下一个排列

实现获取 **下一个排列** 的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列（即，组合出下一个更大的整数）。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须 **原地** 修改，只允许使用额外常数空间。

示例 1：

输入： `nums = [1,2,3]`

输出： `[1,3,2]`

示例 2：

输入： `nums = [3,2,1]`

输出： `[1,2,3]`

示例 3：

输入： `nums = [1,1,5]`

输出： `[1,5,1]`

示例 4：

输入： `nums = [1]`

输出： `[1]`

思路及解法

注意到下一个排列总是比当前排列要大，除非该排列已经是最大的排列。我们希望找到一种方法，能够找到一个大于当前序列的新序列，且变大的幅度尽可能小。具体地：

我们需要将一个左边的「较小数」与一个右边的「较大数」交换，以能够让当前排列变大，从而得到下一个排列。

同时我们要让这个「较小数」尽量靠右，而「较大数」尽可能小。当交换完成后，「较大数」右边的数需要按照升序重新排列。这样可以在保证新排列大于原来排列的情况下，使变大的幅度尽可能小。

```
class Solution {
public:
```

```
    void nextPermutation(vector<int>& nums) {
        int i=nums.size()-2;
        while(i>=0&&nums[i]>=nums[i+1])
        {
            i--;
        }
        if(i>=0)
        {
            int j=nums.size()-1;
            while(j>=0&&nums[i]>=nums[j])
            {
                j--;
            }
            swap(nums[i],nums[j]);
        }
        reverse(nums.begin()+i+1,nums.end());
    }
};
```