Binary Search Tree

Disclaimer: I haven't coded this problem, so not sure the following will work correctly.

Since a connected component of a binary tree is also a binary tree, the only necessary condition for it to be a binary search tree is that the keys should be in increasing order.

We write a method dfs(Node u) that returns the maximum binary search tree rooted at node u. The outline of the algorithm looks like following:

```
Tree dfs(Node u) {
    Tree left = dfs(left_child(u))
    Tree right = dfs(right_child(u))

    return build_result(left, right)
}
```

There are still two things that need to be taken care of:

- How to represent a tree efficiently? Since we know that it is always a binary tree, and only the order is important, you can represent it with an array of integers, containing the keys of the tree.
- How to build result for Node u from result of its left child and right child? To contain node u, we can only use nodes in left subtree having keys less than node u. Similarly, we can only use nodes in right subtree that have keys greater than node u. So, from the result of left subtree, we remove all nodes that have keys greater or equal to node u. Similar for right subtree, and then we can combine these keys to get the result for node u.

Now, we have something that can run in $O(N^2)$. To optimize it to $O(N * logN)$, you can do the following:

When you get the 2 arrays from left subtree and right subtree, reuse the bigger one (adding necessary values to the other one, and return that array)

Code from team ThanQ+: here.