CSE 222
HOMEWORK 7
REPORT

Anıl Mert Ulaşan
171044004

Problem Solution Approach

Part 1

I kept private skiplist and AVLTree objects in my classes. I used skiplist and avltree objects' methods inside necessary method implementations.

Most of the requested methods are not implemented.

Part 2

I created a method Balance() in BinarySearchTree class. Method returns (Left Tree's Height)-(Right subTree's Height).

In my isAVL method, it checks the value of Balance() method. If it's between -1 and 1, returns true, else returns false.

In my isRedBlack method, it returns isRed() method of BinarySearchTree class.

Part 3

When generating random numbers, to avoid repeat, i added them to a local LinkedHashSet object. If add operation returns null, generate another random number and try again. With that way i make sure there is not any repeating number.

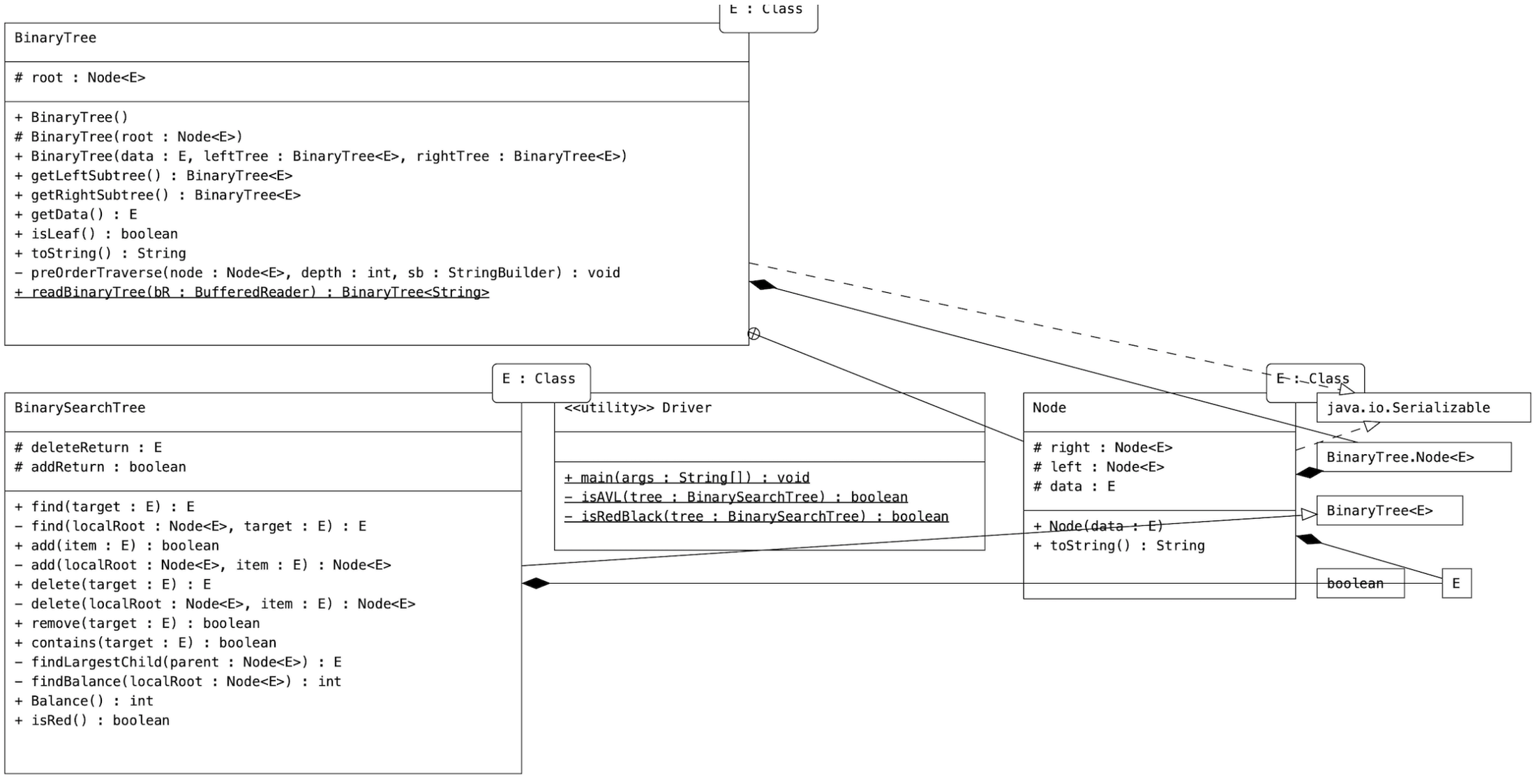
Then i add the numbers to my ArrayList randoms object. This count of values are added to every Tree type. Then measured the time of every tree type's 100 element insertion operation. This is repeated 10 times and returned the average for every tree type.

The above operation is done to every size(10k, 20k…) and printed the averages for every tree type of every size.

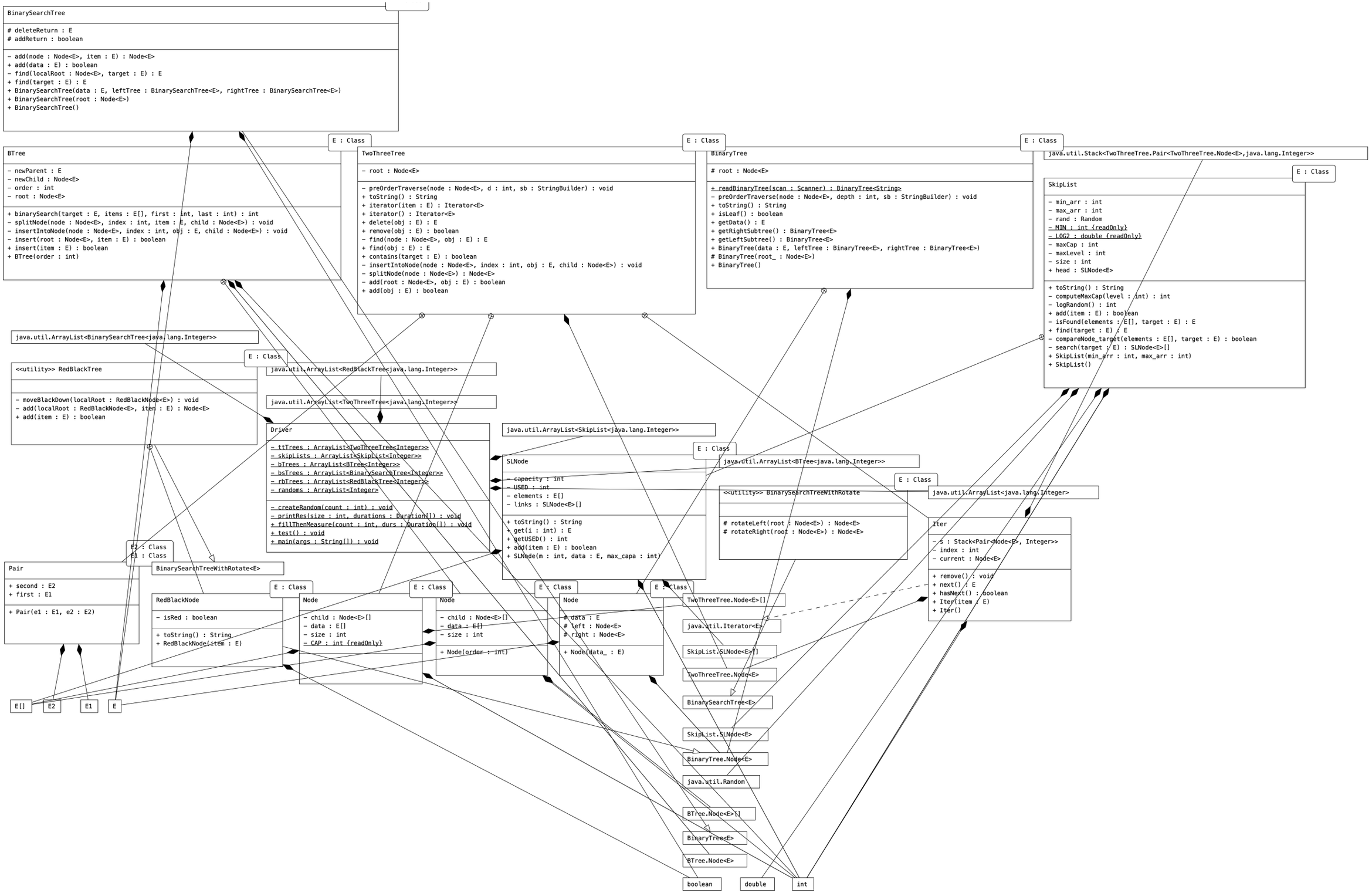
Just printed test results to stdin, no graph.

Class Diagrams

Part 2



Part 3


BinarySearchTree
deleteReturn : E
addReturn : boolean
- add(node : Node<E>, item : E) : Node<E>
+ add(data : E) : boolean
- find(localRoot : Node<E>, target : E) : E
+ find(target : E) : E
+ BinarySearchTree(data : E, leftTree : BinarySearchTree<E>, rightTree : BinarySearchTree<E>)
+ BinarySearchTree(root : Node<E>)
+ BinarySearchTree()
BTree
- newParent : E
- newChild : Node<E>
- order : int
- root : Node<E>
+ binarySearch(target : E, items : E[], first : int, last : int) : int
- splitNode(node : Node<E>, index : int, item : E, child : Node<E>) : void
- insertIntoNode(node : Node<E>, index : int, obj : E, child : Node<E>) : void
- insert(root : Node<E>, item : E) : boolean
+ insert(item : E) : boolean
+ BTree(order : int)
E : Class
TwoThreeTree
- root : Node<E>
- preOrderTraverse(node : Node<E>, d : int, sb : StringBuilder) : void
+ toString() : String
+ iterator(item : E) : Iterator<E>
+ iterator() : Iterator<E>
+ delete(obj : E) : E
+ remove(obj : E) : boolean
- find(node : Node<E>, obj : E) : E
+ find(obj : E) : E
+ contains(target : E) : boolean
- insertIntoNode(node : Node<E>, index : int, obj : E, child : Node<E>) : void
- splitNode(node : Node<E>) : Node<E>
- add(root : Node<E>, obj : E) : boolean
+ add(obj : E) : boolean
E : Class
BinaryTree
root : Node<E>
+ readBinaryTree(scan : Scanner) : BinaryTree<String>
- preOrderTraverse(node : Node<E>, depth : int, sb : StringBuilder) : void
+ toString() : String
+ isLeaf() : boolean
+ getData() : E
+ getRightSubtree() : BinaryTree<E>
+ getLeftSubtree() : BinaryTree<E>
+ BinaryTree(data : E, leftTree : BinaryTree<E>, rightTree : BinaryTree<E>)
# BinaryTree(root_ : Node<E>)
+ BinaryTree()
E : Class
java.util.Stack<TwoThreeTree.Pair<TwoThreeTree.Node<E>,java.lang.Integer>>
E : Class
SkipList
- min_arr : int
- max_arr : int
- rand : Random
- MIN : int {readOnly}
- LOG2 : double {readOnly}
- maxCap : int
- maxLevel : int
- size : int
+ head : SLNode<E>
+ toString() : String
- computeMaxCap(level : int) : int
- logRandom() : int
+ add(item : E) : boolean
- isFound(elements : E[], target : E) : E
+ find(target : E) : E
- compareNode_target(elements : E[], target : E) : boolean
- search(target : E) : SLNode<E>[]
+ SkipList(min_arr : int, max_arr : int)
+ SkipList()
java.util.ArrayList<BinarySearchTree<java.lang.Integer>>
E : Class
java.util.ArrayList<RedBlackTree<java.lang.Integer>>
java.util.ArrayList<TwoThreeTree<java.lang.Integer>>
<<utility>> RedBlackTree
- moveBlackDown(localRoot : RedBlackNode<E>) : void
- add(localRoot : RedBlackNode<E>, item : E) : Node<E>
+ add(item : E) : boolean
Driver
- ttTrees : ArrayList<TwoThreeTree<Integer>>
- skipLists : ArrayList<SkipList<Integer>>
- bTrees : ArrayList<BTree<Integer>>
- bsTrees : ArrayList<BinarySearchTree<Integer>>
- rbTrees : ArrayList<RedBlackTree<Integer>>
- randoms : ArrayList<Integer>
- createRandom(count : int) : void
- printRes(size : int, durations : Duration[]) : void
+ fillThenMeasure(count : int, durs : Duration[]) : void
+ test() : void
+ main(args : String[]) : void
java.util.ArrayList<SkipList<java.lang.Integer>>
E : Class
SLNode
- capacity : int
- USED : int
- elements : E[]
- links : SLNode<E>[]
+ toString() : String
+ get(i : int) : E
+ getUSED() : int
+ add(item : E) : boolean
+ SLNode(m : int, data : E, max_capa : int)
E : Class
java.util.ArrayList<BTree<java.lang.Integer>>
E : Class
java.util.ArrayList<java.lang.Integer>
<<utility>> BinarySearchTreeWithRotate
rotateLeft(root : Node<E>) : Node<E>
rotateRight(root : Node<E>) : Node<E>
Iter
- s : Stack<Pair<Node<E>, Integer>>
- index : int
- current : Node<E>
+ remove() : void
+ next() : E
+ hasNext() : boolean
+ Iter(item : E)
+ Iter()
E2 : Class
E1 : Class
Pair
+ second : E2
+ first : E1
+ Pair(e1 : E1, e2 : E2)
BinarySearchTreeWithRotate<E>
E : Class
RedBlackNode
- isRed : boolean
+ toString() : String
+ RedBlackNode(item : E)
E : Class
Node
- child : Node<E>[]
- data : E[]
- size : int
- CAP : int {readOnly}
E : Class
Node
- child : Node<E>[]
- data : E[]
- size : int
+ Node(order : int)
E : Class
Node
+ data : E
# left : Node<E>
# right : Node<E>
+ Node(data_ : E)
E : Class
TwoThreeTree.Node<E>[]
java.util.Iterator<E>
SkipList.SLNode<E>[]
TwoThreeTree.Node<E>
BinarySearchTree<E>
SkipList.SLNode<E>
BinaryTree.Node<E>
java.util.Random
BTree.Node<E>[]
BinaryTree<E>
BTree.Node<E>
E[]
E2
E1
E
boolean
double
int

Test Cases

Part 2

1) Create a BinarySearchTree object and fill with some values which makes the Right subTree's height way bigger than left.

See isAVL and isRedBlack methods' results.

2) Create a BinarySearchTree object and fill with some values which makes the subTree's height equal

See isAVL and isRedBlack methods' results.

Part 3

1) Create an instance of Binary Search tree, Red-Black tree, B-Tree, 2-3 tree, SkipList

2) Generate 10000 non-repeated numbers and insert them to the trees.

3) Add 100 random numbers to every tree type and measure the time that insertion operation takes.

4) Repeat 2) with different amount of numbers. Do 3) and see how much time insertion take on different sizes.

Running Command&Results

Part 2

```
0
  null
  1
    null
    4
      3
        null
        null
      8
        5
          null
          7
            null
            null
        12
          9
            null
            11
              null
              null
          16
            13
              null
              null
            20
              null
              24
                null
                28
                  null
                  null

==============Test isAVL Method==================
false
==============Test isRedBlack Method==================
false
```

```
8
  0
    null
    1
      null
      2
        null
        3
          null
          4
            null
            5
              null
              6
                null
                7
                  null
                  null
  9
    null
    10
      null
      11
        null
        12
          null
          13
            null
            14
              null
              15
                null
                null

==============Test isAVL Method==================
true
==============Test isRedBlack Method==================
false
```

Part 3

```
=================Test 10000=================
BST average for size:10000 = PT0.0004323S
Btree average for size:10000 = PT0.0004466S
R-BTree average for size:10000 = PT0.0004412S
2-3Tree average for size:10000 = PT0.0004447S
SL average for size:10000 = PT0.0004345S
=================Test 10000=================



=================Test 20000=================
BST average for size:20000 = PT0.0000709S
Btree average for size:20000 = PT0.0000788S
R-BTree average for size:20000 = PT0.0000772S
2-3Tree average for size:20000 = PT0.0000785S
SL average for size:20000 = PT0.000071S
=================Test 20000=================



=================Test 40000=================
BST average for size:40000 = PT0.0000735S
Btree average for size:40000 = PT0.0000825S
R-BTree average for size:40000 = PT0.0000807S
2-3Tree average for size:40000 = PT0.0000819S
SL average for size:40000 = PT0.0000736S
=================Test 40000=================



=================Test 80000=================
BST average for size:80000 = PT0.0000893S
Btree average for size:80000 = PT0.0001002S
R-BTree average for size:80000 = PT0.0000983S
2-3Tree average for size:80000 = PT0.0000994S
SL average for size:80000 = PT0.0000894S
=================Test 80000=================
```


```
=================Test 10000=================
BST average for size:10000 = PT0.0004323S
Btree average for size:10000 = PT0.0004466S
R-BTree average for size:10000 = PT0.0004412S
2-3Tree average for size:10000 = PT0.0004447S
SL average for size:10000 = PT0.0004345S
=================Test 10000=================



=================Test 20000=================
BST average for size:20000 = PT0.0000709S
Btree average for size:20000 = PT0.0000788S
R-BTree average for size:20000 = PT0.0000772S
2-3Tree average for size:20000 = PT0.0000785S
SL average for size:20000 = PT0.000071S
=================Test 20000=================



=================Test 40000=================
BST average for size:40000 = PT0.0000735S
Btree average for size:40000 = PT0.0000825S
R-BTree average for size:40000 = PT0.0000807S
2-3Tree average for size:40000 = PT0.0000819S
SL average for size:40000 = PT0.0000736S
=================Test 40000=================



=================Test 80000=================
BST average for size:80000 = PT0.0000893S
Btree average for size:80000 = PT0.0001002S
R-BTree average for size:80000 = PT0.0000983S
2-3Tree average for size:80000 = PT0.0000994S
SL average for size:80000 = PT0.0000894S
=================Test 80000=================
```