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

Yunus Emre Öztürk
1901042619

1. SYSTEM REQUIREMENTS

Non-Functional System Requirements:

- 100 KB memory (For tests)
- Java Runtime Environment

Functional System Requirements:

Skiplist:

```
    public boolean insert(E item)
```

For insert user should give item, it shouldn’t be null it returns false if item is already in the set.

```
public boolean remove(Object o)
```

For remove user should give an object, it should have same generic type and shouldn’t be null. It will return false if item is not in the set.

```
public Iterator<E> descendingIterator()
```

Returns a descendingIterator to user, which can be used to travel through the set.

AVLTreeSet:

```
public boolean insert(E item)
```

For insert user should give item, it shouldn’t be null.

```
public Iterator<E> iterator()
```

Returns a Iterator to user, which can be used to travel through the set.

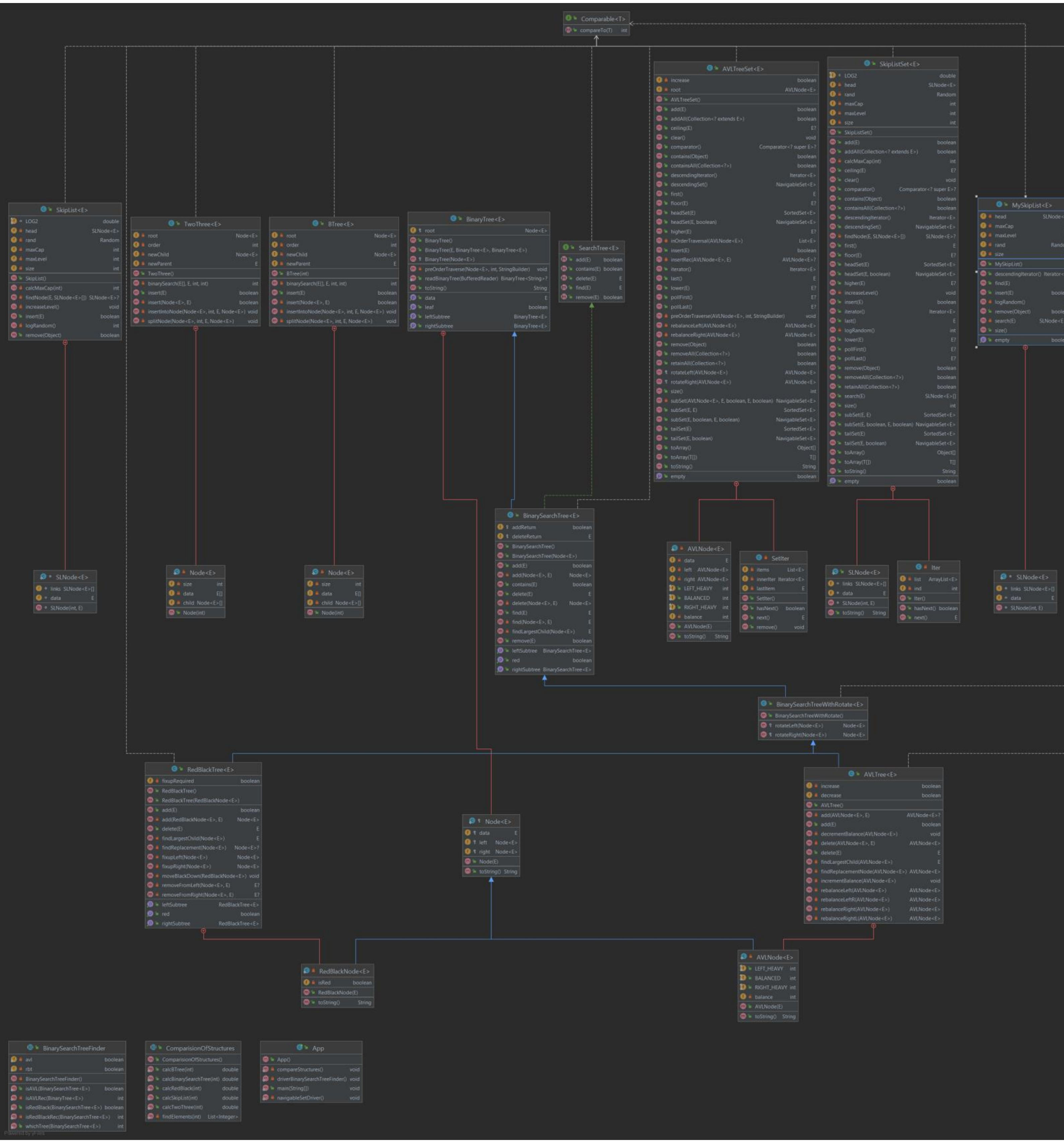
```
public SortedSet<E> headSet(E toElement)
```

Returns a headset which has all the elements lower than given element.

```
public SortedSet<E> tailSet(E fromElement)
```

Returns a headset which has all the elements higher and equal to given element.

2. CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

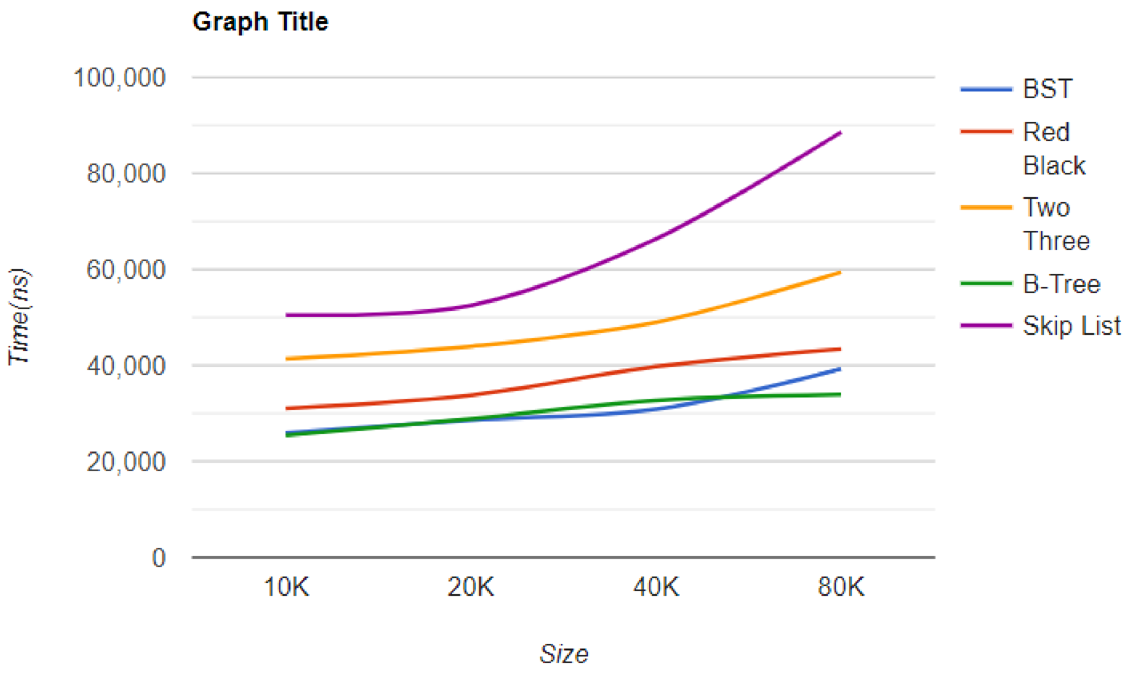
In first part for skiplist I should implement insertion, deletion and descendingIterator. In implementation of insertion I used SkipList diagram and addition to it I checked the given element is in the set or not. For deletion I used SkipList deletion algorithm, to delete head I take the next element if there is and increased it's level to max level then connected it's levels to correct nodes. To create a descending iterator first I put all the datas to an array and then use a cursor to track it.

For AVLTreeSet I used AVLTree insertion algorithm. For iterator first I travelled all set with inorder traversal and then save elements to an list then used that list for iterator methods. For headset I recursively traverse nodes can be lower than given element and then returned the set. For tailset I recursively traverse nodes can be higher and equal to the given element and then returned the set.

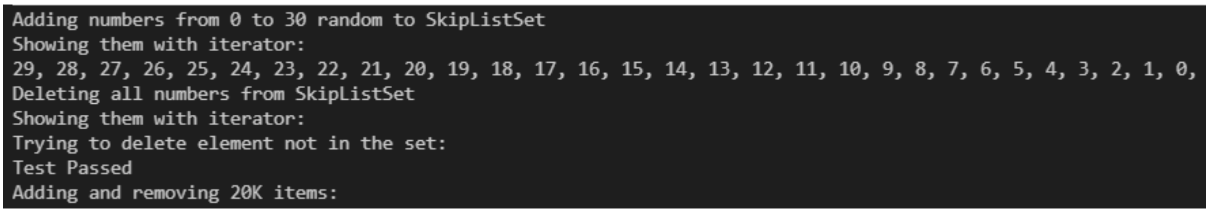
In second part to detect AVLTree I checked all nodes and their biggest left and right heights recursively, if method detects a height difference bigger or equal to two it says it is not an AVLTree. To detect RedBlackTree I used isRed method and black node count as detectors in my algorithm, method looks root node black or not and then recursively looks all nodes and checks if a red node has red child or not. It also looks right and left black node count if there is a difference it detects that the given tree is not a RedBlackTree.

In third part I used methods for each structure which takes an integer as parameter which represents the structure size. It takes given count + 100 different elements and tries 10 iteration and returns the average time consumption in ns.

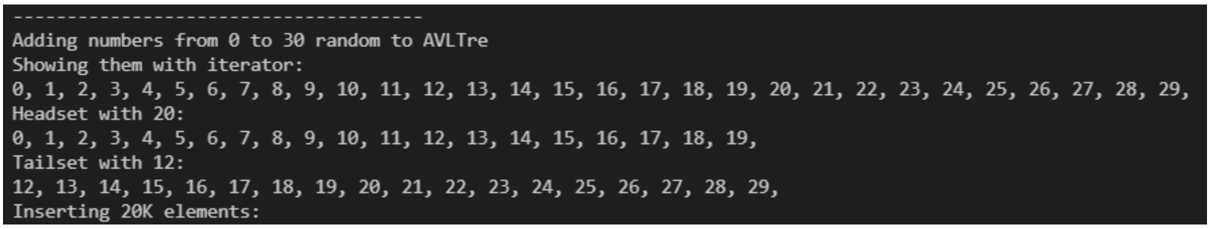
4. TEST CASES / RUNNING AND RESULTS



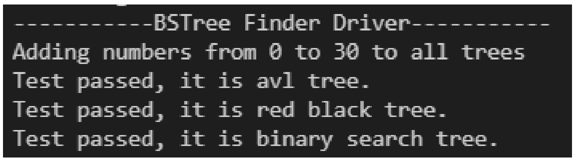
Other than b-tree al the structures that i’ve tested starts to give worse performance after big insertions, especially skiplist.



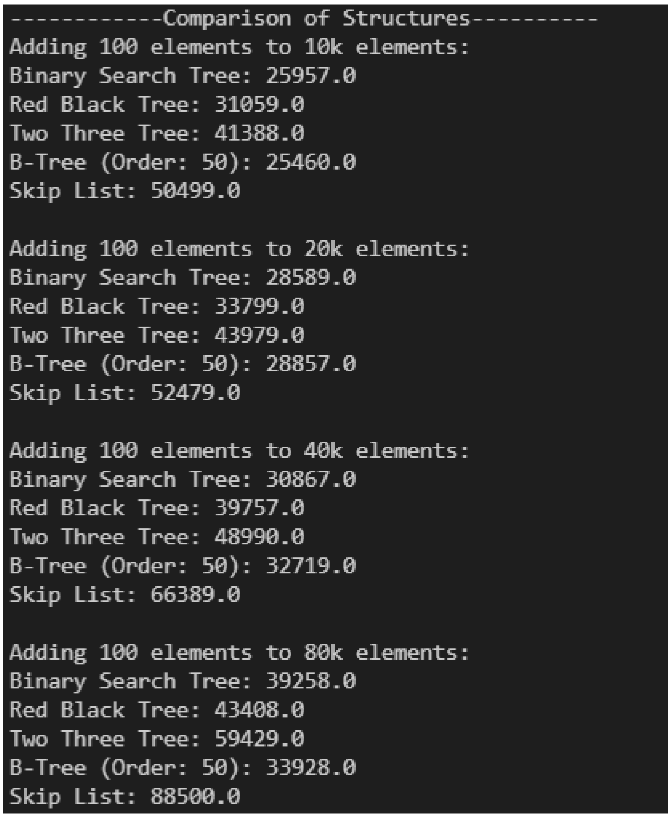
```
Adding numbers from 0 to 30 random to SkipListSet
Showing them with iterator:
29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
Deleting all numbers from SkipListSet
Showing them with iterator:
Trying to delete element not in the set:
Test Passed
Adding and removing 20K items:
```



```
-------------------------------------
Adding numbers from 0 to 30 random to AVLTre
Showing them with iterator:
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,
Headset with 20:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
Tailset with 12:
12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
Inserting 20K elements:
```



```
-----------BSTree Finder Driver-----------
Adding numbers from 0 to 30 to all trees
Test passed, it is avl tree.
Test passed, it is red black tree.
Test passed, it is binary search tree.
```



```
------------Comparison of Structures----------
Adding 100 elements to 10k elements:
Binary Search Tree: 25957.0
Red Black Tree: 31059.0
Two Three Tree: 41388.0
B-Tree (Order: 50): 25460.0
Skip List: 50499.0

Adding 100 elements to 20k elements:
Binary Search Tree: 28589.0
Red Black Tree: 33799.0
Two Three Tree: 43979.0
B-Tree (Order: 50): 28857.0
Skip List: 52479.0

Adding 100 elements to 40k elements:
Binary Search Tree: 30867.0
Red Black Tree: 39757.0
Two Three Tree: 48990.0
B-Tree (Order: 50): 32719.0
Skip List: 66389.0

Adding 100 elements to 80k elements:
Binary Search Tree: 39258.0
Red Black Tree: 43408.0
Two Three Tree: 59429.0
B-Tree (Order: 50): 33928.0
Skip List: 88500.0
```