旋转数组查找LeetCode\_33\_SearchInRotatedSortedArray

# 题目介绍

<https://leetcode.com/problems/search-in-rotated-sorted-array/description/>

Suppose **an array sorted** in ascending order is rotated **at some pivot** unknown to you beforehand.(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a **target** value to search. If found in the array return its index, otherwise return -1.

You may assume **no duplicate exists** in the array.

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

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0

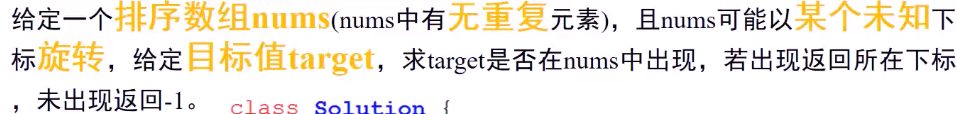
Output: 4

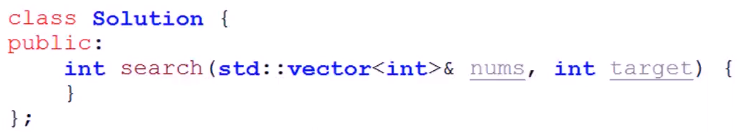
Example 2:

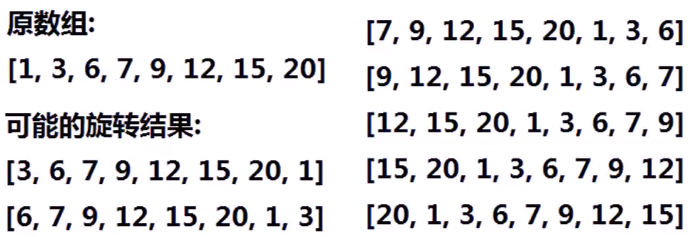
Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

难度：Medium

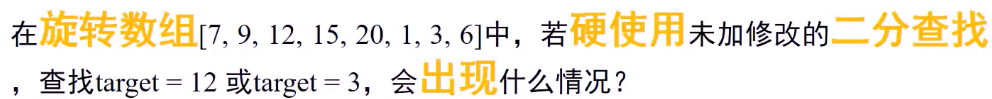


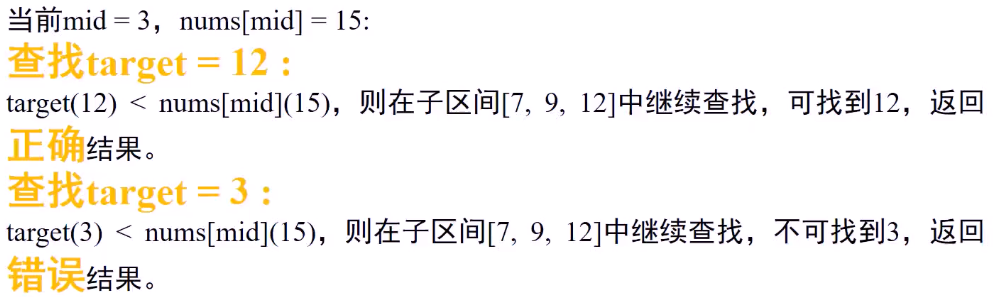


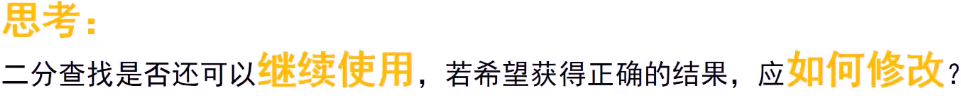


# 分析思路：

## 思考：







## 解决思路

分情况讨论：

首先根据**target与nums[mid]**进行比较，分成三种情况:**等于、大于、小于**。

这样仅仅是进行分类，不能得到什么有效信息。

第二个判断条件nums[begin]与nums[mid]比较重要，可以获知**递增区间与旋转区间的前后**。

* nums[begin]<nums[mid]：可以得出[begin,mid-1]肯定为递增区间，[mid+1,end]为旋转区间。
* nums[begin]<nums[mid]：可以得出[begin,mid-1]肯定为旋转区间，[mid+1,end]为递增区间。

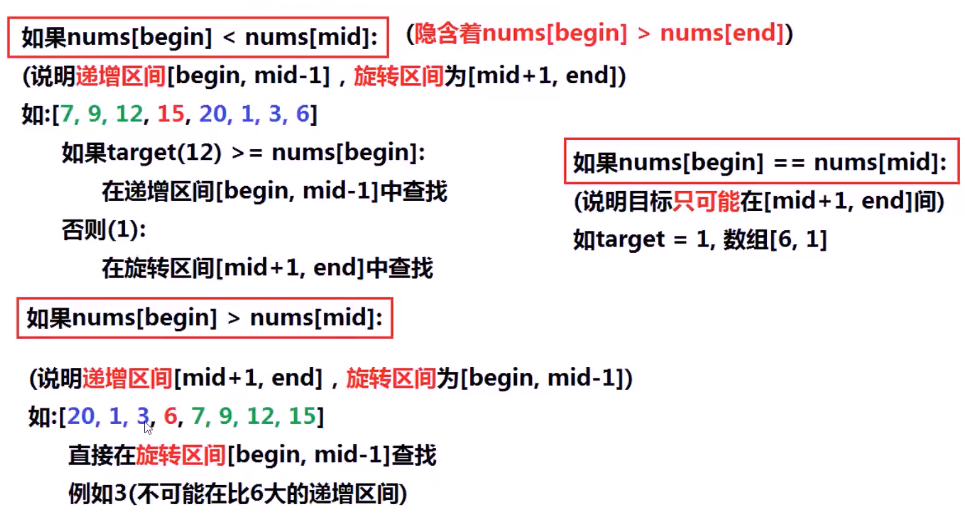
两个判断条件结合在一起就可以知道下一步应该搜索的区间，从而while循环实现。

下面分成7种情况讨论:

* **target==nums[mid];直接返回mid；**
* **target<nums[mid]且nums[begin]<nums[mid];前一段为递增区间，下次搜索哪一段需要由target与nums[begin]决定，若target<=nums[begin],返回后一段区间，low=mid+1; target>nums[begin]返回前一段区间，high=mid-1;**
* **target<nums[mid]且nums[begin]>nums[mid];后一段为递增区间，下次搜索后一段区间。**
* **target>nums[mid]且nums[begin]<nums[mid];前一段为递增区间，下次搜索后一段区间,low = mid+1。**
* **target>nums[mid]且nums[begin]>nums[mid];后一段为递增区间，下次搜索哪一段需要由target与nums[begin]决定，若target<=nums[begin],返回后一段区间，low=mid+1; target>nums[begin]返回前一段区间，high=mid-1;**

### **target=nums[mid]:** 直接返回**mid**。

### **target<nums[mid]**



### **target>nums[mid]**



# 代码实现

## 示例代码





## Java代码实现：

public class **LeetCode\_33\_SearchInRotatedSortedArray** {

@Test

public void test(){

int[] nums = {23,25,34,56,199,2,5,6,10};

for(int i = 0;i < nums.length;i++){

int target = nums[i];

int index = binarySearchInRotatedSortedArray(nums,0,nums.length-1,target);

System.out.println("target+\";index = \" + index = " + target+";index = " + index);

}

int target = 66;

int index = binarySearchInRotatedSortedArray(nums,0,nums.length-1,target);

System.out.println("target+\";index = \" + index = " + target+";index = " + index);

}

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

while(low <= high){

int midIndex = (low + high)/2;

if(target < nums[midIndex]) {

if(nums[low] <= nums[midIndex]){//前一段为递增区间

if(target > nums[low]){

high = midIndex -1;

}else if (target < nums[low]){

low = midIndex+1;

}else{

return low;

}

}else{//nums[low] > nums[midIndex]后一段为递增区间

high = midIndex -1;

}

// else{

// low = midIndex +1;

// }

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

if(nums[low] <= nums[midIndex]){//前一段为递增区间(可以归在一起)

low = midIndex+1;

}else{ //(nums[low] > nums[midIndex])

if(target > nums[low]){

high = midIndex - 1;

}else if (target < nums[low]){

low = midIndex + 1;

}else{

return low;

}

}

// else{//容易忽略

// low = midIndex + 1;

// }

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

return midIndex;

}

}

return -1;

}

}