# 题目

给定一个不含重复元素的整数数组nums。一个以此数组直接递归构建的 最大二叉树 定义如下：

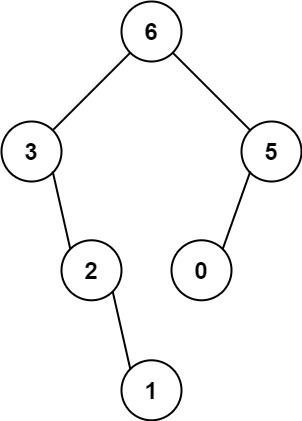
二叉树的根是数组nums中的最大元素。

左子树是通过数组中 最大值左边部分 递归构造出的最大二叉树。

右子树是通过数组中 最大值右边部分 递归构造出的最大二叉树。

返回有给定数组nums构建的最大二叉树 。

示例 1：



输入：nums = [3,2,1,6,0,5]

输出：[6,3,5,null,2,0,null,null,1]

解释：递归调用如下所示：

- [3,2,1,6,0,5] 中的最大值是 6 ，左边部分是 [3,2,1] ，右边部分是 [0,5] 。

- [3,2,1] 中的最大值是 3 ，左边部分是 [] ，右边部分是 [2,1] 。

- 空数组，无子节点。

- [2,1] 中的最大值是 2 ，左边部分是 [] ，右边部分是 [1] 。

- 空数组，无子节点。

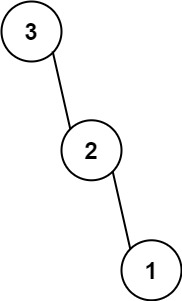
- 只有一个元素，所以子节点是一个值为 1 的节点。

- [0,5] 中的最大值是 5 ，左边部分是 [0] ，右边部分是 [] 。

- 只有一个元素，所以子节点是一个值为 0 的节点。

- 空数组，无子节点。

示例 2：



输入：nums = [3,2,1]

输出：[3,null,2,null,1]

提示：

1 <= nums.length <= 1000

0 <= nums[i] <= 1000

nums 中的所有整数互不相同

# 分析

## 方法一：递归法

本方法非常简单。创建方法construct(nums, l, r)，用于找出在数组nums中从l到r索引（不包含第r个位置）中最大二叉树的根节点。

算法步骤如下：

1. 首先调用construct(nums, 0, n)，其中n是数组nums的长度。
2. 在索引范围(l:r−1)内找到最大值的索引，将nums[max\_i]作为根节点。
3. 调用construct(nums, l, max\_i)创建根节点的左孩子。递归执行此操作，创建根节点的整个左子树。
4. 类似的，调用construct(nums, max\_i + 1, r)创建根节点的右子树。
5. 方法construct(nums, 0, n)返回最大二叉树的根节点。

代码：

public class Solution {

public TreeNode constructMaximumBinaryTree(int[] nums) {

return construct(nums, 0, nums.length);

}

public TreeNode construct(int[] nums, int l, int r) {

if (l == r)

return null;

int max\_i = max(nums, l, r);

TreeNode root = new TreeNode(nums[max\_i]);

root.left = construct(nums, l, max\_i);

root.right = construct(nums, max\_i + 1, r);

return root;

}

public int max(int[] nums, int l, int r) {

int max\_i = l;

for (int i = l; i < r; i++) {

if (nums[max\_i] < nums[i])

max\_i = i;

}

return max\_i;

}

}

复杂度分析

时间复杂度：O(n^2)。方法construct一共被调用n次。每次递归寻找根节点时，需要遍历当前索引范围内所有元素找出最大值。一般情况下，每次遍历的复杂度为O(logn)，总复杂度为O(nlogn)。最坏的情况下，数组nums有序，总的复杂度为O(n^2)。

空间复杂度：O(n)。递归调用深度为n。平均情况下，长度为n的数组递归调用深度为O(logn)。

另一种写法：

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

    TreeNode\* constructMaximumBinaryTree(vector<int>& nums) {

        return dfs(nums,0,nums.size()-1);

    }

    TreeNode \*dfs(vector<int> &nums,int left,int right)

    {

        if(left>right)  return nullptr;

        int idx = left;

        for(int i=left+1;i<=right;i++)

        {

            if(nums[i] > nums[idx])

                idx = i;

        }

        TreeNode \*root = new TreeNode(nums[idx]);

        root->left = dfs(nums,left,idx-1);

        root->right = dfs(nums,idx+1,right);

        return root;

    }

};