GTU Department of Computer Engineering

CSE 222/505 - Spring 2021 Homework 7

Report

Muhammet Fikret ATAR

1801042693

PART 1:

PROBLEM SOLUTION APPROACH:

I looked to the book’s implementation of the methods so that they can give me an example.I extended the class so that it implements the NavigableSet. I used the already implemented functions and made slight adjustments. This helped tremendously.I couldnt do descendingIterator, iterator, headSet ,tailSet

Test Cases:

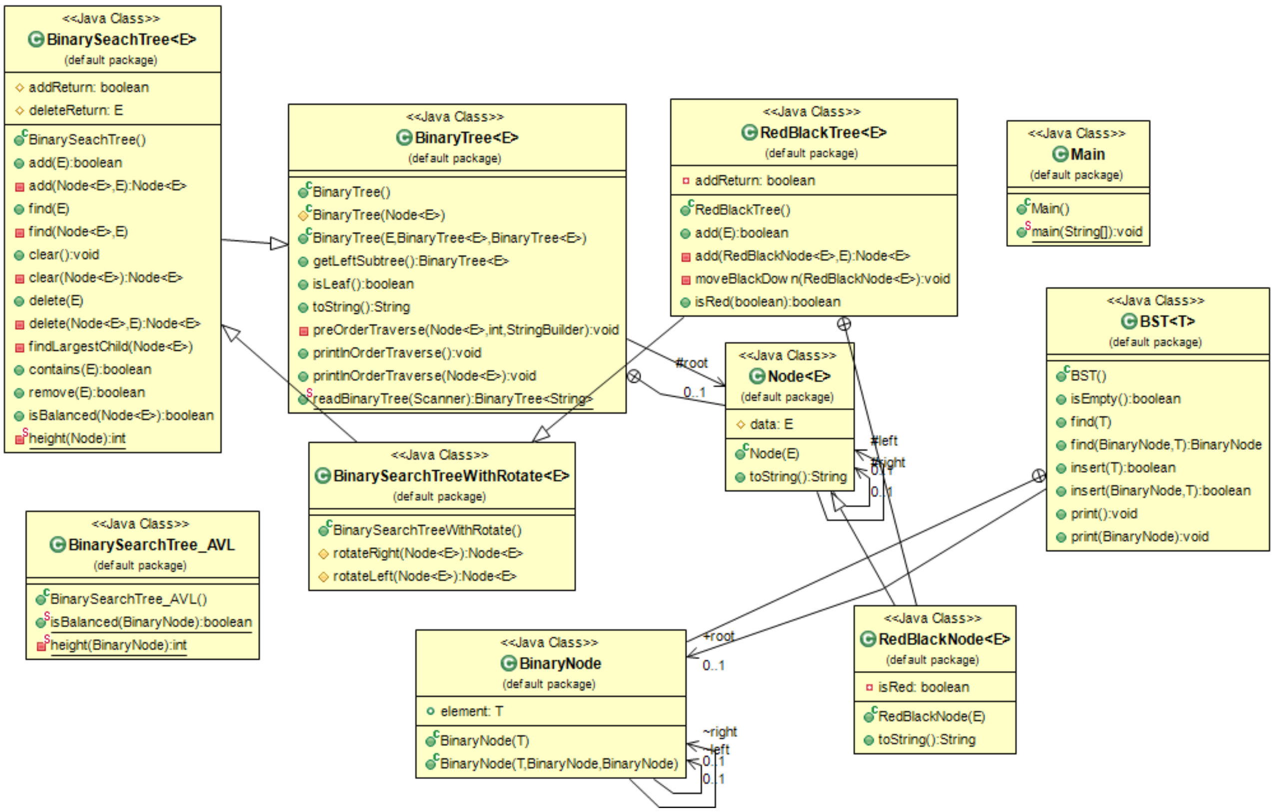
Test Case Id	Test Scenario	Test Steps	Test Data	Excepted Results	Actual Results	Pass/Fail
T1	Call insert function fors kip list	Call method	101,102,103	added	As expected	Pass
T2	Call remove function fors kip list	Call method	101,102,103	removed	As expected	Pass
T3	Call insert function for avl tree	Call method	101,102,103	added	As expected	Pass

Running Result:

```
Skip List Insert Remove
Successful add
Successful add
Successful add
Successful remove
Successful remove
Successful remove
AVL Tree Insert
Successful add
Successful add
Successful add
```

Part 2:

Class Diagrams:



Problem Solution Approach

Firstly I made the decision to create a function that checks whether the tree is actually an Binary Search Tree. Then I made a function to check if the tree is balanced. It goes until the leftmost or the rightmost node recursively.Then the check function realizes these functions and returns true or false.Mostly same things apply to the Red Black Tree as well.The Node is in its own class, so that I wouldn't need to create them for each tree; AVL and Red Black.

Test Cases:

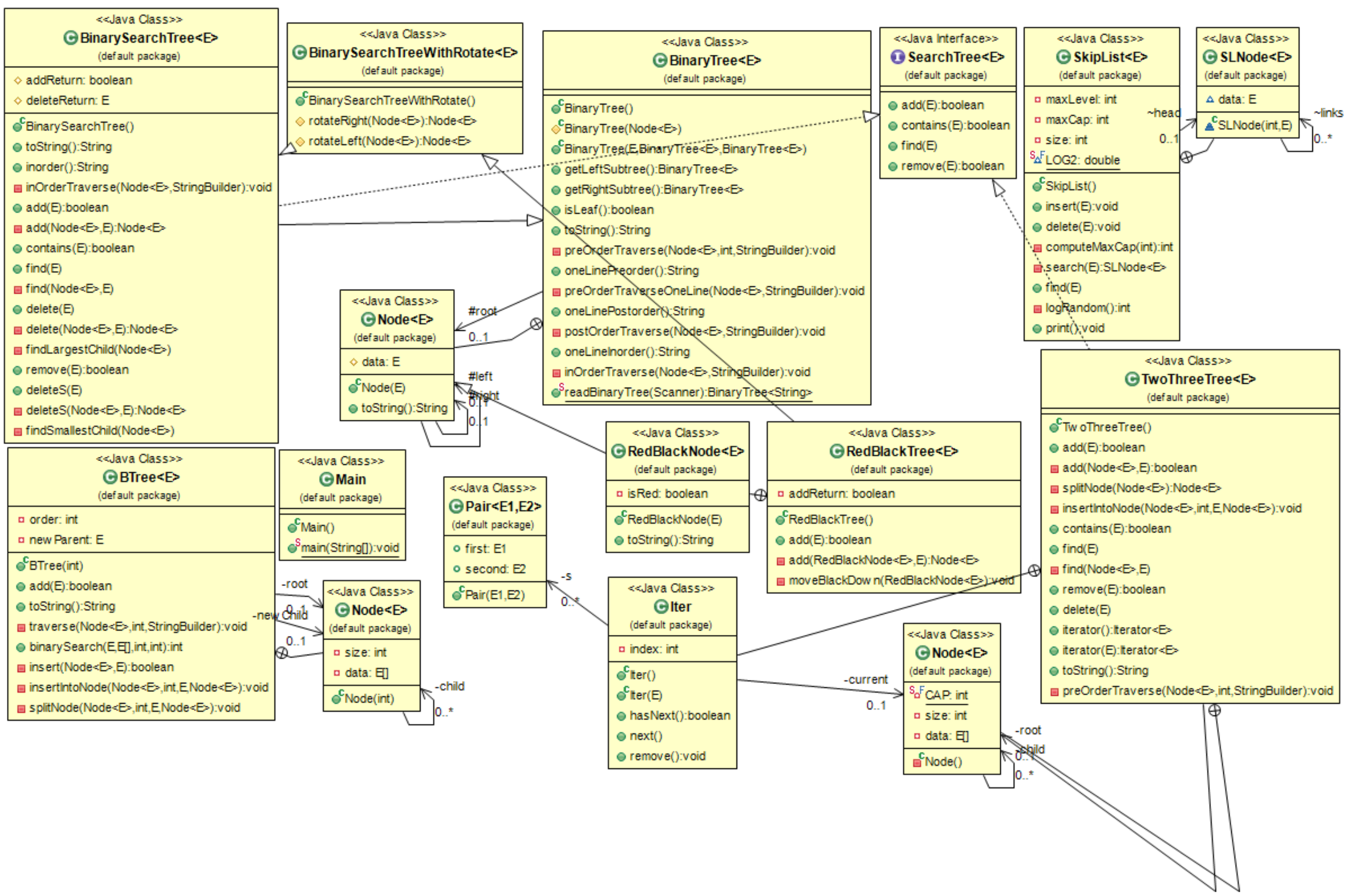
Test Case Id	Test Scenario	Test Steps	Test Data	Excepted Results	Actual Results	Pass/Fail
T1	Checks AVL tree	Call isBlanced Function	45,10,7,12 ,90,50	It is an Avl Tree	Avl Tree	Pass
T2	Checks AVL tree	Call isBlanced Function	-	It is an Avl Tree	Avl Tree	Pass
T3	Checks AVL tree	Call isBlanced Function	1,2,3,4,5	It is not an Avl Tree	Not Avl Tree	Pass
T4	Checks Red-Black tree	Call isRed-Blcktree	1,2,3,4,5,1 ,2,3,4,5	It is not a red Black Tree	Not red Black Tree	Pass
T5	Checks Red-Black tree	Call isRed-Blcktree	45,10,7,90,50 ,45,10,7,90,50	It is a red Black Tree	red Black Tree	Pass

Running Result

T1 `It is an AVL tree`
T2 `It is an AVL tree`
T3 `It is not an AVL tree`
T4 `It is not a Red Black tree`
T5 `It is a Red Black tree`

Part 3:

Class diagrams



PROBLEM SOLUTION APPROACH :

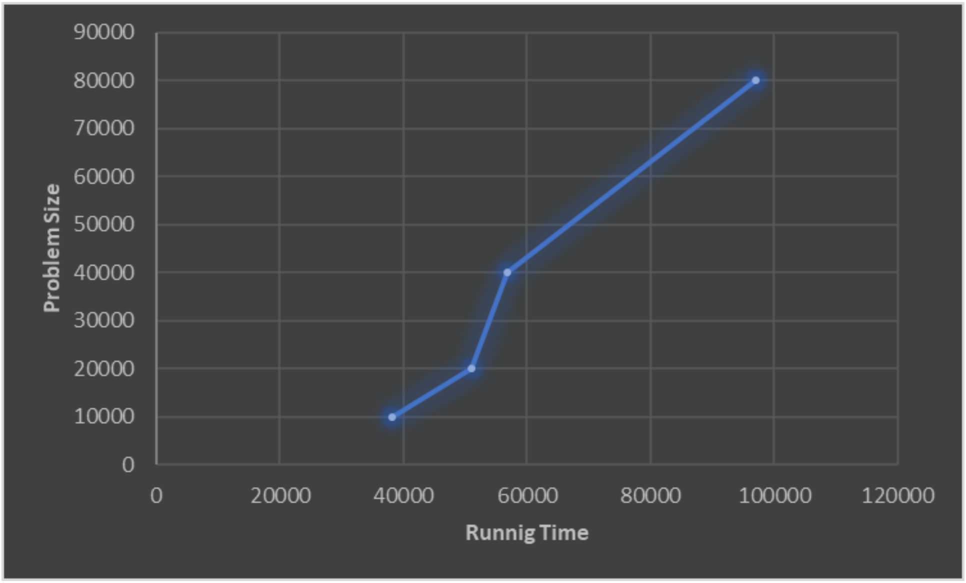
Since we need BST, Red Black Tree, 2-3 Tree, B-Tree and Skip List's implementations to measure their times, I took them exactly as they are from the book. Then, I created an instance of them in the main driver class, with some variables. These variables are the size, number of repetitions, and extra number amount.Instead of filling every instance of a tree every time, I decided to create a method that fills the array and repeats it for 10 times. The time is measured in the test methods and also printed in the same method. I used nanoseconds instead of milliseconds.

Test Cases:

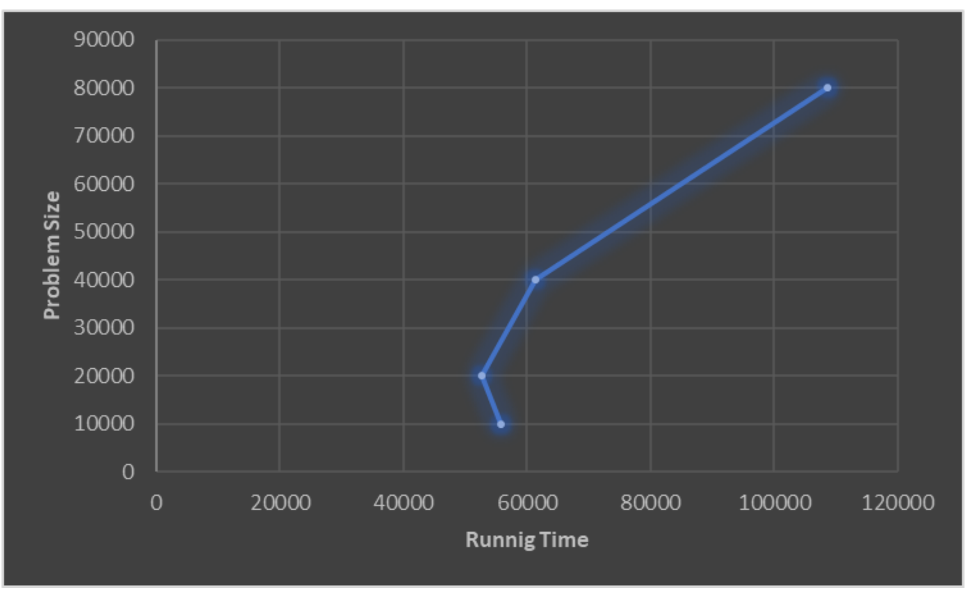
Test Case Id	Test Scenario	Test Steps	Test Data	Excepted Results	Actual Results	Pass/Fail
T1	Check size is 10000	Random number adding this data structures and 10 extra number, then delete all of them	Measure time	Showing measuring time of each data structures	As Excepted	Pass
T2	Check size is 20000	Random number adding this data structures and 10 extra number, then delete all of them	Measure time	Showing measuring time of each data structures	As Excepted	Pass
T3	Check size is 40000	Random number adding this data structures and 10 extra number, then delete all of them	Measure time	Showing measuring time of each data structures	As Excepted	Pass
T4	Check size is 40000	Random number adding this data structures and 10 extra number, then delete all of them	Measure time	Showing measuring time of each data structures	As Excepted	Pass

Graphs:

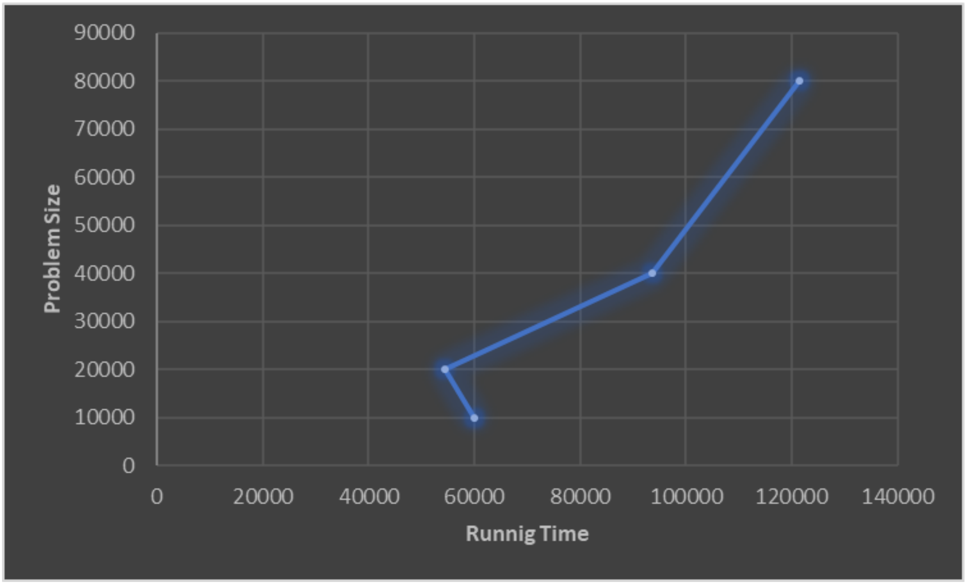
Bst:



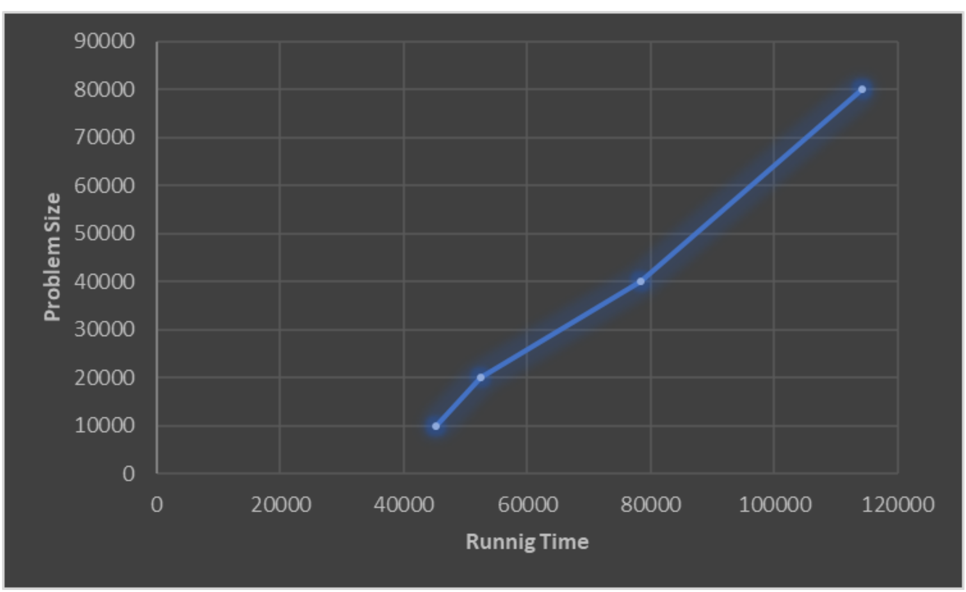
Red Black Tree



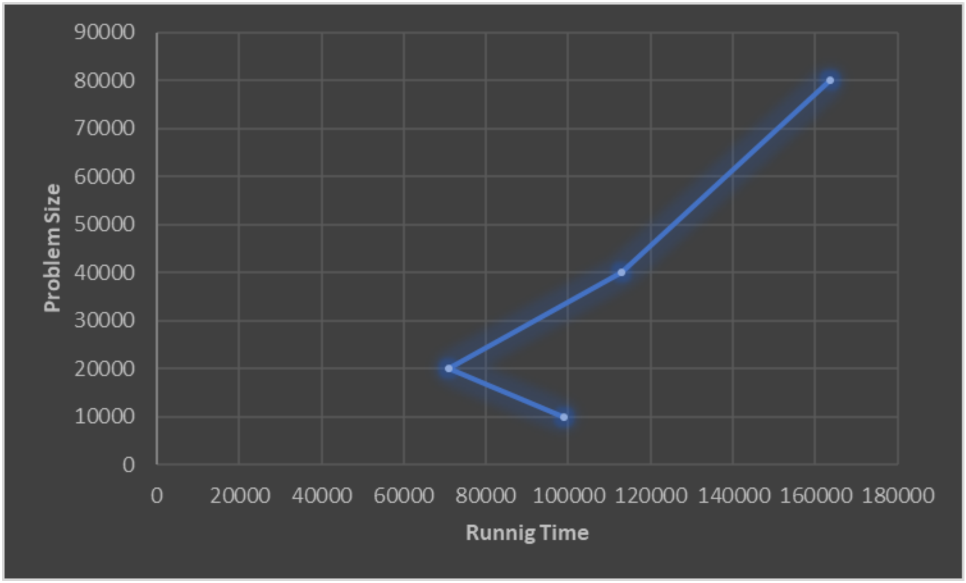
2-3 Tree



B-Tree



Skip List



Result of graph

As you can see, some trees are better with bigger sizes, and some trees are better with smaller sizes. There is also the fact that all trees get worse timings as the size increases. This does not mean they get worse linearly of logarithmically exactly. Some get the form of linear; some get the form of logarithmic. But if we want to generalize, Binary Search Tree is the fastest in all of them in every size.

Runnig Result:

```
PART 3:
Compare insertion performance of the following data structures;

Perform this operation 10 times for 10.000
Binary Search Tree
Average->->38120
Red Black Tree
Average->->55740
2-3 tree
Average->->60060
B-Tree
Average->->45290
Skip list
Average->->98880

Perform this operation 10 times for 20.000
Binary Search Tree
Average->->51090
Red Black Tree
Average->->52770
2-3 tree
Average->->54550
B-Tree
Average->->52530
Skip list
Average->->71040

Perform this operation 10 times for 40.000
Binary Search Tree
Average->->56830
Red Black Tree
Average->->61480
2-3 tree
Average->->93520
B-Tree
Average->->78490
Skip list
Average->->113070
```

```
Perform this operation 10 times for 80.000
Binary Search Tree
Average->->97140
Red Black Tree
Average->->108600
2-3 tree
Average->->121560
B-Tree
Average->->114270
Skip list
Average->->163720
```