区间查找LeetCode\_34. 34. Find First and Last Position of Element in Sorted Array

# 题目：

**34. Find First and Last Position of Element in Sorted Array**

**难度：Medium**

Given an array of integers nums 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]**.

**Example 1:**

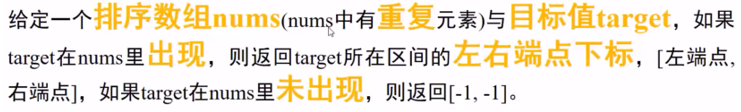
Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

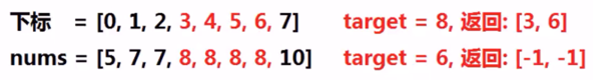
**Example 2:**

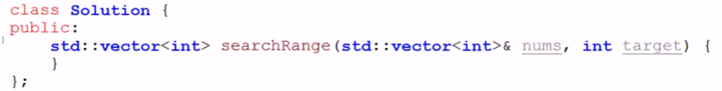
Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]



示例：





# 思路分析

需要分别找出左端点和右端点，最简单的方法是先找左端点，然后寻找右端点。

(当然也存在一起寻找的优秀算法，由于时间关系，这里不深入研究，见LeetCode别人提交的代码)

在二分查找的基础上修改：当target不等于(大于或小于)nums[midIndex]时，需要按照二分查找原则继续查找，直到找到target==nums[midIndex];当target==nums[midIndex]，需要根据查找左端点还是右端点区别对待。

* 查找左端点：**真正是左端点有两种情况：①midIndex==0；②target > nums[midIndex-1]**。首先判断是否满足这两种情况，若满足，直接返回midIndex，否则继续向下调整空间，即high=midIndex-1；
* 查找右端点：**真正是右端点的两种情况：①midIndex==nums.length-1;②target<nums[midIndex+1]**。首先判断是否满足这两种情况，若满足，直接返回midIndex，否则继续向上调整空间，即low=midIndex+1；

# Java代码

* 将**获取左端点**与**获取右端点**封装成单独的函数。

public class SearchRange {

@Test

public void test() {

int[] nums = {5,7,7,8,8,10};

int[] result = searchRange(nums,8);

System.out.println(Arrays.toString(result));

}

public int[] **searchRange**(int[] nums, int target) {

int[] result = new int[2];

result[0] = getLeftBound(nums,0,nums.length - 1,target);

result[1] = getRightBound(nums,0,nums.length - 1,target);

return result;

}

public int **getLeftBound**(int[] nums,int low,int high,int target){

while(low <= high){

int midIndex = (low + high)/2;

if(target < nums[midIndex]){

high = midIndex -1;

}else if(target > nums[midIndex]){

low = midIndex + 1;

}else{//target == nums[midIndex]

**if(midIndex == 0 || target > nums[midIndex-1]){**

**return midIndex;**

**}**

**high = midIndex -1;**

}

}

return -1;

}

public int **getRightBound**(int[] nums,int low,int high,int target){

while(low <= high){

int midIndex = (low + high)/2;

if(target < nums[midIndex]){

high = midIndex -1;

}else if(target > nums[midIndex]){

low = midIndex + 1;

}else{//target == nums[midIndex]

**if(midIndex == nums.length -1|| target < nums[midIndex + 1]){**

**return midIndex;**

**}**

**low = midIndex + 1;**

}

}

return -1;

}

}

* 不拆分成单独函数：

public int[] searchRange2(int[] nums, int target) {

int low = 0;

int high = nums.length - 1;

int[] result = {-1,-1};

while(low <= high){//获取左端点

int midIndex = (low + high)/2;

if(target < nums[midIndex]){

high = midIndex -1;

}else if(target > nums[midIndex]){

low = midIndex + 1;

}else{

if(midIndex == 0 || target > nums[midIndex-1]){

result[0] = midIndex;

break;

}

high = midIndex -1;

}

}

low = 0;

high = nums.length - 1;

while(low <= high){//获取右端点

int midIndex = (low + high)/2;

if(target < nums[midIndex]){

high = midIndex -1;

}else if(target > nums[midIndex]){

low = midIndex + 1;

}else{

if(midIndex == nums.length -1|| target < nums[midIndex + 1]){

result[1] = midIndex;

return result;

}

low = midIndex + 1;

}

}

return result;

}