Jai Shree Ram

>>>>>>>>>>

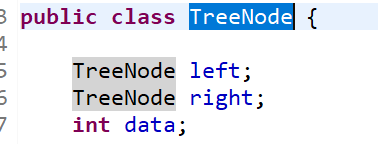
Tree Problems and Solutions

>>>>>

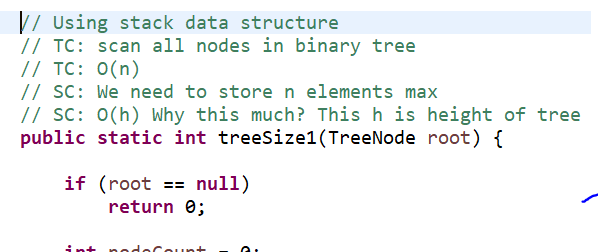
**Problem: Find an efficient algorithm to return the size of binary tree.**

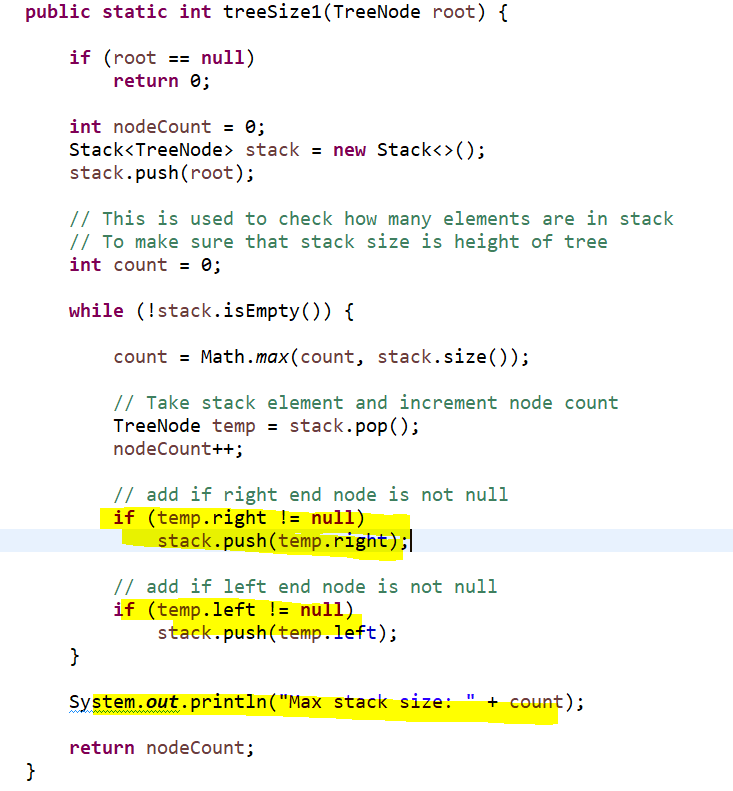
Description: Count number of nodes in a binary tree

Learning: For non-linear problems, think solution using data structure.



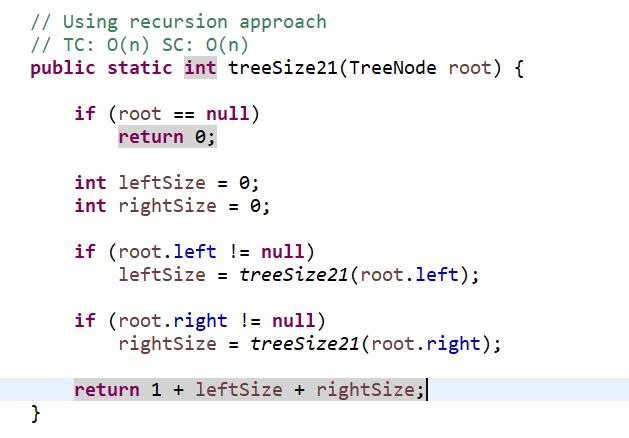
Solution 1: Using Stack as Data Structure





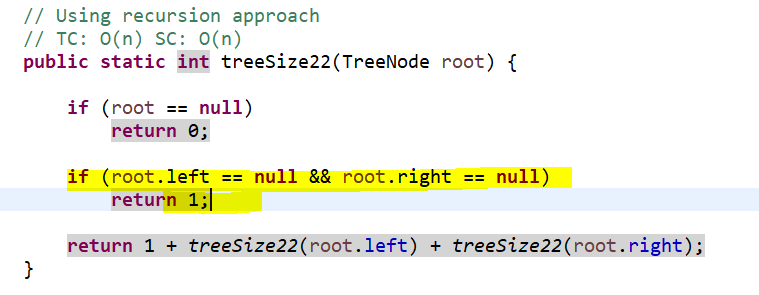
Solution 2: Using Recursion

Left sub tree size and right sub tree will give node count. We will add one and return the result.



In recursion making some optimization to reduce extra call, when root node’s left and right are null.

Approach is same and will give you same result. It will just reduce one extra stack frame.



**Solution 3: Can we solve this problem, but level order node reading?**

We can do level order traversing using queue data structure.

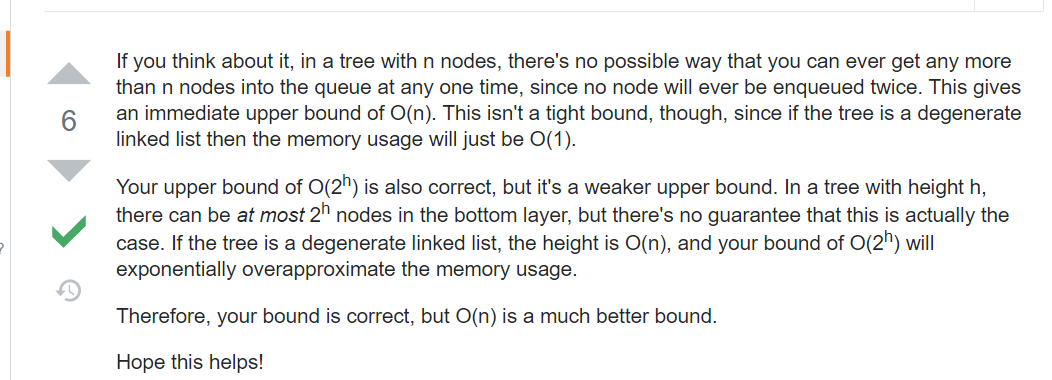
Using Queue as solution for counting number of nodes in a binary tree.

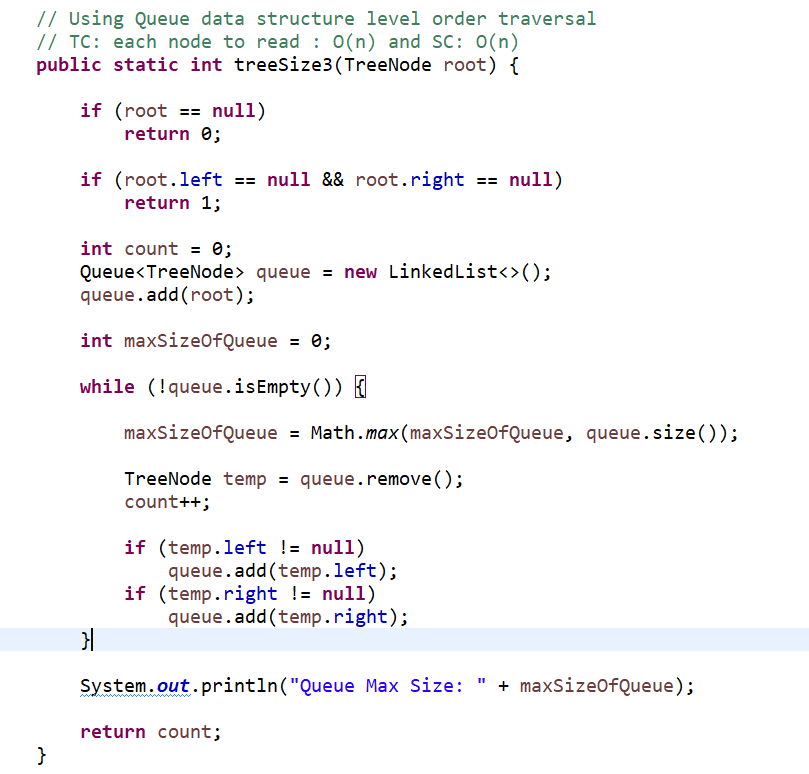
TC: O(n)

SC: O(n)?

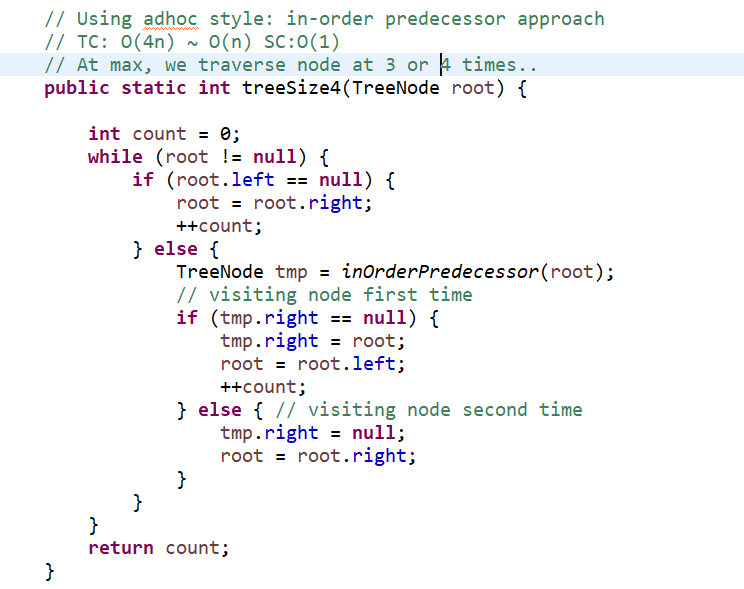
When I tried this program, I found that queue does not add more than height elements in queue. Because if we add more nodes, some nodes definitely get out of queue also.

I found some reasoning about keeping upper bound to O(n) from stack overflow.





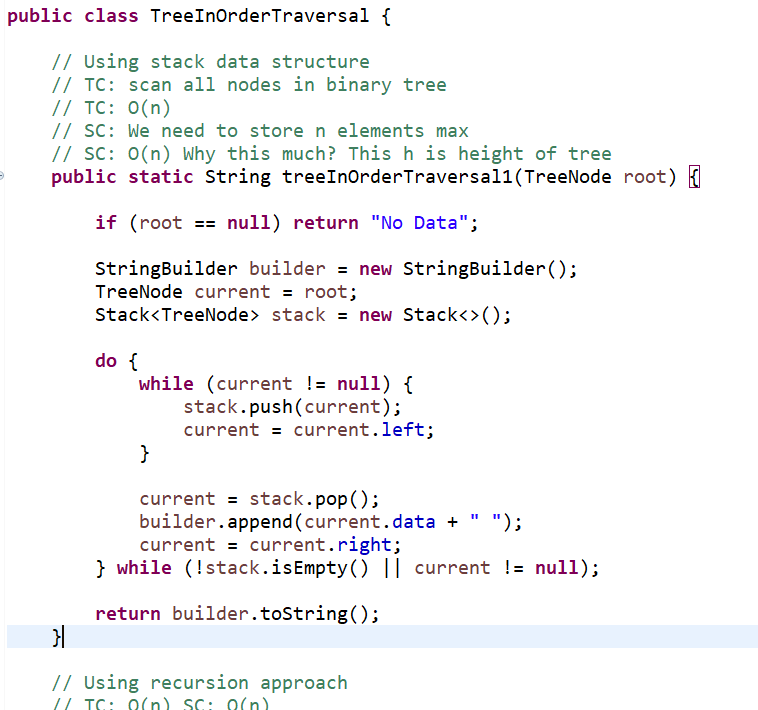
Solution 4: Using adhoc inorder predecessor solution.



>>>>>>>>>>

**Problem Statement: Print tree nodes in in-order.**

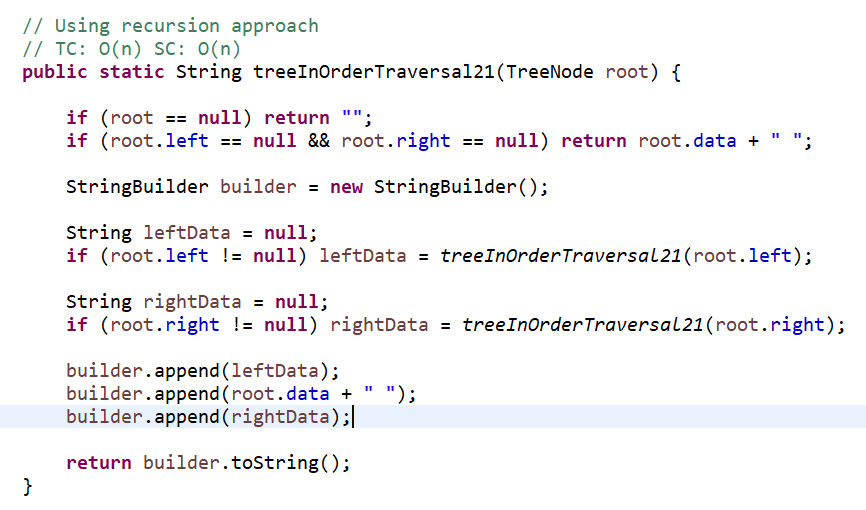
Solution 1. Using stack



Take current pointer and start adding into stack until current.left is not null.

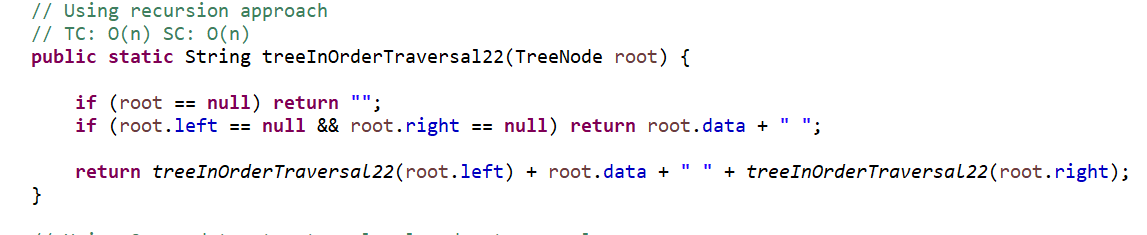
Once current becomes null, then pop the node and print data and move current to right and do the same steps for right side node elements.

Solution 2. Using recursion

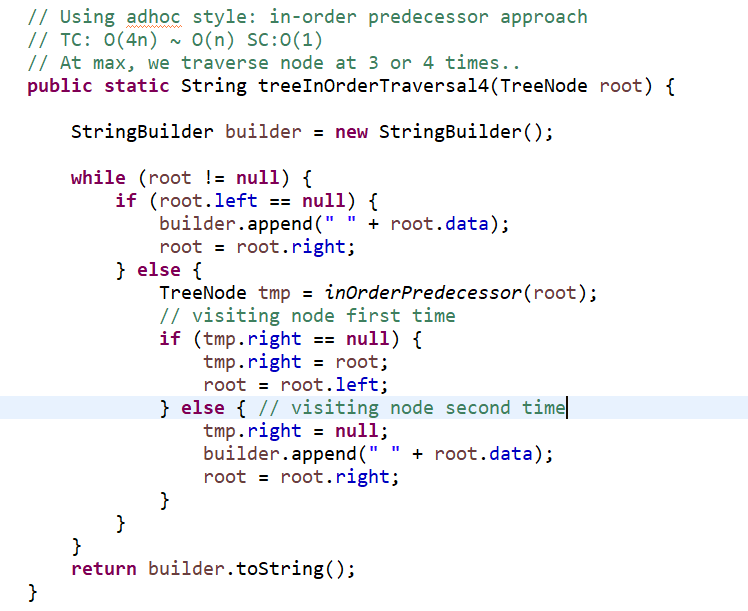


First process left sub tree and then root node and then right sub tree.

Solution 3. Using optimized recursion



Solution 4. Using adhoc in-order predecessor approach



Solution using queue: Can we traverse list in level order using queue and print data in in-order.

For this, we can do augmentation and can try. Or we can keep level and if level is not just lower than previous, we will add that node back to queue and get new node to process. Will have to try.

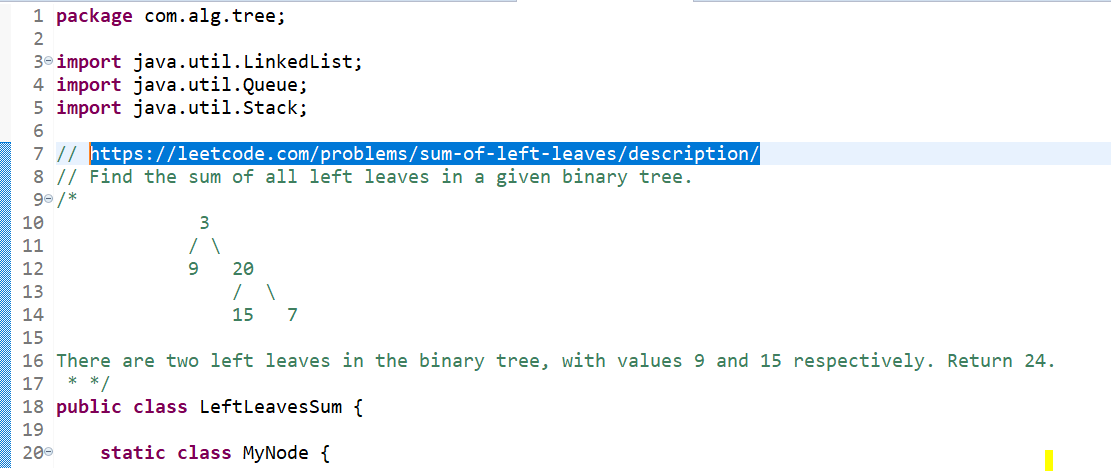
>>>>>>>>>>>>

Similarly, we can try for other post order and pre order traversals.

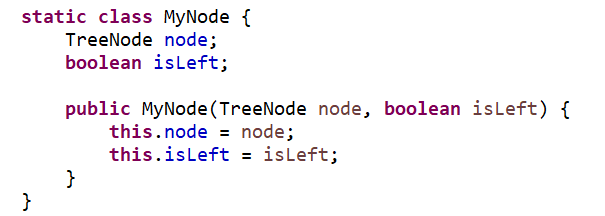
>>>>>>>>>>>>>>

**Problem Statement: Find the sum of all left leaves in a given binary tree.**

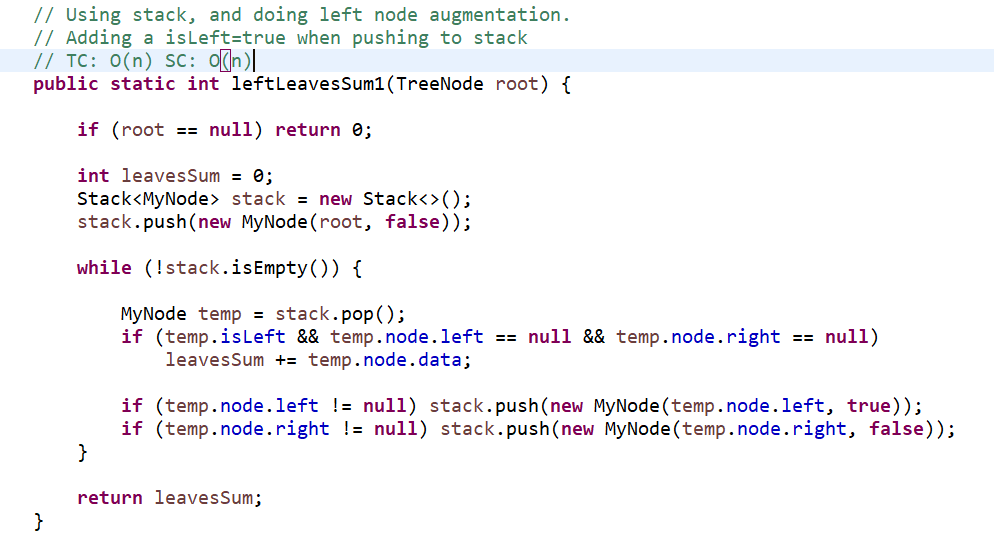
<https://leetcode.com/problems/sum-of-left-leaves/description/>



For augmenting, tree node, added one new class which allows augmentation.

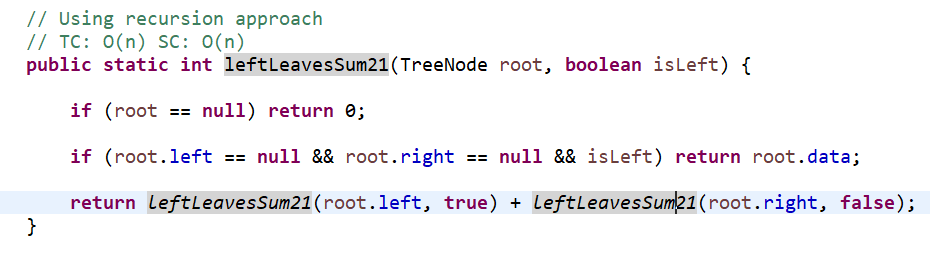


Solution 1: Using stack

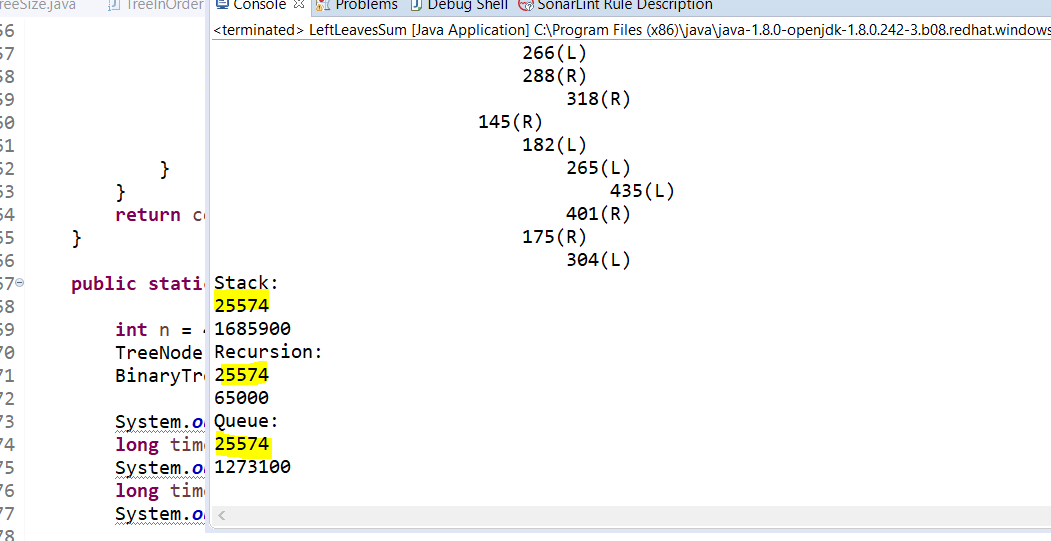


Solution 2: Using Recursion

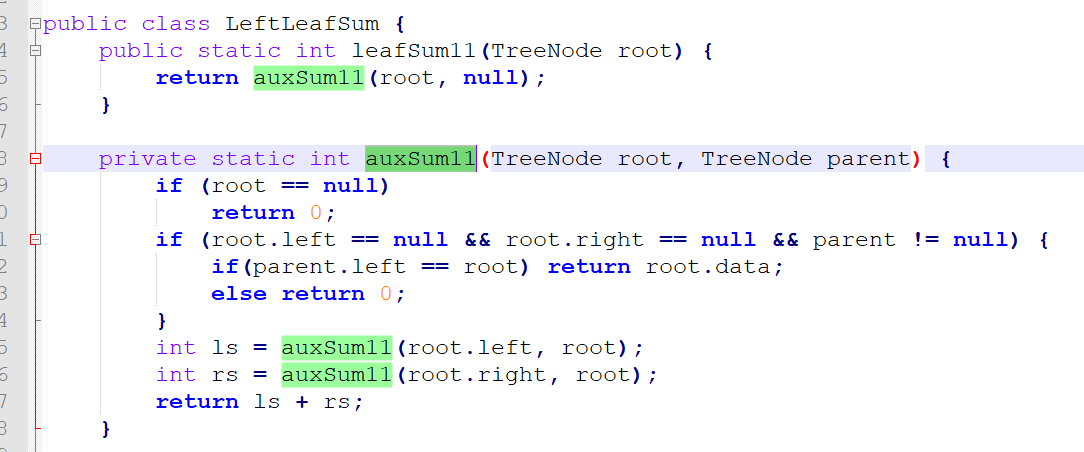
When we use recursion, this problem looks very easy. Get left sub tree left leaves sum and right sub tree’s left leaves sum and return the total.



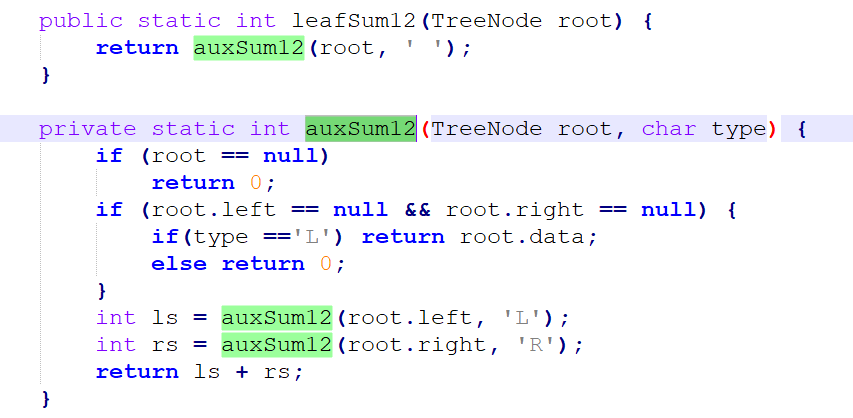
I executed many times and found that recursion is faster than others..

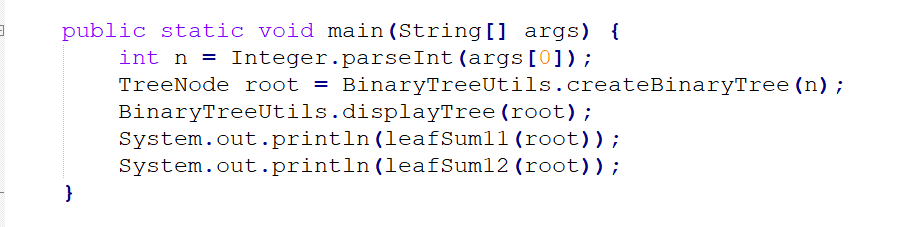


Solution 4: Using adhoc in-order predecessor is possible, when while inserting/creating binary tree, we keep a flag which maintains that whether node is a left or right node. Then we can iterate whole list and if we find a node whose flag is left and left and right are null, then add to sum. This is possible by augmenting the list.

Solution 5: 

Solution 6:

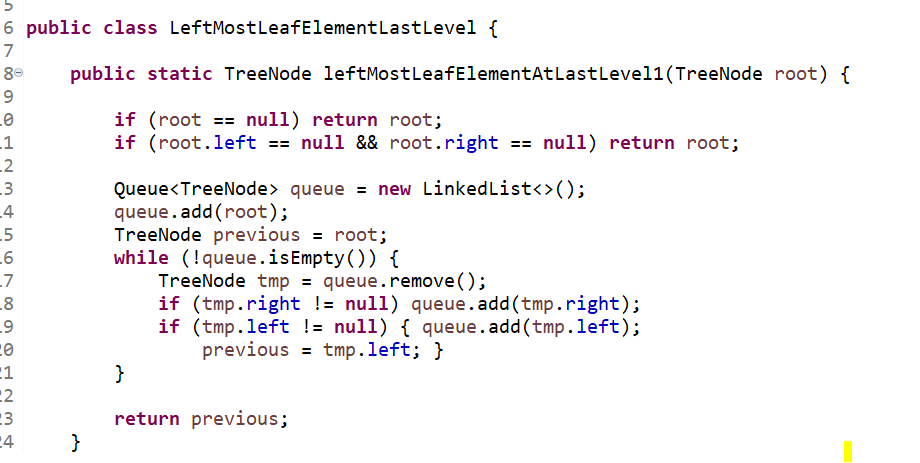




>>>>>>>>>>>>>>

Problem Statement: Find an efficient algorithm which returns most left element of last level.

Solution 1: Using Queue and adding first right node and then left node.



>>>>>>>>>>>>

Problem Statement: Find an efficient algorithm to find an element in given binary tree.

Solution 1. Using stack

TC: O(n) SC: O(n)

Solution 2. Using Queue

TC: O(n) SC: O(n)

Solution 3. Using Recursion and break condition when result found. Worst case, no match found.

TC: O(n) SC: O(n)

Solution 4: Using adhoc in-order predecessor

TC: 4\*n : O(n) SC: O(1)

>>>>>>>>>>>>>>>>>>

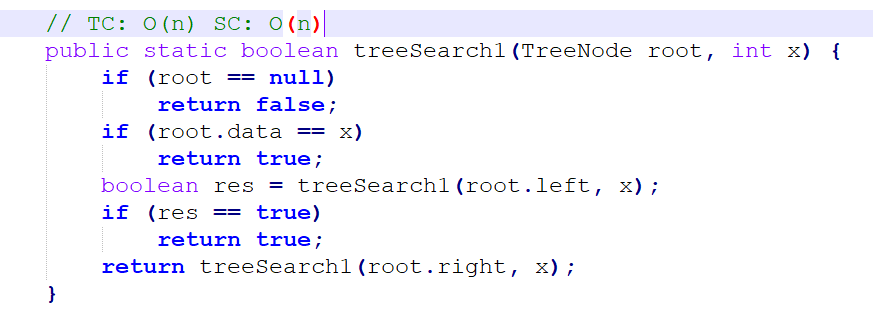
Using NULL marking in queue and that also help in solving couple of problems.

1 NULL 2 3 NULL 4 5 6 NULL etc.

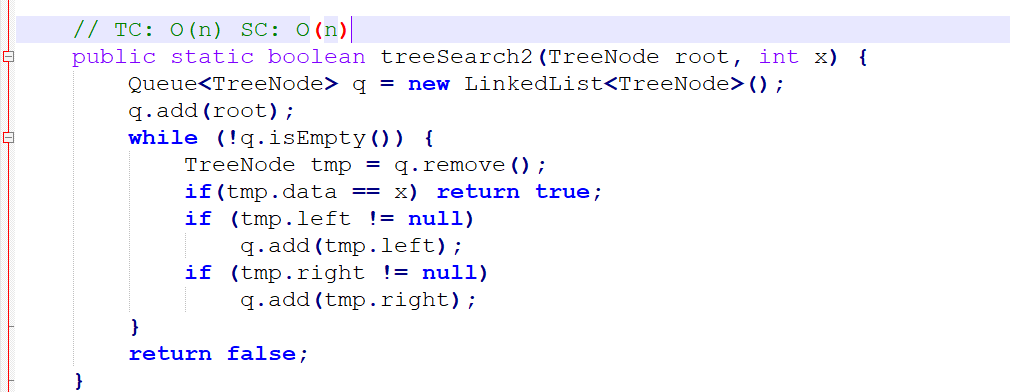
Problem Statement: Find an efficient algorithm to search an element in given binary tree. (Not sorted binary tree)

Return true if element found else false.

Solution 1. Using recursion



Solution 2: Using Queue



Solution 3. Using Stack.TC: O(n) SC: O(n)

Solution 4: Using in-order predecessor Adhoc way

TC: O(n) SC: (1)

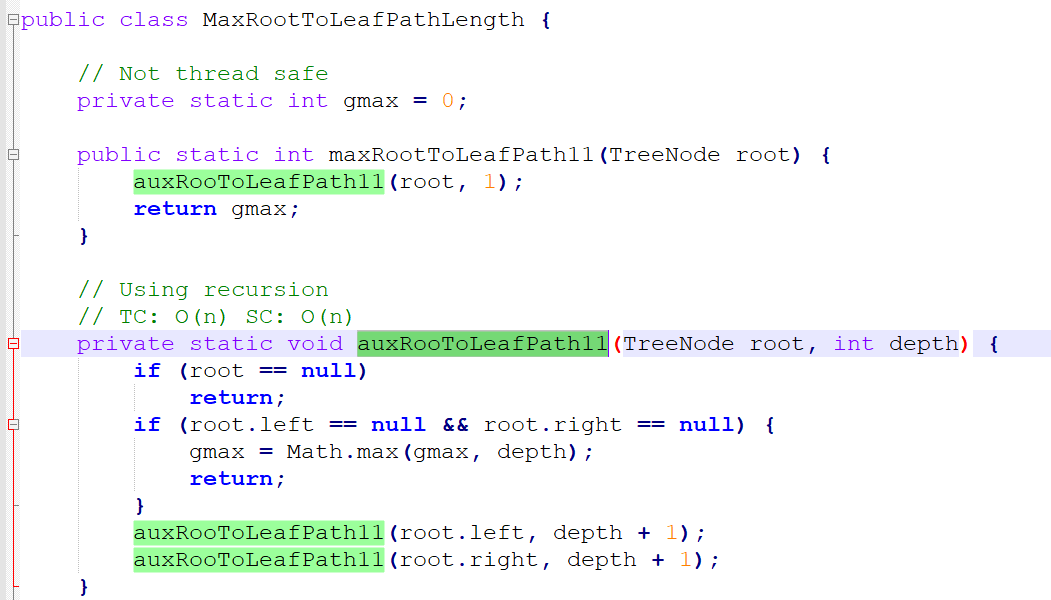
>>>>>>>>>>>>>>>>>>>>>>>>

Tree Path Related Problems

Problem Statement: Find an efficient algorithm that returns longest path from root to leaf in a given binary tree (Not BST, does not matter. Just for clarity)

Solution 1. Using recursion

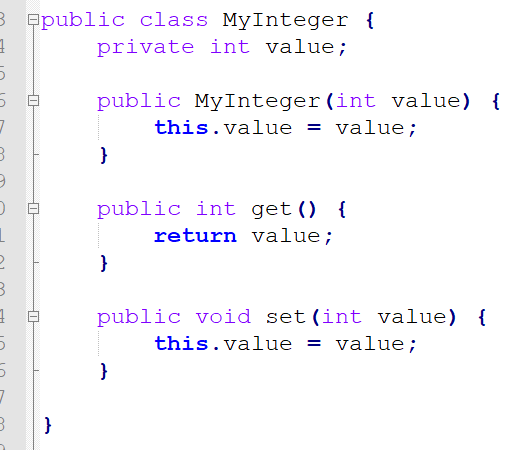
Return Max(Get left sub tree longest path, right sub tree path)

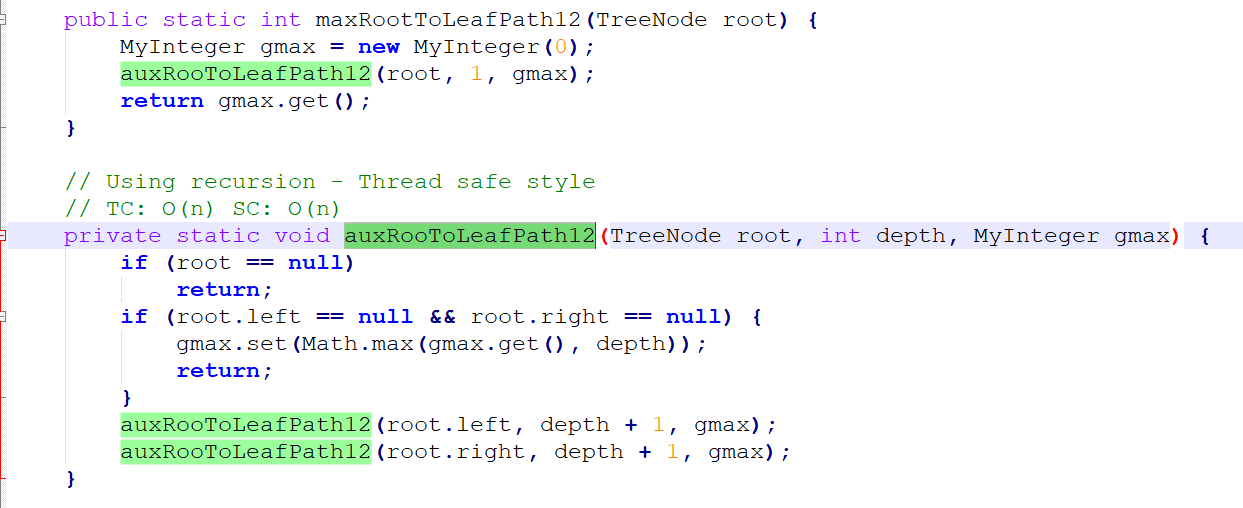


This updated static int field. That is not thread safe, when more than one thread try to access same field.

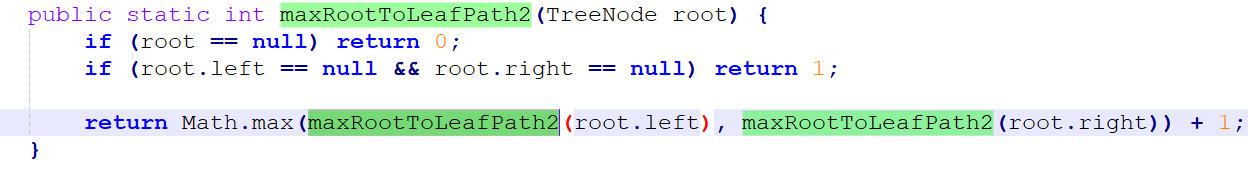
So create one class MyInteger and make this as thread safe.

**Solution 2: Using recursion in thread safe way.**

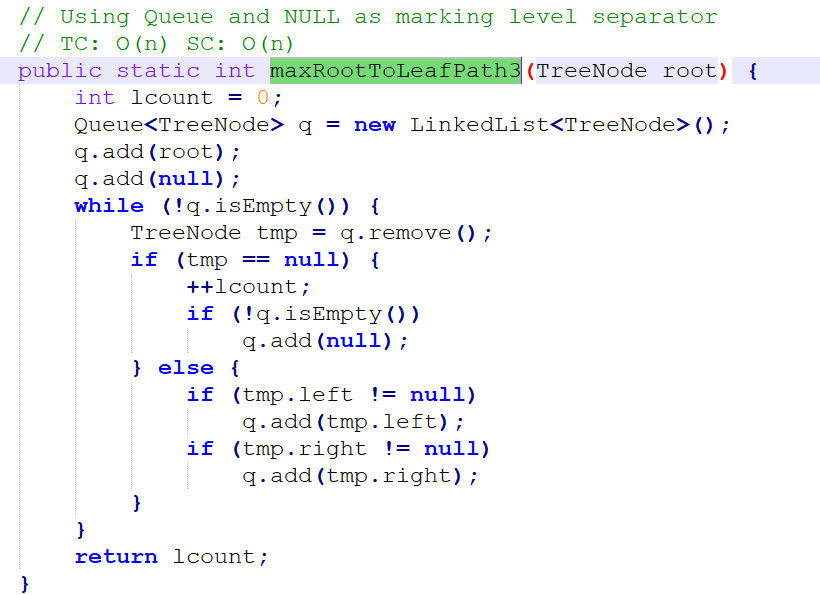




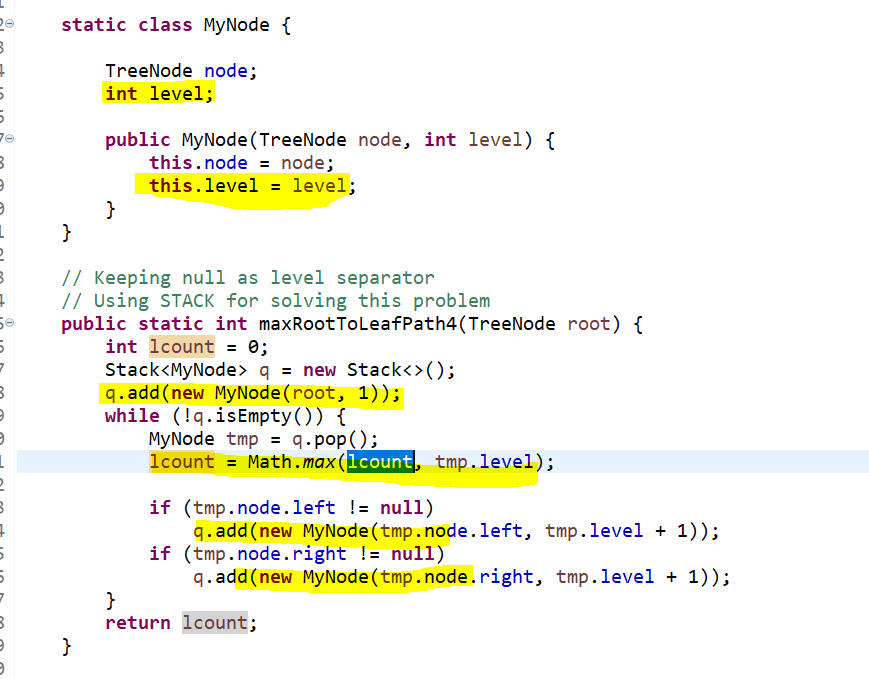
**Solution 3: Using recursion optimized way**



**Solution 4: Using Queue Data Structure and Marking each level with NULL**



**Solution 5: Using Stack with node augmentation. Update the level from previous and iterate all.**



Solution 6: Using in-order predecessor (Adhoc Style)

Will have to try this. Looks we can solve.

>>>>>>>>>>>>>

Problem Statement: Find an efficient algorithm which returns serialized string for given binary tree and we can create binary tree from serialized string (De-serialization into tree)

Formats: JSON, XML, AVRO, String, CSV, PROTOCOL based, etc.

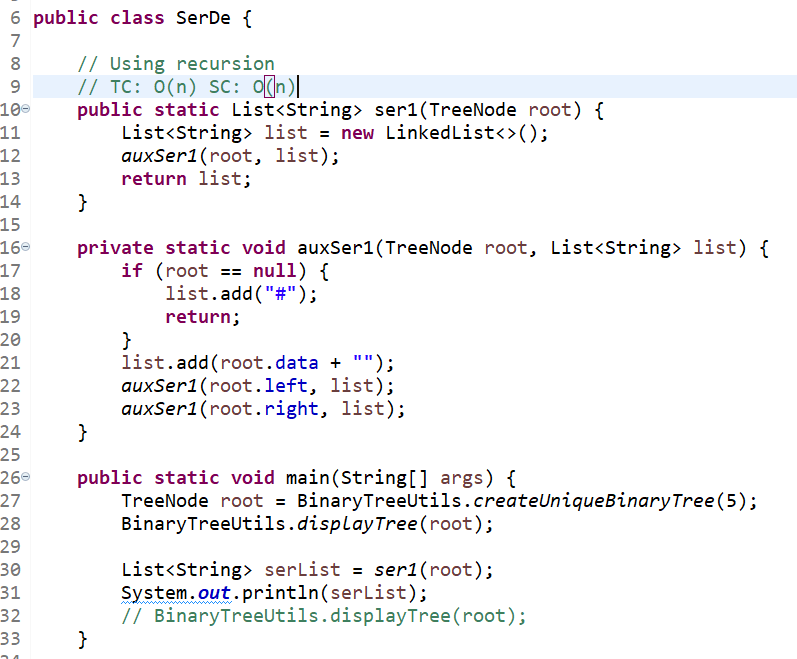
**Solution 1: Using pre-order with # as null marking symbol**

Why Pre-Order? With Pre or Post, we can get the root either in start or in end. But in in-order way, we need to keep extra information that which is root node.

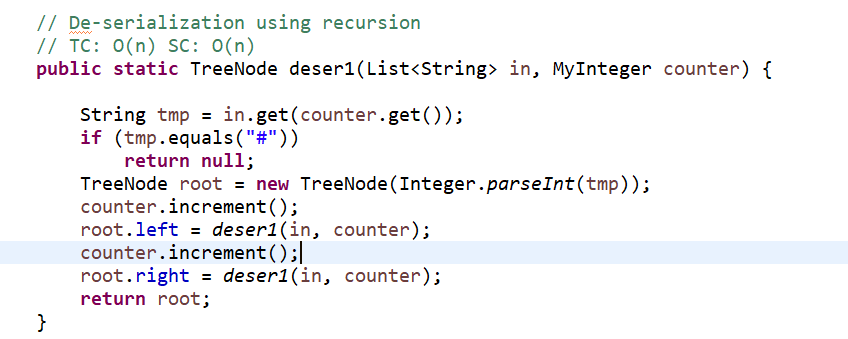
Serialization: TC: O(n) and SC: O(n)

De-Serialization: TC: O(n) and SC: O(n)

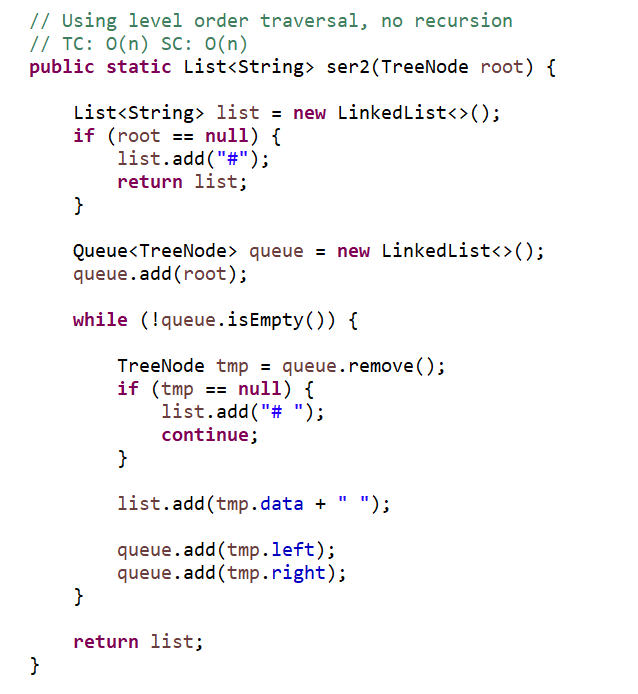
Let’s try with recursion approach first.



**De-serialization using pre-order string into tree back using recursion.**



**Solution 2: Using level order traversing and keeping # as null marking**



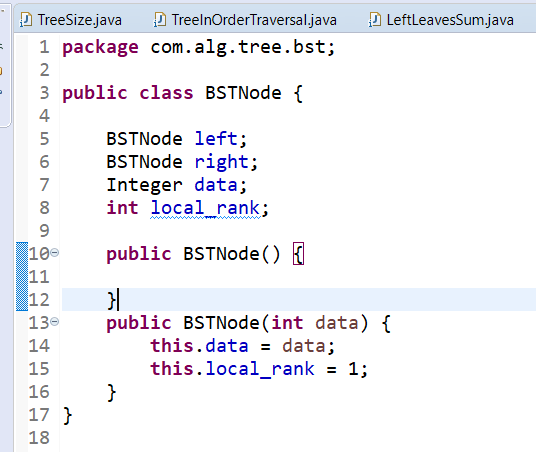
Level Order Tree De-Serialization:

Program is failing Will have to try on this.

>>>>>>>>>>>>

BST (Binary Search Tree) Problems and Solutions

Root – x, Left < x and Right > x



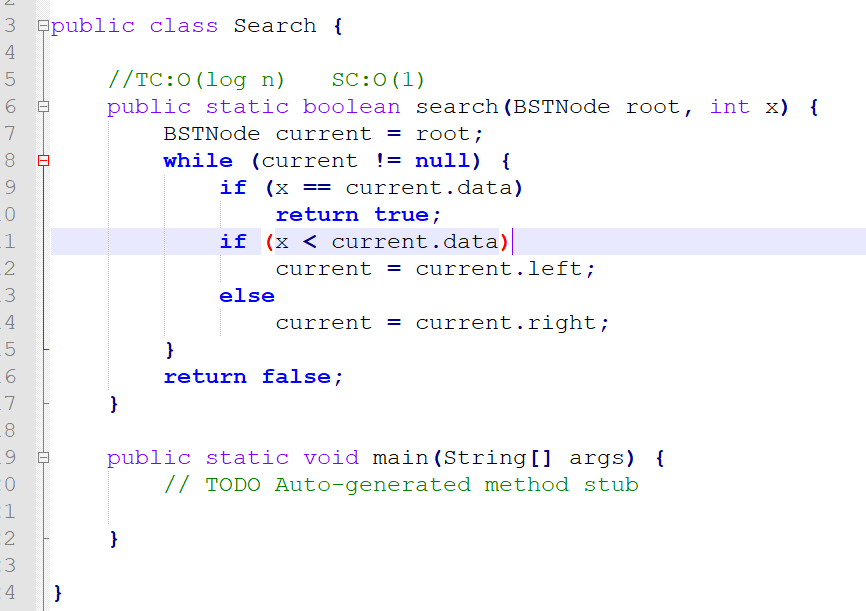
Problem Statement: (BST Search) Find an efficient algorithm to search an element in binary search tree.

Solution 1: Using Level order travel

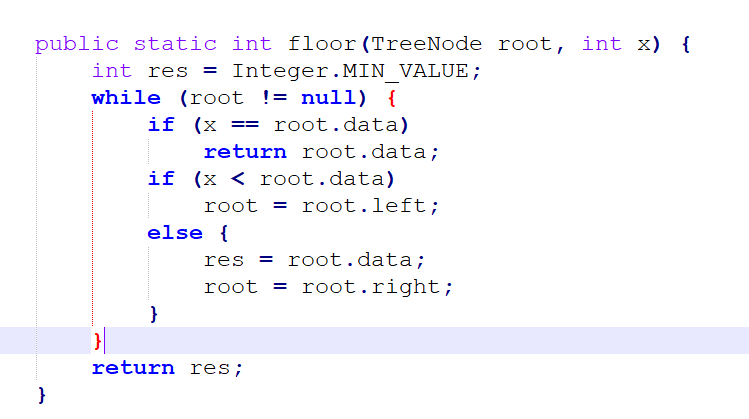
Solution 2: Using Stack (Does not matter)

Solution 3: Using Recursion

Solution 4: Adhoc In-Order Predecessor

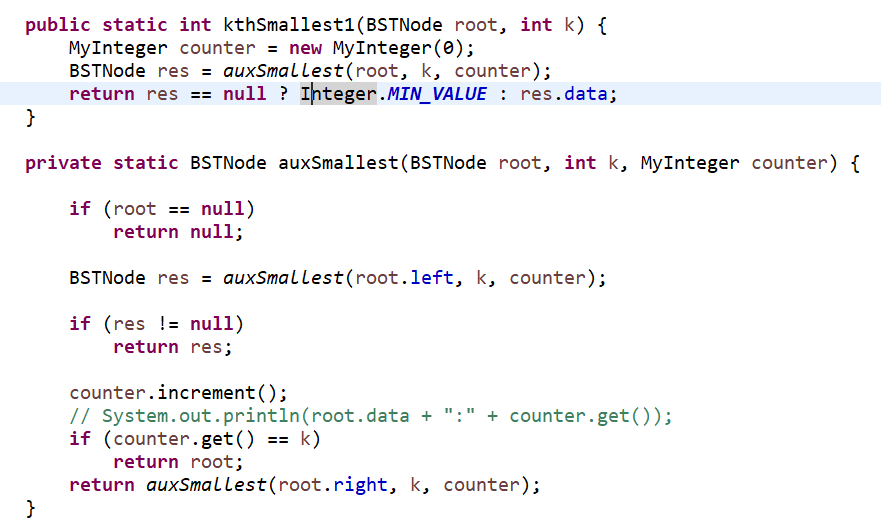


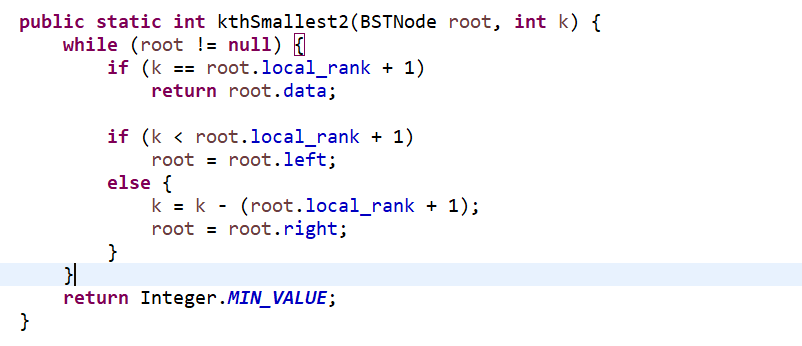
**Problem Statement: Find an efficient algorithm to find floor value of element X in BST.**



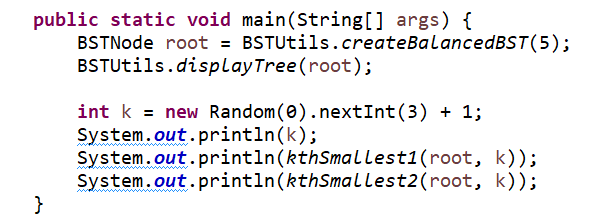
>>>>>>>>>>>>

Problem Statement: Find an efficient algorithm to return Kth smallest element in given BST tree.

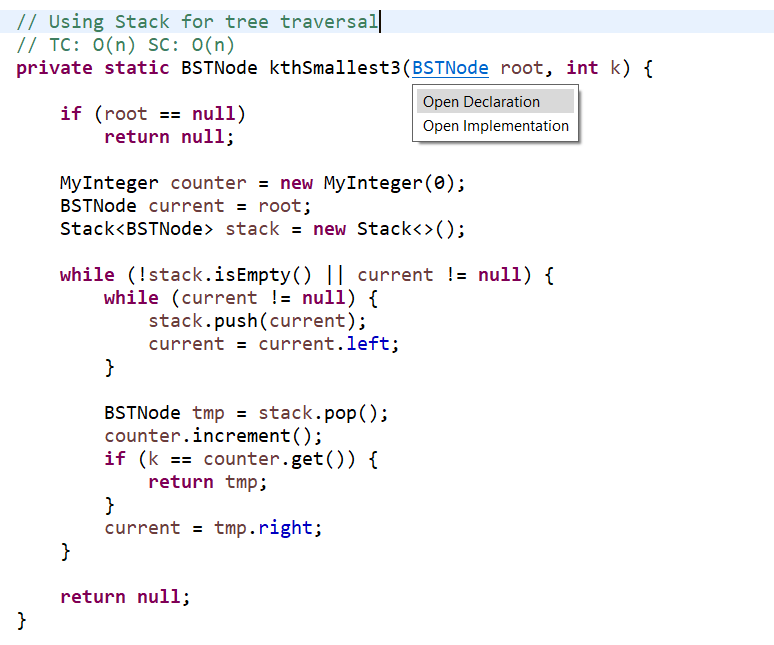




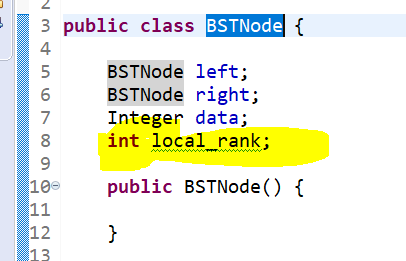
Main Program to test



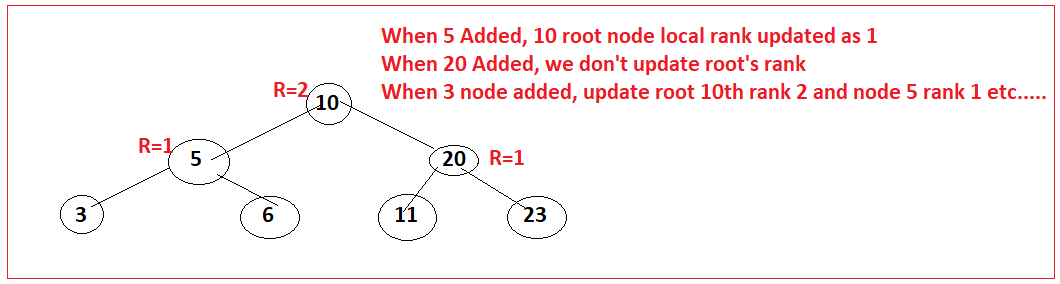
Solution 3: Using Stack for traversing BST and giving the Kth smallest element.



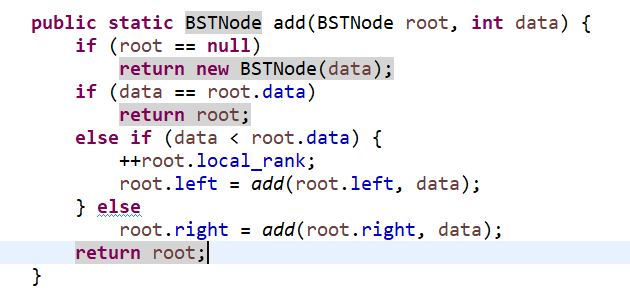
Solution 4: Using level counter. If root element, gets new element in left end, update all nodes level value.



While creating BST tree only, we update the node local rank and keeping updating for left node insertions only.



Add function of BST to update the rank of node.



>>>>>>>>>>>

**Problem Statement:** Find an efficient algorithm to return the minimum cost for merging n given files. At a time, only 2 files allowed to merge.

Example: 5, 3, 6

(5 + 3 = 8 Ops), (8+ 6 = 14 Ops) => 8+ 14 = 22 Ops to merge the files.

(3,6=9 Ops), (9+5= 14 Ops) => 9+14 = 23 Ops

If we learn from this example, we see that we will have to try all the combinations to get the minimum cost of merging the files.

Solution 1: Complete search

TC: O(n^2) and SC: O(1)

Solution 2: If we take minimum numbers and keep taking next minimum and repeat this process, we will get the minimum cost. If we just reverse this and take maximum files and then do, then time will be increased more. So better solution is, take minimum numbers and solve this.

Example:

2,1,4,5

(1,2)=3Ops, (3,4)=7Ops, (7,5)=12 Ops

Total = 22 Ops

For solution 2, we need to sort elements in each step complete.

TC: n.logn + (n-1)Log(n-1) + (n-2)log(n-2) + …..

TC: n^2 \* logn

SC: O(n)

Sorted Data Structure which sort elements on add or remove.

TC: n.logn + (contains switch add) + n.logn

TC: n.logn + 3n + n.logn

TC: n.logn Ops

SC: O(n)

>>>>>>>>>>>>>>>>