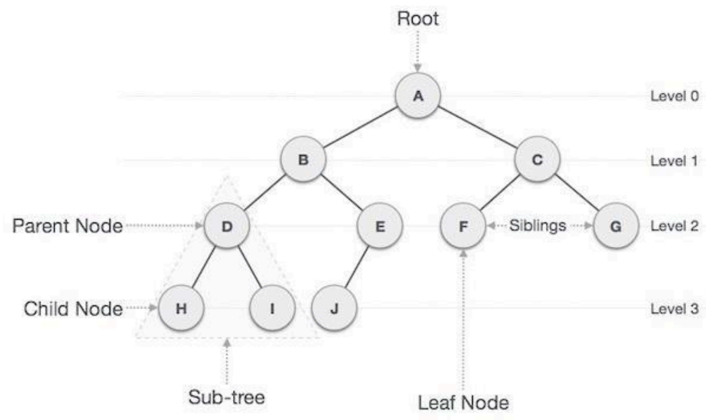
Quiz Section for Program Design (II)

Exercise #8

One said, “Programmers, touch some grass”. So in this exercise, we will implement an abstract data type called a “***Binary Search Tree***”.

First of all, you should know the basics of a ***Tree***. The table below shows some basic names of tree data types, and the graph below is an example.

Name	Tree
`Node`	Data contained in the tree.
`Edge / Link`	Line connects two nodes.
`Child Node`	For a certain node N, the nodes that N can go to. A node can have 0 to infinite child nodes.
`Parent Node`	For a certain node N, the nodes that can reach back to N. A node can only have 1 parent node.
`Root`	The topmost node in the tree, which has no parent node.
`Level`	The distance of a node away from the root.
`Sibling Node`	Nodes in the same level.
`Leaf Node`	Nodes with no child nodes.

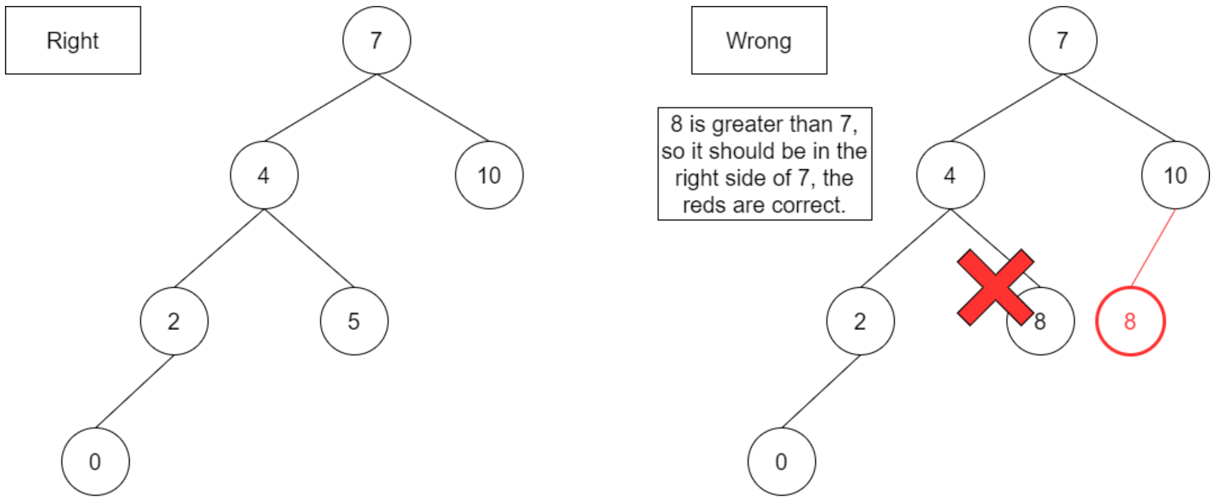


root: A
D's child node: H, I
D's parent node: B
D's sibling node: E
leaf nodes: H, I, J, F, G

The differences between ***Tree*** and ***Binary Search Tree*** are

1. Every node in the binary search tree can only have the utmost 2 child nodes.
2. The value in the left child node must be smaller than its parent node, and the value in the right child node must be greater than the current node (the parent node).

The graph below shows a correct binary search tree and a wrong one.

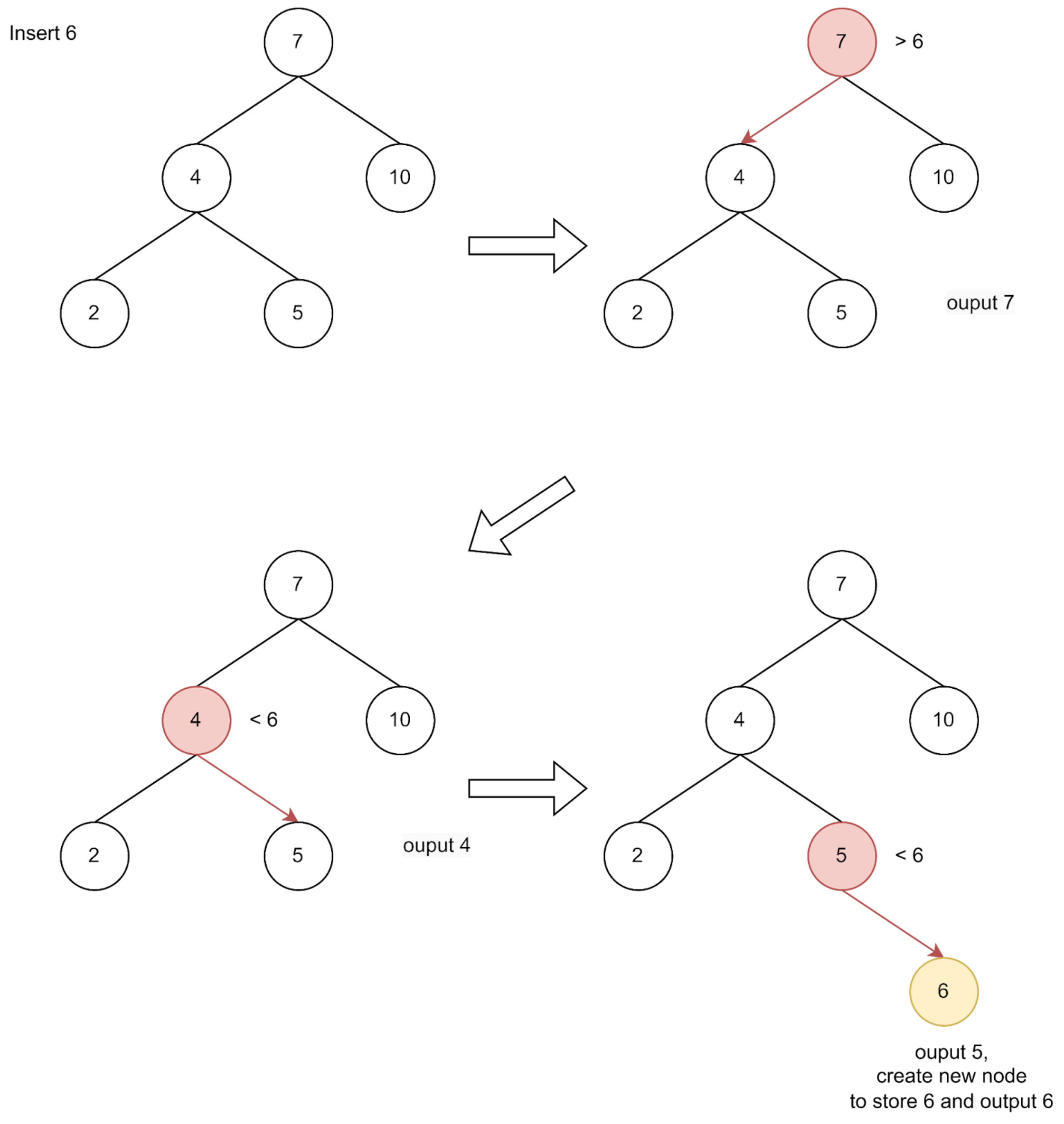

Right
7
4
10
2
5
0
Wrong
8 is greater than 7, so it should be in the right side of 7, the reds are correct.
7
4
10
2
8
8
0

Your task is to complete the sample code that will be uploaded on eCousrse2. You have to finish the functions below.

function	definition
`insert`	Insert new data into ***Binary Search Tree.*** To find the target position where to insert, we start from root and compare the data of root and new data. 1. If the new data is smaller than the data of root, the target position is located in the left child. **a.** If the left child of root is *NULL*, we create a new node for new data and assign it to the left child of root. **b.** Otherwise, if the left child of root isn't *NULL,* we need to do the same process for the left child by recursively calling the insert function or using while-loop. (call the `insert` function to compare the data of the left child and new data). 2. If the new data is greater than the data of root, the target position is located in the right child of root. The rest process is almost the same with 1.

	You should print the data of the node which is compared with new data in order. The figure below shows the example of insert.
`find`	This function is to determine whether the target data exists in the tree. To find the target data, we start from root and compare the data of root and target data. 1. If the target data is equal to the data of root, then we find the target data and return. 2. If the target data is smaller than the data of root, we search the left child of root (recursively call the `find` function) 3. If the target data is greater than the data of root, we search the right child of root (recursively call the `find` function)

The figure below shows how to insert 6 into a tree.



- **Input Format**
  The first line of the input will contain one integer, *numQuery*, which means the number of queries should be processed.
  The following *numQuery* lines describe the queries. Each line describes one query.
  Each line has two integers. The first integer, *OperationType,* represents the type of query. The second integer represents the *Data*.
  If *OperationType* is equal to 1, insert *Data* into the tree and print the node being passed (i.e., insert function).
  If *OperationType* is equal to 2, determine whether *Data* exists in the tree and print the result (i.e., find function).

- **Output Format**
  Refer to the sample output.

- **Technical Specifications**
  - $1 \leq numQuery \leq 2 \times 10^5$
  - $1 \leq Data \leq 10^9$

It is guaranteed that each inserted data is distinct and positive.

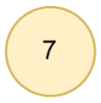
The table below shows the example input and output.

Input	Output

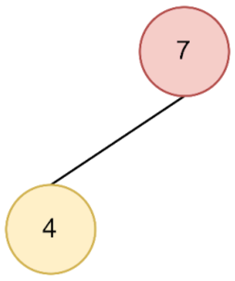
`11`
`1 7`
`1 4`
`2 2`
`2 7`
`1 2`
`1 5`
`2 4`
`1 10`
`2 10`
`2 11`
`1 6`
	`7`
	`7 4`
	`2 is not in tree`
	`7 is in tree`
	`7 4 2`
	`7 4 5`
	`4 is in tree`
	`7 10`
	`10 is in tree`
	`11 is not in tree`
	`7 4 5 6`

Explanation of example

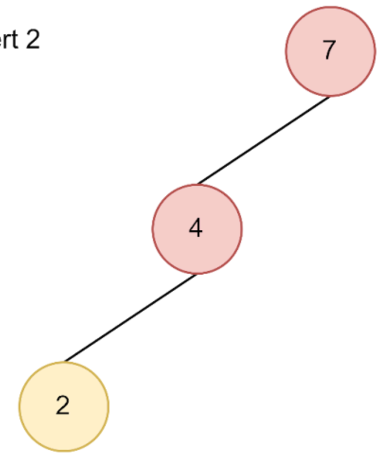
Insert 7



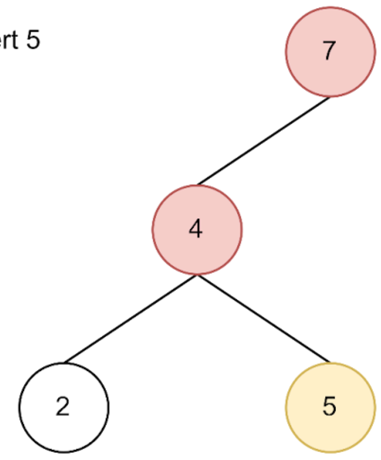
Insert 4



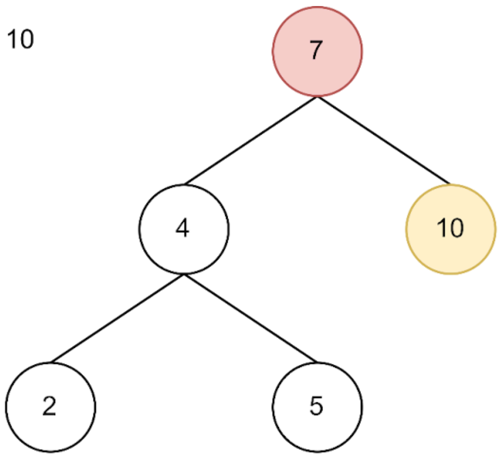
Insert 2



Insert 5



Insert 10



Insert 6