Gebze Technical University
Computer Engineering

CSE 222/505 – Spring 2021

Homework 7 Report

Baran Solmaz
1801042601

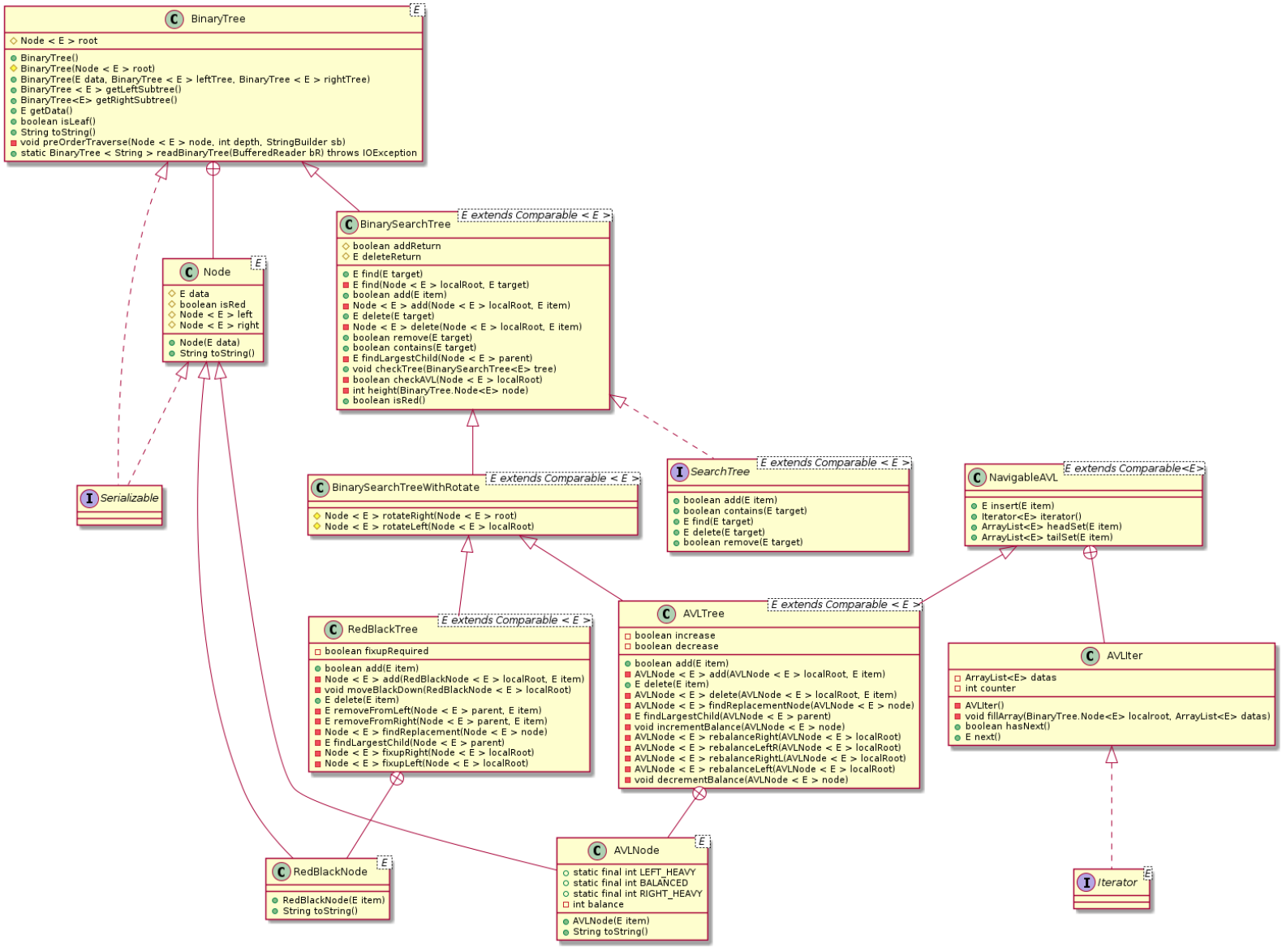
1) Detailed System Requirements

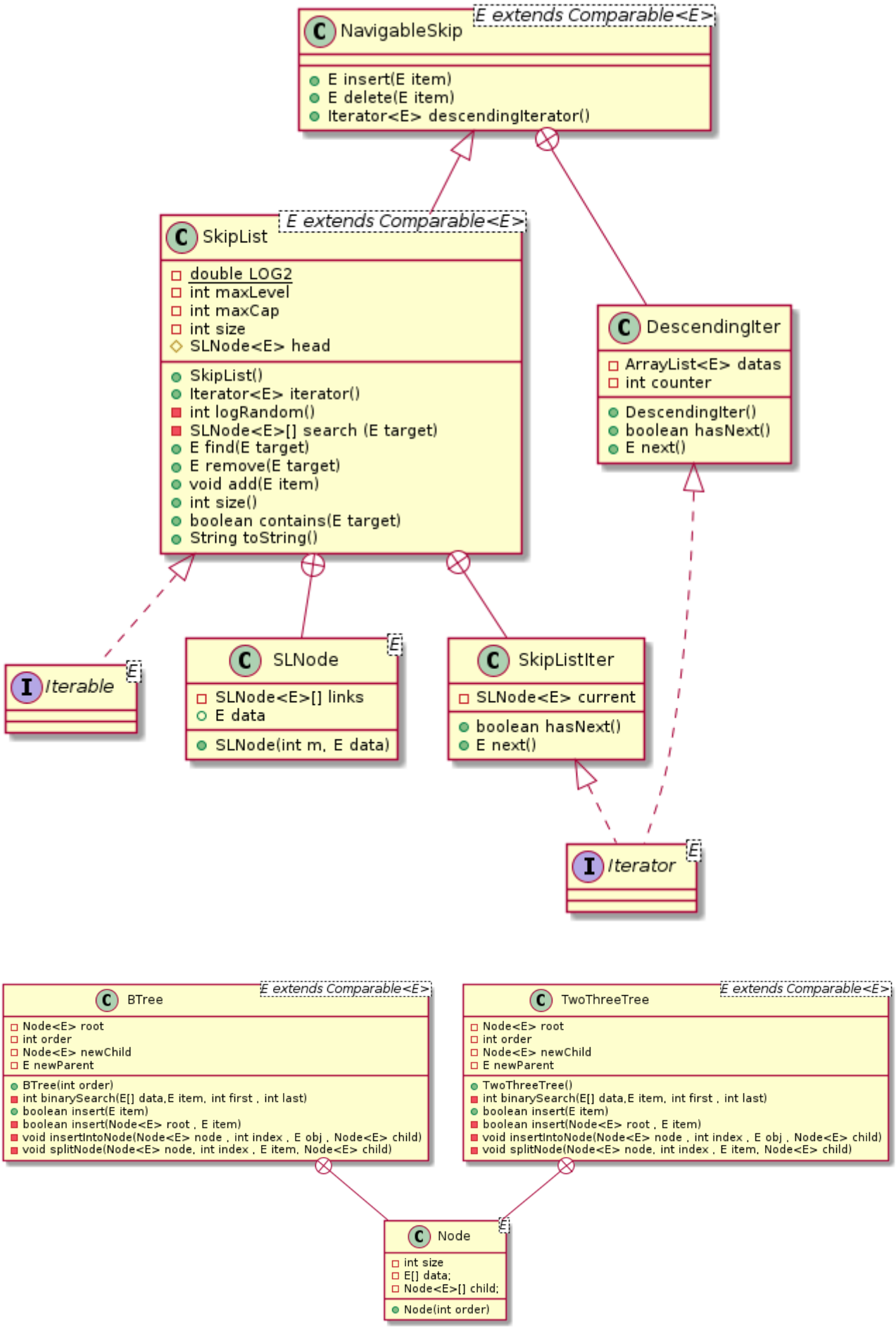
Part 1: You have to generate unique random numbers to insert element into NavigableSkip class that extends SkipList class and NavigableAVL class that extends AVL class .

Part 2: You need at least one AVL Tree , one Red Black Tree and one Binary Search Tree to try method. These trees must be filled.

Part 3: You must have book implementations of wanted data structures and generate unique random number at least 80,100 numbers .

2) Class Diagrams




NavigableSkip
E extends Comparable<E>
E insert(E item)
E delete(E item)
Iterator<E> descendingIterator()
SkipList
E extends Comparable<E>
double LOG2
int maxLevel
int maxCap
int size
SLNode<E> head
SkipList()
Iterator<E> iterator()
int logRandom()
SLNode<E>[] search (E target)
E find(E target)
E remove(E target)
void add(E item)
int size()
boolean contains(E target)
String toString()
DescendingIter
ArrayList<E> datas
int counter
DescendingIter()
boolean hasNext()
E next()
Iterable
E
SLNode
E
SLNode<E>[] links
E data
SLNode(int m, E data)
SkipListIter
SLNode<E> current
boolean hasNext()
E next()
Iterator
E
BTree
E extends Comparable<E>
Node<E> root
int order
Node<E> newChild
E newParent
BTree(int order)
int binarySearch(E[] data,E item, int first , int last)
boolean insert(E item)
boolean insert(Node<E> root , E item)
void insertIntoNode(Node<E> node , int index , E obj , Node<E> child)
void splitNode(Node<E> node, int index , E item, Node<E> child)
TwoThreeTree
E extends Comparable<E>
Node<E> root
int order
Node<E> newChild
E newParent
TwoThreeTree()
int binarySearch(E[] data,E item, int first , int last)
boolean insert(E item)
boolean insert(Node<E> root , E item)
void insertIntoNode(Node<E> node , int index , E obj , Node<E> child)
void splitNode(Node<E> node, int index , E item, Node<E> child)
Node
E
int size
E[] data;
Node<E>[] child;
Node(int order)

3) Problem Solutions Approach

Part 1: I extended AVL Tree and Skip List to implement wanted methods of Navigable Set Interface. For descending iterator, I created new inner class in NavigableSkip class and added all elements of SkipList into the arraylist to hold elements. For insert and delete, I used methods of extended classes. For iterator of NavigableAVL , I created new inner class in NavigableAVL class and added all elements of AVL Tree into the arraylist to hold elements sequentially.

Part 2: I checked height of subtrees for AVL Tree, and roots color for Red Black Tree.

Part 3: I generated 80,100 unique random numbers and added these numbers to structures.

4) Test Cases

```
private static void generateRandoms(ArrayList<Integer> randomNumbers,int value) {
    Random rand=new Random();
    while(randomNumbers.size()<value){
        int temp=rand.nextInt(1000000);
        if (!randomNumbers.contains(temp))
            randomNumbers.add(temp);
    }
}
```

Part 1:

```
System.out.println("----- Generating Unique Random Numbers -----");
ArrayList<Integer> randomNumbers= new ArrayList<Integer>();
generateRandoms(randomNumbers,20);
System.out.println("\n----- NavigableSet Using Skip-List -----");
NavigableSkip<Integer> skip=new NavigableSkip<Integer>();
System.out.println("\n-- Insertion --");
for (int i = 0; i < 10; i++) {
    System.out.print(".insert("+randomNumbers.get(i)+")\t"); skip.insert(randomNumbers.get(i));
    System.out.println(".insert("+randomNumbers.get(10+i)+")"); skip.insert(randomNumbers.get(i+10));
}
System.out.println(skip);
```

```
System.out.println("\n-- Deletion --");
for (int i = 0; i < 4; i++) {
    System.out.print(".delete("+randomNumbers.get(i)+")\t"); skip.delete(randomNumbers.get(i));
    System.out.println(".delete("+randomNumbers.get(10+i)+")"); skip.delete(randomNumbers.get(i+10));
}
System.out.println(skip);
```

```
System.out.println("\n-- Descending Iterator --");
Iterator<Integer> iter=skip.descendingIterator();
while (iter.hasNext()) {
    System.out.print(iter.next()+" ");
}
System.out.println();
```

```
System.out.println("\n----- NavigableSet Using AVL Tree -----");
NavigableAVL<Integer> avl=new NavigableAVL<Integer>();
for (int i = 0; i < 10; i++) {
    System.out.print(".insert("+randomNumbers.get(i)+")\t"); avl.insert(randomNumbers.get(i));
    System.out.println(".insert("+randomNumbers.get(10+i)+")"); avl.insert(randomNumbers.get(i+10));
}
//System.out.println(avl);
```

```
System.out.println("\n-- Iterator --");
Iterator<Integer> iter2=avl.iterator();
while (iter2.hasNext()) {
    System.out.print(iter2.next()+" ");
}
```

```
System.out.println("\n-- Head Set - Tail Set --");
System.out.println(".headSet("+randomNumbers.get(5)+")-> "+avl.headSet(randomNumbers.get(5)));
System.out.println(".tailSet("+randomNumbers.get(5)+")-> "+avl.tailSet(randomNumbers.get(5))+"\n");

System.out.println(".headSet("+randomNumbers.get(15)+")-> "+avl.headSet(randomNumbers.get(15)));
System.out.println(".tailSet("+randomNumbers.get(15)+")-> "+avl.tailSet(randomNumbers.get(15)));
```

Part 2:

```
RedBlackTree<Integer> redBlack= new RedBlackTree<Integer>();
AVLTree<Integer> avl= new AVLTree<Integer>();
RedBlackTree<Integer> both= new RedBlackTree<Integer>();
BinarySearchTree<Integer> bst= new BinarySearchTree<Integer>();

System.out.println("Only Red-Black Tree With {3,2,15,8,4,9}");
redBlack.add(3);
redBlack.add(2);
redBlack.add(15);
redBlack.add(8);
redBlack.add(4);
redBlack.add(9);
System.out.println(redBlack);
bst.checkTree(redBlack);
```

```
System.out.println("\nOnly AVL Tree With {12,8,18,5,11,17,4}");
avl.add(12);
avl.add(8);
avl.add(18);
avl.add(5);
avl.add(11);
avl.add(17);
avl.add(4);
System.out.println(avl);
bst.checkTree(avl);
```

```
System.out.println("\nNeither Tree With Degenerate Tree {8,9,7,6,5,4,3}");
bst.add(8);
bst.add(9);
bst.add(7);
bst.add(6);
bst.add(5);
bst.add(4);
bst.add(3);
System.out.println(bst);
bst.checkTree(bst);
```

```
System.out.println("\nBoth Tree With Perfect Tree {10,5,15,3,6,12,17}");
both.add(10);
both.add(5);
both.add(15);
both.add(3);
both.add(6);
both.add(12);
both.add(17);
System.out.println(both);
bst.checkTree(both);
```

Part 3:

```
ArrayList<Integer> randomNumbers= new ArrayList<Integer>();
generateRandoms(randomNumbers,80100);
long timeSmall,timeSmall2;
float average=0;
```

```
System.out.println("\nAdding 100 Elements On The 10,000 Elements \n");

for (int j = 0; j < 10; j++) {
    BinarySearchTree<Integer> bst1 =new BinarySearchTree<Integer>();

    for (int i = 0; i < 10000; i++)
        bst1.add(randomNumbers.get(i));

    timeSmall= System.nanoTime();
    for (int i = 80000; i < 80100; i++)
        bst1.add(randomNumbers.get(i));
    timeSmall2= System.nanoTime();

    average+=(timeSmall2-timeSmall) / 1000000F;
}
System.out.println("Binary Search Tree 10K+ Time : "+average / 10+" ms.");
average=0;
```

```java
for (int j = 0; j < 10; j++) {
    RedBlackTree<Integer> redBlack1 =new RedBlackTree<Integer>();

    for (int i = 0; i < 10000; i++)
        redBlack1.add(randomNumbers.get(i));

    timeSmall= System.nanoTime();
    for (int i = 80000; i < 80100; i++)
        redBlack1.add(randomNumbers.get(i));
    timeSmall2= System.nanoTime();

    average+=(timeSmall2-timeSmall) / 1000000F;
}
System.out.println("Red Black Tree 10K+ \tTime : "+average / 10+" ms.");
average=0;
```

```java
 for (int j = 0; j < 10; j++) {
     TwoThreeTree<Integer> twoThree1 =new TwoThreeTree<Integer>();

     for (int i = 0; i < 10000; i++)
         twoThree1.insert(randomNumbers.get(i));

     timeSmall= System.nanoTime();
     for (int i = 80000; i < 80100; i++)
         twoThree1.insert(randomNumbers.get(i));
     timeSmall2= System.nanoTime();

     average+=(timeSmall2-timeSmall) / 1000000F;
 }
 System.out.println("2-3 Tree 10K+ \t\tTime : "+average / 10+" ms.");
 average=0;
```

```java
for (int j = 0; j < 10; j++) {
    BTree<Integer> bTree1 =new BTree<Integer>(6);

    for (int i = 0; i < 10000; i++)
        bTree1.insert(randomNumbers.get(i));

    timeSmall= System.nanoTime();
    for (int i = 80000; i < 80100; i++)
        bTree1.insert(randomNumbers.get(i));
    timeSmall2= System.nanoTime();

    average+=(timeSmall2-timeSmall) / 1000000F;
}
System.out.println("B Tree 10K+ \t\tTime : "+average / 10+" ms.");
average=0;
```

```java
for (int j = 0; j < 10; j++) {
    SkipList<Integer> skip1 =new SkipList<Integer>();

    for (int i = 0; i < 10000; i++)
        skip1.add(randomNumbers.get(i));

    timeSmall= System.nanoTime();
    for (int i = 80000; i < 80100; i++)
        skip1.add(randomNumbers.get(i));
    timeSmall2= System.nanoTime();

    average+=(timeSmall2-timeSmall) / 1000000F;
}
System.out.println("SkipList 10K+ \t\tTime : "+average / 10+" ms.");
average=0;
```

***These are just 10K sized structures, process is same in other sizes**

5) Running Command and Results

Part 1:

```
----- Generating Unique Random Numbers -----

----- NavigableSet Using Skip-List -----

-- Insertion --
.insert(419852) .insert(389800)
.insert(617908) .insert(17332)
.insert(18349)  .insert(936015)
.insert(847324) .insert(277888)
.insert(384489) .insert(936787)
.insert(406423) .insert(62532)
.insert(25083)  .insert(570171)
.insert(484570) .insert(283167)
.insert(387658) .insert(62770)
.insert(371722) .insert(68468)
[17332, 18349, 25083, 62532, 62770, 68468, 277888, 283167, 371722, 384489, 387658, 389800, 40642
3, 419852, 484570, 570171, 617908, 847324, 936015, 936787]
```

```
-- Deletion --
.delete(419852) .delete(389800)
.delete(617908) .delete(17332)
.delete(18349)  .delete(936015)
.delete(847324) .delete(277888)
[25083, 62532, 62770, 68468, 283167, 371722, 384489, 387658, 406423, 484570, 570171, 936787]

-- Descending Iterator --
936787 570171 484570 406423 387658 384489 371722 283167 68468 62770 62532 25083
```

```
----- NavigableSet Using AVL Tree -----
.insert(419852) .insert(389800)
.insert(617908) .insert(17332)
.insert(18349)  .insert(936015)
.insert(847324) .insert(277888)
.insert(384489) .insert(936787)
.insert(406423) .insert(62532)
.insert(25083)  .insert(570171)
.insert(484570) .insert(283167)
.insert(387658) .insert(62770)
.insert(371722) .insert(68468)
```

```
-- Iterator --
17332 18349 25083 62532 62770 68468 277888 283167 371722 384489 387658 389800 406423 419852 4845
70 570171 617908 847324 936015 936787
-- Head Set - Tail Set --
.headSet(406423)-> [17332, 18349, 25083, 62532, 62770, 68468, 277888, 283167, 371722, 384489, 38
7658, 389800]
.tailSet(406423)-> [406423, 419852, 484570, 570171, 617908, 847324, 936015, 936787]

.headSet(62532)-> [17332, 18349, 25083]
.tailSet(62532)-> [62532, 62770, 68468, 277888, 283167, 371722, 384489, 387658, 389800, 406423,
419852, 484570, 570171, 617908, 847324, 936015, 936787]
```

Part 2:

```
Only Red-Black Tree With {3,2,15,8,4,9}
Black: 3
  Black: 2
    null
    null
  Red  : 8
    Black: 4
      null
      null
    Black: 15
      Red  : 9
        null
        null
      null

Red Black
```

```
Only AVL Tree With {12,8,18,5,11,17,4}
-1: 12
  -1: 8
    -1: 5
      0: 4
        null
        null
      null
    0: 11
      null
      null
  -1: 18
    0: 17
      null
      null
    null

AVL Tree
```

```
Neither Tree With Degenerate Tree {8,9,7,6,5,4,3}
8
  7
    6
      5
        4
          3
            null
            null
          null
        null
      null
    null
  9
    null
    null

Neither
```

```
Both Tree With Perfect Tree {10,5,15,3,6,12,17}
Black: 10
  Black: 5
    Red  : 3
      null
      null
    Red  : 6
      null
      null
  Black: 15
    Red  : 12
      null
      null
    Red  : 17
      null
      null

Both
```

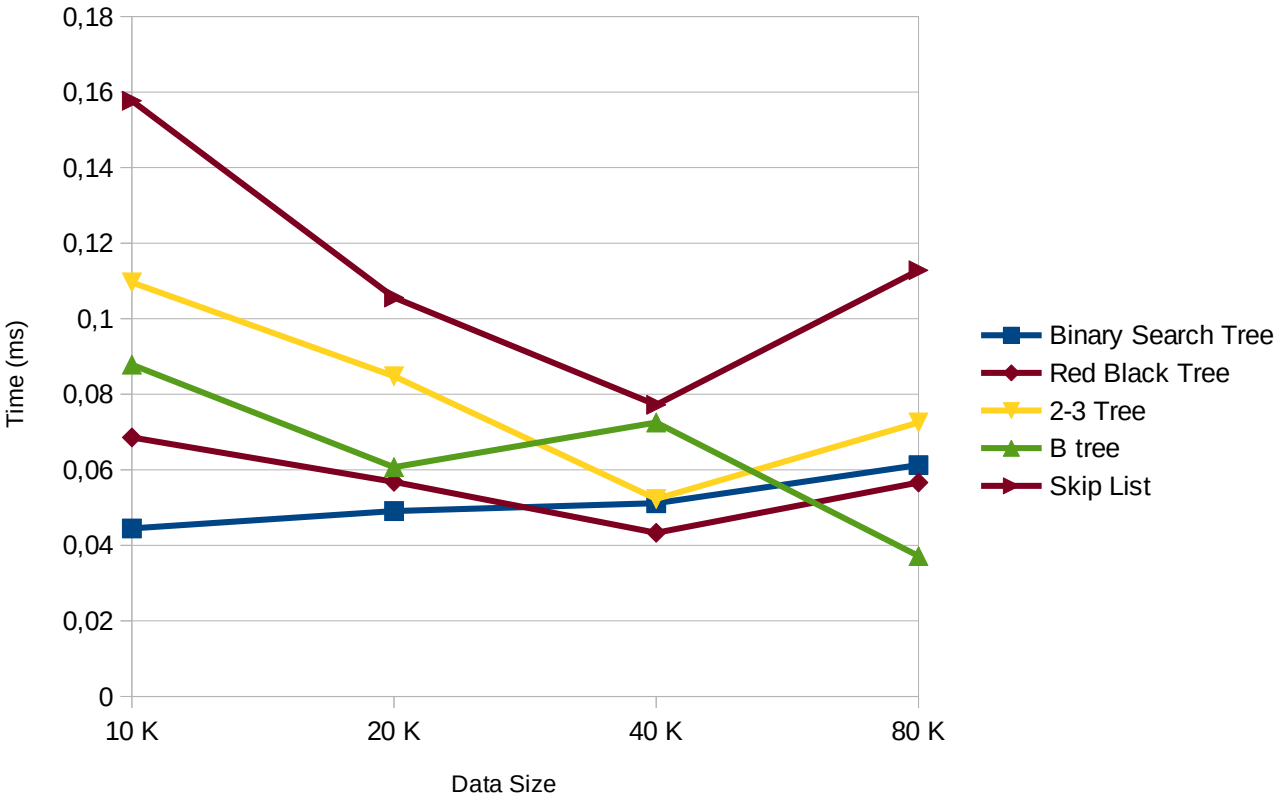
Part 3:

```
###### PART 3 #####


Adding 100 Elements On The 10,000 Elements

Binary Search Tree 10K+ Time : 0.044469796 ms.
Red Black Tree 10K+     Time : 0.068562105 ms.
2-3 Tree 10K+           Time : 0.1096069 ms.
B Tree 10K+             Time : 0.087832004 ms.
SkipList 10K+           Time : 0.15770712 ms.
```

```
Adding 100 Elements On The 20,000 Elements

Binary Search Tree 20K+ Time : 0.049070302 ms.
Red Black Tree 20K+     Time : 0.056825805 ms.
2-3 Tree 20K+           Time : 0.084764 ms.
B Tree 20K+             Time : 0.060679298 ms.
SkipList 20K+           Time : 0.1056066 ms.
```

```
Adding 100 Elements On The 40,000 Elements

Binary Search Tree 40K+ Time : 0.051155698 ms.
Red Black Tree 40K+     Time : 0.043316595 ms.
2-3 Tree 40K+           Time : 0.0522693 ms.
B Tree 40K+             Time : 0.0325935 ms.
SkipList 40K+           Time : 0.0771934 ms.
```

```
Adding 100 Elements On The 80,000 Elements

Binary Search Tree 80K+ Time : 0.061251003 ms.
Red Black Tree 80K+     Time : 0.0566305 ms.
2-3 Tree 80K+           Time : 0.07252079 ms.
B Tree 80K+             Time : 0.037169397 ms.
SkipList 80K+           Time : 0.112829104 ms.
```


0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
Time (ms)
10 K
20 K
40 K
80 K
Data Size
Binary Search Tree
Red Black Tree
2-3 Tree
B tree
Skip List