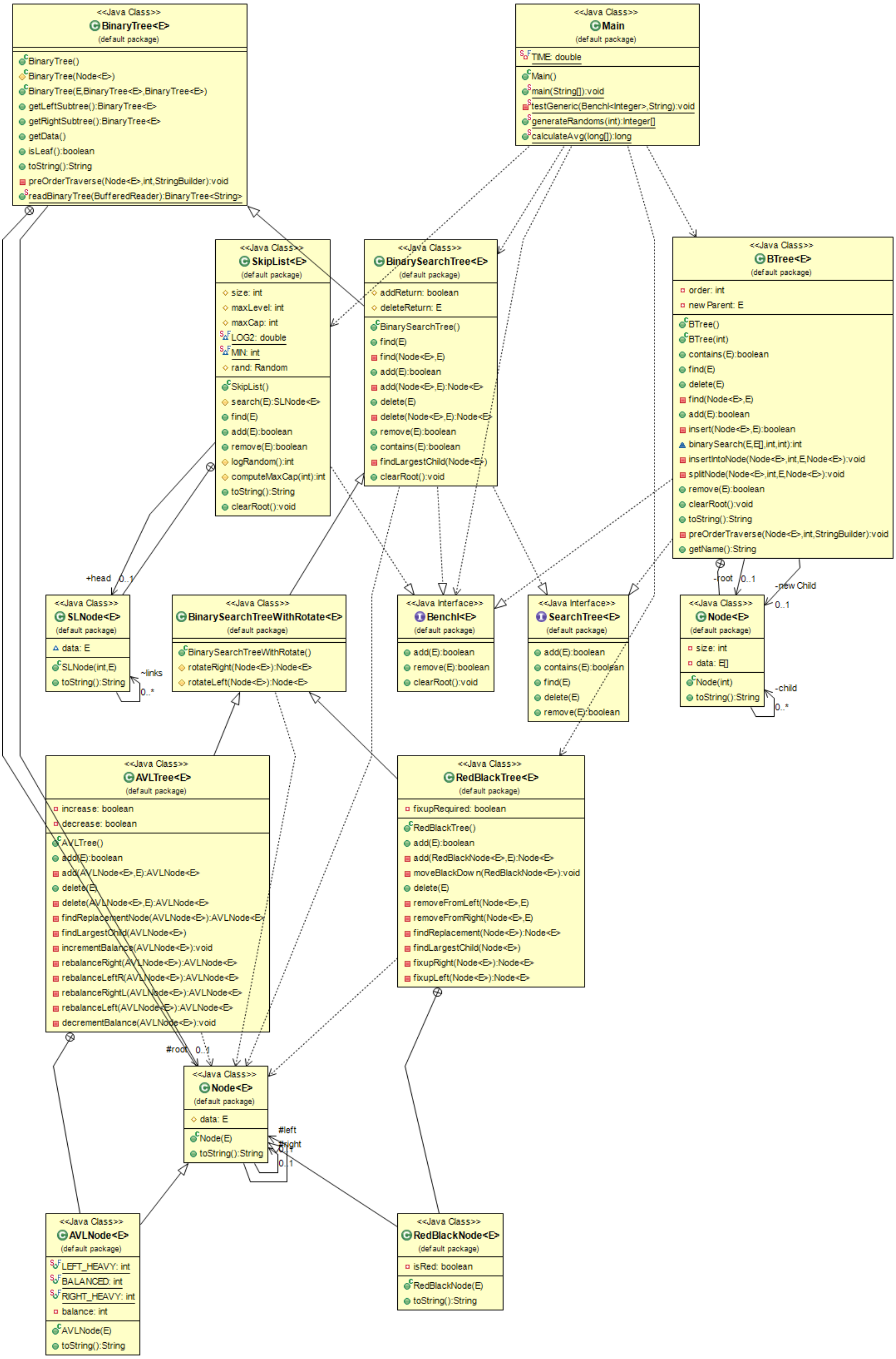
Graph for part3



3 – Class Diagrams Part1



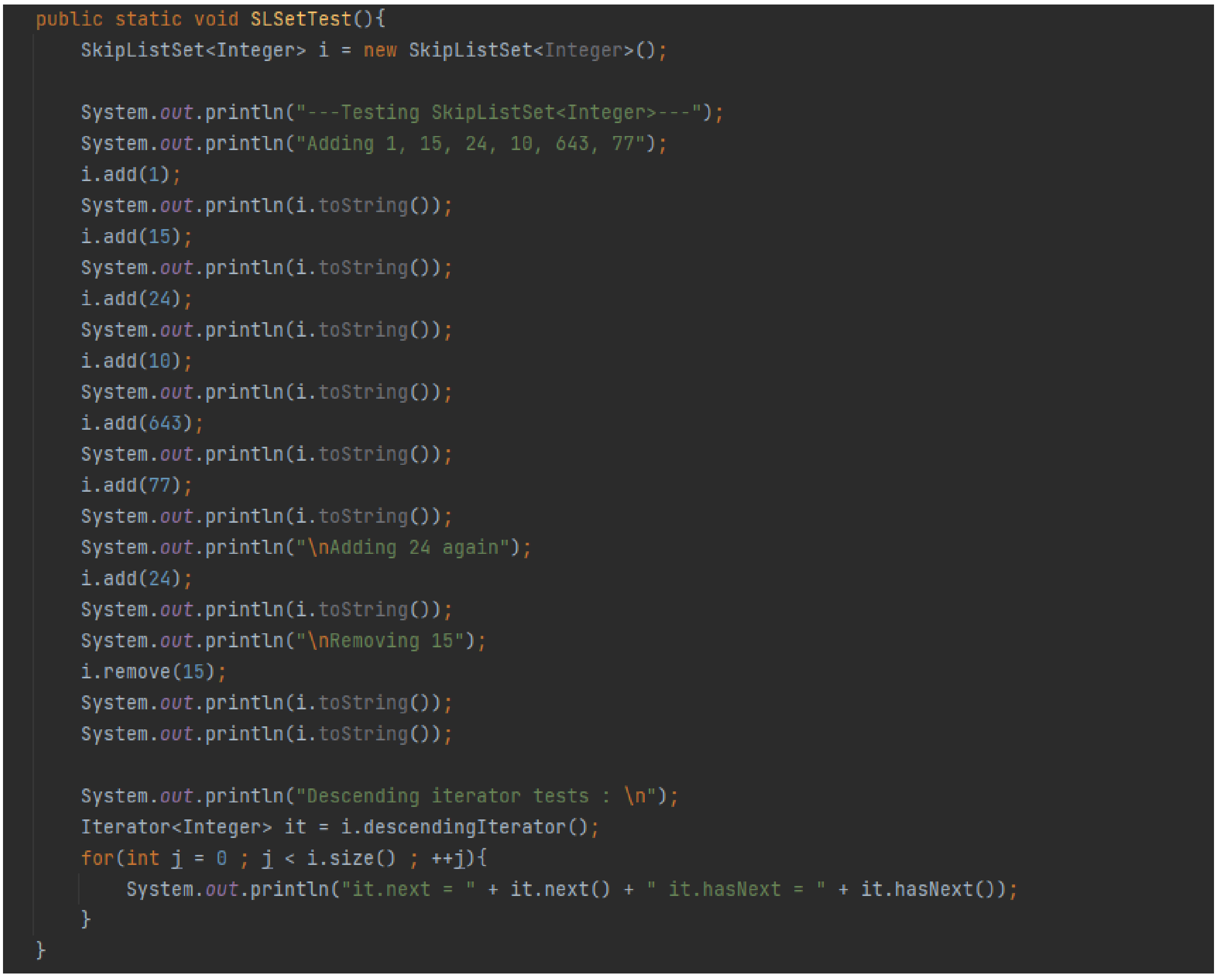
Part3



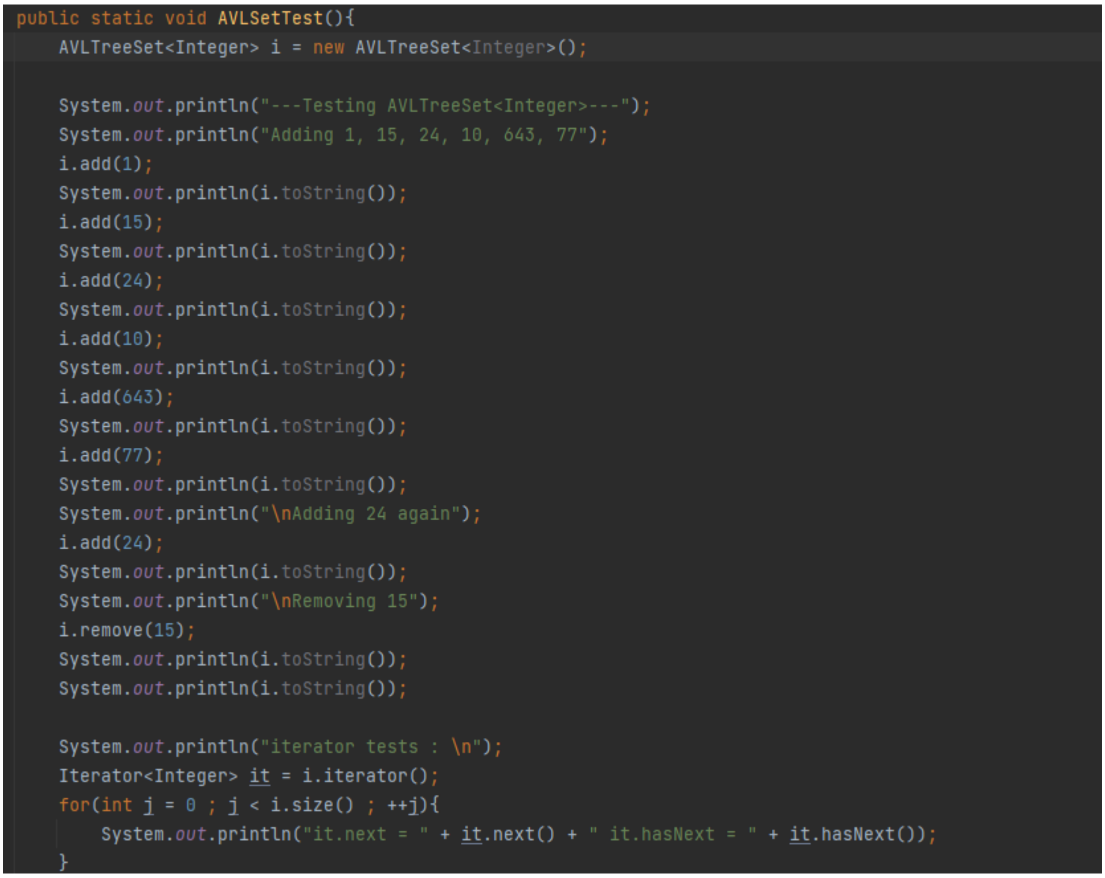
<<Java Class>>
BinaryTree<E>
(default package)
BinaryTree()
BinaryTree(Node<E>)
BinaryTree(E,BinaryTree<E>,BinaryTree<E>)
getLeftSubtree():BinaryTree<E>
getRightSubtree():BinaryTree<E>
getData()
isLeaf():boolean
toString():String
preOrderTraverse(Node<E>,int,StringBuilder):void
readBinaryTree(BufferedReader):BinaryTree<String>
<<Java Class>>
Main
(default package)
TIME: double
Main()
main(String[]):void
testGeneric(Benchl<Integer>,String):void
generateRandoms(int):Integer[]
calculateAvg(long[]):long
<<Java Class>>
SkipList<E>
(default package)
size: int
maxLevel: int
maxCap: int
LOG2: double
MIN: int
rand: Random
SkipList()
search(E):SLNode<E>
find(E)
add(E):boolean
remove(E):boolean
logRandom():int
computeMaxCap(int):int
toString():String
clearRoot():void
<<Java Class>>
BinarySearchTree<E>
(default package)
addReturn: boolean
deleteReturn: E
BinarySearchTree()
find(E)
find(Node<E>,E)
add(E):boolean
add(Node<E>,E):Node<E>
delete(E)
delete(Node<E>,E):Node<E>
remove(E):boolean
contains(E):boolean
findLargestChild(Node<E>)
clearRoot():void
<<Java Class>>
BTree<E>
(default package)
order: int
new Parent: E
BTree()
BTree(int)
contains(E):boolean
find(E)
delete(E)
find(Node<E>,E)
add(E):boolean
insert(Node<E>,E):boolean
binarySearch(E,E[],int,int):int
insertIntoNode(Node<E>,int,E,Node<E>):void
splitNode(Node<E>,int,E,Node<E>):void
remove(E):boolean
clearRoot():void
toString():String
preOrderTraverse(Node<E>,int,StringBuilder):void
getName():String
+head
0..1
<<Java Class>>
SLNode<E>
(default package)
data: E
SLNode(int,E)
toString():String
~links
0..*
<<Java Class>>
BinarySearchTreeWithRotate<E>
(default package)
BinarySearchTreeWithRotate()
rotateRight(Node<E>):Node<E>
rotateLeft(Node<E>):Node<E>
<<Java Interface>>
Benchl<E>
(default package)
add(E):boolean
remove(E):boolean
clearRoot():void
<<Java Interface>>
SearchTree<E>
(default package)
add(E):boolean
contains(E):boolean
find(E)
delete(E)
remove(E):boolean
-root
0..1
-new Child
0..1
<<Java Class>>
Node<E>
(default package)
size: int
data: E[]
Node(int)
toString():String
-child
0..*
<<Java Class>>
AVLTree<E>
(default package)
increase: boolean
decrease: boolean
AVLTree()
add(E):boolean
add(AVLNode<E>,E):AVLNode<E>
delete(E)
delete(AVLNode<E>,E):AVLNode<E>
findReplacementNode(AVLNode<E>):AVLNode<E>
findLargestChild(AVLNode<E>)
incrementBalance(AVLNode<E>):void
rebalanceRight(AVLNode<E>):AVLNode<E>
rebalanceLeftR(AVLNode<E>):AVLNode<E>
rebalanceRightL(AVLNode<E>):AVLNode<E>
rebalanceLeft(AVLNode<E>):AVLNode<E>
decrementBalance(AVLNode<E>):void
<<Java Class>>
RedBlackTree<E>
(default package)
fixupRequired: boolean
RedBlackTree()
add(E):boolean
add(RedBlackNode<E>,E):Node<E>
moveBlackDown(RedBlackNode<E>):void
delete(E)
removeFromLeft(Node<E>,E)
removeFromRight(Node<E>,E)
findReplacement(Node<E>):Node<E>
findLargestChild(Node<E>)
fixupRight(Node<E>):Node<E>
fixupLeft(Node<E>):Node<E>
#root
0..1
<<Java Class>>
Node<E>
(default package)
data: E
Node(E)
toString():String
#left
#right
0..1
0..1
<<Java Class>>
AVLNode<E>
(default package)
LEFT_HEAVY: int
BALANCED: int
RIGHT_HEAVY: int
balance: int
AVLNode(E)
toString():String
<<Java Class>>
RedBlackNode<E>
(default package)
isRed: boolean
RedBlackNode(E)
toString():String

3 – Test cases

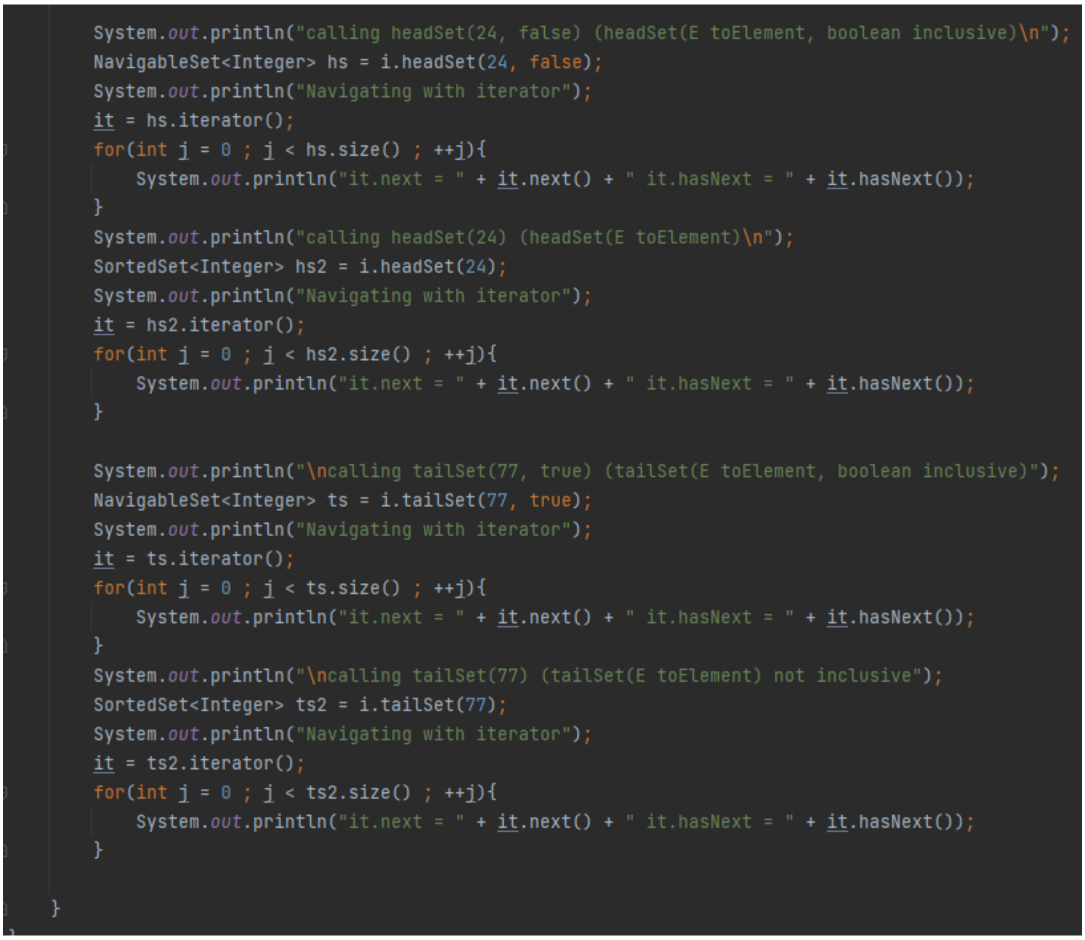
part1



```
public static void SLSetTest(){
    SkipListSet<Integer> i = new SkipListSet<Integer>();

    System.out.println("---Testing SkipListSet<Integer>---");
    System.out.println("Adding 1, 15, 24, 10, 643, 77");
    i.add(1);
    System.out.println(i.toString());
    i.add(15);
    System.out.println(i.toString());
    i.add(24);
    System.out.println(i.toString());
    i.add(10);
    System.out.println(i.toString());
    i.add(643);
    System.out.println(i.toString());
    i.add(77);
    System.out.println(i.toString());
    System.out.println("\nAdding 24 again");
    i.add(24);
    System.out.println(i.toString());
    System.out.println("\nRemoving 15");
    i.remove(15);
    System.out.println(i.toString());
    System.out.println(i.toString());

    System.out.println("Descending iterator tests : \n");
    Iterator<Integer> it = i.descendingIterator();
    for(int j = 0 ; j < i.size() ; ++j){
        System.out.println("it.next = " + it.next() + " it.hasNext = " + it.hasNext());
    }
}
```

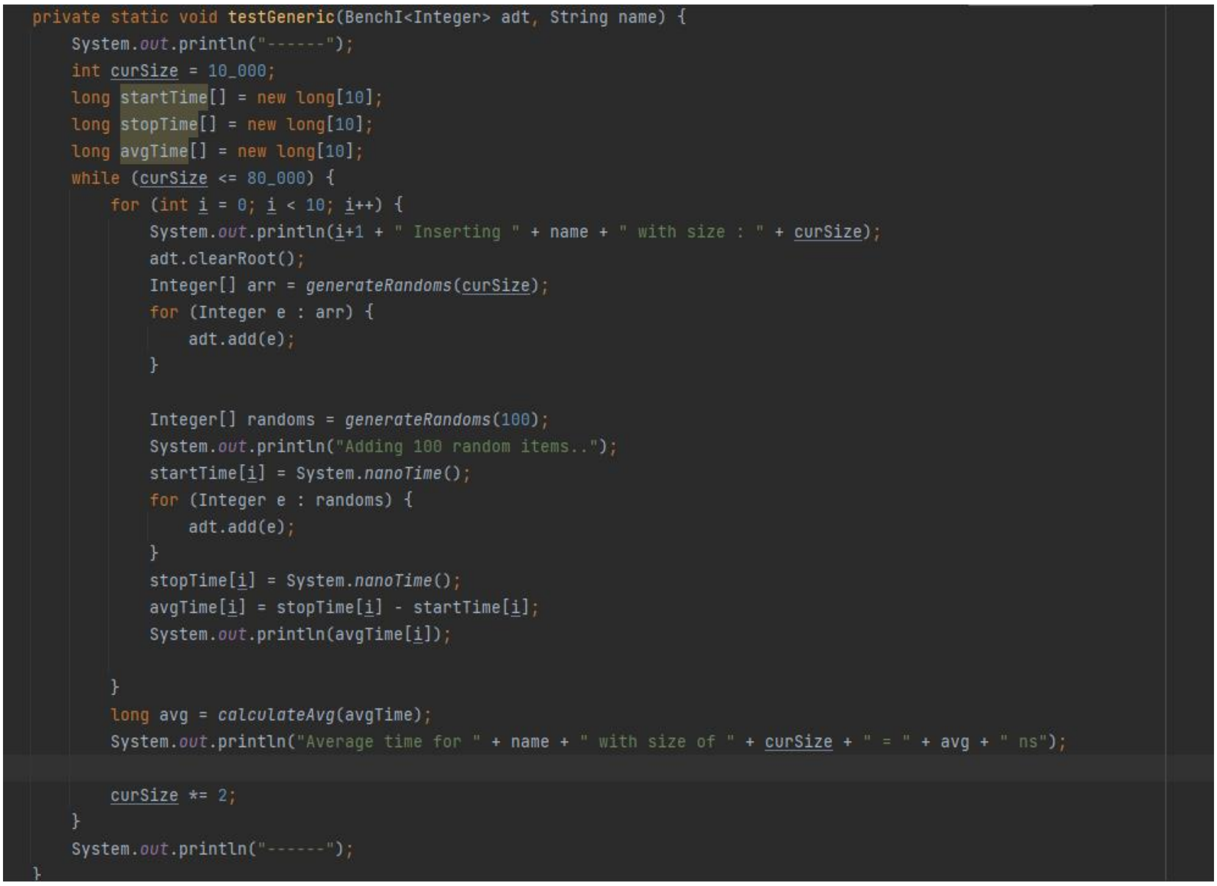


```java
public static void AVLSetTest(){
    AVLTreeSet<Integer> i = new AVLTreeSet<Integer>();

    System.out.println("---Testing AVLTreeSet<Integer>---");
    System.out.println("Adding 1, 15, 24, 10, 643, 77");
    i.add(1);
    System.out.println(i.toString());
    i.add(15);
    System.out.println(i.toString());
    i.add(24);
    System.out.println(i.toString());
    i.add(10);
    System.out.println(i.toString());
    i.add(643);
    System.out.println(i.toString());
    i.add(77);
    System.out.println(i.toString());
    System.out.println("\nAdding 24 again");
    i.add(24);
    System.out.println(i.toString());
    System.out.println("\nRemoving 15");
    i.remove(15);
    System.out.println(i.toString());
    System.out.println(i.toString());

    System.out.println("iterator tests : \n");
    Iterator<Integer> it = i.iterator();
    for(int j = 0 ; j < i.size() ; ++j){
        System.out.println("it.next = " + it.next() + " it.hasNext = " + it.hasNext());
    }
```

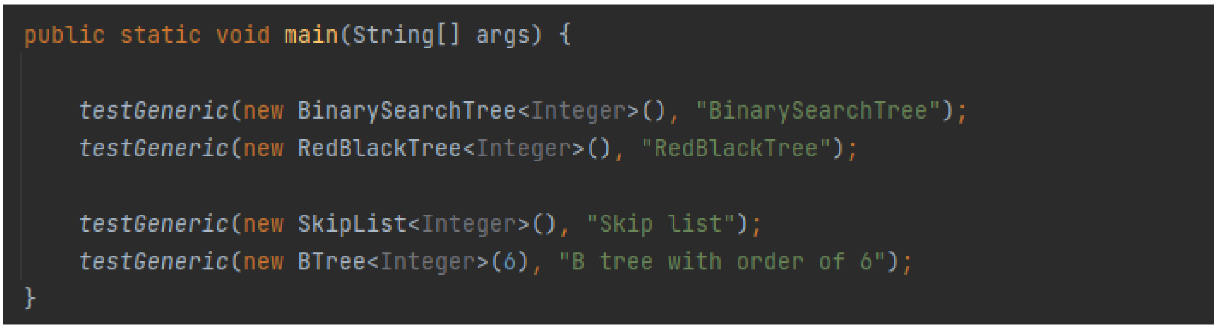


```java
        System.out.println("calling headSet(24, false) (headSet(E toElement, boolean inclusive)\n");
        NavigableSet<Integer> hs = i.headSet(24, false);
        System.out.println("Navigating with iterator");
        it = hs.iterator();
        for(int j = 0 ; j < hs.size() ; ++j){
            System.out.println("it.next = " + it.next() + " it.hasNext = " + it.hasNext());
        }
        System.out.println("calling headSet(24) (headSet(E toElement)\n");
        SortedSet<Integer> hs2 = i.headSet(24);
        System.out.println("Navigating with iterator");
        it = hs2.iterator();
        for(int j = 0 ; j < hs2.size() ; ++j){
            System.out.println("it.next = " + it.next() + " it.hasNext = " + it.hasNext());
        }

        System.out.println("\ncalling tailSet(77, true) (tailSet(E toElement, boolean inclusive)");
        NavigableSet<Integer> ts = i.tailSet(77, true);
        System.out.println("Navigating with iterator");
        it = ts.iterator();
        for(int j = 0 ; j < ts.size() ; ++j){
            System.out.println("it.next = " + it.next() + " it.hasNext = " + it.hasNext());
        }
        System.out.println("\ncalling tailSet(77) (tailSet(E toElement) not inclusive");
        SortedSet<Integer> ts2 = i.tailSet(77);
        System.out.println("Navigating with iterator");
        it = ts2.iterator();
        for(int j = 0 ; j < ts2.size() ; ++j){
            System.out.println("it.next = " + it.next() + " it.hasNext = " + it.hasNext());
        }

    }
```

part3



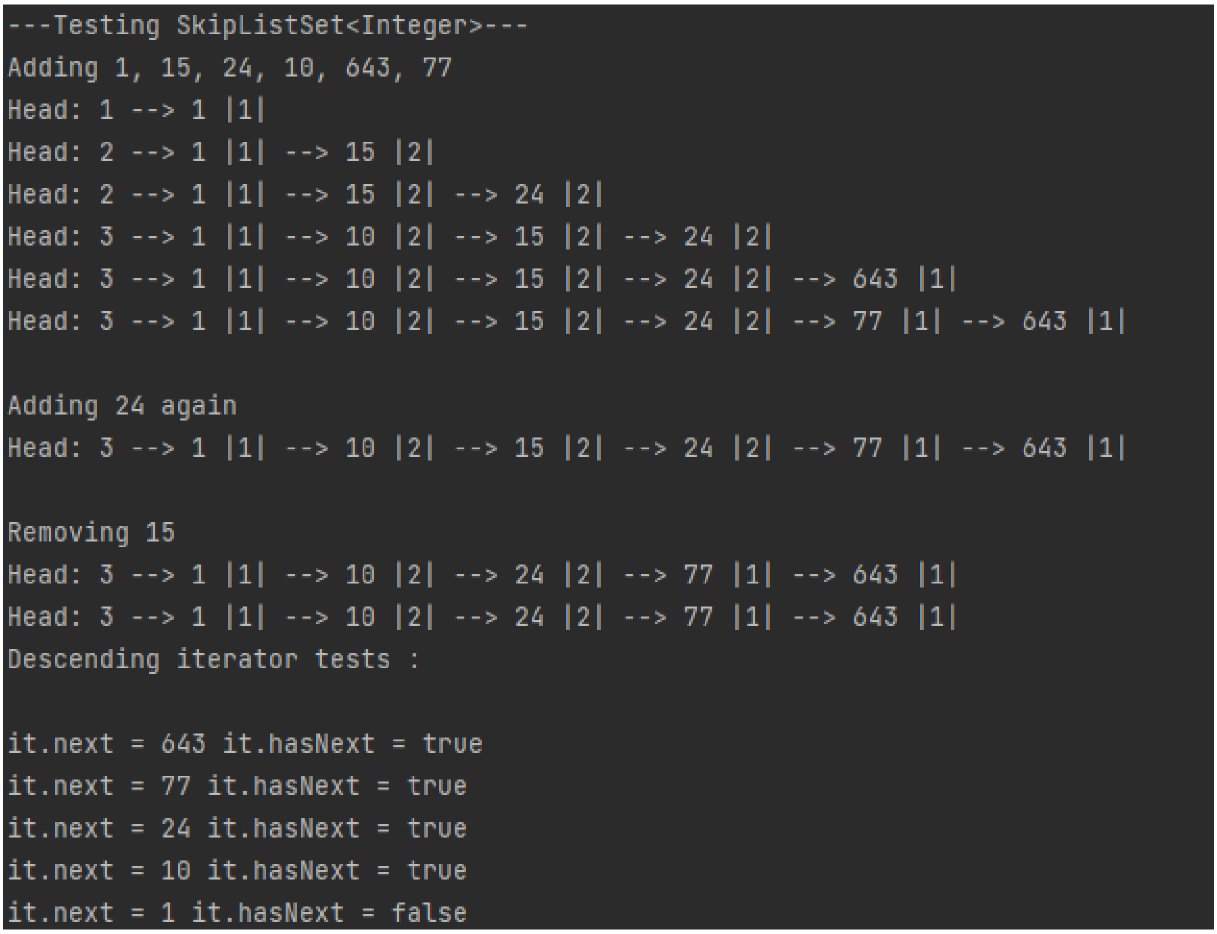
```java
private static void testGeneric(BenchI<Integer> adt, String name) {
    System.out.println("------");
    int curSize = 10_000;
    long startTime[] = new long[10];
    long stopTime[] = new long[10];
    long avgTime[] = new long[10];
    while (curSize <= 80_000) {
        for (int i = 0; i < 10; i++) {
            System.out.println(i+1 + " Inserting " + name + " with size : " + curSize);
            adt.clearRoot();
            Integer[] arr = generateRandoms(curSize);
            for (Integer e : arr) {
                adt.add(e);
            }

            Integer[] randoms = generateRandoms(100);
            System.out.println("Adding 100 random items..");
            startTime[i] = System.nanoTime();
            for (Integer e : randoms) {
                adt.add(e);
            }
            stopTime[i] = System.nanoTime();
            avgTime[i] = stopTime[i] - startTime[i];
            System.out.println(avgTime[i]);

        }
        long avg = calculateAvg(avgTime);
        System.out.println("Average time for " + name + " with size of " + curSize + " = " + avg + " ns");

        curSize *= 2;
    }
    System.out.println("------");
}
```



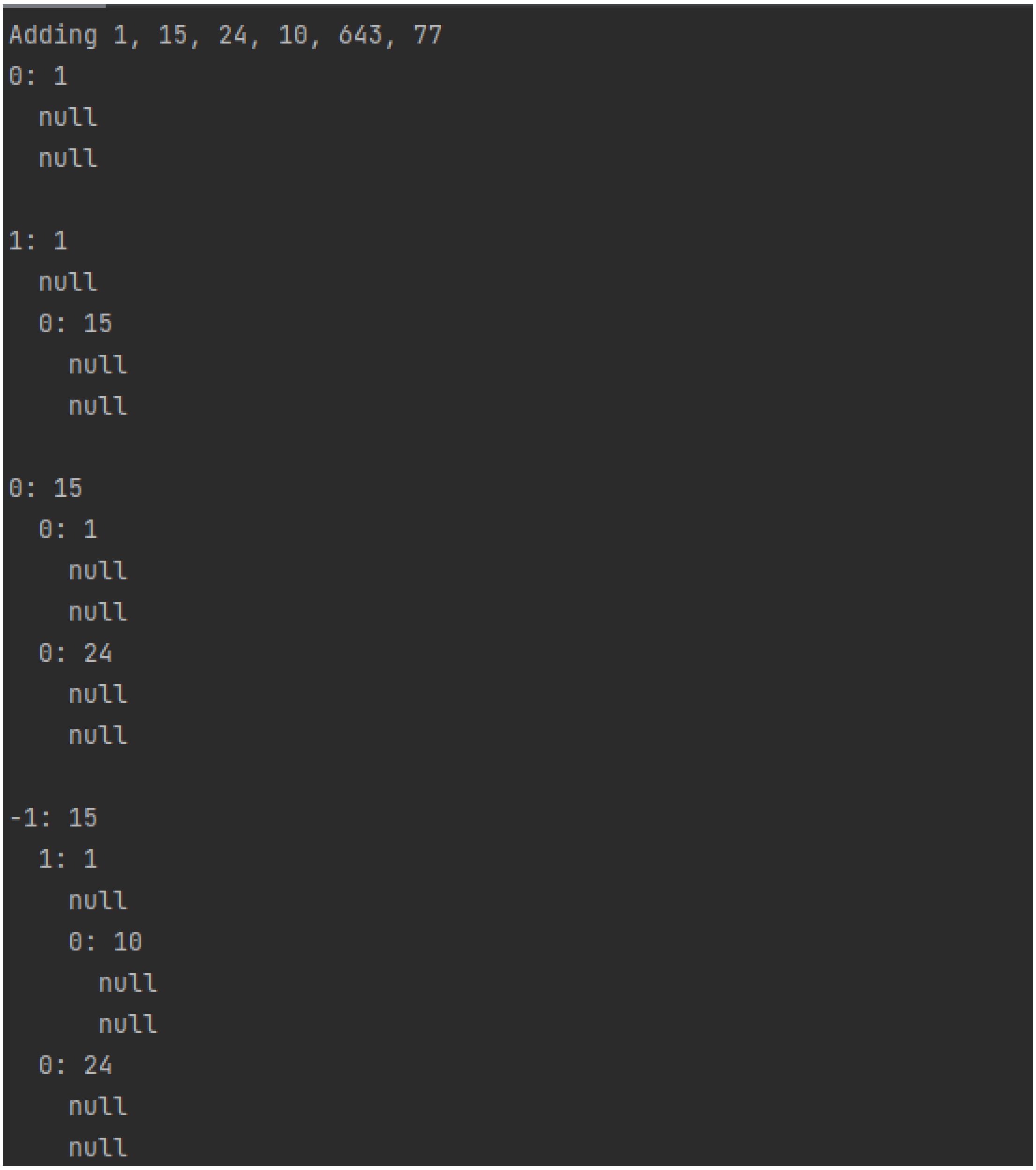
```java
public static void main(String[] args) {

    testGeneric(new BinarySearchTree<Integer>(), "BinarySearchTree");
    testGeneric(new RedBlackTree<Integer>(), "RedBlackTree");

    testGeneric(new SkipList<Integer>(), "Skip list");
    testGeneric(new BTree<Integer>(6), "B tree with order of 6");
}
```

4 – Running commands and results

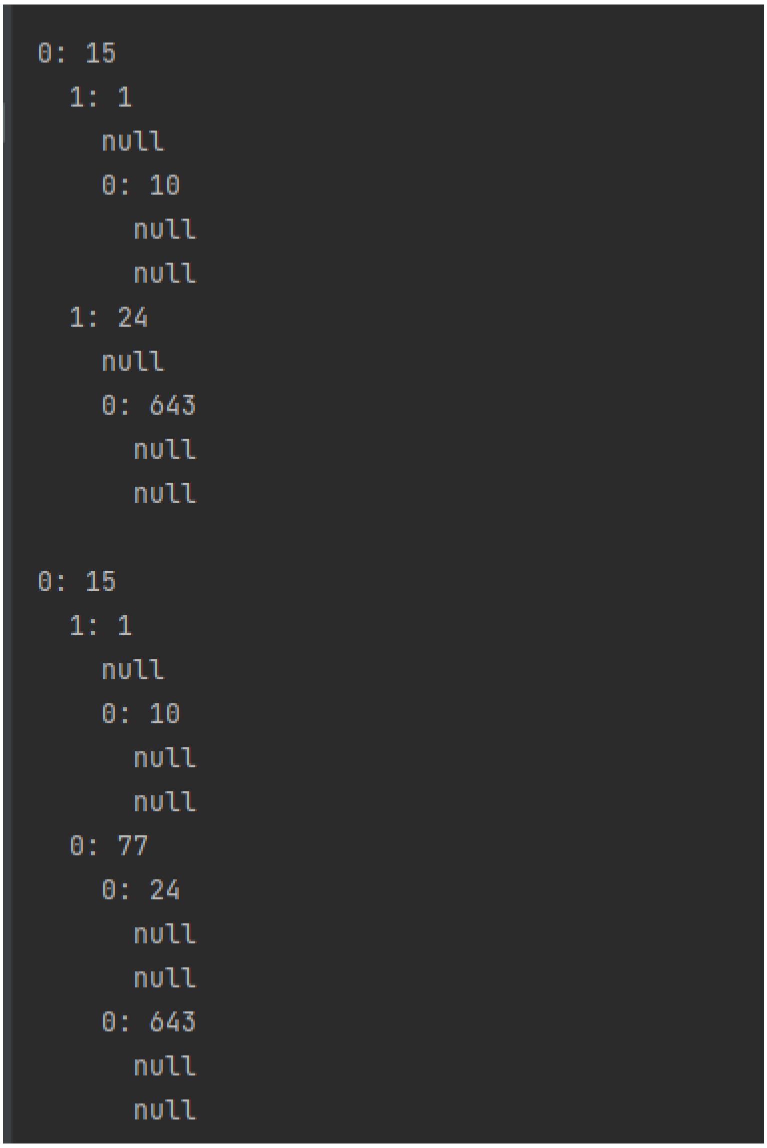
results of part1 :



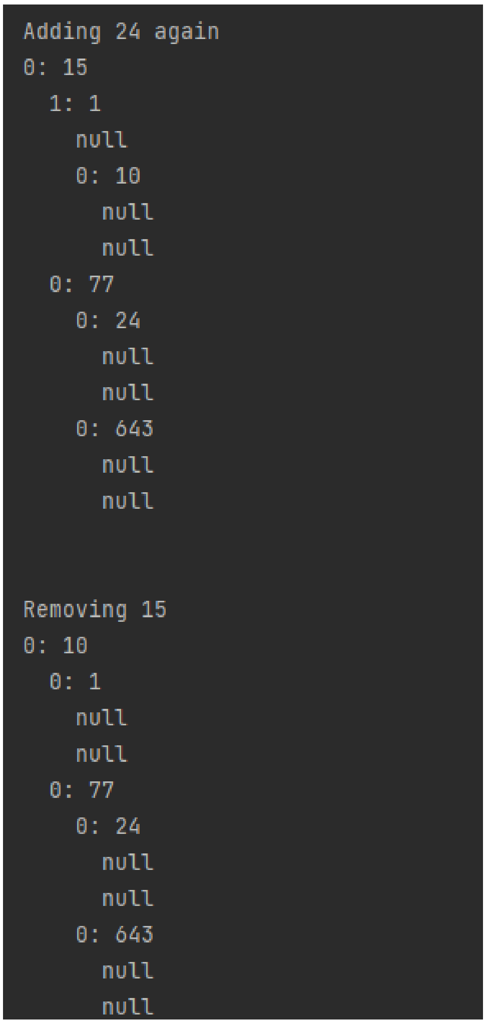
```
---Testing SkipListSet<Integer>---
Adding 1, 15, 24, 10, 643, 77
Head: 1 --> 1 |1|
Head: 2 --> 1 |1| --> 15 |2|
Head: 2 --> 1 |1| --> 15 |2| --> 24 |2|
Head: 3 --> 1 |1| --> 10 |2| --> 15 |2| --> 24 |2|
Head: 3 --> 1 |1| --> 10 |2| --> 15 |2| --> 24 |2| --> 643 |1|
Head: 3 --> 1 |1| --> 10 |2| --> 15 |2| --> 24 |2| --> 77 |1| --> 643 |1|

Adding 24 again
Head: 3 --> 1 |1| --> 10 |2| --> 15 |2| --> 24 |2| --> 77 |1| --> 643 |1|

Removing 15
Head: 3 --> 1 |1| --> 10 |2| --> 24 |2| --> 77 |1| --> 643 |1|
Head: 3 --> 1 |1| --> 10 |2| --> 24 |2| --> 77 |1| --> 643 |1|
Descending iterator tests :

it.next = 643 it.hasNext = true
it.next = 77 it.hasNext = true
it.next = 24 it.hasNext = true
it.next = 10 it.hasNext = true
it.next = 1 it.hasNext = false
```



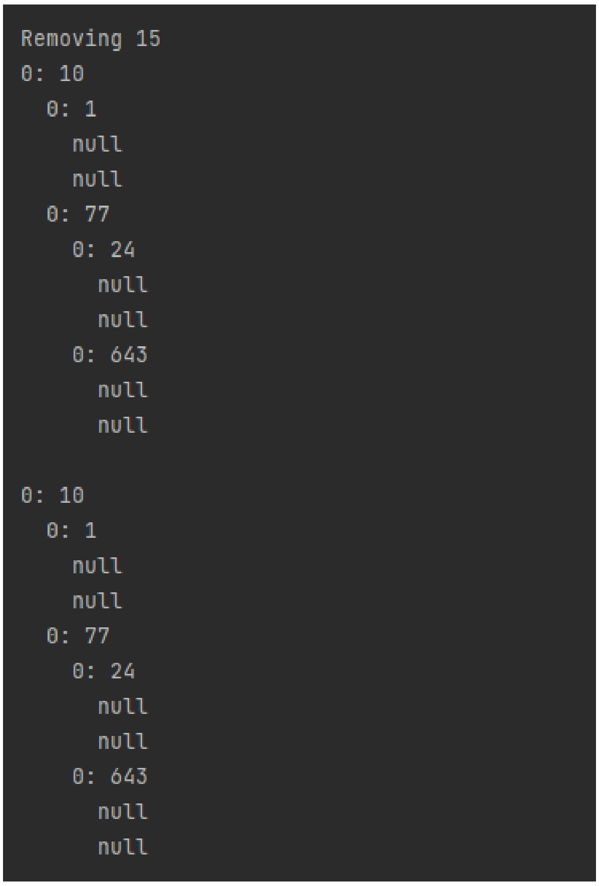
```
Adding 1, 15, 24, 10, 643, 77
0: 1
  null
  null

1: 1
  null
  0: 15
    null
    null

0: 15
  0: 1
    null
    null
  0: 24
    null
    null

-1: 15
  1: 1
    null
    0: 10
      null
      null
  0: 24
    null
    null
```



```
0: 15
  1: 1
    null
    0: 10
      null
      null
  1: 24
    null
    0: 643
      null
      null

0: 15
  1: 1
    null
    0: 10
      null
      null
  0: 77
    0: 24
      null
      null
    0: 643
      null
      null
```

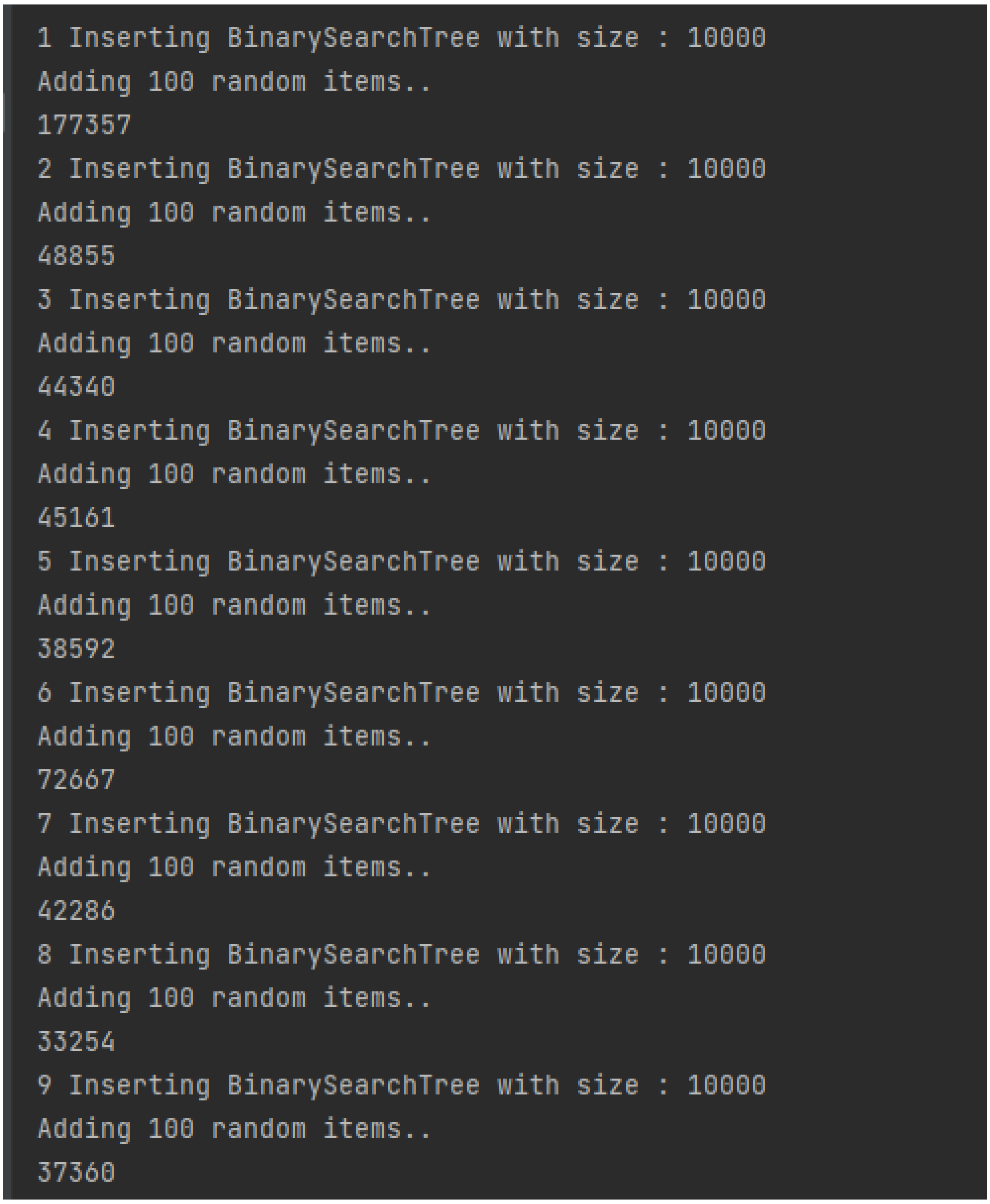


```
Adding 24 again
0: 15
  1: 1
    null
    0: 10
      null
      null
  0: 77
    0: 24
      null
      null
    0: 643
      null
      null


Removing 15
0: 10
  0: 1
    null
    null
  0: 77
    0: 24
      null
      null
    0: 643
      null
      null
```

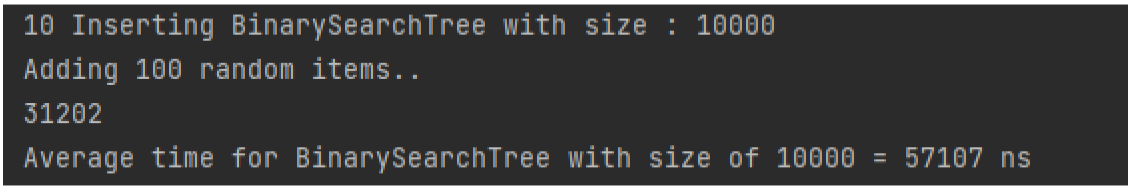


```
Removing 15
0: 10
  0: 1
    null
    null
  0: 77
    0: 24
      null
      null
    0: 643
      null
      null

0: 10
  0: 1
    null
    null
  0: 77
    0: 24
      null
      null
    0: 643
      null
      null
```

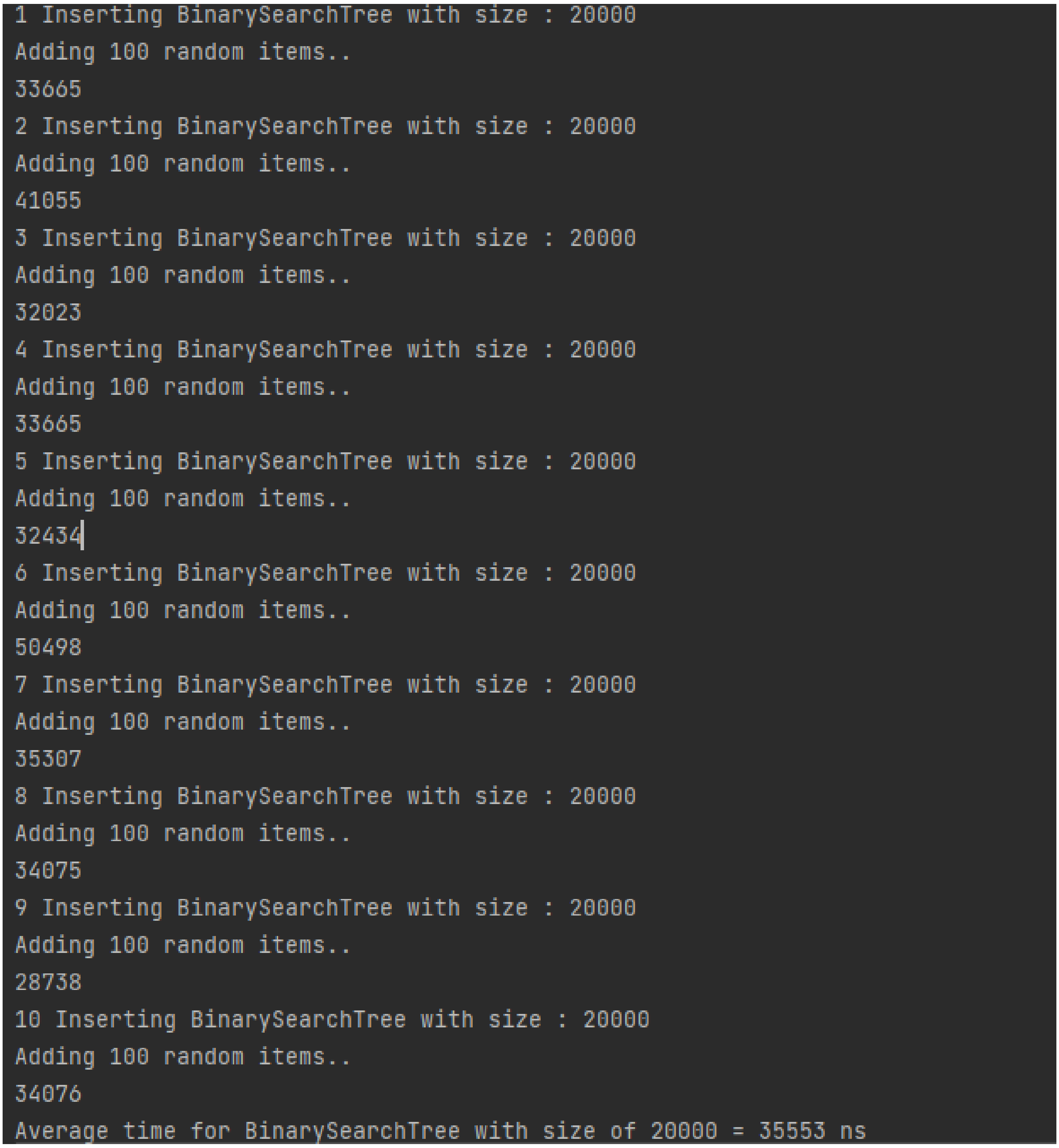
Results of part3 :



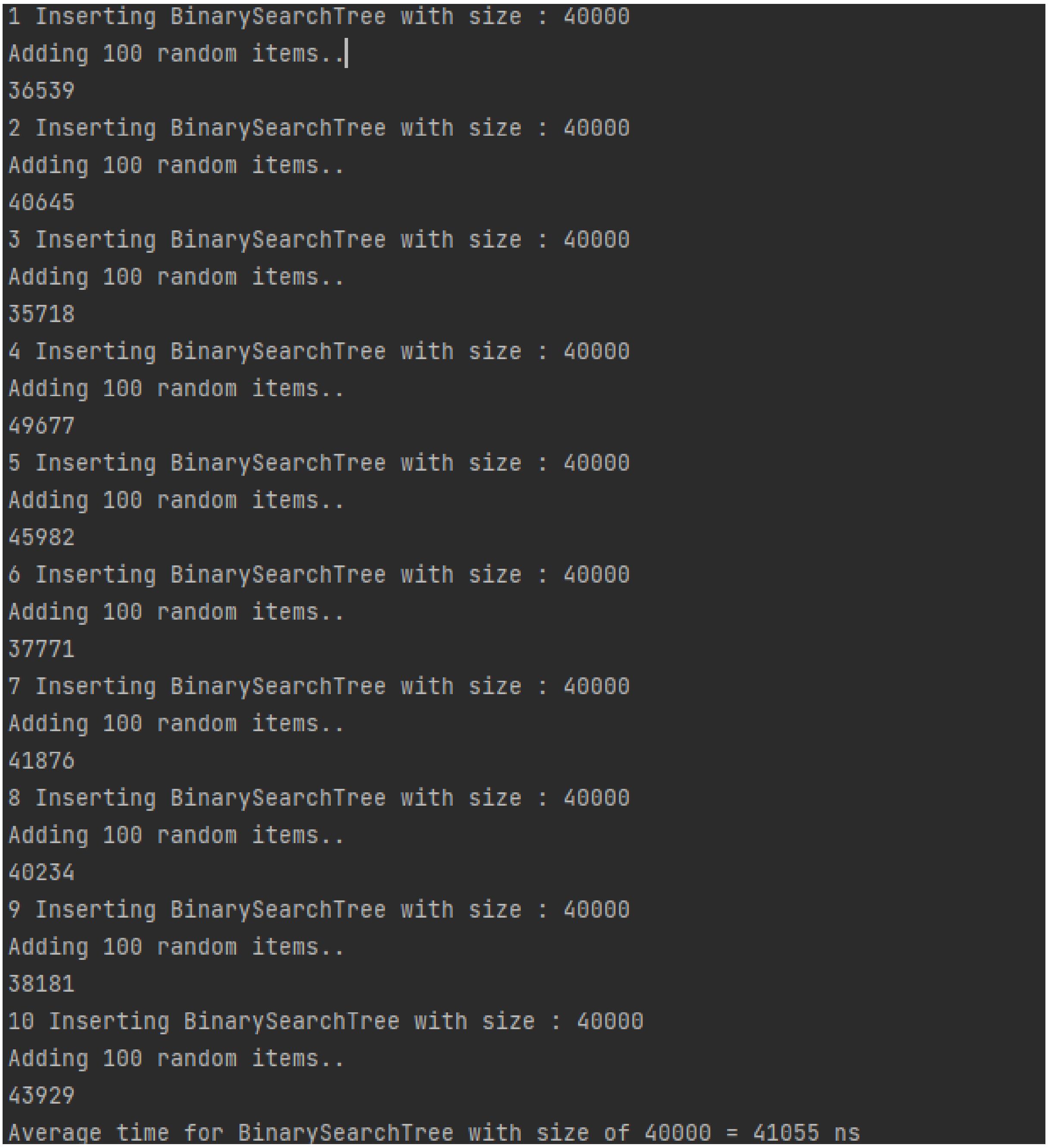
```
1 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
177357
2 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
48855
3 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
44340
4 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
45161
5 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
38592
6 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
72667
7 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
42286
8 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
33254
9 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
37360
```



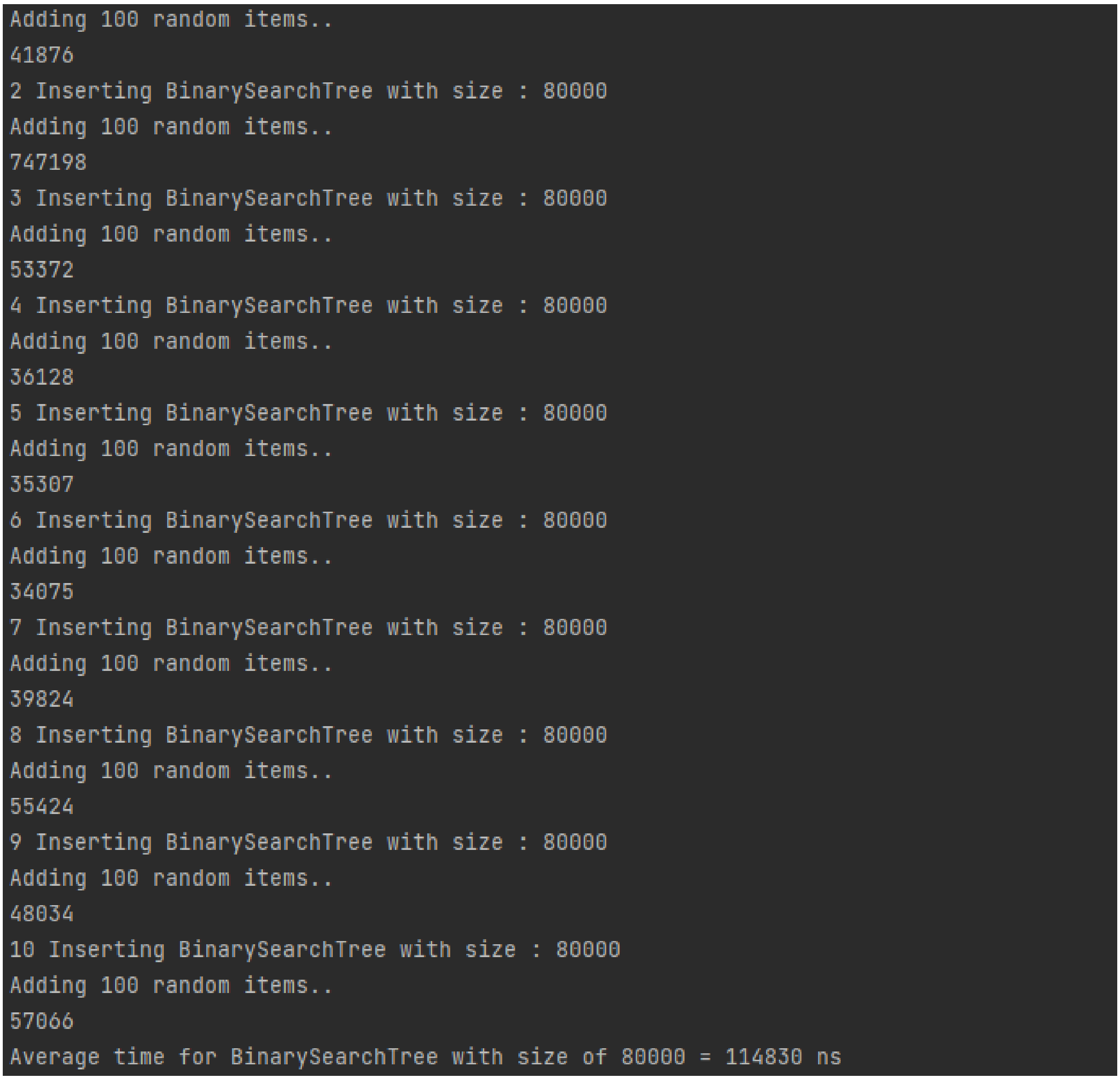
```
10 Inserting BinarySearchTree with size : 10000
Adding 100 random items..
31202
Average time for BinarySearchTree with size of 10000 = 57107 ns
```



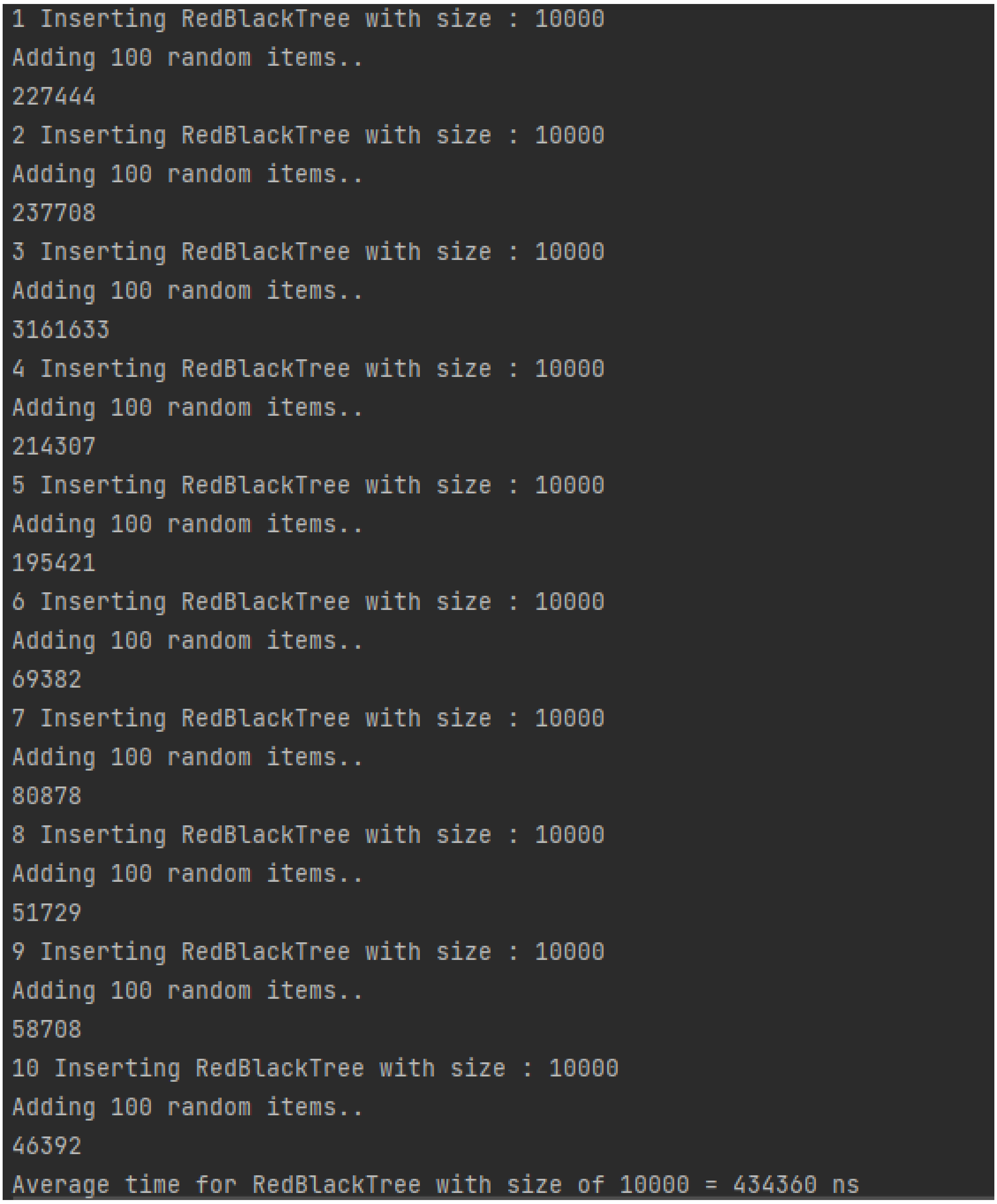
```
1 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
33665
2 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
41055
3 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
32023
4 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
33665
5 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
32434
6 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
50498
7 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
35307
8 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
34075
9 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
28738
10 Inserting BinarySearchTree with size : 20000
Adding 100 random items..
34076
Average time for BinarySearchTree with size of 20000 = 35553 ns
```



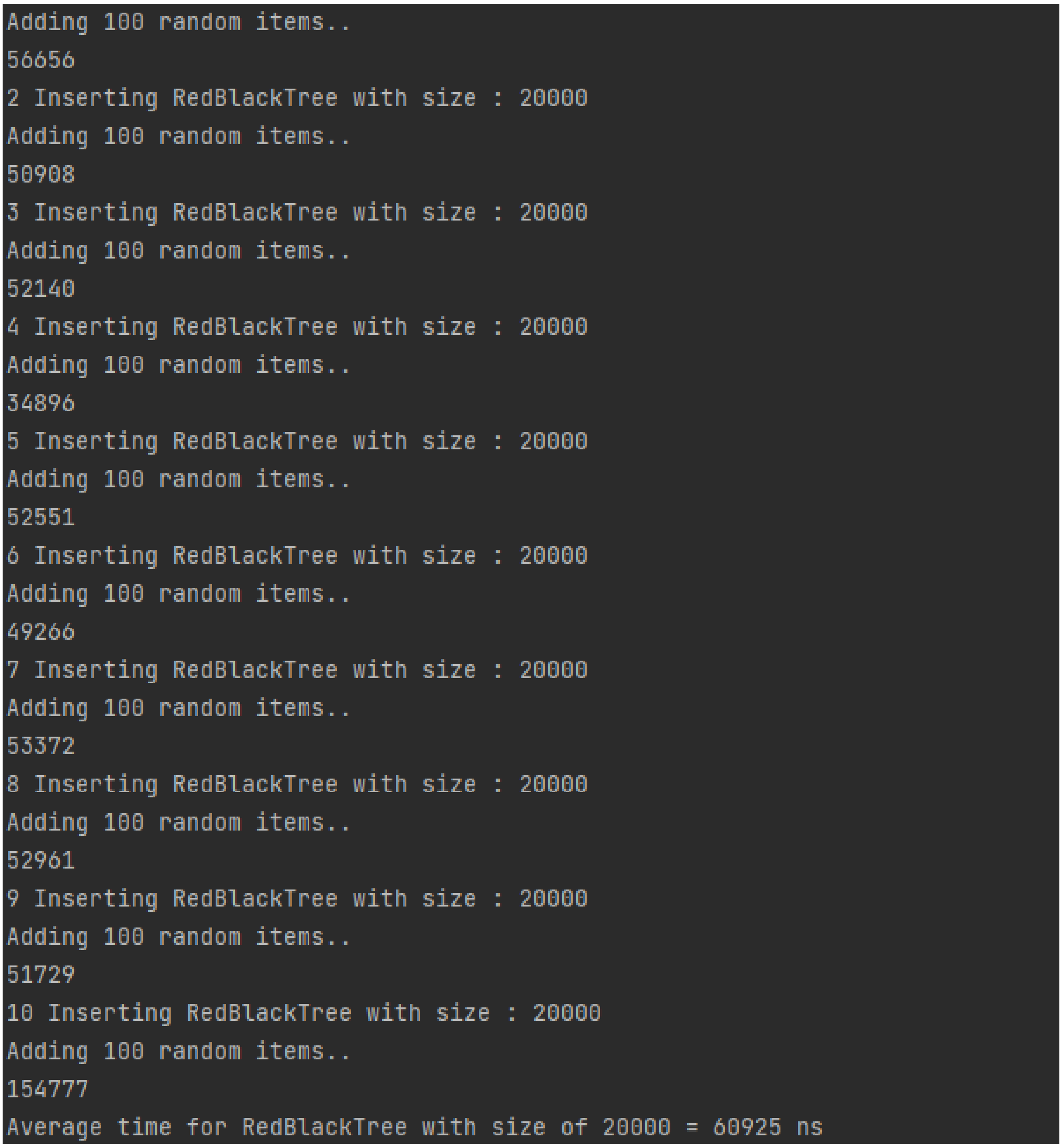
```
1 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
36539
2 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
40645
3 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
35718
4 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
49677
5 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
45982
6 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
37771
7 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
41876
8 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
40234
9 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
38181
10 Inserting BinarySearchTree with size : 40000
Adding 100 random items..
43929
Average time for BinarySearchTree with size of 40000 = 41055 ns
```



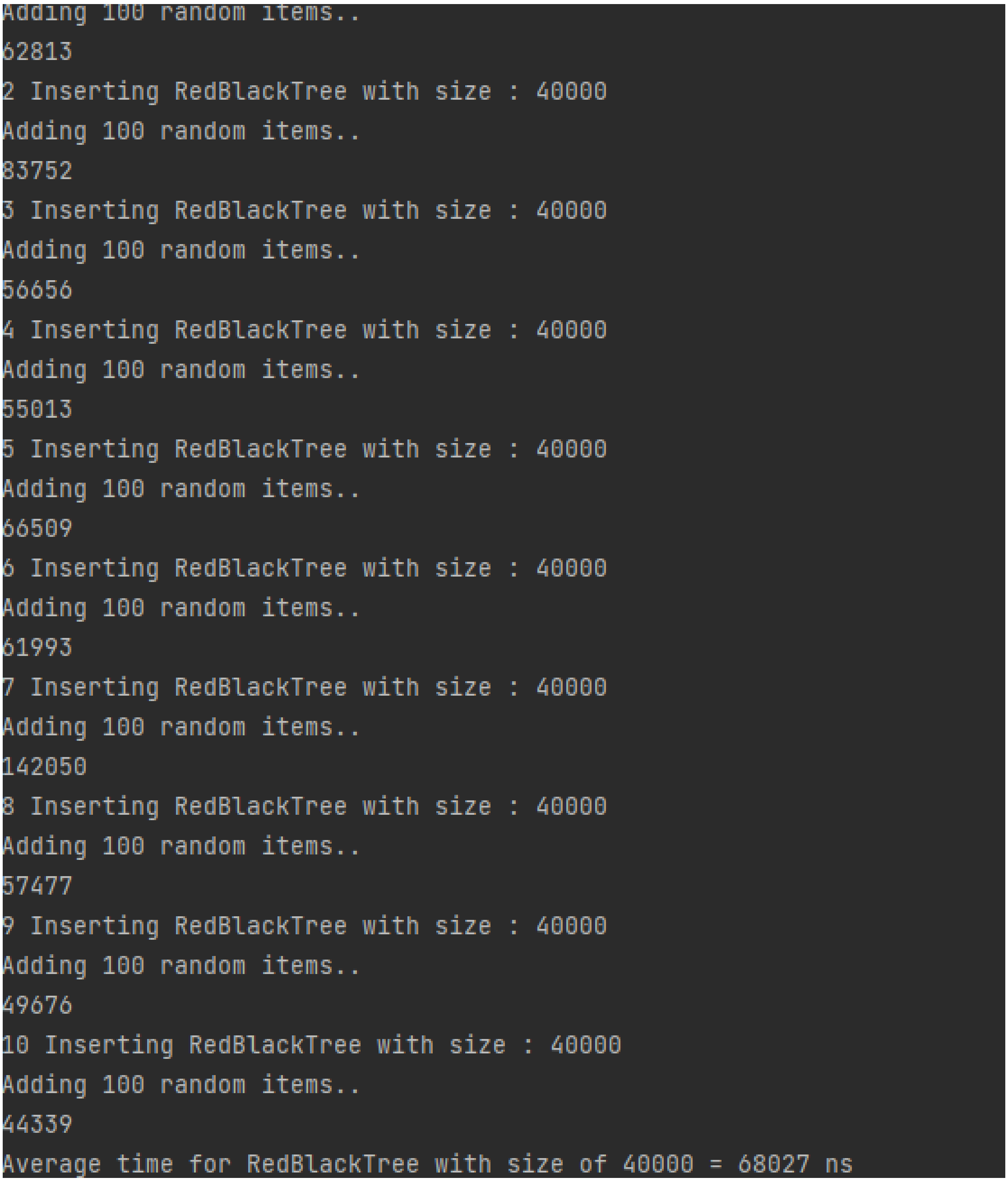
```
Adding 100 random items..
41876
2 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
747198
3 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
53372
4 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
36128
5 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
35307
6 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
34075
7 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
39824
8 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
55424
9 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
48034
10 Inserting BinarySearchTree with size : 80000
Adding 100 random items..
57066
Average time for BinarySearchTree with size of 80000 = 114830 ns
```



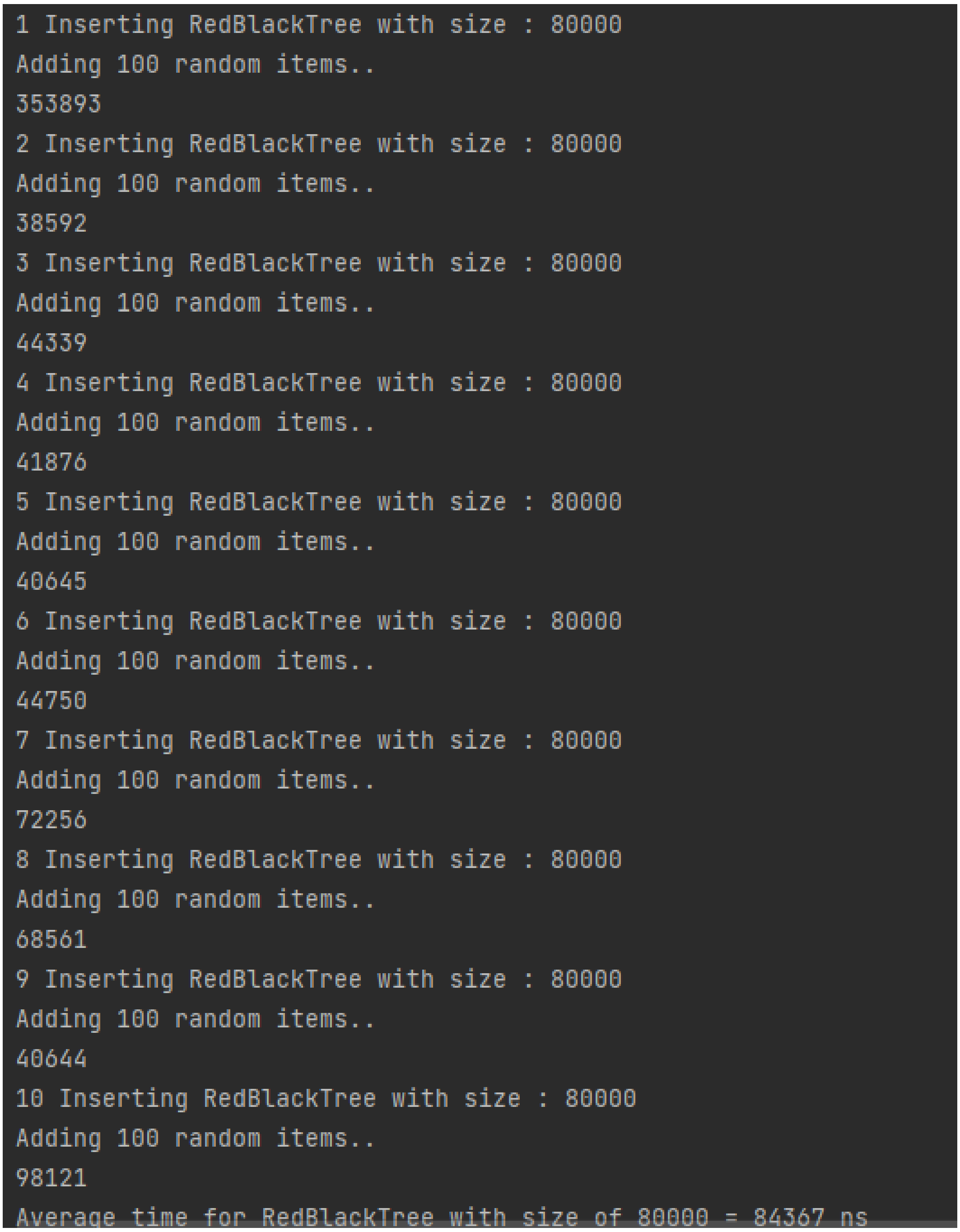
```
1 Inserting RedBlackTree with size : 10000
Adding 100 random items..
227444
2 Inserting RedBlackTree with size : 10000
Adding 100 random items..
237708
3 Inserting RedBlackTree with size : 10000
Adding 100 random items..
3161633
4 Inserting RedBlackTree with size : 10000
Adding 100 random items..
214307
5 Inserting RedBlackTree with size : 10000
Adding 100 random items..
195421
6 Inserting RedBlackTree with size : 10000
Adding 100 random items..
69382
7 Inserting RedBlackTree with size : 10000
Adding 100 random items..
80878
8 Inserting RedBlackTree with size : 10000
Adding 100 random items..
51729
9 Inserting RedBlackTree with size : 10000
Adding 100 random items..
58708
10 Inserting RedBlackTree with size : 10000
Adding 100 random items..
46392
Average time for RedBlackTree with size of 10000 = 434360 ns
```



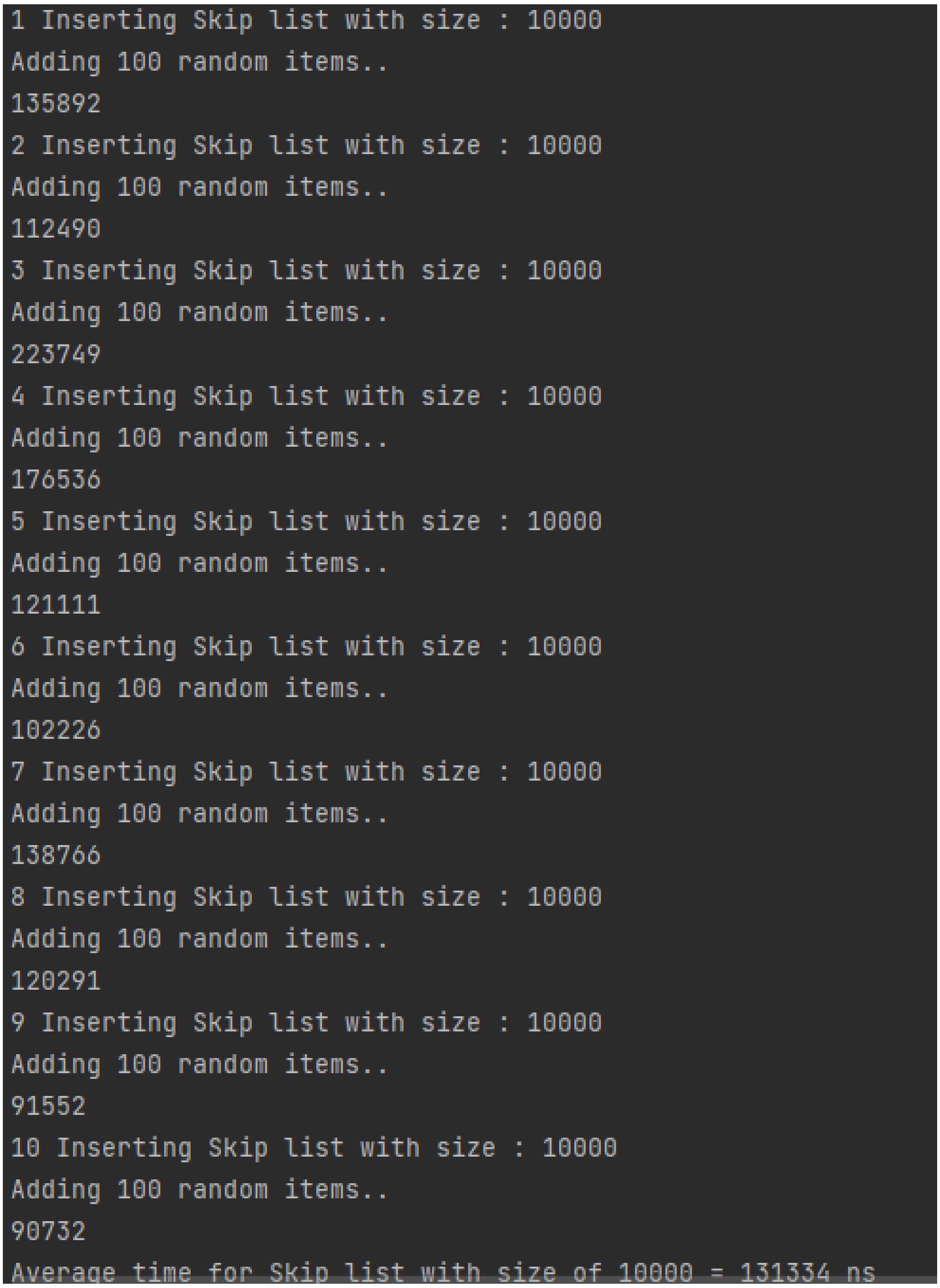
```
Adding 100 random items..
56656
2 Inserting RedBlackTree with size : 20000
Adding 100 random items..
50908
3 Inserting RedBlackTree with size : 20000
Adding 100 random items..
52140
4 Inserting RedBlackTree with size : 20000
Adding 100 random items..
34896
5 Inserting RedBlackTree with size : 20000
Adding 100 random items..
52551
6 Inserting RedBlackTree with size : 20000
Adding 100 random items..
49266
7 Inserting RedBlackTree with size : 20000
Adding 100 random items..
53372
8 Inserting RedBlackTree with size : 20000
Adding 100 random items..
52961
9 Inserting RedBlackTree with size : 20000
Adding 100 random items..
51729
10 Inserting RedBlackTree with size : 20000
Adding 100 random items..
154777
Average time for RedBlackTree with size of 20000 = 60925 ns
```



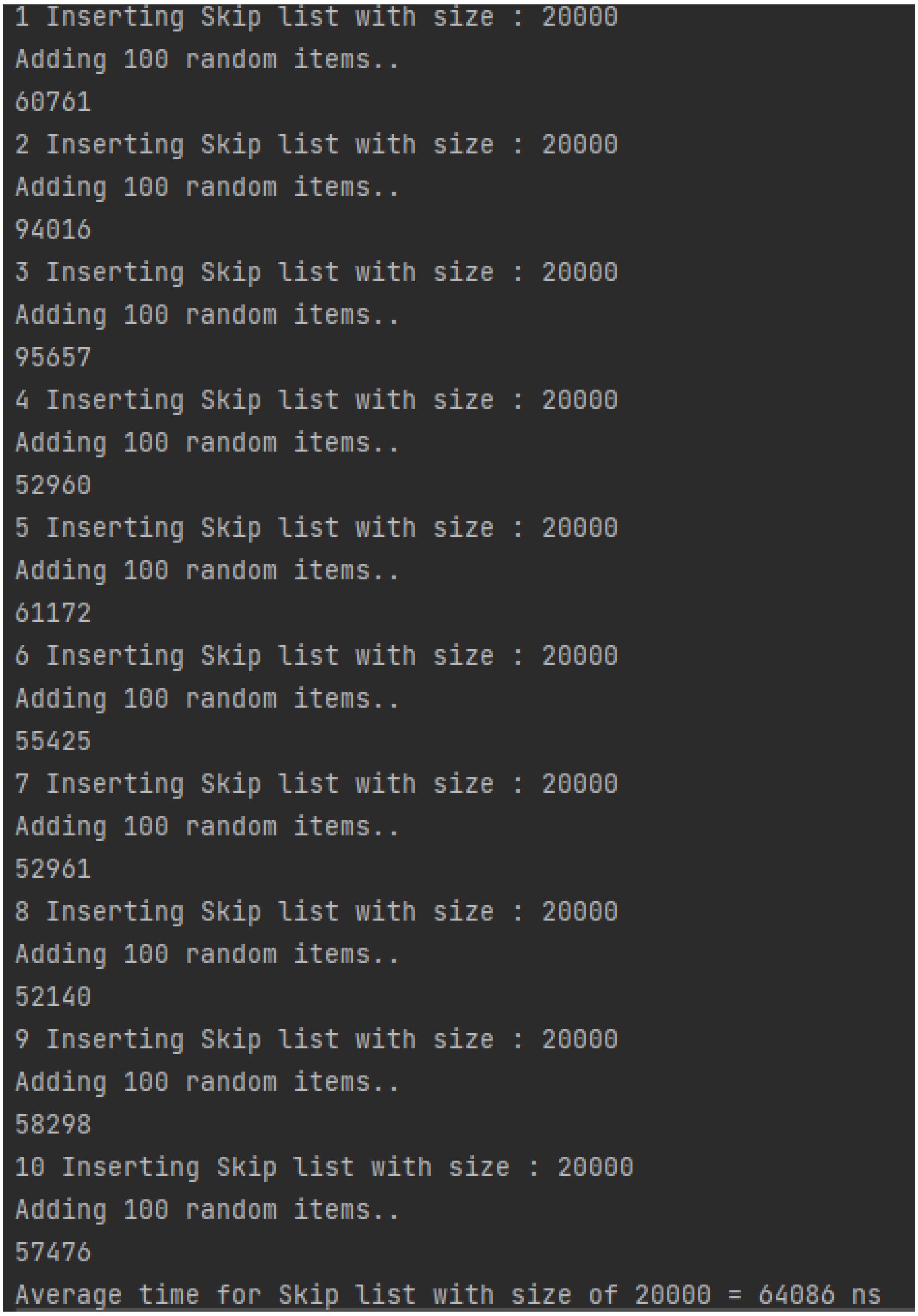
```
Adding 100 random items..
62813
2 Inserting RedBlackTree with size : 40000
Adding 100 random items..
83752
3 Inserting RedBlackTree with size : 40000
Adding 100 random items..
56656
4 Inserting RedBlackTree with size : 40000
Adding 100 random items..
55013
5 Inserting RedBlackTree with size : 40000
Adding 100 random items..
66509
6 Inserting RedBlackTree with size : 40000
Adding 100 random items..
61993
7 Inserting RedBlackTree with size : 40000
Adding 100 random items..
142050
8 Inserting RedBlackTree with size : 40000
Adding 100 random items..
57477
9 Inserting RedBlackTree with size : 40000
Adding 100 random items..
49676
10 Inserting RedBlackTree with size : 40000
Adding 100 random items..
44339
Average time for RedBlackTree with size of 40000 = 68027 ns
```



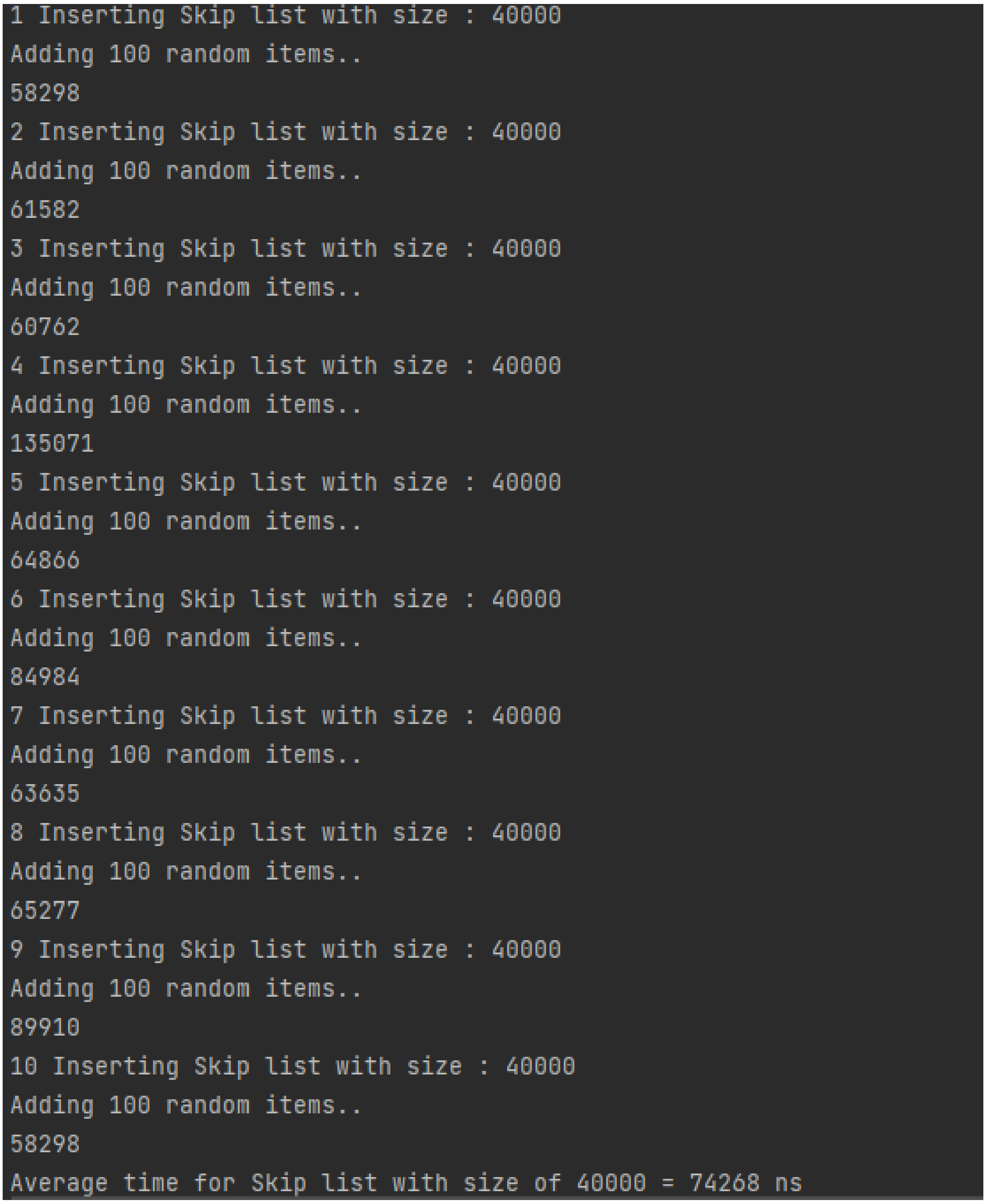
```
1 Inserting RedBlackTree with size : 80000
Adding 100 random items..
353893
2 Inserting RedBlackTree with size : 80000
Adding 100 random items..
38592
3 Inserting RedBlackTree with size : 80000
Adding 100 random items..
44339
4 Inserting RedBlackTree with size : 80000
Adding 100 random items..
41876
5 Inserting RedBlackTree with size : 80000
Adding 100 random items..
40645
6 Inserting RedBlackTree with size : 80000
Adding 100 random items..
44750
7 Inserting RedBlackTree with size : 80000
Adding 100 random items..
72256
8 Inserting RedBlackTree with size : 80000
Adding 100 random items..
68561
9 Inserting RedBlackTree with size : 80000
Adding 100 random items..
40644
10 Inserting RedBlackTree with size : 80000
Adding 100 random items..
98121
Average time for RedBlackTree with size of 80000 = 84367 ns
```



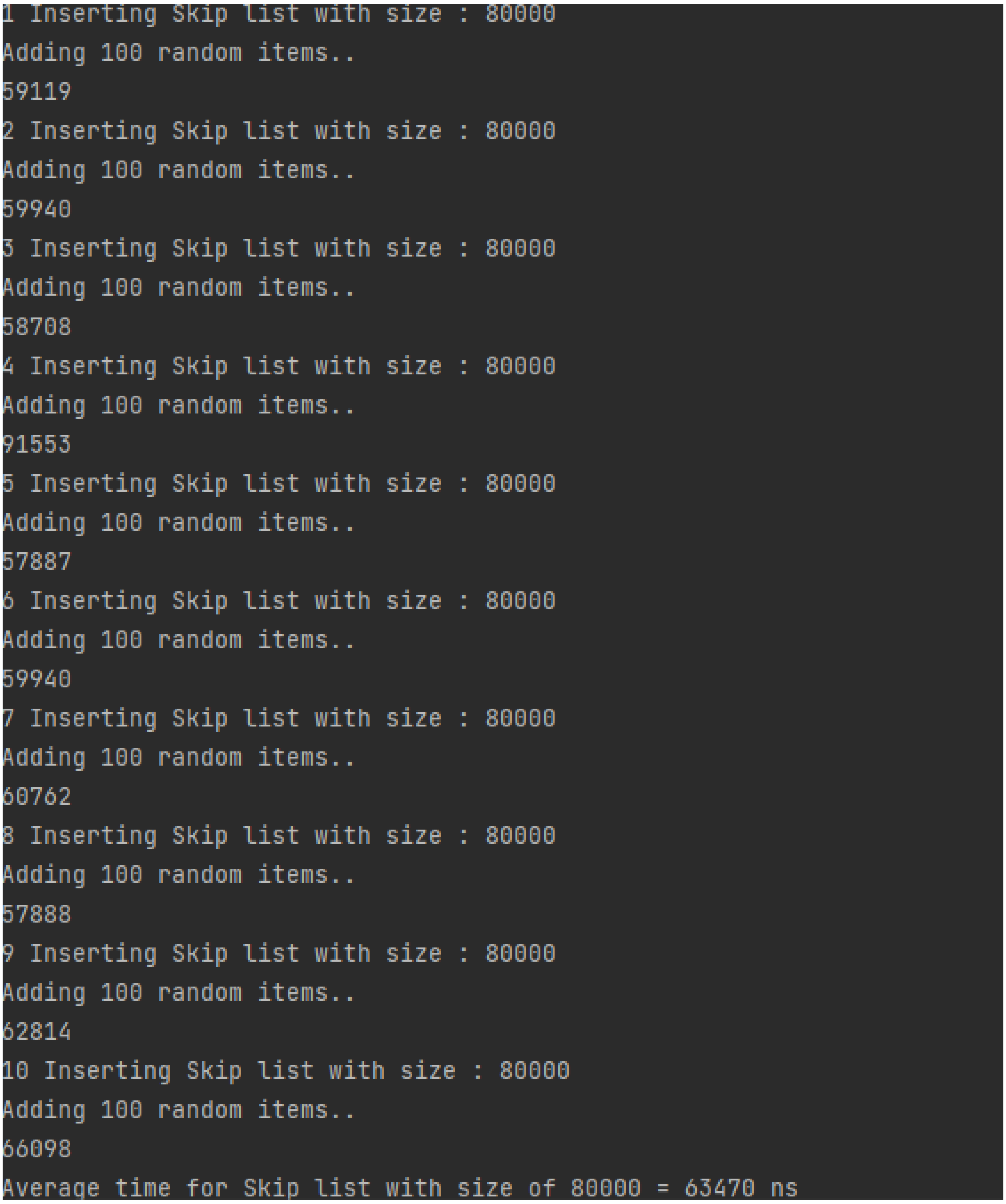
```
1 Inserting Skip list with size : 10000
Adding 100 random items..
135892
2 Inserting Skip list with size : 10000
Adding 100 random items..
112490
3 Inserting Skip list with size : 10000
Adding 100 random items..
223749
4 Inserting Skip list with size : 10000
Adding 100 random items..
176536
5 Inserting Skip list with size : 10000
Adding 100 random items..
121111
6 Inserting Skip list with size : 10000
Adding 100 random items..
102226
7 Inserting Skip list with size : 10000
Adding 100 random items..
138766
8 Inserting Skip list with size : 10000
Adding 100 random items..
120291
9 Inserting Skip list with size : 10000
Adding 100 random items..
91552
10 Inserting Skip list with size : 10000
Adding 100 random items..
90732
Average time for Skip list with size of 10000 = 131334 ns
```



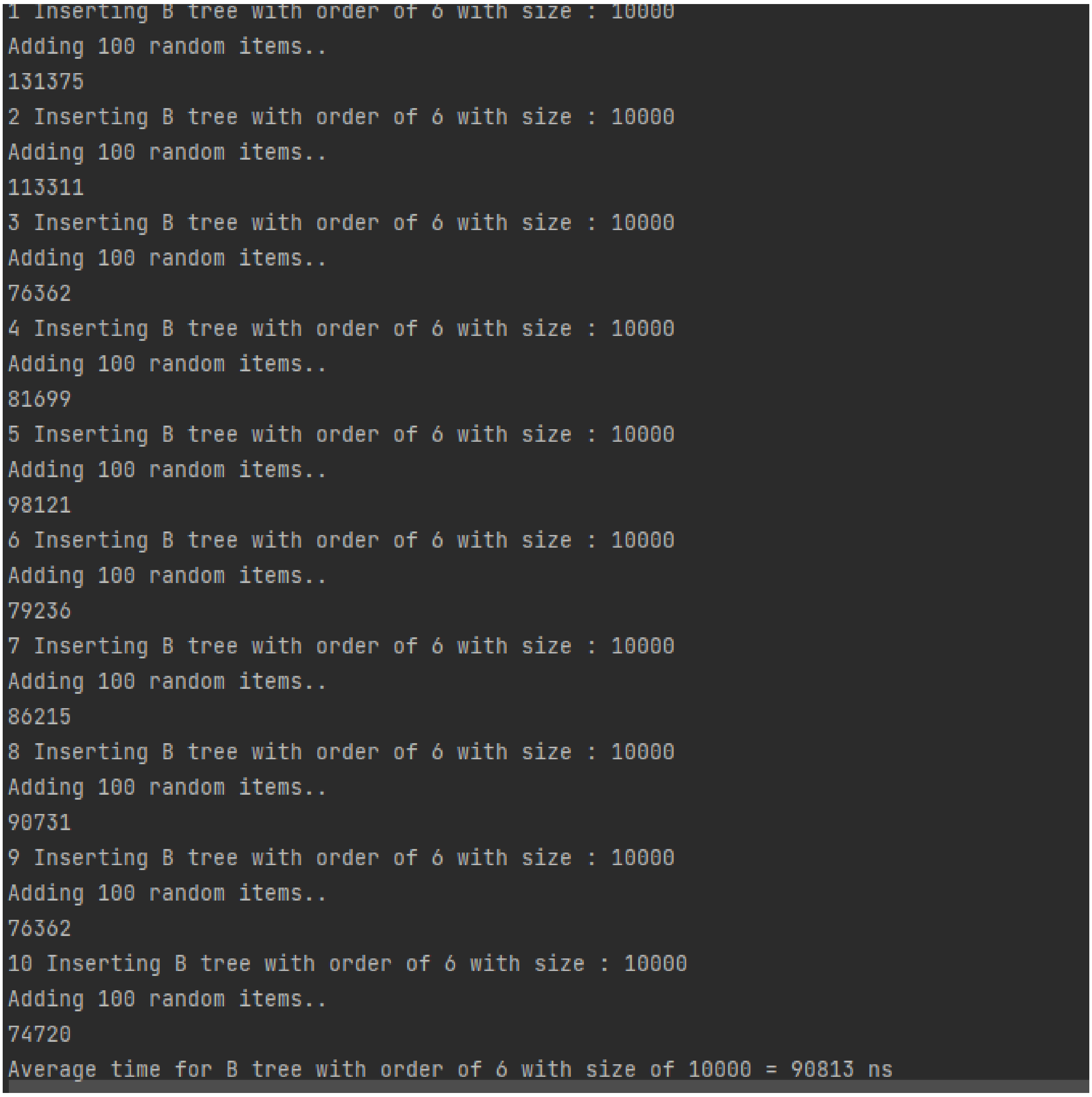
```
1 Inserting Skip list with size : 20000
Adding 100 random items..
60761
2 Inserting Skip list with size : 20000
Adding 100 random items..
94016
3 Inserting Skip list with size : 20000
Adding 100 random items..
95657
4 Inserting Skip list with size : 20000
Adding 100 random items..
52960
5 Inserting Skip list with size : 20000
Adding 100 random items..
61172
6 Inserting Skip list with size : 20000
Adding 100 random items..
55425
7 Inserting Skip list with size : 20000
Adding 100 random items..
52961
8 Inserting Skip list with size : 20000
Adding 100 random items..
52140
9 Inserting Skip list with size : 20000
Adding 100 random items..
58298
10 Inserting Skip list with size : 20000
Adding 100 random items..
57476
Average time for Skip list with size of 20000 = 64086 ns
```



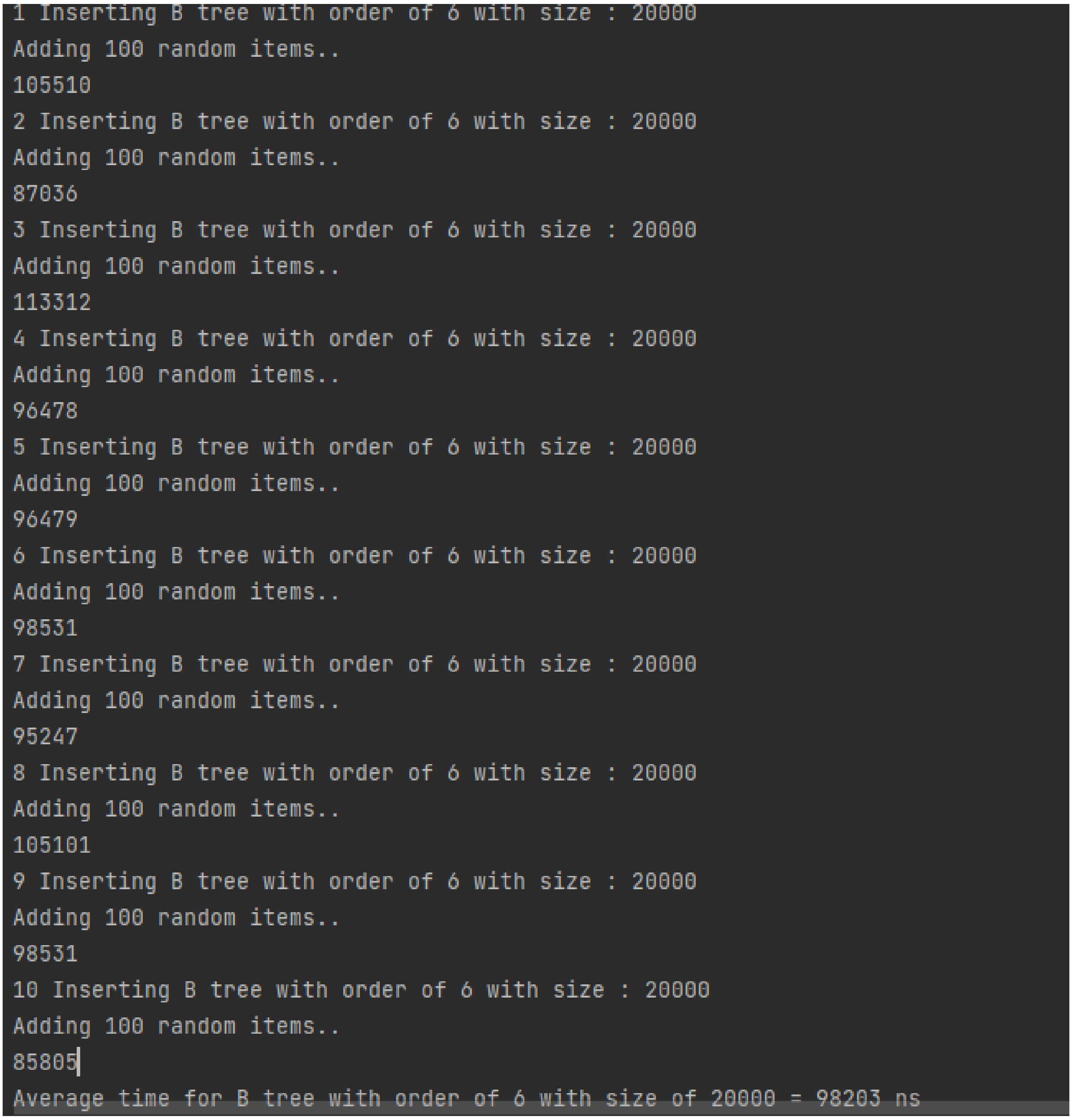
```
1 Inserting Skip list with size : 40000
Adding 100 random items..
58298
2 Inserting Skip list with size : 40000
Adding 100 random items..
61582
3 Inserting Skip list with size : 40000
Adding 100 random items..
60762
4 Inserting Skip list with size : 40000
Adding 100 random items..
135071
5 Inserting Skip list with size : 40000
Adding 100 random items..
64866
6 Inserting Skip list with size : 40000
Adding 100 random items..
84984
7 Inserting Skip list with size : 40000
Adding 100 random items..
63635
8 Inserting Skip list with size : 40000
Adding 100 random items..
65277
9 Inserting Skip list with size : 40000
Adding 100 random items..
89910
10 Inserting Skip list with size : 40000
Adding 100 random items..
58298
Average time for Skip list with size of 40000 = 74268 ns
```



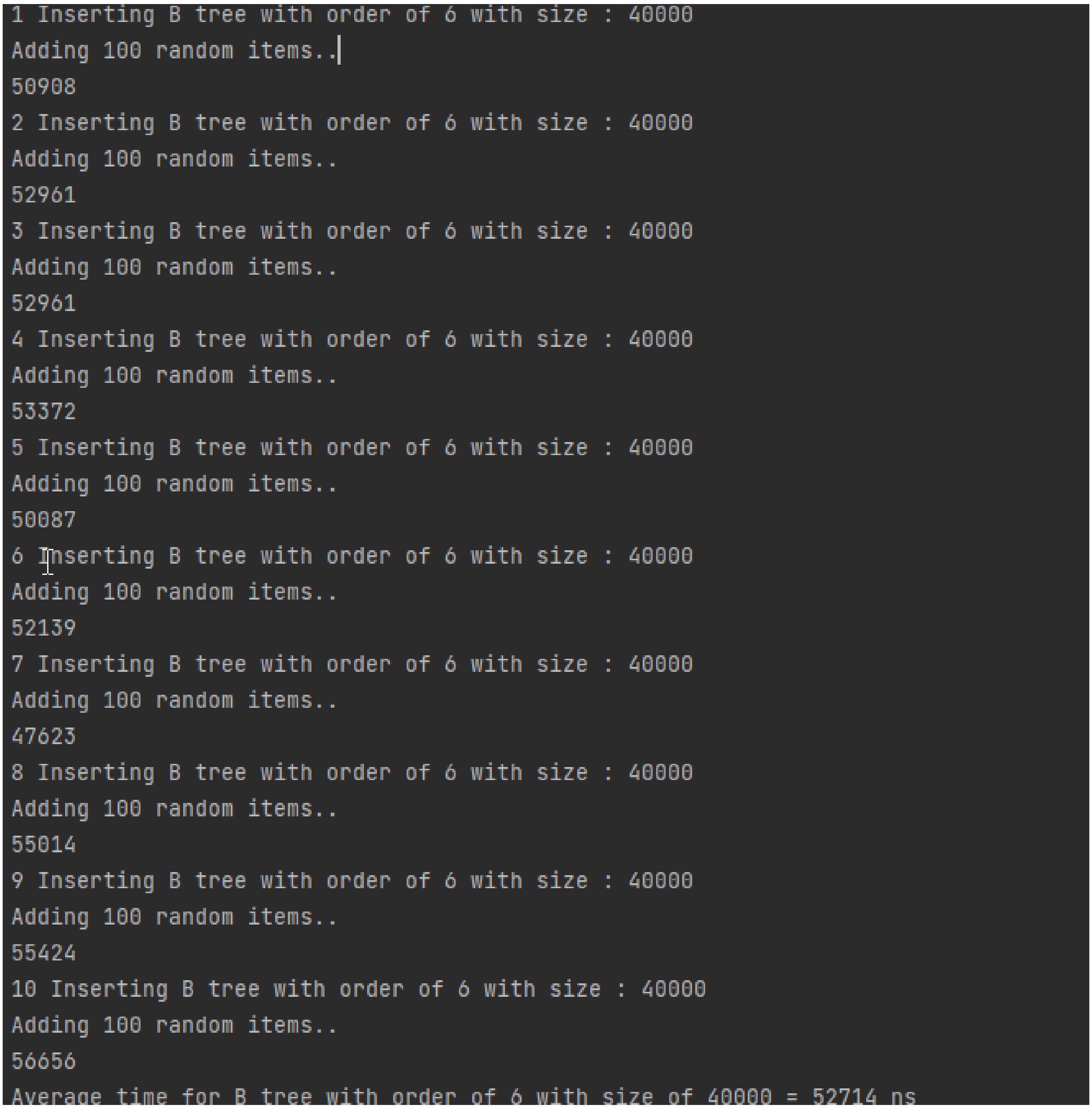
```
1 Inserting Skip list with size : 80000
Adding 100 random items..
59119
2 Inserting Skip list with size : 80000
Adding 100 random items..
59940
3 Inserting Skip list with size : 80000
Adding 100 random items..
58708
4 Inserting Skip list with size : 80000
Adding 100 random items..
91553
5 Inserting Skip list with size : 80000
Adding 100 random items..
57887
6 Inserting Skip list with size : 80000
Adding 100 random items..
59940
7 Inserting Skip list with size : 80000
Adding 100 random items..
60762
8 Inserting Skip list with size : 80000
Adding 100 random items..
57888
9 Inserting Skip list with size : 80000
Adding 100 random items..
62814
10 Inserting Skip list with size : 80000
Adding 100 random items..
66098
Average time for Skip list with size of 80000 = 63470 ns
```



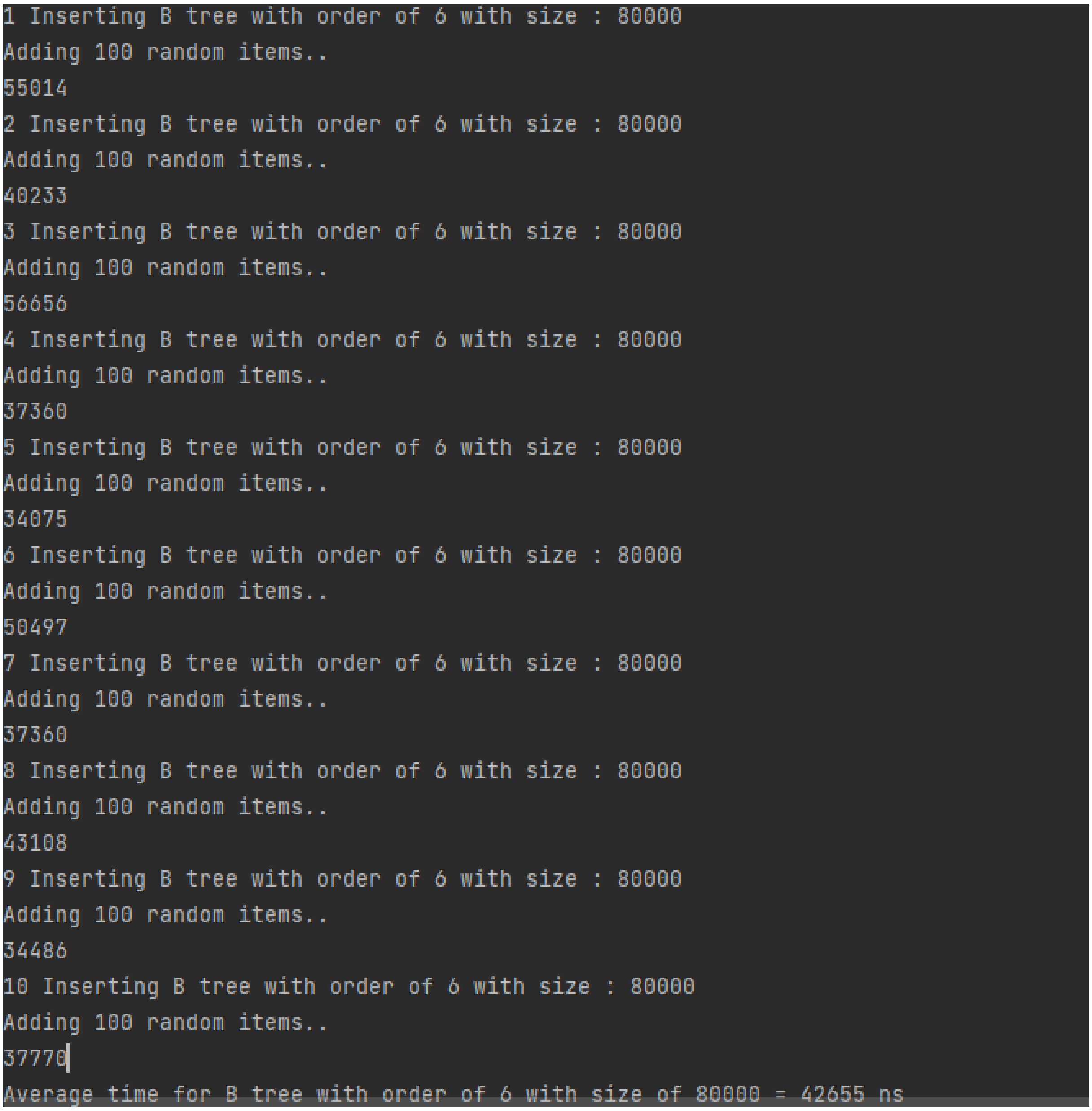
```
1 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
131375
2 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
113311
3 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
76362
4 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
81699
5 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
98121
6 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
79236
7 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
86215
8 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
90731
9 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
76362
10 Inserting B tree with order of 6 with size : 10000
Adding 100 random items..
74720
Average time for B tree with order of 6 with size of 10000 = 90813 ns
```



```
1 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
105510
2 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
87036
3 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
113312
4 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
96478
5 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
96479
6 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
98531
7 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
95247
8 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
105101
9 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
98531
10 Inserting B tree with order of 6 with size : 20000
Adding 100 random items..
85805
Average time for B tree with order of 6 with size of 20000 = 98203 ns
```



```
1 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
50908
2 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
52961
3 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
52961
4 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
53372
5 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
50087
6 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
52139
7 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
47623
8 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
55014
9 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
55424
10 Inserting B tree with order of 6 with size : 40000
Adding 100 random items..
56656
Average time for B tree with order of 6 with size of 40000 = 52714 ns
```



```
1 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
55014
2 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
40233
3 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
56656
4 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
37360
5 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
34075
6 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
50497
7 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
37360
8 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
43108
9 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
34486
10 Inserting B tree with order of 6 with size : 80000
Adding 100 random items..
37770
Average time for B tree with order of 6 with size of 80000 = 42655 ns
```