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

**Yeşim Yalçın**

**200104004094**

**1-) System Requirements**

**-**The system is written using Java jdk 15.0.1

-You can use the makefile then use java DriverCode2 command to run the program.

**2-) Class Diagram**

**-**The class diagram is added as ClassDiagram.png to the homework file.

**3-) Problem Solution Aproach**

-In part 2, I have implemented the isRed method as requested in the homework PDF. isRed method throws and exception in BinarySearchTree because in regular BinarySearchTrees there are no colouring therefore checking for the root’s colour is meaningless. AVLTree I have implemented extends from BinarySearchTree and also throws exception in this method for the same reasons. However, in RedBlackTree this method returns false because in RedBlackTrees the root node is always black. The method AVLOrRedBlack decides if the given BinarySearchTree is an AVLTree, RedBlackTree or a regular BinarySearchTree according to what happens in isRed method. If it return false, it means it is a RedBlackTree. If it throws an exception the type is decided according to the given exception message from the classes.

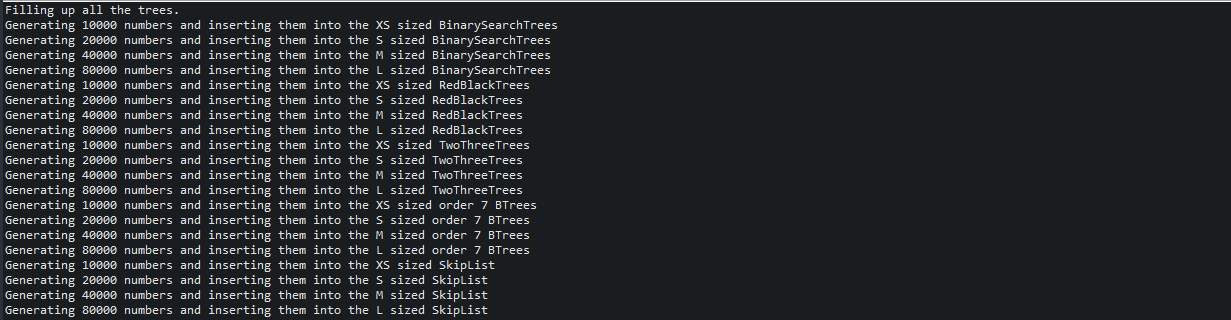
-In part 3, I have firstly filled all the trees and the SkipLists. I have created 10 from XS (10.000), S(20.000), M(40.000), L(80.000) sized containers. 10 from each size for all the different type of containers. I have randomized the elements by choosing an upperbound of x5 of their sizes so that the random choosing can be efficient enough. Also I have chosen the order of BTree as 7 to have a significant difference from the TwoThreeTree.

After filling all the containers, I have randomized 100 more elements by choosing a lowerbound of their previous upperbound and an upperbound of lowerbound+500 to avoid the same elements to be chosen as much as possible. I have calculated the total and average running time of all sized containers separately.

**4-) Test Cases and Running Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Scenario** | **Expected Result** | **Real Result** | **Pass/Fail** |
| **1** | Creating 10 from each size of all containers. | Successful creation | The same | Pass |
| **2** | Testing the running time of XS(10.000) sized BinarySearchTree. | 22460ns (0.0225ms)  Runtime average result received. | The same | Pass |
| **3** | Testing the running time of S(20.000) sized BinarySearchTree. | 22600ns (0.0226ms) Runtime average result received. | The same | Pass |
| **4** | Testing the running time of M(40.000) sized BinarySeacrhTree. | 23350ns (0.0233ms) Runtime average result received. | The same | Pass |
| **5** | Testing the running time of L(80.000) sized BinarySearchTree. | 24320ns (0.0243ms) Runtime average result received. | The same | Pass |
| **6** | Testing the running time of XS(10.000) sized RedBlackTree. | 27860ns (0.0279ms) Runtime average result received. | The same | Pass |
| **7** | Testing the running time of S(20.000) sized RedBlackTree. | 29090ns  (0.0290ms) Runtime average result received. | The same | Pass |
| **8** | Testing the running time of M(40.000) sized RedBlackTree. | 35710ns  (0.0357ms) Runtime average result received. | The same | Pass |
| **9** | Testing the running time of L(80.000) sized RedBlackTree. | 31610ns  (0.0316ms) Runtime average result received. | The same | Pass |
| **10** | Testing the running time of XS(10.000) sized TwoThreeTree. | 30270ns  (00303ms) Runtime average result received. | The same | Pass |
| **11** | Testing the running time of S(20.000) sized TwoThreeTree. | 29780ns  (0.0298ms) Runtime average result received. | The same | Pass |
| **12** | Testing the running time of M(40.000) sized TwoThreeTree. | 30500ns  (0.0305ms) Runtime average result received. | The same | Pass |
| **13** | Testing the running time of L(80.000) sized TwoThreeTree. | 31800ns  (0.0318ms) Runtime average result received. | The same | Pass |
| **14** | Testing the running time of XS(10.000) sized BTree. | 25960ns  (0.0260ms) Runtime average result received. | The same | Pass |
| **15** | Testing the running time of S(20.000) sized BTree. | 25070ns  (0.02507ms) Runtime average result received. | The same | Pass |
| **16** | Testing the running time of M(40.000) sized BTree. | 26560ns  (0.2656ms) Runtime average result received. | The same | Pass |
| **17** | Testing the running time of L(80.000) sized BTree | 26170ns  (0.0261ms) Runtime average result received. | The same | Pass |
| **18** | Testing the running time of XS(10.000) sized SkipList. | 28290ns  (0.0283ms) Runtime average result received. | The same | Pass |
| **19** | Testing the running time of S(20.000) sized SkipList | 28490ns  (0.0285ms) Runtime average result received. | The same | Pass |
| **20** | Testing the running time of M(40.000) sized SkipList. | 28510ns  (0.0285ms) Runtime average result received. | The same | Pass |
| **21** | Testing the running time of L(80.000) sized SkipList. | 34510ns  (0.0345ms) Runtime average result received. | The same | Pass |
| **22** | Testing AVLOrRedBlack method with an AVLTree. | Exception thrown, AVLTree is detected. | The same | Pass |
| **23** | Testing AVLOrRedBlack method with a RedBlackTree | RedBlackTree is detected. | The same | Pass |
| **24** | Testing AVLOrRedBlack method with a regular BinarySearchTree. | Exception thrown, regular BinarySearchTree is detected. | The same | Pass |

1-) Creating 10 from each size of all containers.



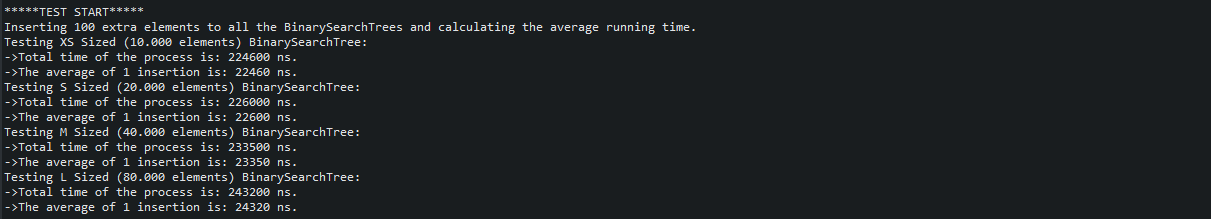
2-)

Testing the running time of 10 instances of XS(10.000) sized BinarySearchTree.

Testing the running time of 10 instances of S(20.000) sized BinarySearchTree

Testing the running time of 10 instances of M(40.000) sized BinarySeacrhTree.

Testing the running time of 10 instances of L(80.000) sized BinarySearchTree



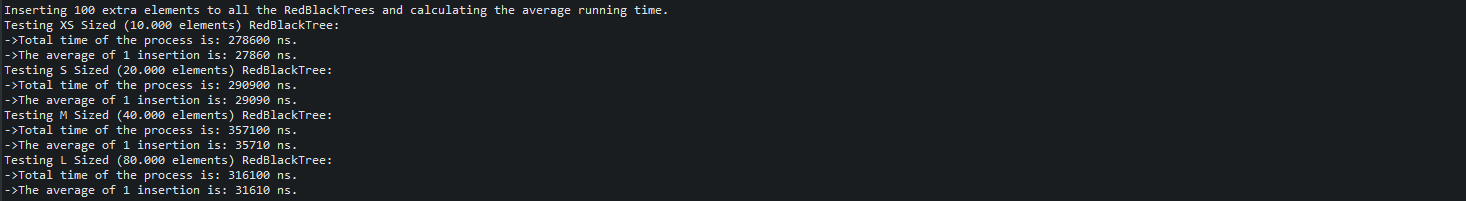
3-)

Testing the running time of 10 instances of XS(10.000) sized RedBlackTree.

Testing the running time of 10 instances of S(20.000) sized RedBlackTree

Testing the running time of 10 instances of M(40.000) sized RedBlackTree.

Testing the running time of 10 instances of L(80.000) sized RedBlackTree



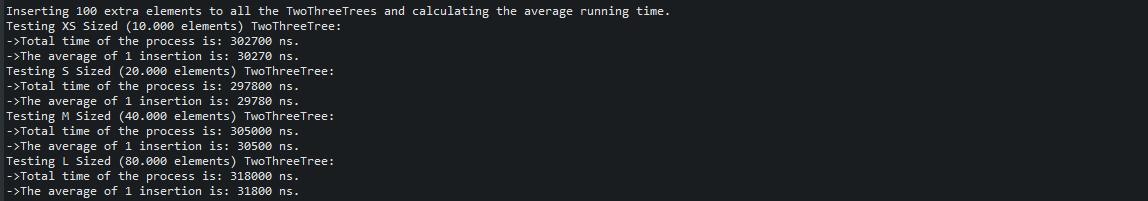
4-)

Testing the running time of 10 instances of XS(10.000) sized TwoThreeTree.

Testing the running time of 10 instances of S(20.000) sized TwoThreeTree

Testing the running time of 10 instances of M(40.000) sized TwoThreeTree.

Testing the running time of 10 instances of L(80.000) sized TwoThreeTree



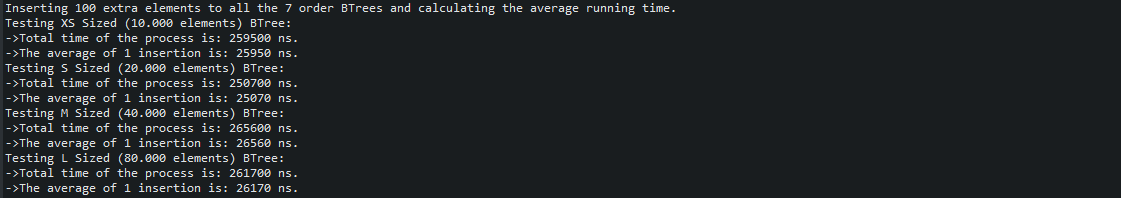
5-)

Testing the running time of 10 instances of XS(10.000) sized BTree.

Testing the running time of 10 instances of S(20.000) sized BTree.

Testing the running time of 10 instances of M(40.000) sized BTree.

Testing the running time of 10 instances of L(80.000) sized BTree



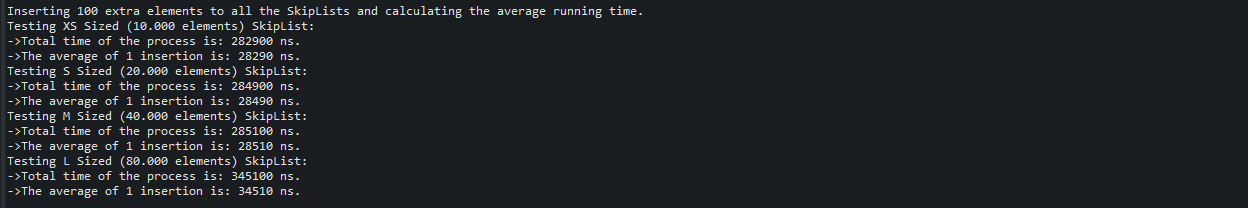
6-)

Testing the running time of 10 instances of XS(10.000) sized SkipList.

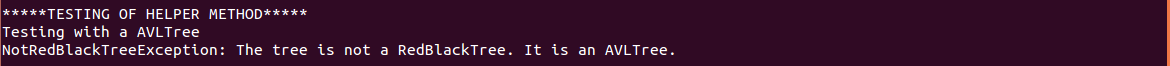
Testing the running time of 10 instances of S(20.000) sized SkipList

Testing the running time of 10 instances of M(40.000) sized SkipList.

Testing the running time of 10 instances of L(80.000) sized SkipList.



7-) Testing AVLOrRedBlack method with an AVLTree.



8-) Testing AVLOrRedBlack method with a RedBlackTree



9-) Testing AVLOrRedBlack method with a regular BinarySearchTree.



**5-) Results Analysis**

**A-) Average Runtime Table Calculated from 10 Runs of Each Occurrence**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BinarySearchTree** | **RedBlackTree** | **TwoThreeTree** | **BTree** | **SkipList** |
| **XS (10.000)** | 22460ns (0.0225ms) | 27860ns (0.0279ms) | 30270ns  (00303ms) | 25960ns  (0.0260ms) | 28290ns  (0.0283ms) |
| **S (20.000)** | 22600ns (0.0226ms) | 29090ns  (0.0290ms) | 29780ns  (0.0298ms) | 25070ns  (0.02507ms) | 28490ns  (0.0285ms) |
| **M (40.000)** | 23350ns (0.0233ms) | 35710ns  (0.0357ms) | 30500ns  (0.0305ms) | 26560ns  (0.2656ms) | 28510ns  (0.0285ms) |
| **L (80.000)** | 24320ns (0.0243ms) | 31610ns  (0.0316ms) | 31800ns  (0.0318ms) | 26170ns  (0.0261ms) | 34510ns  (0.0345ms) |

**B-) Run-Time vs Problem Size Graphs**

**BinarySearchTree RedBlackTree**

**TwoThreeTree BTree**

**SkipList**

**All Data Types Combined**

**C-) Comparison Conclusion**

-The average increase rate of BinarySearchTree is %2.7 when calculated.

-The average increase rate of RedBlackTree is %5.8 when calculated.

-The average increase rate of TwoThreeTree is %1.7 when calculated.

-The average increase rate of BTree is %1 when calculated.

-The average increase rate of SkipList is %6 when calculated.

-According to all the datas the average running time comparison is:

For XS (10.000) size: TwoThreeThree > RedBlackTree > SkipList > BTree > BinarySearchTree

For S (20.000) size: TwoThreeThree > RedBlackTree > SkipList > BTree > BinarySearchTree

For M (40.000) size: RedBlackTree > TwoThreeTree > SkipList > BTree > BinarySearchTree

For L (80.000) size: SkipList > TwoThreeTree > RedBlackTree > BTree > BinarySearchTree

-According to all the datas the average increase rate comparison is:

SkipList > RedBlackTree > BinarySearchTree > TwoThreeTree > BTree

-In general, it is possible to say BTree performed the best with the least increase rate and with the second least average running time. It is possible for BTree to have better performance compared to others with big data insertion. However, for small data insertion BinarySearchTree performed better.

It is also possible to say RedBlackThree performed the worst with the second highest increase rate and with second or third most average running time. However, it is still better than some other data types like SkipList and TwoThreeTree with smaller data insertion.