GIT Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 7 Report

Serdil Anıl Ünlü
1801042672

1. SYSTEM REQUIREMENTS

a. Functional Requirements

PART1:

- There are 3 methods that is implemented for NavigableSet using Skiplist.
    a. boolean add(E e)
    b. Boolean remove(E e)
    c. Iterator<E>descendingIterator()
- I created a private skipList field inside the navigableSkipList class and used the add method through that value. And This method add the element to the NavigableSet. Uses SkipList methods to do it.
- Remove method remove given parameter from the NavigableSet if the element is exist in the set.
- Descending iterator, on the other hand, sorts the numbers given in the set descending order.
- There are 4 methods that is implemented for NavigableSet using AVL Tree in navigableSetAVL class:

  1-)Boolean insert(E e)

  2-)Iterator<E> iterator()

  3-)NavigableSet<E> headset(E headElement)

  4-)NavigableSet<E>tailSet(E fromElement)

- I add element to NavigableSet using methods of AVL tree class.
- To use the Iterator, I first add the elements of the tree to the arraylist I created by looping from left to right, and then I iterate that array.
- In headSet, I compared the elements using the compareTo function in the headSet method and them to the array accordingly.
- In the tailSet method, similar to the headSet method, I compare the elements with the compareTo method and throw them into the array I created with the arraylist and return that array.

PART2:

- There is 1 method isAvlRedBlack(BinarySearchTree<Integer>). And this method checks if the Binary Search Tree written to the parameter is an AVL or Red-BlackTree also if Binary Search Tree doesn't provide both, I print it as well. I am looking at the left and right height differences of the tree written to the parameter inside this method. I suppress whether it is balanced according to him.

PART3:

- In part3, I tested the data structures available in the book by inserting them in Main and measured the time they took.
- These Data Structures are:

  1-)BinarySearchTree

  2-)Red-Black Tree

  3-)2-3 Tree

  4-)B-Tree

  5-)SkipList

b.Non-Functional Requirements:

Part1:

- Add methods shouldn't insert same elements to the NavigableSet
- Remove method should return null if there is no element in the NavigableSet.
- Iterator and descendingIterator must iterate all elements once
- headSet method in returned NavigableSet shouldn't contain any bigger number than given parameter.

- tailSet method shouldn’t contain any less number than given parameter.

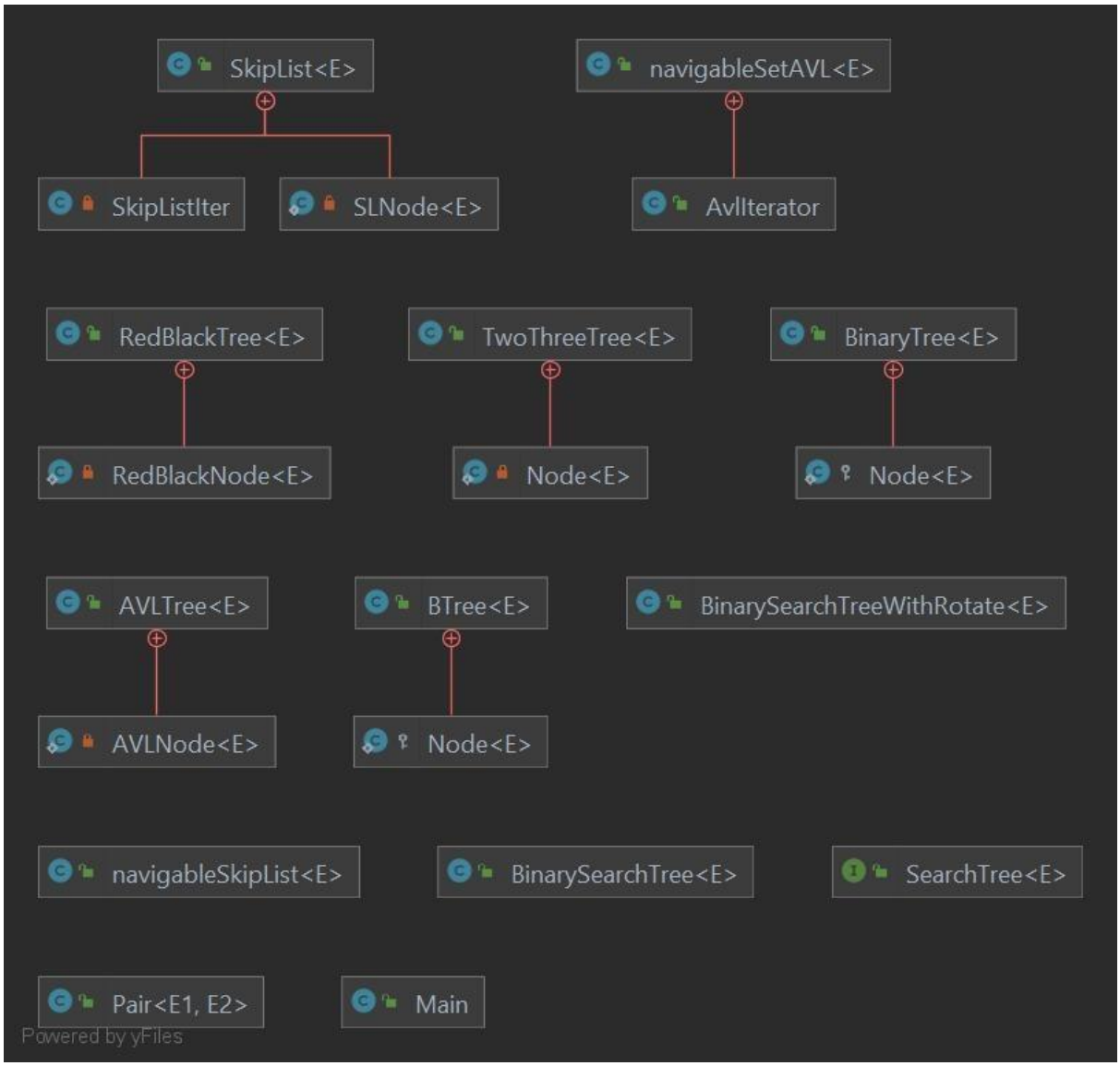
PART2:

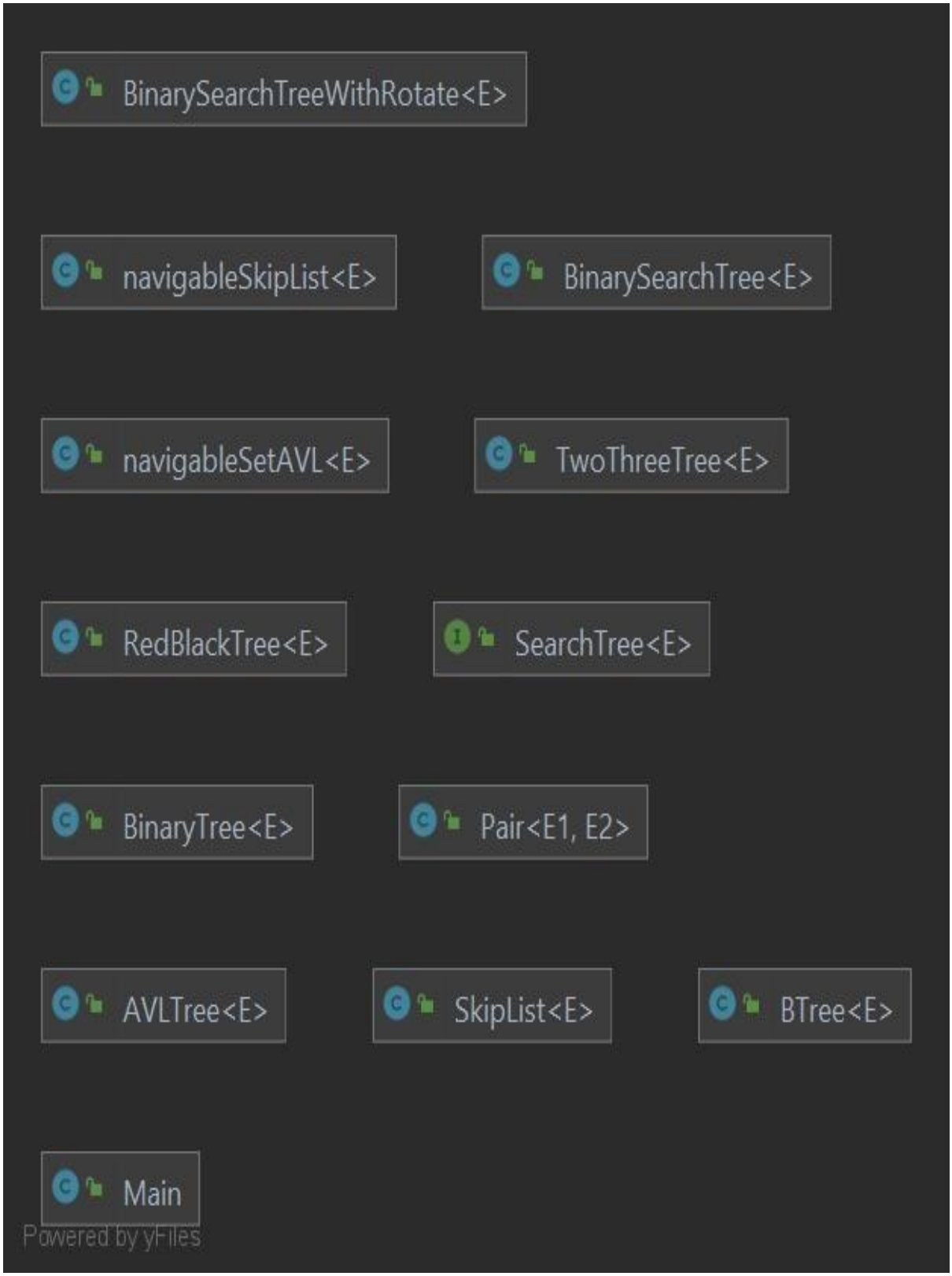
- The isAvlRedBlack method looks at the height differences on the right and left sides of the entered as a parameter.

PART3:

- Driver code must generate randomly generated (non-repeating) numbers with the size of 10k, 20k, 40k, 80k.
- Then insert these random numbers into the 5 different data structure.

2. **CLASS DIAGRAMS**




BinarySearchTreeWithRotate<E>
navigableSkipList<E>
BinarySearchTree<E>
navigableSetAVL<E>
TwoThreeTree<E>
RedBlackTree<E>
SearchTree<E>
BinaryTree<E>
Pair<E1, E2>
AVLTree<E>
SkipList<E>
BTree<E>
Main
Powered by yFiles


BTree<E>
root Node<E>
order int
newParent E
newChild Node<E>
BTree(int)
add(E) boolean
binarySearch(E, E[], int, int) int
clear() void
contains(E) boolean
delete(E) E
find(E) E
find(Node<E>, E) E
inOrderTraverse(Node<E>, List<E>) void
insert(Node<E>, E) boolean
insertIntoNode(Node<E>, int, E, Node<E>) void
preOrderTraverse(Node<E>, int, StringBuilder) void
remove(E) boolean
splitNode(Node<E>, int, E, Node<E>) void
toList() List<E>
toString() String
Node<E>
size int
data E[]
child Node<E>[]
Node(int)
toString() String
AVLTree<E>
increase boolean
decrease boolean
AVLTree()
add(AVLNode<E>, E) AVLNode<E>
add(E) boolean
decrementBalance(AVLNode<E>) void
delete(AVLNode<E>, E) AVLNode<E>
delete(E) E
findLargestChild(AVLNode<E>) E
findReplacementNode(AVLNode<E>) AVLNode<E>
incrementBalance(AVLNode<E>) void
printInorder(Node<E>) void
rebalanceLeft(AVLNode<E>) AVLNode<E>
rebalanceLeftR(AVLNode<E>) AVLNode<E>
rebalanceRight(AVLNode<E>) AVLNode<E>
rebalanceRightL(AVLNode<E>) AVLNode<E>
AVLNode<E>
LEFT_HEAVY int
BALANCED int
RIGHT_HEAVY int
balance int
AVLNode(E)
toString() String
BinaryTree<E>
Node int
node Object
root Node<E>
BinaryTree()
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
BinaryTree(Node<E>)
preOrderTraverse(Node<E>, int, StringBuilder) void
readBinaryTree(BufferedReader) BinaryTree<String>?
toString() String
data E
leaf boolean
leftSubtree BinaryTree<E>
rightSubtree BinaryTree<E>
Node<E>
data E
left Node<E>
right Node<E>
Node(E)
toString() String
SkipList<E>
LOG2 double
maxLevel int
maxCap int
size int
head SLNode<E>
SkipList()
add(E) void
find(E) E
iterator() Iterator<E>
logRandom() int
remove(E) E
search(E) SLNode<E>[]
size() int
toString() String
SkipListIter
current SLNode<E>
SkipListIter()
hasNext() boolean
next() E
SLNode<E>
links SLNode<E>[]
data E
SLNode(int, E)
navigableSetAVL<E>
value AVLTree<E>
arr2 ArrayList<E>
navigableSetAVL()
headSet(E) ArrayList<E>
insert(E) boolean
iterator() AvlIterator
itrArray(Node<E>) void
printInorder() void
printInorder(Node<E>) void
tailSet(E) ArrayList<E>
toString() String
AvlIterator
current int
last int
arr ArrayList<E>
AvlIterator()
hasNext() boolean
itrArray(Node<E>) void
next() E
TwoThreeTree<E>
root Node<E>
order int
newChild Node<E>
newParent E
TwoThreeTree(int)
binarySearch(E[], E, int, int) int
contains(E) boolean
find(E) E
find(Node<E>, E) E
insert(E) boolean
insert(Node<E>, E) boolean
insertIntoNode(Node<E>, int, E, Node<E>) void
splitNode(Node<E>, int, E, Node<E>) void
Node<E>
size int
data E[]
child Node<E>[]
Node(int)
RedBlackTree<E>
fixupRequired boolean
RedBlackTree()
add(E) boolean
add(RedBlackNode<E>, E) Node<E>
delete(E) E
findLargestChild(Node<E>) E
findReplacement(Node<E>) Node<E>?
fixupLeft(Node<E>) Node<E>
fixupRight(Node<E>) Node<E>
moveBlackDown(RedBlackNode<E>) void
removeFromLeft(Node<E>, E) E?
removeFromRight(Node<E>, E) E?
RedBlackNode<E>
isRed boolean
RedBlackNode(E)
toString() String
BinarySearchTree<E>
addReturn boolean
AVLdata int
deleteReturn E
left BinarySearchTree<E>
right BinarySearchTree<E>
BinarySearchTree()
add(E) boolean
add(Node<E>, E) Node<E>
contains(E) boolean
delete(E) E
delete(Node<E>, E) Node<E>
find(E) E
find(Node<E>, E) E
findLargestChild(Node<E>) E
remove(E) boolean
Main
Main()
isAvlBst(Node<Integer>) boolean
isAvlRedBlack(BinarySearchTree<Integer>) int
isRedBlack(Node<Integer>) boolean
main(String[]) void
maxHeight(Node<Integer>) int
minHeight(Node<Integer>) int
navigableSkipList<E>
data SkipList<E>
navigableSkipList()
add(E) boolean
descendingIterator() Iterator<E>
remove(E) E
toString() String
SearchTree<E>
add(E) boolean
contains(E) boolean
delete(E) E
find(E) E
remove(E) boolean
BinarySearchTreeWithRotate<E>
BinarySearchTreeWithRotate()
rotateLeft(Node<E>) Node<E>
rotateRight(Node<E>) Node<E>
Pair<E1, E2>
first E1
second E2
Pair(E1, E2)

3-PROBLEM SOLUTION APPROACH

1- PART1:

- firstly, I implement the NavigableSet interface by using Skip List.
- Also I using stack data structure for navigable set by using Skip List while using descending Iterator in navigableSkipList.
- For part 1 navigableSet with Avl Tree , I create AvlIterator for iterate the elements in the Avl Tree.
- And in navigableSetAVL class I record the array elements which in the Avl Tree.
- So while using headSet and tailSet I returned the array.

2-) PART2

• I can find out if the tree written in the parameter is AvlTree, Red Black Tree or both.

3-)PART3

• I use the 5 data structure using book implementation and I testing in Main class by using insert operation and whileI measure time I using nanotime and for I using non repeating elements I use find and contains method.

4. TEST CASES

```
navigableSkipList<Integer> nv = new navigableSkipList<>();
System.out.println();



ArrayList<Integer> tp = new ArrayList<>();
tp.add(12);
tp.add(10);
tp.add(9);
tp.add(1);
tp.add(26);
tp.add(14);
tp.add(4);
tp.add(92);
tp.add(19);
tp.add(6);
tp.add(110);
tp.add(95);
tp.add(138);
tp.add(150);
tp.add(222);


System.out.println("******* Adding ******");
System.out.println("Elements to Add");

for(int i=0;i<tp.size();i++){
    System.out.print(tp.get(i)+", ");
}
```

```
System.out.println("Before Adding To SkipListSet:");
System.out.println(nv);

nv.add(12);
nv.add(10);
nv.add(9);
nv.add(1);
nv.add(26);
nv.add(14);
nv.add(4);
nv.add(92);
nv.add(19);
nv.add(6);
nv.add(110);
nv.add(95);
nv.add(138);
nv.add(150);
nv.add(222);
System.out.println("After Adding To SkipListSet:");
System.out.println(nv);
```

```
System.out.println("Before Removing: ");
System.out.println(nv);
nv.remove(92);
nv.remove(14);
System.out.println("After Removing: ");
System.out.println(nv);
System.out.println();
```

```java
System.out.println("********* Descending Iterator: ***********");
Iterator<Integer> x = nv.descendingIterator();

while (x.hasNext()){
    System.out.print(x.next()+ ", ");
}
```

```java
System.out.println("************************* AVLTreeSet Tests *********************************");
System.out.println();
System.out.println("********* Adding **********");
navigableSetAVL<Integer> na = new navigableSetAVL<>();
navigableSetAVL<Integer> na2 = new navigableSetAVL<>();

System.out.println("Elements To Add: ");
ArrayList<Integer> rec = new ArrayList<>();

rec.add(23);
rec.add(10);
rec.add(2);
rec.add(1);
rec.add(22);
rec.add(12);
rec.add(130);
rec.add(51);
rec.add(93);
rec.add(111);
rec.add(229);
rec.add(87);
rec.add(45);
rec.add(79);
rec.add(100);

for(int i=0;i<rec.size();i++){
    System.out.print(rec.get(i)+", ");
}
```

```
System.out.println("Before Adding");
System.out.println(na);

na.insert(23);
na.insert(10);
na.insert(2);
na.insert(1);
na.insert(22);
na.insert(12);
na.insert(130);
na.insert(51);
na.insert(93);
na.insert(111);
na.insert(229);
na.insert(87);
na.insert(45);
na.insert(79);
na.insert(100);


System.out.println("After Adding To AVLTreeSet: ");
na.printInorder();
System.out.println();
System.out.println();
```

```
navigableSetAVL.AvlIterator it = na.iterator();

System.out.println("ITERATOR: ");
while(it.hasNext()){
    System.out.print(it.next()+" ");
}
```

```
System.out.println();
System.out.println();
System.out.println("********** headSet ***********");
System.out.println("To Element: 23");
ArrayList<Integer> arrAvl = na.headSet(23);

System.out.println(arrAvl);
```

```
na2.insert(23);
na2.insert(10);
na2.insert(2);
na2.insert(1);
na2.insert(22);
na2.insert(12);
na2.insert(130);
na2.insert(51);
na2.insert(93);
na2.insert(111);
na2.insert(229);
na2.insert(87);
na2.insert(45);
na2.insert(79);
na2.insert(100);
System.out.println();
System.out.println();
System.out.println("********** tailSet ************");
System.out.println("From Element: 100");
ArrayList<Integer> arrAvl2 = na2.tailSet(100);
System.out.println(arrAvl2);
```

```
System.out.println();
System.out.println("******************");
BST4.add(12);
BST4.add(8);
BST4.add(4);
BST4.add(5);
BST4.add(11);
BST4.add(18);
BST4.add(17);
BST4.add(2);
BST4.add(7);

System.out.println("Binary Tree is :\n"+ BST4);

int result4 = isAvlRedBlack(BST4);
if(result4 == 1){
    System.out.println("THE Tree Above is Avl Tree And Red-Black Tree");
}

if(result4 == 2){
    System.out.println("The Tree Above is Red-Black Tree");

}

else{
    System.out.println("The Above Tree is Neither the Avl Tree nor the Red Black Tree");
}
```

```
System.out.println();
System.out.println("************************ BinarySearchTree-Test ***************************");
System.out.println();

BinarySearchTree<Integer> [] binarySearchTree10K = new BinarySearchTree[10];
BinarySearchTree<Integer> [] binarySearchTree20K = new BinarySearchTree[10];
BinarySearchTree<Integer> [] binarySearchTree40K = new BinarySearchTree[10];
BinarySearchTree<Integer> [] binarySearchTree80K = new BinarySearchTree[10];

RedBlackTree<Integer> [] redBlackTree10K = new RedBlackTree[10];
RedBlackTree<Integer> [] redBlackTree20K = new RedBlackTree[10];
RedBlackTree<Integer> [] redBlackTree40K = new RedBlackTree[10];
RedBlackTree<Integer> [] redBlackTree80K = new RedBlackTree[10];

TwoThreeTree<Integer> [] TwoThreeTree10K = new TwoThreeTree[10];
TwoThreeTree<Integer> [] TwoThreeTree20K = new TwoThreeTree[10];
TwoThreeTree<Integer> [] TwoThreeTree40K = new TwoThreeTree[10];
TwoThreeTree<Integer> [] TwoThreeTree80K = new TwoThreeTree[10];

BTree<Integer> [] bTree10K = new BTree[10];
BTree<Integer> [] bTree20K = new BTree[10];
BTree<Integer> [] bTree40K = new BTree[10];
BTree<Integer> [] bTree80K = new BTree[10];

SkipList<Integer> [] skipList10K = new SkipList[10];
SkipList<Integer> [] skipList20K = new SkipList[10];
SkipList<Integer> [] skipList40K = new SkipList[10];
SkipList<Integer> [] skipList80K = new SkipList[10];
```

```
for( int i = 0 ; i < 10 ; ++i ){
    binarySearchTree10K[i] = new BinarySearchTree<>();
    for( int j = 0 ; j < 10000 ; ++j ) {
        if(!(binarySearchTree10K[i].contains( extraNumber = random.nextInt() ))){
            binarySearchTree10K[i].add( extraNumber );
        }

        else{
            j--;
        }
    }
}



for(int i=0;i<10;i++){
    ArrayList<Integer>array = new ArrayList<>();

    while(array.size() != 100){
        if(!array.contains(extraNumber = random.nextInt() )){
            array.add(extraNumber);
        }
    }
    start = System.nanoTime();

    for(int j = 0;j<array.size();j++){
        binarySearchTree10K[i].add(array.get(j));
    }

    end = System.nanoTime();
    time += (end-start);
}

System.out.println("Average time for Binary Search Tree for 10k elements while inserting 100 random elements: "+ time/10+
```

```java
for( int i = 0 ; i < 10 ; ++i ){
    binarySearchTree20K[i] = new RedBlackTree<>();
    for( int j = 0 ; j < 20000 ; ++j ) {
        if(!(binarySearchTree20K[i].contains( extraNumber = random.nextInt() ))){
            binarySearchTree20K[i].add( extraNumber );
        }

        else{
            j--;
        }
    }
}


for(int i=0;i<10;i++){
    ArrayList<Integer>array = new ArrayList<>();

    while(array.size() != 100){
        if(!array.contains(extraNumber = random.nextInt() )){
            array.add(extraNumber);
        }
    }
    start = System.nanoTime();

    for(int j = 0;j<array.size();j++){
        binarySearchTree20K[i].add(array.get(j));
    }

    end = System.nanoTime();
    time2 += (end-start);
}

System.out.println("Average time for Binary Search Tree for 20k elements while inserting 100 random elements: "+ time2/10
```

5. RUNNING AND RESULTS

```
********************************** PART1 ***************************************

***************** SkipListSet Testing **************

******* Adding ******
Elements to Add
12, 10, 9, 1, 26, 14, 4, 92, 19, 6, 110, 95, 138, 150, 222,
Before Adding To SkipListSet:
[]
After Adding To SkipListSet:
[1, 4, 6, 9, 10, 12, 14, 19, 26, 92, 95, 110, 138, 150, 222]

************ Removing ************
ELEMENT TO REMOVE : 92,14
Before Removing:
[1, 4, 6, 9, 10, 12, 14, 19, 26, 92, 95, 110, 138, 150, 222]
After Removing:
[1, 4, 6, 9, 10, 12, 19, 26, 95, 110, 138, 150, 222]

********* Descending Iterator: ************
222, 150, 138, 110, 95, 26, 19, 12, 10, 9, 6, 4, 1,
```

```
************************** AVLTreeSet Tests **********************************

******** Adding ***********
Elements To Add:
23, 10, 2, 1, 22, 12, 130, 51, 93, 111, 229, 87, 45, 79, 100,
Before Adding
null

After Adding To AVLTreeSet:
1 2 10 12 22 23 45 51 79 87 93 100 111 130 229

ITERATOR:
1 2 10 12 22 23 45 51 79 87 93 100 111 130 229

********** headSet ***********
To Element: 23
[1, 2, 10, 12, 22]


********* tailSet ***********
From Element: 100
[100, 111, 130, 229]
```

```
********************************** PART2 ************************************
Binary Tree is :
5
  4
    null
    null
  20
    18
      null
      null
    23
      null
      null

THE TREE ABOVE IS AVL TREE AND Red-Black TREE
```

```
**********
Binary Tree is :
12
  8
    4
      null
      5
        null
        null
    11
      null
      null
  18
    17
      null
      null
    null

THE TREE ABOVE IS AVL TREE AND Red-Black TREE
```

```
**********
Binary Tree is :
5
  4
    null
    null
  20
    18
      6
        null
        null
      null
    23
      null
      null

The Tree Above is Red-Black Tree
```

```
********************
Binary Tree is :
12
  8
    4
      2
        null
        null
      5
        null
        7
          null
          null
    11
      null
      null
  18
    17
      null
      null
    null

The Above Tree is Neither the Avl Tree nor the Red Black Tree
```

```
********************** BinarySearchTree-Test **************************

Average time for Binary Search Tree for 10k elements while inserting 100 random elements: 640432 ns
Average time for Binary Search Tree for 20k elements while inserting 100 random elements: 91697 ns
Average time for Binary Search Tree for 40k elements while inserting 100 random elements: 140386 ns
Average time for Binary Search Tree for 80k elements while inserting 100 random elements: 132954 ns

************************** Red-Black Tree-Test ************************************

Average time for Red Black Tree for 10k elements while inserting 100 random elements: 141468 ns
Average time for Red Black Tree for 20k elements while inserting 100 random elements: 117428 ns
Average time for Red Black Tree for 40k elements while inserting 100 random elements: 90540 ns
Average time for Red Black Tree for 80k elements while inserting 100 random elements: 121969 ns


****************************** Two-Three Tree- Test  ***************************************

Average time for TwoThree Tree for 10k elements while inserting 100 random elements: 120035 ns
Average time for TwoThree Tree for 20k elements while inserting 100 random elements: 108509 ns
Average time for TwoThree Tree for 40k elements while inserting 100 random elements: 143639 ns
Average time for TwoThree Tree for 80k elements while inserting 100 random elements: 171680 ns

********************************** B-Tree-Test **********************************

Average time for B-Tree for 10k elements while inserting 100 random elements: 89306 ns
Average time for B-Tree for 20k elements while inserting 100 random elements: 534595 ns
Average time for B-Tree for 40k elements while inserting 100 random elements: 119803 ns
Average time for B-Tree for 80k elements while inserting 100 random elements: 116477 ns


********************************* SkipList-Test *************************************

Average time for Skip List for 10k elements while inserting 100 random elements: 114249 ns
Average time for Skip List for 20k elements while inserting 100 random elements: 160469 ns
Average time for Skip List for 40k elements while inserting 100 random elements: 189036 ns
Average time for Skip List for 80k elements while inserting 100 random elements: 205490 ns
```