GTU DEPARTMENT OF COMUPETER ENGINEERING CSE 222/505 – SPRING 2021 HOMEWORK 7 REPORT

OZAN GEÇKİN

1801042103

PART 1:

1.System Requirements

We must implement the insertion, deletion, and iterator methods of Navigable Set using Skip list and AVL Tree.

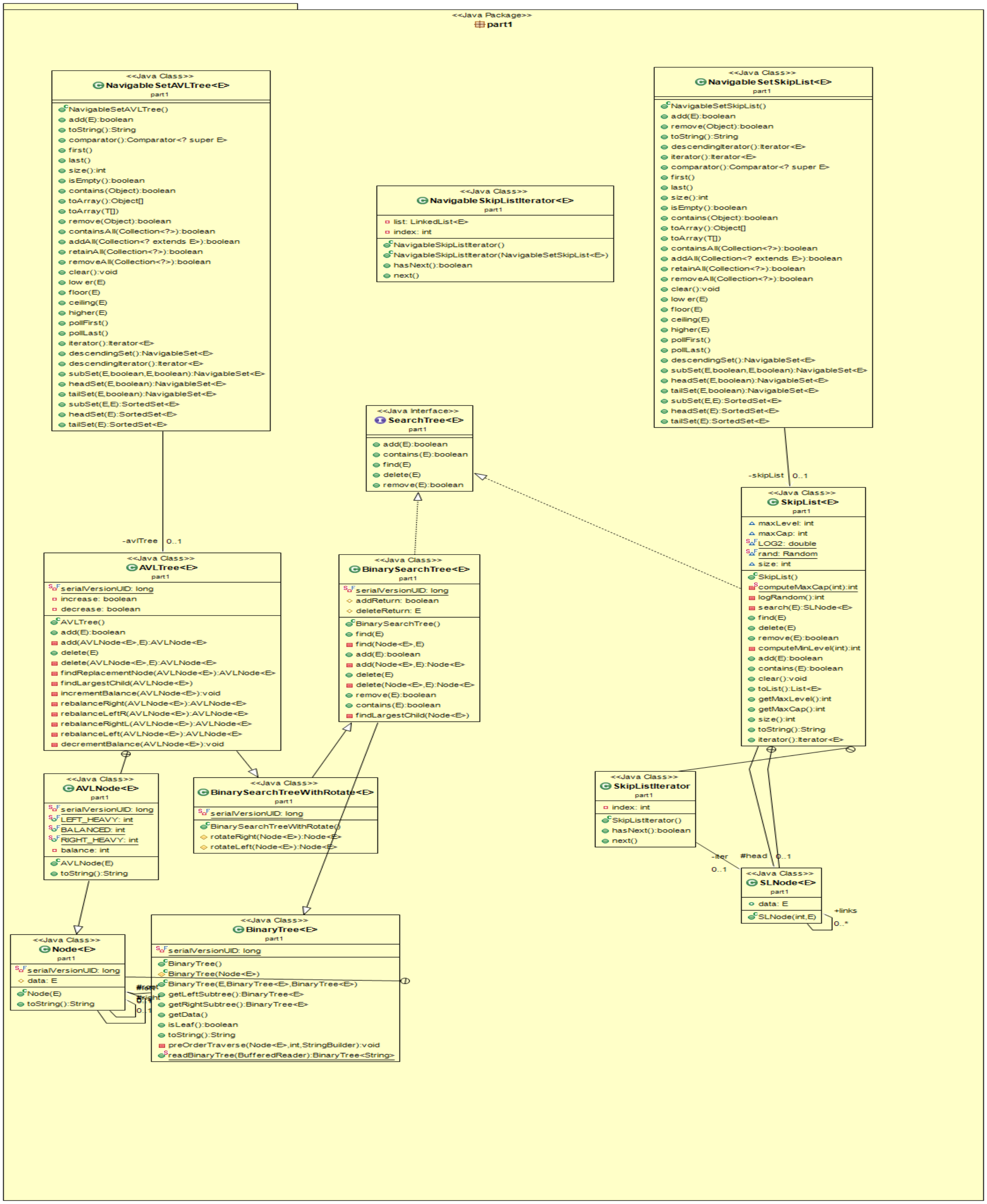
insert(E element): Takes element and adds while using SkipList or AVLTree.

delete(E element): Takes element , Searchs the element in the set and deletes it. Does this while using skip list or AVL Tree.

descendingIterator(): A standard iterator. Instead of traversing the elements from the start, it does this from the last place.

Note: I developed my program in eclipse java 1.8 on windows

2.Class Diagram



3.Problem Solution Approach

My Problem solution steps are;

-Specify the problem requirements

-Analyze the problem

-Design an algorithm and Program

-Implement the algorithm

-Test and verify the program

3.1) Specify the problem requirements: I understand the problem.

3.2) Analyze the problem: I identify; input data, output data, Additional requirements and constraints.

3.3)Design an algorithm and program: I implemented the data structures I need for Part 1 from the implement text book. I wrote the skipList iterator class as an inner class within the SkipList class. NavigableSetSkipList implements NavigableSet interface. It contains SkipList. It keeps data in SkipList. I have a NavigableSetSkipListIterator class. This is my iterator class. I made the necessary implementations. You can see the detailed explanation from the javadoc. I could only override the insert method in NavigableSetAVLTree.

4.Test Case

T	Test Case	Step	Expected Result	Actual Result	Pass/Fail
T1	Insert element (AVL)	Call insert method	Value inserted	Value inserted	Pass
T2	Insert element (Skip list)	Call insert method	Value inserted	Value inserted	Pass
T3	delete element (skip list)	Call delete method	Value deleted	Value inserted	Pass
T4	DescendingIterator (skip list)	Use while loop	Prints elements	implemented	Pass

5.Running and Results

T1,T2,T3,T4

```
Fill NavigableSetSkipList :
Level 0:1, 2, 3, 4, 5, 6, 7, 8,
Level 1:2, 3, 6,
Level 2:
Level 3:

Remove value 2 and 3 :
Level 0:1, 4, 5, 6, 7, 8,
Level 1:6,
Level 2:

Testing descending Iterator:
8
7
6
5
4
1

 Testing Filling NavigableSetSkipList:
Fill NavigableSetSkipList :
Level 0:10, 21, 32, 43, 54, 65, 76, 87,
Level 1:10, 32, 43, 65,
Level 2:43,
Level 3:

Remove value 21 and 32 :
Level 0:10, 43, 54, 65, 76, 87,
Level 1:10, 43, 65,
Level 2:43,

Testing descending Iterator:
87
76
65
54
43
10
1: 4
  0: 2
    0: 1
      null
      null
    0: 3
      null
      null
  1: 6
    0: 5
      null
      null
    1: 7
      null
      0: 8
        null
        null
```

PART 2:

1.System Requirements

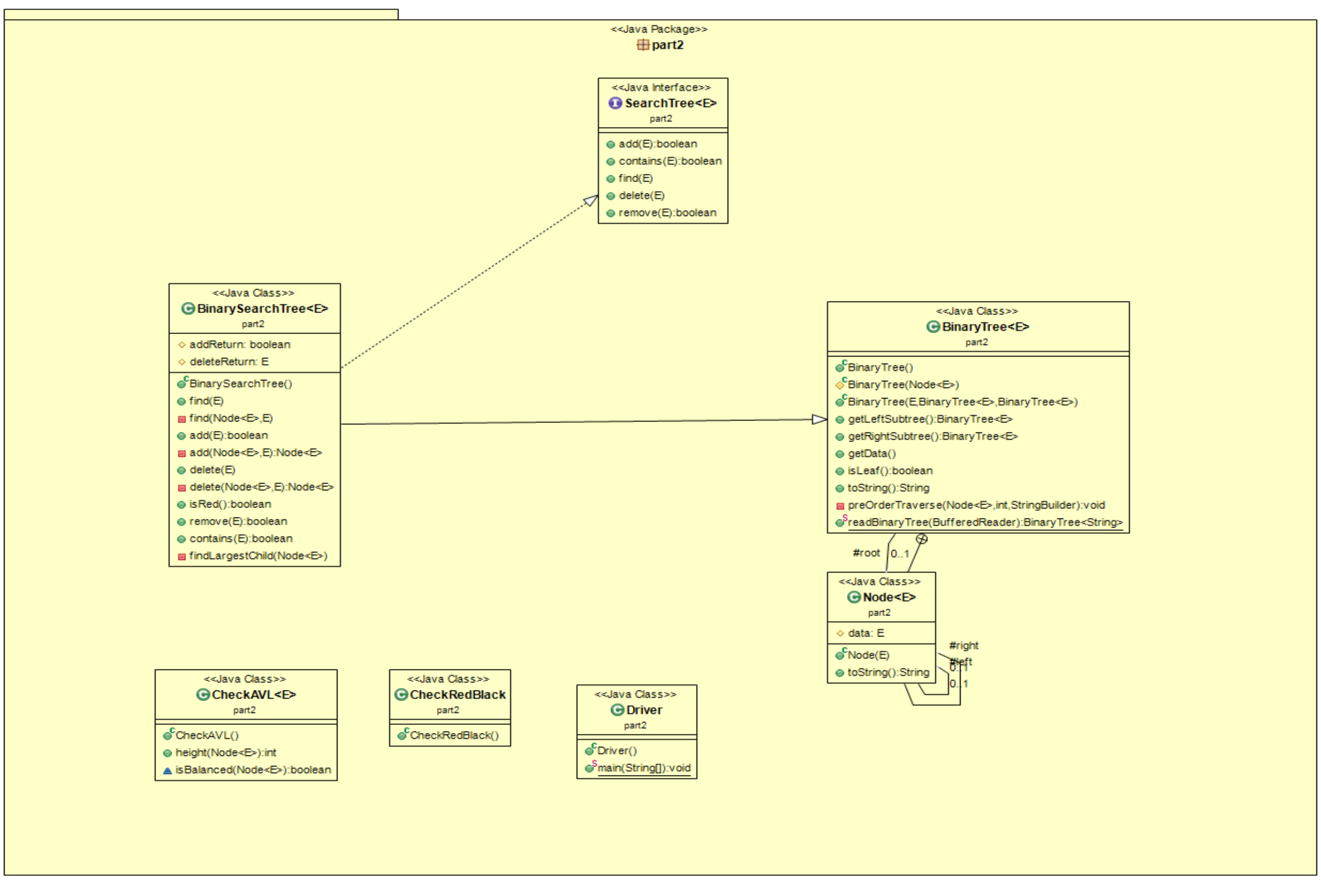
- isAVLTree(BinarySearchTree<E> tree): This method is required. It checks the BST as an AVL tree or not.

-isBalance(Node<E> node): This method is necessary because a BST tree must be in balance to be an AVL tree.

-height(Node<E> root) : This method is required. I need the height of a tree to decide if it is in balance.

Note: I developed my program in eclipse java 1.8 on windows

2.CLASS DIAGRAMS



3.PROBLEM SOLUTION APPROCH

My Problem solution steps are;

-Specify the problem requirements

-Analyze the problem

-Design an algorithm and Program

-Implement the algorithm

-Test and verify the program

3.1) Specify the problem requirements: I understand the problem.

3.2) Analyze the problem: I identify; input data, output data, Additional requirements and constraints.

3.3)Design an algorithm and program:

Binary Search Tree There are some rules for a tree to be an AVL tree or a Red Black Tree.I did my coding in line with these rules. First of all, I wrote my check classes. For a binary Search Tree to be an AVL tree, it must be in balance, so I wrote a function that checks for this condition. I've used it in auxiliary functions. For Red Black Tree Every node in the tree is either red or black. The root node is always black. All leaf nodes are black All children of any red node are black. All paths from any node to the leaf node contain an equal number of black nodes. I did my checks according to these rules. I could not perform the Red Black Tree implementation.

4.TEST CASE

TEST ID	TEST CASE	TEST STEPS	TEST DATA	EXPECTED RESULTS	ACTUAL RESULTS	PASS/ FAIL
T1	Check Binary Search Tree ,AVLTree	1.Create BST 2.Fill BST 3.Check BST is AVLTree	BST fill “100,50,49, 150, 48, 51, 149, 151	This Binary Search Tree is AVLTree	As Expected	Pass
T2	Check Binary Search Tree AVLTree	1.Create BST 2.Fill BST 3.Check BST is AVLTree	BST fill “1, 2, 3, 4, 5, 6, 7”	This Binary Search Tree is not AVLTree	As Expected	Pass
T3	Check Binary Search Tree ,Red Blcak Tree	1.Create BST 2.Fill BST	BST fill “100,50,49, 150, 48, 51, 149, 151	This Binary Search Tree is Red Blcak Tree	As Expected	Fail

		3.Check BST is Red Blcak Tree				
T4	Check Binary Search Tree Red Blcak Tree	1.Create BST 2.Fill BST 3.Check BST is Red Blcak Tree	BST fill “1, 2, 3, 4, 5, 6, 7”	This Binary Search Tree is not Red Blcak Tree	As Expected	Pass

5.RUNNING AND RESULTS

<table>
<tr><th>TEST ID</th><th>TEST RESULT</th></tr>
<tr><td>T1</td><td>100
  50
    49
      48
        null
        null
      null
    51
      null
      null
  150
    149
      null
      null
    151
      null
      null

This Binary Search Tree is AVLTree</td></tr>
<tr><td>T2</td><td>1
  null
  2
    null
    3
      null
      4
        null
        5
          null
          6
            null
            7
              null
              null

This Binary Search Tree is not AVLTree</td></tr>
</table>

T3	`This Binary Search Tree is not Red Black Tree`
T4	`This Binary Search Tree is not Red Black Tree`

PART 3:

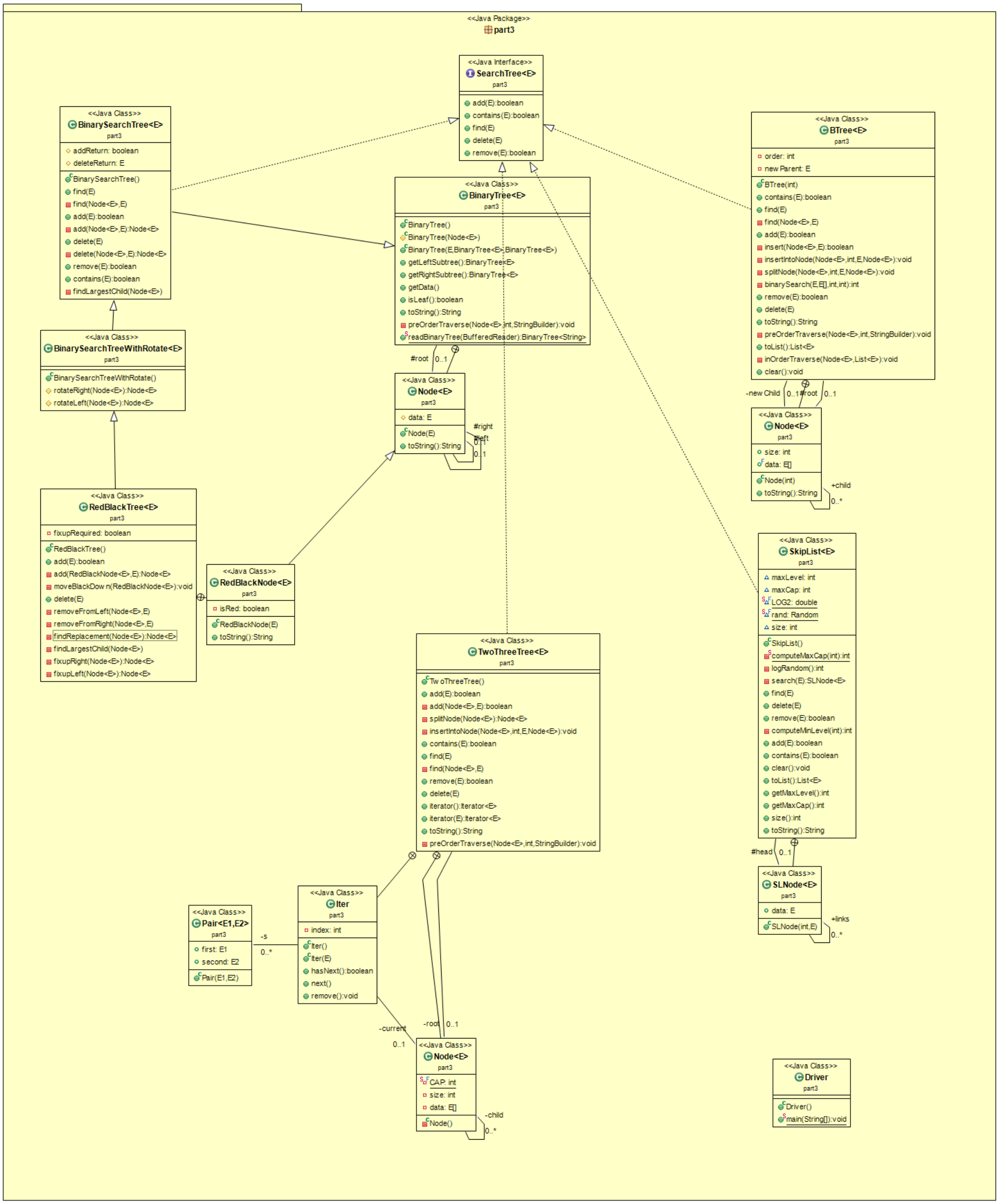
1.System Requirement

Implementation of Binary Search Tree, Implementation of Red Black Tree, Implementation of 2-3 Tree, Implementation of B-Tree, Implementation of Skip List,

A way to measure time Instances of every tree for 4 times with sizes 10k, 20k, 40k and 80k

Test every instance for 10 times. This will result in 200 time results. Graph these results.

2.CLASS DIAGRAMS



3.PROBLEM SOLUTION APPROACH

Problem Solution Approach

My Problem solution steps are;

-Specify the problem requirements

-Analyze the problem

-Design an algorithm and Program

-Implement the algorithm

-Test and verify the program

3.1) Specify the problem requirements: I understand the problem.

3.2) Analyze the problem: I identify; input data, output data, Additional requirements and constraints.

3.3)Design an algorithm and program: This part is to performance test of data structures given to me and show it in graphics. I implemented the necessary data structures in the text book. I implemented the interfaces and classes required for the data structures I implemented.

- Binary Search Tree
- Red-Black Tree
- 2-3 Tree
- B-tree
- Skip List

3.4) Implement the algorithm:

I initiate the data structures I implemented in my Driver class. I added 10000, 20000, 40000, 80000 random and unique numbers to each of these data structures 10 times. I added 100 random and unique numbers on it. I kept the running times while adding an extra 100 numbers to each data structure. And I took the averages and had them printed. I have graphed these results using excel.

4.TEST CASE

TEST ID	TEST CASE	TEST STEPS	TEST DATA	EXPECTED RESULTS	ACTUAL RESULTS	PASS/ FAIL
T1	Test Binary Search Tree	1)Create Random and unique Numbers 2)Insert 3)Insert extra 100 random and unique 4) Performance test of data structure 10 times and calculation of average performance time	10.000, 20.000, 40.000 and 80.000 random numbers	Test is successful	As Expected	Pass
T2	Test Red-Black Tree	1)Create Random and unique Numbers 2)Insert 3)Insert extra 100 random and unique 4) Performance test of data structure 10 times and calculation of average performance time	10.000, 20.000, 40.000 and 80.000 random numbers	Test is successful	As Expected	Pass
T3	Test 2-3 Tree	1)Create Random and unique Numbers 2)Insert 3)Insert extra 100 random and unique 4) Performance test of data structure 10 times and calculation of average performance time	10.000, 20.000, 40.000 and 80.000 random numbers	Test is successful	As Expected	Pass
T4	Test B-tree	1)Create Random and unique Numbers 2)Insert 3)Insert extra 100 random and unique 4) Performance test of data structure 10 times and calculation of average performance time	10.000, 20.000, 40.000 and 80.000 random numbers	Test is successful	As Expected	Pass
T5	Test Skip List	1)Create Random and unique Numbers	10.000, 20.000, 40.000 and	Test is successful	As Expected	Pass

		2)Insert 3)Insert extra 100 random and unique 4) Performance test of data structure 10 times and calculation of average performance time	80.000 random numbers			

5.RUNNING AND RESULTS

TEST ID	TEST RESULT
T1	adding 100 extra elements to the Binary Search Tree, which is filled with 10000 random and unique elements 10 times : 39 µs adding 100 extra elements to the Binary Search Tree, which is filled with 20000 random and unique elements 10 times : 37 µs adding 100 extra elements to the Binary Search Tree, which is filled with 40000 random and unique elements 10 times : 45 µs adding 100 extra elements to the Binary Search Tree, which is filled with 80000 random and unique elements 10 times : 52 µs
T2	adding 100 extra elements to the Red Black Tree, which is filled with 10000 random and unique elements 10 times : 44 µs adding 100 extra elements to the Red Black Tree, which is filled with 20000 random and unique elements 10 times : 31 µs adding 100 extra elements to the Red Black Tree, which is filled with 40000 random and unique elements 10 times : 38 µs adding 100 extra elements to the Red Black Tree, which is filled with 80000 random and unique elements 10 times : 49 µs
T3	adding 100 extra elements to the 2-3 Tree, which is filled with 10000 random and unique elements 10 times : 102 µs adding 100 extra elements to the 2-3 Tree, which is filled with 20000 random and unique elements 10 times : 49 µs adding 100 extra elements to the 2-3 Tree, which is filled with 40000 random and unique elements 10 times : 54 µs adding 100 extra elements to the 2-3 Tree, which is filled with 80000 random and unique elements 10 times : 66 µs
T4	adding 100 extra elements to the B-Tree, which is filled with 10000 random and unique elements 10 times : 53 µs adding 100 extra elements to the B-Tree, which is filled with 20000 random and unique elements 10 times : 42 µs adding 100 extra elements to the B-Tree, which is filled with 40000 random and unique elements 10 times : 36 µs adding 100 extra elements to the B-Tree, which is filled with 80000 random and unique elements 10 times : 43 µs
T5	adding 100 extra elements to the Skip List, which is filled with 10000 random and unique elements 10 times : 59 µs adding 100 extra elements to the Skip List, which is filled with 20000 random and unique elements 10 times : 64 µs adding 100 extra elements to the Skip List, which is filled with 40000 random and unique elements 10 times : 80 µs adding 100 extra elements to the Skip List, which is filled with 80000 random and unique elements 10 times : 98 µs

6.GRAPHICS