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

Atakan ALTIN
1801042668

1. SYSTEM REQUIREMENTS

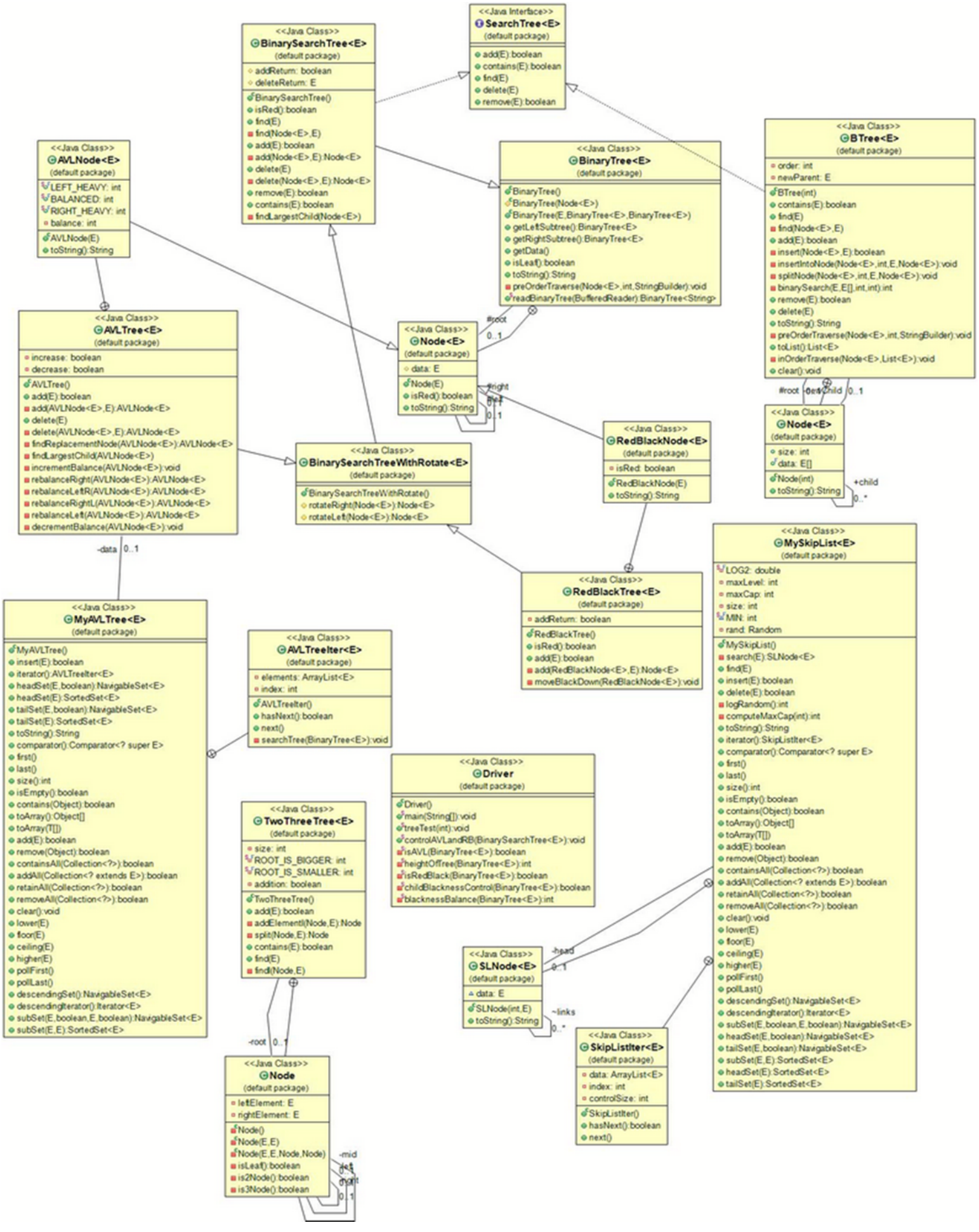
1.1 Functional Requirements

- User should be able to insert any generic element into the skip-list via insert function. Insert function takes one parameter which will be added item. Insert function calls search function to be able to put in correct position in the skip-list. If the capacity is full, function increases capacity.
- User should be able to delete any element from the skip-list. Delete function takes one parameter which will be deleted item. Delete function calls search function to be able to delete from correct position in the skip-list. Function also links the remaining elements to prevent disconnection.
- User should be able to use descendingIterator to see elements with descending order. Skip-list stores elements with increasing order so iterator first store all elements in array-list than it moves reverse-order to provide descending order.
- User should be able to insert any generic element into the AVL tree via insert function. Insert function takes one parameter which will be added item. Function must rebalance both left and right rotation after insert operation.
- User should be able to see all items in the AVL tree via iterator. Iterator traverses all tree just like search operation in the binary tree class.
- User should be able to see all elements which are smaller than given parameter. headSet function traverses all tree via iterator and adds smaller elements into the AVL tree which is Navigable Set then returns it. There is also overloaded version which takes Boolean parameter to be able to indicate given parameter is inclusive or exclusive.
- User should be able to see all elements which are bigger than given parameter. tailSet function traverses all tree via iterator and adds bigger elements into the AVL tree which is Navigable Set then returns it. There is also overloaded version which takes Boolean parameter to be able to indicate given parameter is inclusive or exclusive.
- There is a static controlAVLandRB function in the Driver class. It takes BinarySearchTree and checks both if the tree is AVL Tree or Red-Black tree. It prints the result in the terminal screen. This function calls some helper functions to check if the given tree satisfies AVL and Red-Black conditions.

1.2 Non-Functional Requirements

- Hardware should be able to run at least Java SE13.

2. USE CASE AND CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

MyAVLTree and MySkipList are created and they implements NavigableSet Interface. Skip-list class directly implements skip-list structure and asked methods. AVL tree class uses book implementation as a data field and implements asked methods. Asked functions implementations explained in detail in system requirements. While controlling if given binary tree is an AVL tree, each node's height is calculated and checked if the absolute value of balance is equal or smaller than 1 which is condition of being AVL tree. While controlling if given binary tree is a Red-Black tree, root of tree is checked whether it is red or not. Secondly, childs of the parent node are checked whether they are both black or not. Finally, number of black childs are checked for both left and right sides whether they are equal or not. If all conditions are satisfied, function decides given binary search tree is a Red-Black tree. In the last part, all implementations of asked data structures are compared with their insertion performance. Insert operation performs 10 times for all data structures with different sizes. Inserted elements are chosen randomly and uniquely. Results of average running times are shown in the terminal screen.

4. TEST CASES

- Add some elements into skip list.
- Remove an existing element from skip list.
- Remove a non-existing element from skip list.
- Print all elements in the skip list via skip list iterator.
- Try to reach un-accessible element via skip list iterator.
- Add some elements into AVL tree.
- Print all elements in the AVL tree via AVL tree iterator.
- Try to reach un-accessible element via AVL tree iterator.
- Create Headset with some value in the AVL tree.
- Create Tailset with some value in the AVL tree.
- Create Headset with non-existing value in the AVL tree.
- Create Tailset with non-existing value in the AVL tree.
- Check if ordinary binary search tree is AVL tree or Red-Black tree
- Check if AVL tree is AVL tree or Red-Black tree
- Check if Red-Black tree is AVL tree or Red-Black tree
- Compare insertion performance of BST, Red-Black tree,2-3 Tree, B-tree and skip- list. Perform insertion for 10 times with the 10.000, 20.000, 40.000 and 80.000 random numbers for all structures. Then add 100 additional numbers to all structures to calculate average performance.

5. RUNNING AND RESULTS

```
atakan@DESKTOP-7U8OIGQ:/mnt/e/CSE/CSE222/HW7$ javac *.java
atakan@DESKTOP-7U8OIGQ:/mnt/e/CSE/CSE222/HW7$ java Driver
------------ Adding elements to SkipList --------------

MaxLevel: 4 --> 5 |2| --> 10 |1| --> 15 |1| --> 20 |1| --> 25 |2| --> 30 |1| --> 35 |1| --> 40 |1| --> 45 |1| --> 50 |1|
```

```
------------Removing 35(exist) from SkipList--------------

MaxLevel: 4 --> 5 |2| --> 10 |1| --> 15 |1| --> 20 |1| --> 25 |2| --> 30 |1| --> 40 |1| --> 45 |1| --> 50 |1|
```

```
------------Removing 27(non-exist) from SkipList--------------

Element is not exist.
MaxLevel: 4 --> 5 |2| --> 10 |1| --> 15 |1| --> 20 |2| --> 25 |2| --> 30 |1| --> 40 |1| --> 45 |2| --> 50 |2|
```

```
--------------ITERATOR TEST---------------

50
45
40
30
25
20
15
10
5
```

```
------------Trying to reach next element after iteration done for all items--------------

Iterator Cannot Reach This Element.
```

```
------------Adding Elements to AVL Tree--------------
0: 5
  -1: 3
    0: 2
      null
      null
    null
  0: 9
    0: 6
      null
      null
    0: 10
      null
      null

------------Deleting Elements is not tested because it is not asked--------------
```

```
--------------ITERATOR TEST---------------

5
3
2
9
6
10
```

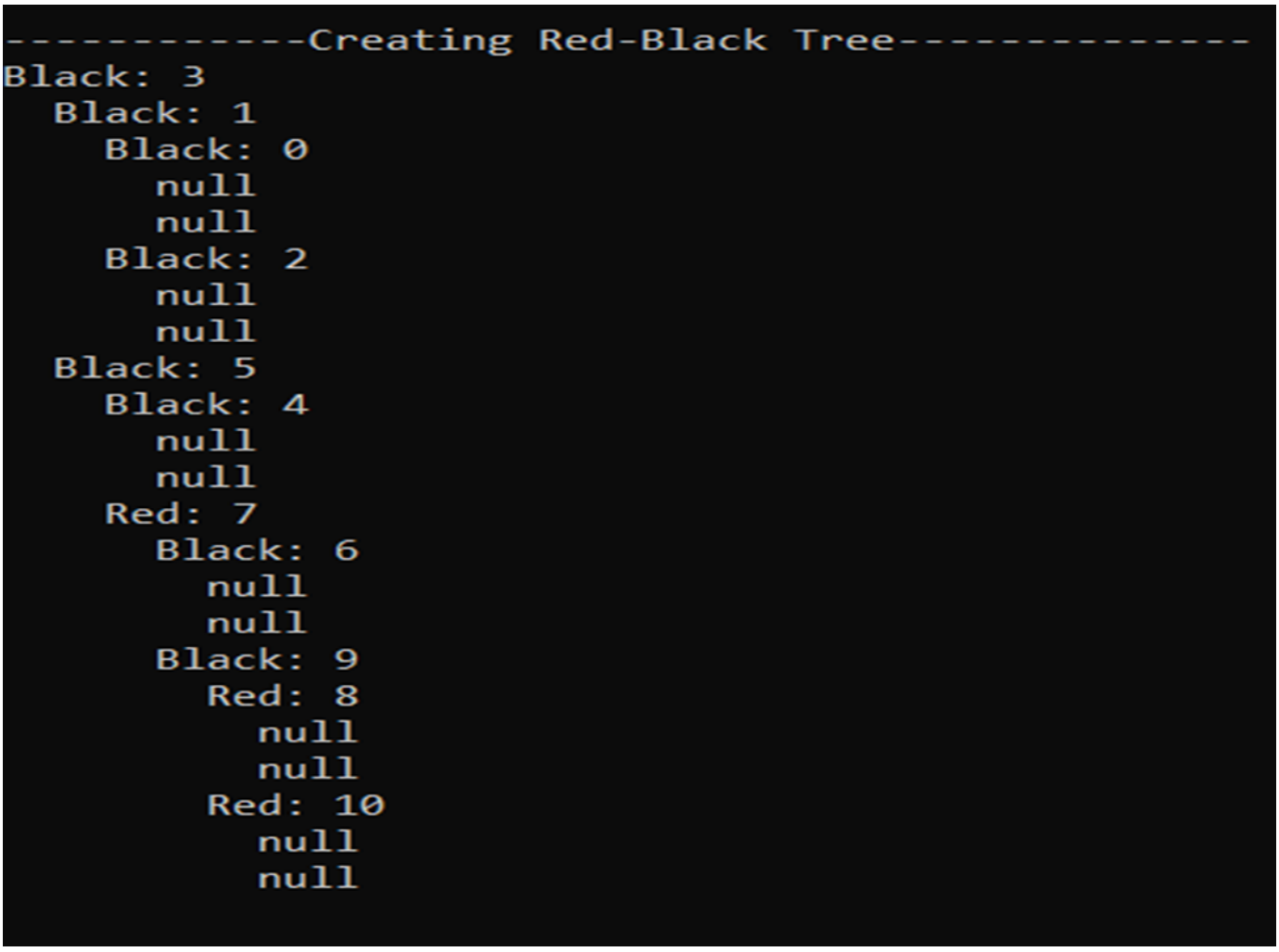
```
------------Trying to reach next element after iteration done for all items-------------

Iterator Cannot Reach This Element.
```

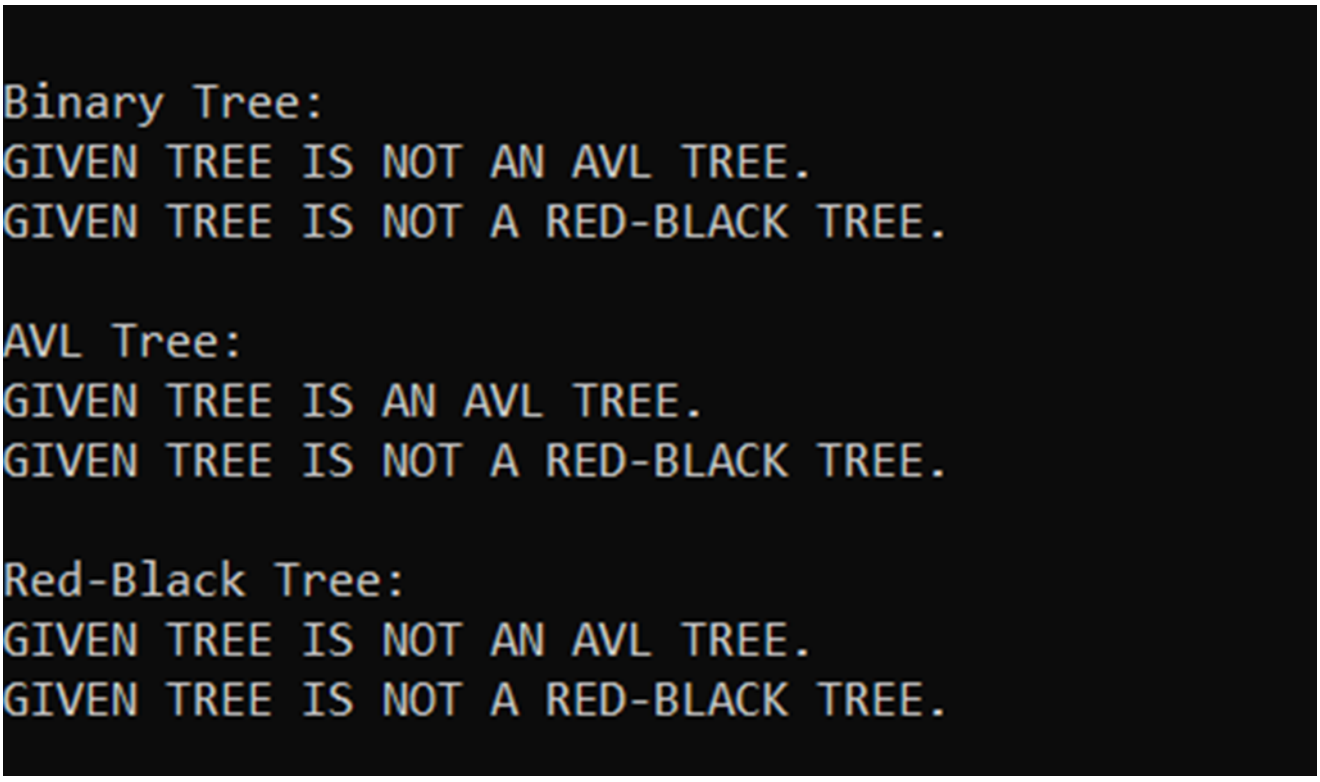
```
------------Testing Headset for parameter 5(inclusive)--------------
3
2
5
------------Testing Tailset for parameter 5(exclusive)--------------
9
6
10
```

```
-----------Testing Headset for parameter 33(Non-existing)-------------
-----------Testing Tailset for parameter 33(Non-existing)-------------
```

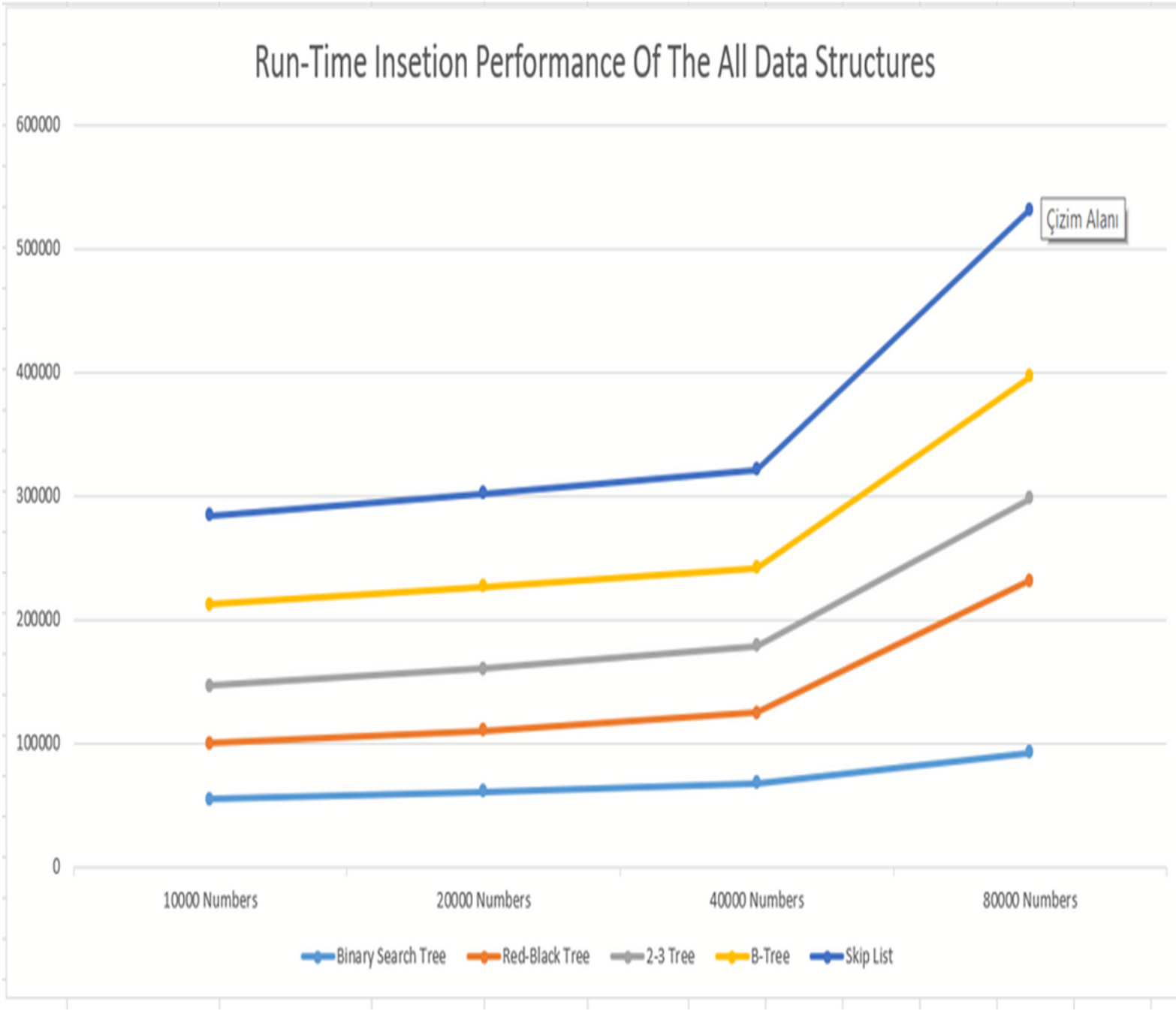
```
-----------Creating Ordinary Binary Search Tree--------------
1
  null
  2
    null
    3
      null
      4
        null
        5
          null
          null
```

```
-------------Creating AVL Tree---------------
-1: 4
  0: 2
    0: 1
      null
      null
    0: 3
      null
      null
  0: 5
    null
    null
```


----------Creating Red-Black Tree----------
Black: 3
Black: 1
Black: 0
null
null
Black: 2
null
null
Black: 5
Black: 4
null
null
Red: 7
Black: 6
null
null
Black: 9
Red: 8
null
null
Red: 10
null
null


Binary Tree:
GIVEN TREE IS NOT AN AVL TREE.
GIVEN TREE IS NOT A RED-BLACK TREE.
AVL Tree:
GIVEN TREE IS AN AVL TREE.
GIVEN TREE IS NOT A RED-BLACK TREE.
Red-Black Tree:
GIVEN TREE IS NOT AN AVL TREE.
GIVEN TREE IS NOT A RED-BLACK TREE.

```
-----------PART3-------------
TESTING FOR  10000 ITEMS:
BINARY SEARCH TREE AVG. INSERTION TIME: 54810 nano seconds
RED-BLACK TREE AVG. INSERTION TIME: 45390 nano seconds
TWO THREE TREE AVG. INSERTION TIME: 46820 nano seconds
B-TREE AVG. INSERTION TIME: 65550 nano seconds
SKIP LIST AVG. INSERTION TIME: 72140 nano seconds

TESTING FOR  20000 ITEMS:
BINARY SEARCH TREE AVG. INSERTION TIME: 60910 nano seconds
RED-BLACK TREE AVG. INSERTION TIME: 49630 nano seconds
TWO THREE TREE AVG. INSERTION TIME: 50120 nano seconds
B-TREE AVG. INSERTION TIME: 66510 nano seconds
SKIP LIST AVG. INSERTION TIME: 75220 nano seconds

TESTING FOR  40000 ITEMS:
BINARY SEARCH TREE AVG. INSERTION TIME: 68410 nano seconds
RED-BLACK TREE AVG. INSERTION TIME: 56560 nano seconds
TWO THREE TREE AVG. INSERTION TIME: 53860 nano seconds
B-TREE AVG. INSERTION TIME: 63240 nano seconds
SKIP LIST AVG. INSERTION TIME: 79580 nano seconds

TESTING FOR  80000 ITEMS:
BINARY SEARCH TREE AVG. INSERTION TIME: 92590 nano seconds
RED-BLACK TREE AVG. INSERTION TIME: 139400 nano seconds
TWO THREE TREE AVG. INSERTION TIME: 66330 nano seconds
B-TREE AVG. INSERTION TIME: 98340 nano seconds
SKIP LIST AVG. INSERTION TIME: 135040 nano seconds
atakan@DESKTOP-7U8OIGQ:/mnt/e/CSE/CSE222/HW7$
```


Run-Time Insetion Performance Of The All Data Structures
600000
500000
400000
300000
200000
100000
0
Çizim Alanı
10000 Numbers
20000 Numbers
40000 Numbers
80000 Numbers
Binary Search Tree
Red-Black Tree
2-3 Tree
B-Tree
Skip List