Maximum Binary Tree

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

1. The root is the maximum number in the array.
2. The left subtree is the maximum tree constructed from left part subarray divided by the maximum number.
3. The right subtree is the maximum tree constructed from right part subarray divided by the maximum number.

Construct the maximum tree by the given array and output the root node of this tree.

Example 1:

```
Input: [3,2,1,6,0,5]
Output: return the tree root node representing the following tree:

      6
    /   \
   3     5
    \    /
     2  0
       \
        1
```

Note:

1. The size of the given array will be in the range [1,1000].

Solution 1

This solution is inspired by @votrubac
Here is his brilliant solution
https://discuss.leetcode.com/topic/98454/c-9-lines-o-n-log-n-map

The key idea is:

1. We scan numbers from left to right, build the tree one node by one step;
2. We use a stack to keep some (not all) tree nodes and ensure a decreasing order;
3. For each number, we keep pop the stack until empty or a bigger number; The bigger number (if exist, it will be still in stack) is current number's root, and the last popped number (if exist) is current number's right child (temporarily, this relationship may change in the future); Then we push current number into the stack.

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* constructMaximumBinaryTree(vector<int>& nums) {
        vector<TreeNode*> stk;
        for (int i = 0; i < nums.size(); ++i)
        {
            TreeNode* cur = new TreeNode(nums[i]);
            while (!stk.empty() && stk.back()->val < nums[i])
            {
                cur->left = stk.back();
                stk.pop_back();
            }
            if (!stk.empty())
                stk.back()->right = cur;
            stk.push_back(cur);
        }
        return stk.front();
    }
};
```

This solution will be slightly faster than normal recursive solution.

Again, great thanks to @votrubac !!!

written by Mrsuyi original link here

Solution 2

A typical recursion problem.

```
public class Solution {
    public TreeNode constructMaximumBinaryTree(int[] nums) {
        if (nums == null) return null;
        return build(nums, 0, nums.length - 1);
    }

    private TreeNode build(int[] nums, int start, int end) {
        if (start > end) return null;

        int idxMax = start;
        for (int i = start + 1; i <= end; i++) {
            if (nums[i] > nums[idxMax]) {
                idxMax = i;
            }
        }

        TreeNode root = new TreeNode(nums[idxMax]);

        root.left = build(nums, start, idxMax - 1);
        root.right = build(nums, idxMax + 1, end);

        return root;
    }
}
```

written by shawngao original link here

Solution 3

```
def constructMaximumBinaryTree(self, nums):
    dummy = TreeNode(None)
    def d(root, nums):
        if not nums:
            return
        i = nums.index(max(nums))
        root.val = max(nums)
        if nums[:i]:
            root.left = TreeNode(None)
            d(root.left, nums[:i])
        if nums[i+1:]:
            root.right = TreeNode(None)
            d(root.right, nums[i+1:])
    d(dummy, nums)
    return dummy
```

written by TianQ original link here