283. Move Zeroes

给定一个数组 nums，编写一个函数将所有 0 移动到数组的末尾，同时保持非零元素的相对顺序。
示例:
输入: [0,1,0,3,12]
输出: [1,3,12,0,0]
说明:
必须在原数组上操作，不能拷贝额外的数组。
尽量减少操作次数。

Analysis

- **暴力枚举，双循环，满足条件交换数据(O(nlongn))，效率不高**

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        for (int i = 0; i < (int)nums.size() - 1; i++) {
            if (nums[i] == 0) {
                for(int j = i + 1; j < (int)nums.size(); j++) {
                    if(nums[j] != 0) {
                        swap(nums[i++], nums[j]);
                    }
                }
            }
        }
    }
};
```

- **两次循环O(n)**
- 第一次循环把非0的数放到指定位置，循环完成则记录下了最后一个非0数的后面一个位置
- 第二次循环是把之前记录的位置开始到数组结束的所有位置都置为0

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int pos = 0;
        for (int i = 0; i < (int)nums.size(); i++) {
            if (nums[i] != 0) {
                nums[pos++] = nums[i];
            }
        }
        for(;pos < nums.size(); pos++) {
```

```
            nums[pos] = 0;
        }
    }
};
```

- **单循环，遇见非0就交换(O(n)) – 推荐**
  - 最差的情况时，如果全是非0的，每一个位置都会重新写一下自己位置的值

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int pos = 0;
        for (int i = 0; i < (int)nums.size(); i++) {
            if (nums[i] != 0) {
                swap(nums[pos++], nums[i]);
            }
        }
    }
};
```

- **使用stl里的stable_partition和自定义的compare函数 O(n) – 推荐**
  - 自定义一个compare函数来判断是否非0
  - stable_partition会根据传入compare函数数组划分成2组，非组0在前，0组在后，并且保证每个组里的数的相对位置不变
  - 如果使用partition的话，也是非组0在前，0组在后，但每组中的数的相对位置不保证和原来的一样。

```
class Solution {
public:
    static bool compareNonZero(int x){
        return (x !=0);
    }
    void moveZeroes(vector<int>& nums) {
        stable_partition(nums.begin(), nums.end(), compareNonZero);
    }
};
```

- **使用stl里的stable_partition和lambda表达式 O(n) – 推荐**
  - 原理同上，只是比较函数直接使用lambda表达式替代了自定义的compare函数。

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        stable_partition(nums.begin(), nums.end(), [](int x)
{return (x != 0);});
    }
};
```

- **使用stl里的stable_partition和logical_not O(n)**
  - 原理同上，只是比较函数直接使用stl里的ogical_not表达式来判断是否是非0。
  - 但这时需要反向遍历数组，因为logical_not是为0返回真，不为0返回假。

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        stable_partition(nums.rbegin(), nums.rend(), logical_not<int>());
    }
};
```

Code

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
// double loop with swap O(nlongn)
// void moveZeroes(vector<int>& nums) {
//     for (int i = 0; i < (int)nums.size() - 1; i++) {
//         if (nums[i] == 0) {
//             for(int j = i + 1; j < (int)nums.size(); j++) {
//                 if(nums[j] != 0) {
//                     swap(nums[i++], nums[j]);
//                 }
//             }
//         }
//     }
// }
// first pos that should be set to 0 O(n)
// void moveZeroes(vector<int>& nums) {
//     int pos = 0;
//     for (int i = 0; i < (int)nums.size(); i++) {
//         if (nums[i] != 0) {
//             nums[pos++] = nums[i];
//         }
//     }
//     for(;pos < nums.size(); pos++) {
//         nums[pos] = 0;
//     }
// }
//last non-zero position with swap O(n)
// void moveZeroes(vector<int>& nums) {
//     int pos = 0;
//     for (int i = 0; i < (int)nums.size(); i++) {
//         if (nums[i] != 0 && i != pos) {
//             swap(nums[pos++], nums[i]);
//         }
//     }
// }
//using stl-stable_partition & compare function customsied O(n)
// bool compareNonZero(int x){
```

```
//     return (x !=0);
// }
// void moveZeroes(vector<int>& nums) {
//     stable_partition(nums.begin(), nums.end(), compareNonZero);
// }
//using stl-stable_partition & lambda O(n)
// void moveZeroes(vector<int>& nums) {
//     stable_partition(nums.begin(), nums.end(), [](int x)
{return (x != 0);});
// }
//
//using stl-stable_partition and logical_not during reverse traversal
void moveZeroes(vector<int>& nums) {
    stable_partition(nums.rbegin(), nums.rend(), logical_not<int>());
}
int main()
{
    int nums[] = {0,1,0,3,12};
    vector<int> array(nums, nums + 5);
    moveZeroes(array);
    //print
    std::cout << "[" ;
    for (int i = 0; i < (int)array.size(); i++) {
        std::cout << array[i];
        if (i != (int)array.size() - 1) {
            std::cout << ",";
        }
    }
    std::cout << "]" <<std::endl;
    return 0;
}
```