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

Erdi Bayır
171044063

PROBLEM SOLUTION APPROACH

For Part 1:

For NavigableSetWithSkipList:

In this part, we have to implement a NavigableSet with Skip List and some methods.
I use Skip List with inner class where I implement NavigableSet interface.
Also I create iterator class in SkipList and implement descending iterator in this class.
Then called insert delete and descending iterator method from NavigableSet.

For NavigableSetWithAVL:

In this part, we have to implement a NavigableSet with AVL Tree and some methods.
I use AVL Tree with extend class where I implement NavigableSet interface.
Also I create iterator class in AVL and implement iterator in this class.
Then called insert headSet tailSet and iterator method from Navigable Set.

For Part 2:

In the second part, we need to write method that takes a BinarySearchTree and checks whether the tree is AVL or Red Black Tree .I use AVL and Red Black Tree from book.
For check is AVL or not : I found and compared the heights of left and right children for all nodes in the AVL. According to the AVL rule, it can be examined (right or left) and there can be at most 1 node.
For check is Red Black or not : I found the maximum height, left height and right height for each node. I needed to keep a reference and for this I used Atomic Integer so as not to lose the values. Then I compared the values I found according to the red black rule and returned the result.

For Part 3:

I wrote a separate test method for all the desired classes in the book and got the results. Then I found the average running times with these results and turned it into a table in the terminal. Using this data, I drew the necessary graphs and compared them.

Note: Teacher,I realized in the last second while was sending the homework, I am stating it here because I do not have a correction time written on the terminal.
“10000 extra insertion takes average time 48319.0 ns.”
“20000 extra insertion takes average time 48319.0 ns.”
“40000 extra insertion takes average time 48319.0 ns.”
“80000 extra insertion takes average time 48319.0 ns.”
There is a typo error here. Instead of these numbers, all of them should have "100 extra insertions". As in the table.

SYSTEM REQUIREMENTS

Functional Requirements

- **For Navigable Set With Skip List;**

  Must add element.

  Must delete element.

  Must use descending iterator.

- **For Navigable Set With AVL Tree;**

  Must add element.

  Must use iterator .

  Must find head set.

  Must find tail set.

- **For Check Binary Tree AVL or Red Black Tree;**

  Must determine Binary Tree is AVL or not.

  Must determine Binary Tree is Red Black or not.

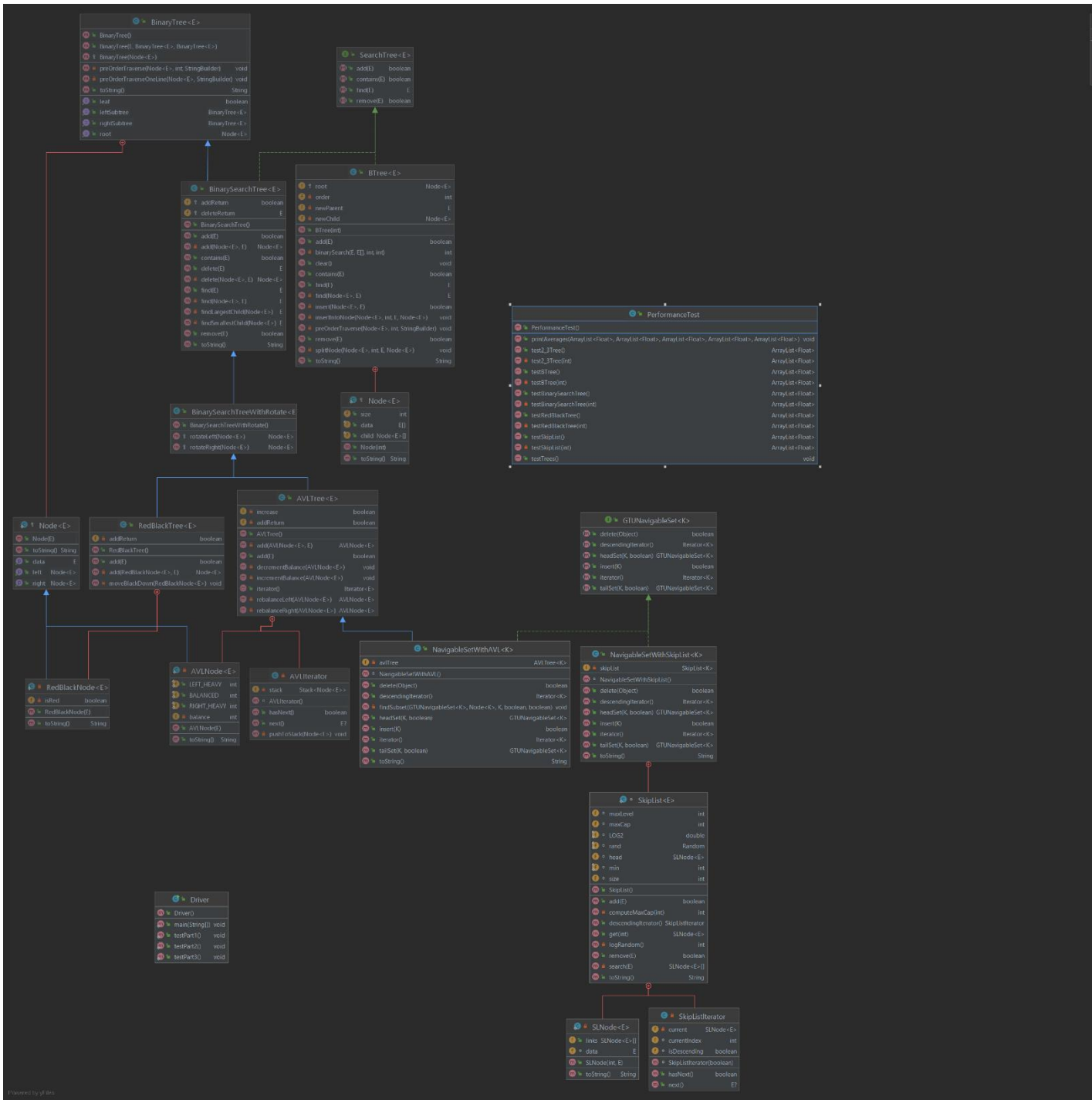
  Must determine Binary Tree is not AVL or Red Black Tree.

- **For Other Books Implementations;**

  Must determine average running time results.

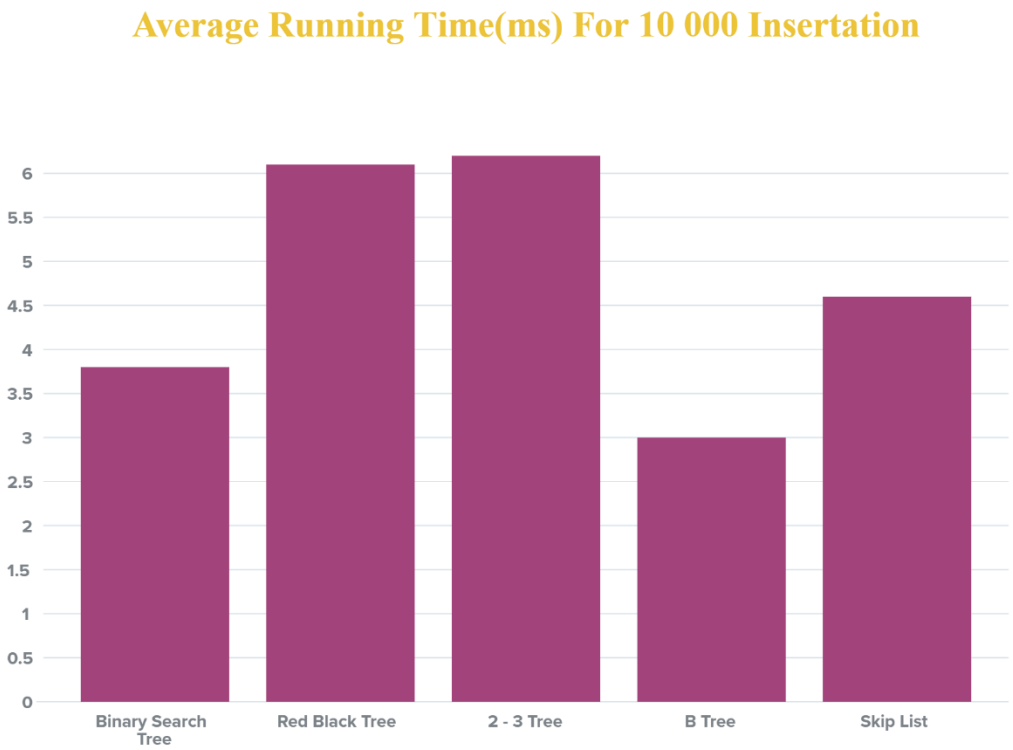
  Must determine extra insertion average running time results.

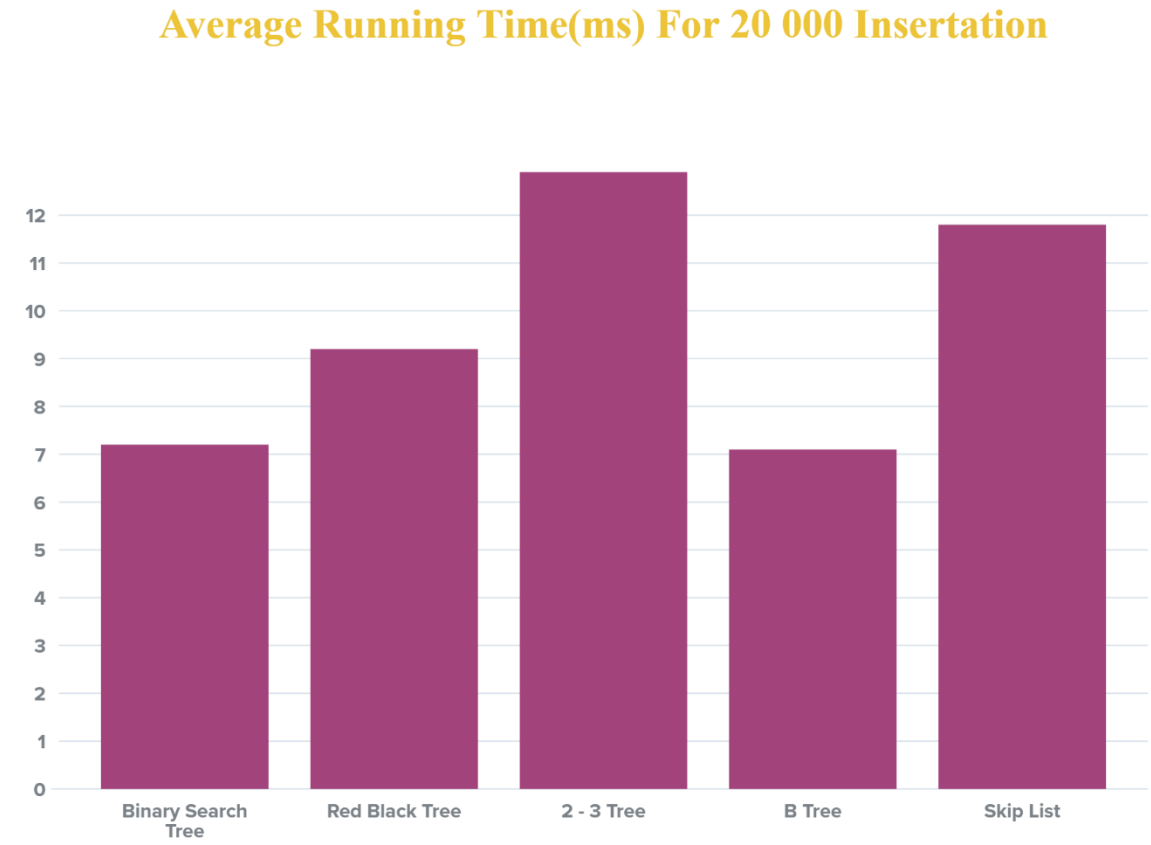
CLASS DIAGRAM



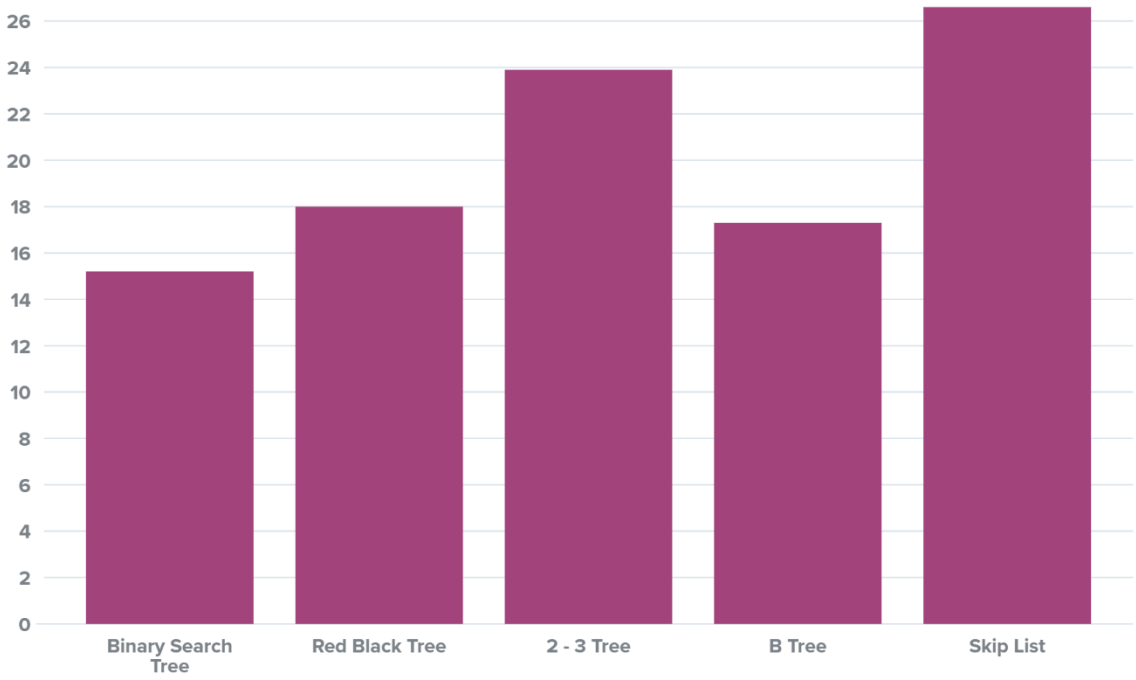
Performance ANALYSIS

Bar Plot Graphs

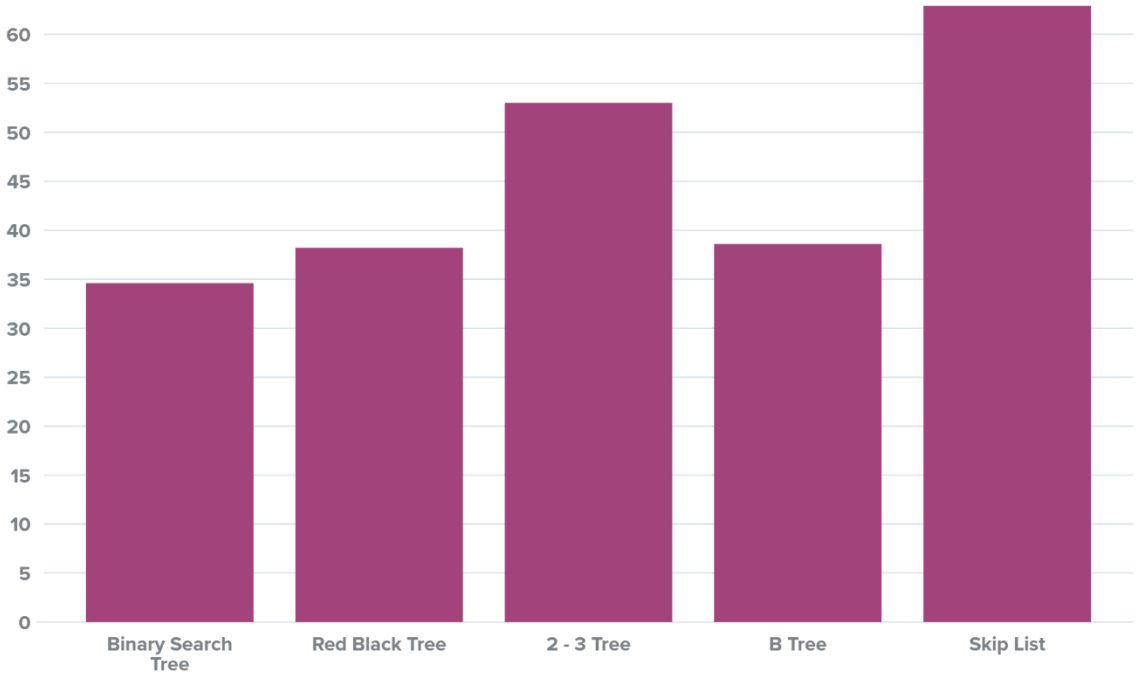




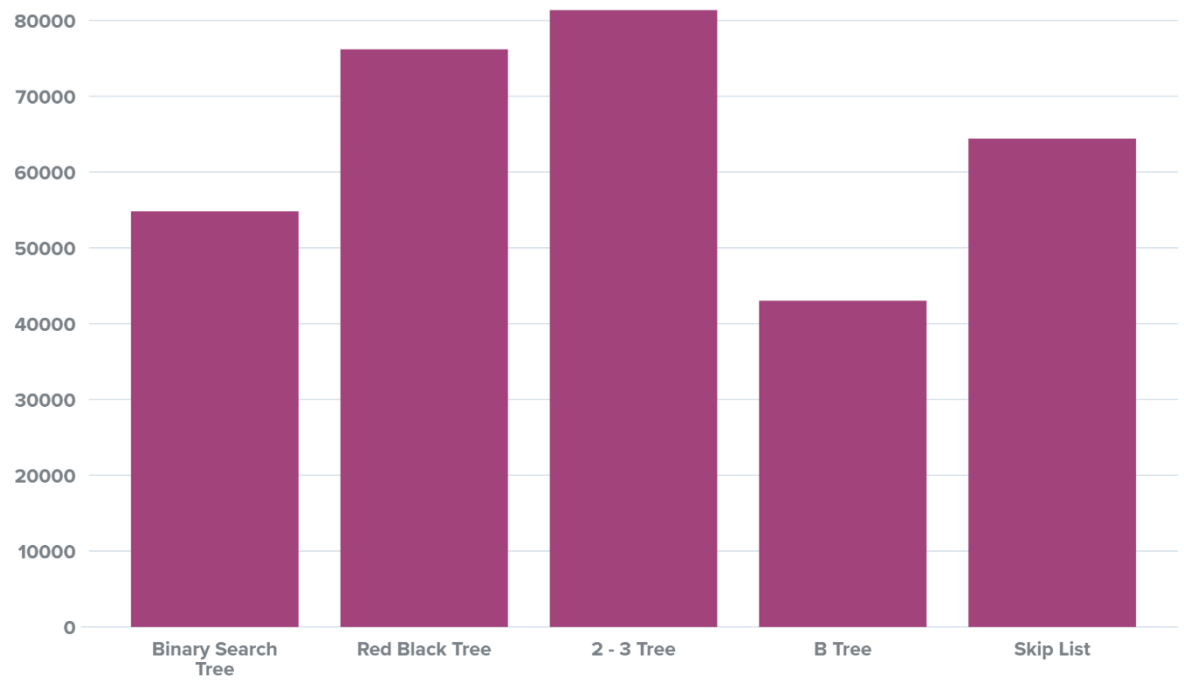
Average Running Time(ms) For 40 000 Insertation



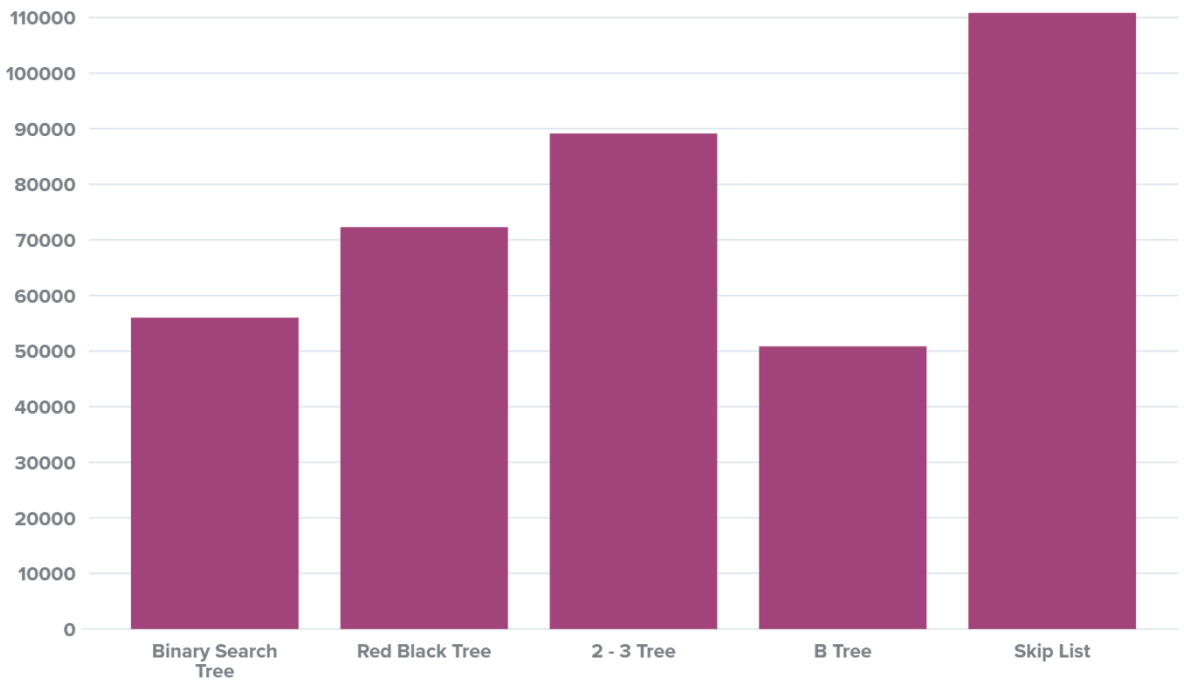
Average Running Time(ms) For 80 000 Insertation



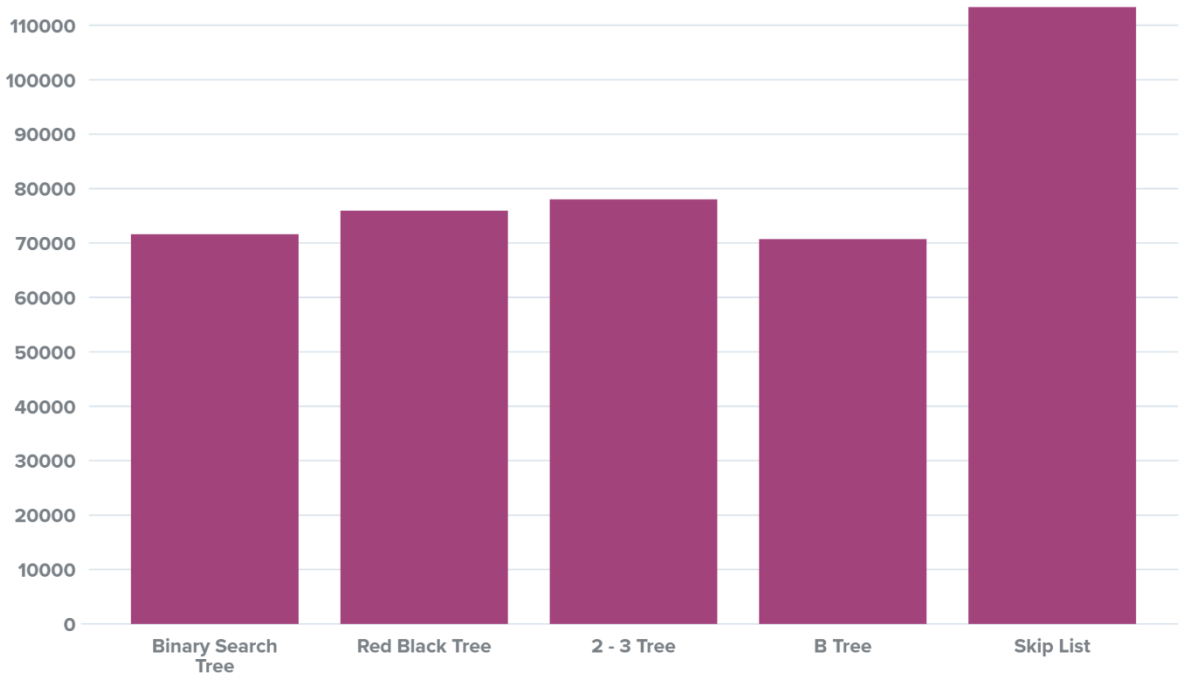
Extra 100 Insertion Average Running Time(ns) For 10 000 Insertation



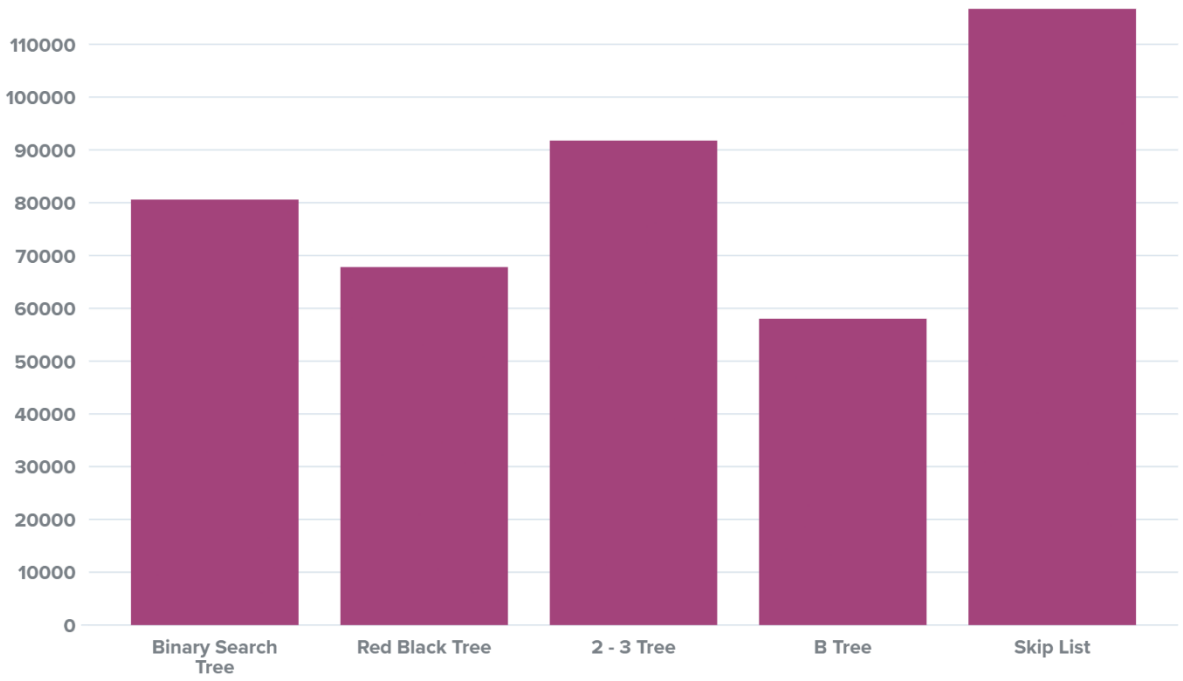
Extra 100 Insertion Average Running Time(ns) For 20 000 Insertation



Extra 100 Insertion Average Running Time(ns) For 40 000 Insertation



Extra 100 Insertion Average Running Time(ns) For 80 000 Insertation



Running Times And Increase Rate Analysis

While adding values such as 10 000 20 000 etc. to these structures, Binary Search Tree and B Tree give the most efficient performance. However between these two structure, when problem size is small like 10 000 add operation , B tree performs better than binary search tree, As the problem size gets bigger such as 40 000, Binary search tree starts to perform better. kip List is the structure that gives the worst performance regardless of the problem. The reason for this is that many nodes are connected to each other and the memory it uses is lower than the others. In the case of extra insertion, we can say that the data is similar to each other. However, while the 2-3 Tree problem gives you worse performance than the skip list when the problem is small, it performs better than the skip list because the increase rate of the skip list is larger as the problem grows.

TEST CASES And Running Command Results

For Part 1:

a)Navigable Set With Skip List

1) Create Navigable Set With Skip List
2) Insert new element
3) Delete element
4) Create map descending iterator
5) Testing next() method
6) Testing hasNext() method

a)Navigable Set With AVL Tree

1) Create Navigable Set With AVL Tree
2) Insert new element
3) Create map iterator
4) Testing next() method
5) Testing hasNext() method
6) Test headSet() method
7) Test tailSet() method

For Part 2:

1) Test BST is AVL or not
2) Test BST is Red Black Tree or not

For Part 3:

1) 10 000,20 000 , 40 000, 80 000 insertion test for all structures
2) Extra 100 insertion test for all structures
3) Find average running time for insertions
4) Find extra 100 insertion average running time for insertions

Running Command Results

```
--------- Test Part 1 ---------
------------ NavigableSet With Skip List Test ------------


Create object from NavigableSetWithSkipList...
Insert Some Elements to Navigable Set...
NavigableSetSkip{
Data = 1
Level Of Data: 1

Data = 2
Level Of Data: 2

Data = 3
Level Of Data: 2

Data = 5
Level Of Data: 1

Data = 6
Level Of Data: 1

Data = 12
Level Of Data: 1

Data = 34
Level Of Data: 2

Data = 45
Level Of Data: 2
```

```
Data = 51
Level Of Data: 1

Data = 112
Level Of Data: 1

}
----Descending Iterator Test----
Has next = true
Next:51
Has next = true
Next:45
Has next = true
Next:34
Has next = true
Next:12
Has next = true
Next:6
Has next = true
Next:5
Has next = true
Next:3
Has next = true
Next:2
Has next = true
Next:1
```

```
Delete Method Test With Delete Some elements
---Delete 12---
NavigableSetSkip{
Data = 1
Level Of Data: 1

Data = 2
Level Of Data: 2

Data = 3
Level Of Data: 2

Data = 5
Level Of Data: 1

Data = 6
Level Of Data: 1

Data = 34
Level Of Data: 2

Data = 45
Level Of Data: 2

Data = 51
Level Of Data: 1

Data = 112
Level Of Data: 1

}
```

```
---Delete 1---
NavigableSetSkip{
Data = 2
Level Of Data: 2

Data = 3
Level Of Data: 2

Data = 5
Level Of Data: 1

Data = 6
Level Of Data: 1

Data = 34
Level Of Data: 2

Data = 45
Level Of Data: 2

Data = 51
Level Of Data: 1

Data = 112
Level Of Data: 1

}
```

```
---Delete 112---
NavigableSetSkip{
Data = 2
Level Of Data: 2

Data = 3
Level Of Data: 2

Data = 5
Level Of Data: 1

Data = 6
Level Of Data: 1

Data = 34
Level Of Data: 2

Data = 45
Level Of Data: 2

Data = 51
Level Of Data: 1

}
```

```
------------ NavigableSet With AVL Tree Test ------------


Create object from NavigableSetWithAVL...
Insert Some Elements to Navigable Set...
NavigableSetWithAVL{
-1: 13
  1: 5
    0: 1
      -
      -
    -1: 12
      0: 7
        -
        -
      -
  0: 19
    0: 16
      -
      -
    0: 27
      -
      -
}
```

```
----Iterator Test----
Has next = true
Next:1
Has next = true
Next:5
Has next = true
Next:7
Has next = true
Next:12
Has next = true
Next:13
Has next = true
Next:16
Has next = true
Next:19
Has next = true
Next:27
```

```
----HeadSet and TailSet Test----
Head set to Element : 19 and inclusive : true -> HeadSet is:
NavigableSetWithAVL{
0: 12
  0: 5
    0: 1
      -
      -
    0: 7
      -
      -
  0: 16
    0: 13
      -
      -
    0: 19
      -
      -
}
Head set to Element : 16 and inclusive : false -> HeadSet is:
NavigableSetWithAVL{
1: 5
  0: 1
    -
    -
  0: 12
    0: 7
      -
      -
    0: 13
      -
      -
}
```

```
Head set to Element : 8 and inclusive : true -> HeadSet is:
NavigableSetWithAVL{
0: 5
  0: 1
    -
    -
  0: 7
    -
    -
}
Tail set from Element : 12 and inclusive : true -> TailSet is:
NavigableSetWithAVL{
1: 13
  0: 12
    -
    -
  0: 19
    0: 16
      -
      -
    0: 27
      -
      -
}
```

```
Tail set from Element : 5 and inclusive : false -> TailSet is:
NavigableSetWithAVL{
0: 16
  0: 12
    0: 7
      -
      -
    0: 13
      -
      -
  1: 19
    -
    0: 27
      -
      -
}
Tail set from Element : 25 and inclusive : true -> TailSet is:
NavigableSetWithAVL{
0: 27
  -
  -
}
--------- Test Part 2 ---------
--------- Create Binary Search Trees And Check is AVL or Red Black Tree ---------
7
  3
    -
    -
  18
    10
      8
        -
        -
      11
        -
        -
    22
      -
      26
        -
        -
```

```
It is a Red Black Tree!
12
  8
    5
      4
        2
          -
          -
        -
      -
    9
      -
      -
  18
    -
    -
```

```
It is a Red Black and AVL Tree!
40
  35
    27
      13
        -
        -
      -
    38
      -
      -
  42
    -
    -

It is a Red Black Tree!
```

```
--------- Test Part 3 ---------
1.BinarySearchTree 10000 insertion takes 9 ms.
1.BinarySearchTree extra 100 insertion takes 106485 ns.
2.BinarySearchTree 10000 insertion takes 4 ms.
2.BinarySearchTree extra 100 insertion takes 47772 ns.
3.BinarySearchTree 10000 insertion takes 3 ms.
3.BinarySearchTree extra 100 insertion takes 46678 ns.
4.BinarySearchTree 10000 insertion takes 3 ms.
4.BinarySearchTree extra 100 insertion takes 52877 ns.
5.BinarySearchTree 10000 insertion takes 3 ms.
5.BinarySearchTree extra 100 insertion takes 47407 ns.
6.BinarySearchTree 10000 insertion takes 3 ms.
6.BinarySearchTree extra 100 insertion takes 47042 ns.
7.BinarySearchTree 10000 insertion takes 4 ms.
7.BinarySearchTree extra 100 insertion takes 47407 ns.
8.BinarySearchTree 10000 insertion takes 3 ms.
8.BinarySearchTree extra 100 insertion takes 66735 ns.
9.BinarySearchTree 10000 insertion takes 3 ms.
9.BinarySearchTree extra 100 insertion takes 44855 ns.
10.BinarySearchTree 10000 insertion takes 3 ms.
10.BinarySearchTree extra 100 insertion takes 40843 ns.
10000 insertion takes average time 3.8 ms.
10000 extra insertion takes average time 54810.1 ns.
```

```
1.BinarySearchTree 20000 insertion takes 6 ms.
1.BinarySearchTree extra 100 insertion takes 53607 ns.
2.BinarySearchTree 20000 insertion takes 4 ms.
2.BinarySearchTree extra 100 insertion takes 47408 ns.
3.BinarySearchTree 20000 insertion takes 6 ms.
3.BinarySearchTree extra 100 insertion takes 35738 ns.
4.BinarySearchTree 20000 insertion takes 6 ms.
4.BinarySearchTree extra 100 insertion takes 47407 ns.
5.BinarySearchTree 20000 insertion takes 7 ms.
5.BinarySearchTree extra 100 insertion takes 57983 ns.
6.BinarySearchTree 20000 insertion takes 7 ms.
6.BinarySearchTree extra 100 insertion takes 55795 ns.
7.BinarySearchTree 20000 insertion takes 6 ms.
7.BinarySearchTree extra 100 insertion takes 43761 ns.
8.BinarySearchTree 20000 insertion takes 8 ms.
8.BinarySearchTree extra 100 insertion takes 110860 ns.
9.BinarySearchTree 20000 insertion takes 13 ms.
9.BinarySearchTree extra 100 insertion takes 56159 ns.
10.BinarySearchTree 20000 insertion takes 9 ms.
10.BinarySearchTree extra 100 insertion takes 51419 ns.
20000 insertion takes average time 7.2 ms.
20000 extra insertion takes average time 56013.7 ns.
```

```
1.BinarySearchTree 40000 insertion takes 11 ms.
1.BinarySearchTree extra 100 insertion takes 48137 ns.
2.BinarySearchTree 40000 insertion takes 10 ms.
2.BinarySearchTree extra 100 insertion takes 67099 ns.
3.BinarySearchTree 40000 insertion takes 13 ms.
3.BinarySearchTree extra 100 insertion takes 191453 ns.
4.BinarySearchTree 40000 insertion takes 22 ms.
4.BinarySearchTree extra 100 insertion takes 61994 ns.
5.BinarySearchTree 40000 insertion takes 20 ms.
5.BinarySearchTree extra 100 insertion takes 56889 ns.
6.BinarySearchTree 40000 insertion takes 19 ms.
6.BinarySearchTree extra 100 insertion takes 70382 ns.
7.BinarySearchTree 40000 insertion takes 17 ms.
7.BinarySearchTree extra 100 insertion takes 86063 ns.
8.BinarySearchTree 40000 insertion takes 15 ms.
8.BinarySearchTree extra 100 insertion takes 51054 ns.
9.BinarySearchTree 40000 insertion takes 13 ms.
9.BinarySearchTree extra 100 insertion takes 44126 ns.
10.BinarySearchTree 40000 insertion takes 12 ms.
10.BinarySearchTree extra 100 insertion takes 39020 ns.
40000 insertion takes average time 15.2 ms.
40000 extra insertion takes average time 71621.7 ns.
```

```
1.BinarySearchTree 80000 insertion takes 29 ms.
1.BinarySearchTree extra 100 insertion takes 76217 ns.
2.BinarySearchTree 80000 insertion takes 31 ms.
2.BinarySearchTree extra 100 insertion takes 97367 ns.
3.BinarySearchTree 80000 insertion takes 38 ms.
3.BinarySearchTree extra 100 insertion takes 137846 ns.
4.BinarySearchTree 80000 insertion takes 38 ms.
4.BinarySearchTree extra 100 insertion takes 66006 ns.
5.BinarySearchTree 80000 insertion takes 50 ms.
5.BinarySearchTree extra 100 insertion takes 104661 ns.
6.BinarySearchTree 80000 insertion takes 33 ms.
6.BinarySearchTree extra 100 insertion takes 58712 ns.
7.BinarySearchTree 80000 insertion takes 26 ms.
7.BinarySearchTree extra 100 insertion takes 51054 ns.
8.BinarySearchTree 80000 insertion takes 29 ms.
8.BinarySearchTree extra 100 insertion takes 70017 ns.
9.BinarySearchTree 80000 insertion takes 30 ms.
9.BinarySearchTree extra 100 insertion takes 68194 ns.
10.BinarySearchTree 80000 insertion takes 42 ms.
10.BinarySearchTree extra 100 insertion takes 75852 ns.
80000 insertion takes average time 34.6 ms.
80000 extra insertion takes average time 80592.6 ns.
```

```
1.RebBlackTree 10000 insertion takes 16 ms.
1.RedBlack extra 100 insertion takes 151703 ns.
2.RebBlackTree 10000 insertion takes 11 ms.
2.RedBlack extra 100 insertion takes 117060 ns.
3.RebBlackTree 10000 insertion takes 7 ms.
3.RedBlack extra 100 insertion takes 87157 ns.
4.RebBlackTree 10000 insertion takes 8 ms.
4.RedBlack extra 100 insertion takes 112319 ns.
5.RebBlackTree 10000 insertion takes 3 ms.
5.RedBlack extra 100 insertion takes 48137 ns.
6.RebBlackTree 10000 insertion takes 5 ms.
6.RedBlack extra 100 insertion takes 71476 ns.
7.RebBlackTree 10000 insertion takes 5 ms.
7.RedBlack extra 100 insertion takes 54336 ns.
8.RebBlackTree 10000 insertion takes 2 ms.
8.RedBlack extra 100 insertion takes 47407 ns.
9.RebBlackTree 10000 insertion takes 2 ms.
9.RedBlack extra 100 insertion takes 36103 ns.
10.RebBlackTree 10000 insertion takes 2 ms.
10.RedBlack extra 100 insertion takes 36103 ns.
10000 insertion takes average time 6.1 ms.
10000 extra insertion takes average time 76180.1 ns.
```

```
1.RebBlackTree 20000 insertion takes 12 ms.
1.RedBlack extra 100 insertion takes 78769 ns.
2.RebBlackTree 20000 insertion takes 12 ms.
2.RedBlack extra 100 insertion takes 83145 ns.
3.RebBlackTree 20000 insertion takes 11 ms.
3.RedBlack extra 100 insertion takes 73299 ns.
4.RebBlackTree 20000 insertion takes 13 ms.
4.RedBlack extra 100 insertion takes 82416 ns.
5.RebBlackTree 20000 insertion takes 11 ms.
5.RedBlack extra 100 insertion takes 97367 ns.
6.RebBlackTree 20000 insertion takes 8 ms.
6.RedBlack extra 100 insertion takes 87157 ns.
7.RebBlackTree 20000 insertion takes 6 ms.
7.RedBlack extra 100 insertion takes 44854 ns.
8.RebBlackTree 20000 insertion takes 5 ms.
8.RedBlack extra 100 insertion takes 41208 ns.
9.RebBlackTree 20000 insertion takes 7 ms.
9.RedBlack extra 100 insertion takes 88250 ns.
10.RebBlackTree 20000 insertion takes 7 ms.
10.RedBlack extra 100 insertion takes 46313 ns.
20000 insertion takes average time 9.2 ms.
20000 extra insertion takes average time 72277.8 ns.
```

```
1.RebBlackTree 40000 insertion takes 21 ms.
1.RedBlack extra 100 insertion takes 119977 ns.
2.RebBlackTree 40000 insertion takes 19 ms.
2.RedBlack extra 100 insertion takes 96639 ns.
3.RebBlackTree 40000 insertion takes 19 ms.
3.RedBlack extra 100 insertion takes 44490 ns.
4.RebBlackTree 40000 insertion takes 22 ms.
4.RedBlack extra 100 insertion takes 86427 ns.
5.RebBlackTree 40000 insertion takes 17 ms.
5.RedBlack extra 100 insertion takes 53971 ns.
6.RebBlackTree 40000 insertion takes 20 ms.
6.RedBlack extra 100 insertion takes 112683 ns.
7.RebBlackTree 40000 insertion takes 17 ms.
7.RedBlack extra 100 insertion takes 80957 ns.
8.RebBlackTree 40000 insertion takes 15 ms.
8.RedBlack extra 100 insertion takes 52148 ns.
9.RebBlackTree 40000 insertion takes 15 ms.
9.RedBlack extra 100 insertion takes 61265 ns.
10.RebBlackTree 40000 insertion takes 15 ms.
10.RedBlack extra 100 insertion takes 50689 ns.
40000 insertion takes average time 18.0 ms.
40000 extra insertion takes average time 75924.6 ns.
```

```
1.RebBlackTree 80000 insertion takes 36 ms.
1.RedBlack extra 100 insertion takes 87156 ns.
2.RebBlackTree 80000 insertion takes 41 ms.
2.RedBlack extra 100 insertion takes 58347 ns.
3.RebBlackTree 80000 insertion takes 44 ms.
3.RedBlack extra 100 insertion takes 53242 ns.
4.RebBlackTree 80000 insertion takes 35 ms.
4.RedBlack extra 100 insertion takes 52513 ns.
5.RebBlackTree 80000 insertion takes 32 ms.
5.RedBlack extra 100 insertion takes 52877 ns.
6.RebBlackTree 80000 insertion takes 31 ms.
6.RedBlack extra 100 insertion takes 50690 ns.
7.RebBlackTree 80000 insertion takes 44 ms.
7.RedBlack extra 100 insertion takes 133469 ns.
8.RebBlackTree 80000 insertion takes 41 ms.
8.RedBlack extra 100 insertion takes 55430 ns.
9.RebBlackTree 80000 insertion takes 44 ms.
9.RedBlack extra 100 insertion takes 79498 ns.
10.RebBlackTree 80000 insertion takes 34 ms.
10.RedBlack extra 100 insertion takes 54701 ns.
80000 insertion takes average time 38.2 ms.
80000 extra insertion takes average time 67792.3 ns.
```

```
1.2-3 Tree 10000 insertion takes 9 ms.
1.2-3 Tree extra 100 insertion takes 85698 ns.
2.2-3 Tree 10000 insertion takes 6 ms.
2.2-3 Tree extra 100 insertion takes 75852 ns.
3.2-3 Tree 10000 insertion takes 5 ms.
3.2-3 Tree extra 100 insertion takes 69653 ns.
4.2-3 Tree 10000 insertion takes 6 ms.
4.2-3 Tree extra 100 insertion takes 71840 ns.
5.2-3 Tree 10000 insertion takes 6 ms.
5.2-3 Tree extra 100 insertion takes 75123 ns.
6.2-3 Tree 10000 insertion takes 6 ms.
6.2-3 Tree extra 100 insertion takes 98097 ns.
7.2-3 Tree 10000 insertion takes 7 ms.
7.2-3 Tree extra 100 insertion takes 137117 ns.
8.2-3 Tree 10000 insertion takes 7 ms.
8.2-3 Tree extra 100 insertion takes 66005 ns.
9.2-3 Tree 10000 insertion takes 5 ms.
9.2-3 Tree extra 100 insertion takes 66370 ns.
10.2-3 Tree 10000 insertion takes 5 ms.
10.2-3 Tree extra 100 insertion takes 67829 ns.
10000 insertion takes average time 6.2 ms.
10000 extra insertion takes average time 81358.4 ns.
```

```
1.2-3 Tree 20000 insertion takes 12 ms.
1.2-3 Tree extra 100 insertion takes 76216 ns.
2.2-3 Tree 20000 insertion takes 11 ms.
2.2-3 Tree extra 100 insertion takes 84604 ns.
3.2-3 Tree 20000 insertion takes 12 ms.
3.2-3 Tree extra 100 insertion takes 76946 ns.
4.2-3 Tree 20000 insertion takes 14 ms.
4.2-3 Tree extra 100 insertion takes 97002 ns.
5.2-3 Tree 20000 insertion takes 16 ms.
5.2-3 Tree extra 100 insertion takes 71111 ns.
6.2-3 Tree 20000 insertion takes 13 ms.
6.2-3 Tree extra 100 insertion takes 83874 ns.
7.2-3 Tree 20000 insertion takes 14 ms.
7.2-3 Tree extra 100 insertion takes 117060 ns.
8.2-3 Tree 20000 insertion takes 13 ms.
8.2-3 Tree extra 100 insertion takes 126177 ns.
9.2-3 Tree 20000 insertion takes 12 ms.
9.2-3 Tree extra 100 insertion takes 86063 ns.
10.2-3 Tree 20000 insertion takes 12 ms.
10.2-3 Tree extra 100 insertion takes 72205 ns.
20000 insertion takes average time 12.9 ms.
20000 extra insertion takes average time 89125.8 ns.
```

```
1.2-3 Tree 40000 insertion takes 26 ms.
1.2-3 Tree extra 100 insertion takes 73299 ns.
2.2-3 Tree 40000 insertion takes 22 ms.
2.2-3 Tree extra 100 insertion takes 106849 ns.
3.2-3 Tree 40000 insertion takes 20 ms.
3.2-3 Tree extra 100 insertion takes 76945 ns.
4.2-3 Tree 40000 insertion takes 21 ms.
4.2-3 Tree extra 100 insertion takes 118518 ns.
5.2-3 Tree 40000 insertion takes 25 ms.
5.2-3 Tree extra 100 insertion takes 77311 ns.
6.2-3 Tree 40000 insertion takes 32 ms.
6.2-3 Tree extra 100 insertion takes 71840 ns.
7.2-3 Tree 40000 insertion takes 32 ms.
7.2-3 Tree extra 100 insertion takes 62359 ns.
8.2-3 Tree 40000 insertion takes 21 ms.
8.2-3 Tree extra 100 insertion takes 61265 ns.
9.2-3 Tree 40000 insertion takes 22 ms.
9.2-3 Tree extra 100 insertion takes 73299 ns.
10.2-3 Tree 40000 insertion takes 18 ms.
10.2-3 Tree extra 100 insertion takes 58347 ns.
40000 insertion takes average time 23.9 ms.
40000 extra insertion takes average time 78003.2 ns.
```

```
1.2-3 Tree 80000 insertion takes 54 ms.
1.2-3 Tree extra 100 insertion takes 164832 ns.
2.2-3 Tree 80000 insertion takes 95 ms.
2.2-3 Tree extra 100 insertion takes 101743 ns.
3.2-3 Tree 80000 insertion takes 56 ms.
3.2-3 Tree extra 100 insertion takes 96638 ns.
4.2-3 Tree 80000 insertion takes 48 ms.
4.2-3 Tree extra 100 insertion takes 57254 ns.
5.2-3 Tree 80000 insertion takes 40 ms.
5.2-3 Tree extra 100 insertion takes 77676 ns.
6.2-3 Tree 80000 insertion takes 40 ms.
6.2-3 Tree extra 100 insertion takes 75123 ns.
7.2-3 Tree 80000 insertion takes 40 ms.
7.2-3 Tree extra 100 insertion takes 68558 ns.
8.2-3 Tree 80000 insertion takes 64 ms.
8.2-3 Tree extra 100 insertion takes 105390 ns.
9.2-3 Tree 80000 insertion takes 52 ms.
9.2-3 Tree extra 100 insertion takes 95180 ns.
10.2-3 Tree 80000 insertion takes 41 ms.
10.2-3 Tree extra 100 insertion takes 75122 ns.
80000 insertion takes average time 53.0 ms.
80000 extra insertion takes average time 91751.6 ns.
```

```
1.BTree 10000 insertion takes 3 ms.
1.BTree extra 100 insertion takes 44126 ns.
2.BTree 10000 insertion takes 3 ms.
2.BTree extra 100 insertion takes 41938 ns.
3.BTree 10000 insertion takes 3 ms.
3.BTree extra 100 insertion takes 41573 ns.
4.BTree 10000 insertion takes 3 ms.
4.BTree extra 100 insertion takes 41208 ns.
5.BTree 10000 insertion takes 3 ms.
5.BTree extra 100 insertion takes 45948 ns.
6.BTree 10000 insertion takes 3 ms.
6.BTree extra 100 insertion takes 43396 ns.
7.BTree 10000 insertion takes 3 ms.
7.BTree extra 100 insertion takes 43760 ns.
8.BTree 10000 insertion takes 3 ms.
8.BTree extra 100 insertion takes 53242 ns.
9.BTree 10000 insertion takes 3 ms.
9.BTree extra 100 insertion takes 37926 ns.
10.BTree 10000 insertion takes 3 ms.
10.BTree extra 100 insertion takes 37196 ns.
10000 insertion takes average time 3.0 ms
10000 extra insertion takes average time 43031.3 ns.
```

```
1.BTree 20000 insertion takes 6 ms.
1.BTree extra 100 insertion takes 41937 ns.
2.BTree 20000 insertion takes 7 ms.
2.BTree extra 100 insertion takes 44490 ns.
3.BTree 20000 insertion takes 6 ms.
3.BTree extra 100 insertion takes 42302 ns.
4.BTree 20000 insertion takes 8 ms.
4.BTree extra 100 insertion takes 64912 ns.
5.BTree 20000 insertion takes 8 ms.
5.BTree extra 100 insertion takes 86428 ns.
6.BTree 20000 insertion takes 6 ms.
6.BTree extra 100 insertion takes 47772 ns.
7.BTree 20000 insertion takes 6 ms.
7.BTree extra 100 insertion takes 40478 ns.
8.BTree 20000 insertion takes 9 ms.
8.BTree extra 100 insertion takes 48866 ns.
9.BTree 20000 insertion takes 6 ms.
9.BTree extra 100 insertion takes 43031 ns.
10.BTree 20000 insertion takes 9 ms.
10.BTree extra 100 insertion takes 48137 ns.
20000 insertion takes average time 7.1 ms
20000 extra insertion takes average time 50835.3 ns.
```

```
1.BTree 40000 insertion takes 18 ms.
1.BTree extra 100 insertion takes 45949 ns.
2.BTree 40000 insertion takes 18 ms.
2.BTree extra 100 insertion takes 45949 ns.
3.BTree 40000 insertion takes 17 ms.
3.BTree extra 100 insertion takes 46314 ns.
4.BTree 40000 insertion takes 17 ms.
4.BTree extra 100 insertion takes 57619 ns.
5.BTree 40000 insertion takes 18 ms.
5.BTree extra 100 insertion takes 72205 ns.
6.BTree 40000 insertion takes 23 ms.
6.BTree extra 100 insertion takes 72205 ns.
7.BTree 40000 insertion takes 17 ms.
7.BTree extra 100 insertion takes 55430 ns.
8.BTree 40000 insertion takes 15 ms.
8.BTree extra 100 insertion takes 47408 ns.
9.BTree 40000 insertion takes 14 ms.
9.BTree extra 100 insertion takes 81322 ns.
10.BTree 40000 insertion takes 16 ms.
10.BTree extra 100 insertion takes 182701 ns.
40000 insertion takes average time 17.3 ms
40000 extra insertion takes average time 70710.2 ns.
```

```
1.BTree 80000 insertion takes 42 ms.
1.BTree extra 100 insertion takes 53242 ns.
2.BTree 80000 insertion takes 45 ms.
2.BTree extra 100 insertion takes 55794 ns.
3.BTree 80000 insertion takes 36 ms.
3.BTree extra 100 insertion takes 51783 ns.
4.BTree 80000 insertion takes 33 ms.
4.BTree extra 100 insertion takes 52512 ns.
5.BTree 80000 insertion takes 33 ms.
5.BTree extra 100 insertion takes 50689 ns.
6.BTree 80000 insertion takes 40 ms.
6.BTree extra 100 insertion takes 88615 ns.
7.BTree 80000 insertion takes 42 ms.
7.BTree extra 100 insertion takes 65641 ns.
8.BTree 80000 insertion takes 45 ms.
8.BTree extra 100 insertion takes 53606 ns.
9.BTree 80000 insertion takes 37 ms.
9.BTree extra 100 insertion takes 54701 ns.
10.BTree 80000 insertion takes 33 ms.
10.BTree extra 100 insertion takes 53607 ns.
80000 insertion takes average time 38.6 ms
80000 extra insertion takes average time 58019.0 ns.
```

```
1.SkipList 10000 insertion takes 8 ms.
1.SkipList extra 100 insertion takes 74758 ns.
2.SkipList 10000 insertion takes 4 ms.
2.SkipList extra 100 insertion takes 61629 ns.
3.SkipList 10000 insertion takes 4 ms.
3.SkipList extra 100 insertion takes 60900 ns.
4.SkipList 10000 insertion takes 4 ms.
4.SkipList extra 100 insertion takes 63818 ns.
5.SkipList 10000 insertion takes 4 ms.
5.SkipList extra 100 insertion takes 57254 ns.
6.SkipList 10000 insertion takes 4 ms.
6.SkipList extra 100 insertion takes 59806 ns.
7.SkipList 10000 insertion takes 6 ms.
7.SkipList extra 100 insertion takes 91533 ns.
8.SkipList 10000 insertion takes 5 ms.
8.SkipList extra 100 insertion takes 53607 ns.
9.SkipList 10000 insertion takes 4 ms.
9.SkipList extra 100 insertion takes 56160 ns.
10.SkipList 10000 insertion takes 3 ms.
10.SkipList extra 100 insertion takes 64547 ns.
10000 insertion takes average time 4.6 ms.
10000 extra insertion takes average time 64401.2 ns.
```

```
1.SkipList 20000 insertion takes 10 ms.
1.SkipList extra 100 insertion takes 83509 ns.
2.SkipList 20000 insertion takes 12 ms.
2.SkipList extra 100 insertion takes 64547 ns.
3.SkipList 20000 insertion takes 14 ms.
3.SkipList extra 100 insertion takes 110495 ns.
4.SkipList 20000 insertion takes 13 ms.
4.SkipList extra 100 insertion takes 133470 ns.
5.SkipList 20000 insertion takes 11 ms.
5.SkipList extra 100 insertion takes 389470 ns.
6.SkipList 20000 insertion takes 10 ms.
6.SkipList extra 100 insertion takes 64912 ns.
7.SkipList 20000 insertion takes 13 ms.
7.SkipList extra 100 insertion takes 68559 ns.
8.SkipList 20000 insertion takes 13 ms.
8.SkipList extra 100 insertion takes 60900 ns.
9.SkipList 20000 insertion takes 14 ms.
9.SkipList extra 100 insertion takes 71840 ns.
10.SkipList 20000 insertion takes 8 ms.
10.SkipList extra 100 insertion takes 60535 ns.
20000 insertion takes average time 11.8 ms.
20000 extra insertion takes average time 110823.7 ns.
```

```
1.SkipList 40000 insertion takes 27 ms.
1.SkipList extra 100 insertion takes 101743 ns.
2.SkipList 40000 insertion takes 24 ms.
2.SkipList extra 100 insertion takes 114872 ns.
3.SkipList 40000 insertion takes 24 ms.
3.SkipList extra 100 insertion takes 88251 ns.
4.SkipList 40000 insertion takes 28 ms.
4.SkipList extra 100 insertion takes 238130 ns.
5.SkipList 40000 insertion takes 28 ms.
5.SkipList extra 100 insertion takes 113413 ns.
6.SkipList 40000 insertion takes 30 ms.
6.SkipList extra 100 insertion takes 89710 ns.
7.SkipList 40000 insertion takes 31 ms.
7.SkipList extra 100 insertion takes 101378 ns.
8.SkipList 40000 insertion takes 27 ms.
8.SkipList extra 100 insertion takes 103567 ns.
9.SkipList 40000 insertion takes 25 ms.
9.SkipList extra 100 insertion takes 89344 ns.
10.SkipList 40000 insertion takes 22 ms.
10.SkipList extra 100 insertion takes 92991 ns.
40000 insertion takes average time 26.6 ms.
40000 extra insertion takes average time 113339.9 ns.
```

```
1.SkipList 80000 insertion takes 58 ms.
1.SkipList extra 100 insertion takes 125812 ns.
2.SkipList 80000 insertion takes 79 ms.
2.SkipList extra 100 insertion takes 115237 ns.
3.SkipList 80000 insertion takes 65 ms.
3.SkipList extra 100 insertion takes 105755 ns.
4.SkipList 80000 insertion takes 55 ms.
4.SkipList extra 100 insertion takes 105025 ns.
5.SkipList 80000 insertion takes 64 ms.
5.SkipList extra 100 insertion takes 122530 ns.
6.SkipList 80000 insertion takes 65 ms.
6.SkipList extra 100 insertion takes 115601 ns.
7.SkipList 80000 insertion takes 56 ms.
7.SkipList extra 100 insertion takes 152068 ns.
8.SkipList 80000 insertion takes 58 ms.
8.SkipList extra 100 insertion takes 63453 ns.
9.SkipList 80000 insertion takes 62 ms.
9.SkipList extra 100 insertion takes 122165 ns.
10.SkipList 80000 insertion takes 67 ms.
10.SkipList extra 100 insertion takes 139305 ns.
80000 insertion takes average time 62.9 ms.
80000 extra insertion takes average time 116695.1 ns.
```

Average Insertion Times	10 000	20 000	40 000	80 000
BST	3.8ms	7.2ms	15.2ms	34.6ms
RBT	6.1ms	9.2ms	18.0ms	38.2ms
2-3 Tree	6.2ms	12.9ms	23.9ms	53.0ms
BTree	3.0ms	7.1ms	17.3ms	38.6ms
Skip List	4.6ms	11.8ms	26.6ms	62.9ms

Extra 100 Insertion Average Times	10 000	20 000	40 000	80 000
BST	54810.1ns	56013.7ns	71621.7ns	80592.6ns
RBT	76180.1ns	72277.8ns	75924.6ns	67792.3ns
2-3 Tree	81358.4ns	89125.8ns	78003.2ns	91751.6ns
BTree	43031.3ns	50835.3ns	70710.2ns	58019.0ns
Skip List	64401.2ns	110823.7ns	113339.9ns	116695.1ns

```
Process finished with exit code 0
```