/\*

Given an array of integers sorted in ascending order, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example,

Given [5, 7, 7, 8, 8, 10] and target value 8,

return [3, 4].

要求算法复杂度为O(log n)

way-1 从两边往中间夹，感觉会超过O(log n)

way-2 二分查找

写法非常巧妙，

先找满足的第一个的前一个，然后mid+1就到了第一个

先找满足的最后一个的后一个，然后mid-1就到最后一个

\*/

class Solution {

public:

int binarySearchLow(vector<int>& nums, int target, int begin, int end)

{

//先找满足的第一个的前一个，然后mid+1就到了第一个

if(begin > end)

return begin;

int mid = begin + (end - begin) / 2;

if(nums[mid] < target)

return binarySearchLow(nums, target, mid + 1, end);

else

return binarySearchLow(nums, target, begin, mid - 1);

}

int binarySearchUp(vector<int>& nums, int target, int begin, int end)

{

//先找满足的最后一个的后一个，然后mid-1就到最后一个

if(begin > end)

return end;

int mid = begin + (end - begin) / 2;

if(nums[mid] > target)

return binarySearchUp(nums, target, begin, mid - 1);

else

return binarySearchUp(nums, target, mid + 1, end);

}

vector<int> searchRange(vector<int>& nums, int target)

{

//way-1

/\*

vector<int> result(2,-1);

int left=0;

int right=nums.size()-1;

bool ldaowei=0;

bool rdaowei=0;

while(left<=right)

{

if(!ldaowei)

{

if(nums[left]<target)

left++;

else if(nums[left]==target)

ldaowei=1;

else

break;

}

if(!rdaowei)

{

if(nums[right]>target)

right--;

else if(nums[right]==target)

rdaowei=1;

else

break;

}

if(ldaowei && rdaowei)

{

result[0]=left;

result[1]=right;

break;

}

}

return result;

\*/

//way-2

vector<int> ret(2, -1);

if(nums.empty())

return ret;

int high = binarySearchUp(nums, target, 0, nums.size() - 1);

int low = binarySearchLow(nums, target, 0, nums.size() - 1);

if(high >= low)

{

ret[0] = low;

ret[1] = high;

}

return ret;

}

};