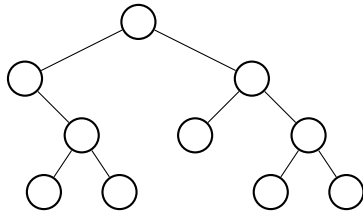


Quiz 3 covers everything up until and including the lecture on Jun 6. To prepare for the quiz, you should review your assignment(s) and the lecture notes. For further practice, we're providing some extra problems below. We're also giving you condensed solutions at the end of this handout. You should attempt these problems prior to looking at the solutions.

1 Structure Of A Binary Search Tree

The tree figure below is a binary search tree, though their keys are not shown.



- (1) Mark the nodes that are the two largest keys. Indicate which is which.
- (2) Mark the nodes that are the two smallest keys. Indicate which is which.

2 Second Key

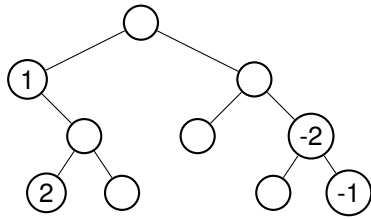
```
class TreeNode {  
    int key;  
    TreeNode left;  
    TreeNode right;  
}
```

Like the tree nodes you have seen, this `TreeNode` class keeps its key in the field `key`, and the left child and right child in `left` and `right`, respectively.

Where's the second smallest key in a binary search tree? You'll answer this question by implementing a function (in Java/pseudo-Java) `int secondKey(TreeNode t)`, where `t` contains at least 2 keys, that returns the second-smallest key in `t`. Write helper functions as necessary. You can assume that there is a `firstKey(t)` function that works perfectly and can refer to it in your code.

3 Condensed Solutions

1. *BST Structure:* The two smallest keys are labeled 1 and 2, and the two largest keys are -2 and -1 (the largest).



2. *Second Key:* Where exactly is the second smallest key in a binary search tree? A moment's thought reveals that it is either **(i)** the parent of the minimum-key node or **(ii)** the minimum key in the right subtree of the minimum-key node. Play with a couple of binary search trees to get a sense of this. Once we know this, it's just a matter of writing a program that walks the tree according to this recipe. Remember that the smallest key can be found by walking left in the tree until it is no longer possible.