假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0,1,2,4,5,6,7]` 可能变为 `[4,5,6,7,0,1,2]`)。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 `-1` 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 *O*(log *n*) 级别。

示例 1:

```
输入: nums = [4,5,6,7,0,1,2], target = 0
输出: 4
```

示例 2:

```
输入: nums = [4,5,6,7,0,1,2], target = 3
输出: -1
```

解题思路：

这道题让在旋转数组中搜索一个给定值，若存在返回坐标，若不存在返回-1。我们还是**考虑二分搜索法**，但是这道题的**难点在于我们不知道原数组在哪旋转了**，我们还是用题目中给的例子来分析，对于数组[0 1 2 4 5 6 7] 共有下列七种旋转方法：

0 1 2 **4 5 6 7**

7 0 1 **2 4 5 6**

6 7 0 **1 2 4 5**

5 6 7 **0 1 2 4**

4 5 6 7 0 1 2

2 4 5 6 7 0 1

1 2 4 5 6 7 0

二分搜索法的关键在于获得了中间数后，判断下面要搜索左半段还是右半段，我们观察上面红色加粗的数字都是升序的，由此我们可以观察出规律，如果中间的数小于最右边的数，则右半段是有序的，若中间数大于最右边数，则左半段是有序的，我们只要在有序的半段里用首尾两个数组来判断目标值是否在这一区域内，这样就可以确定保留哪半边了，代码如下

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int i=0,j=nums.size()-1;
        while(i<=j){
            int wz=(i+j)/2;
            if(nums[wz]==target){
                return wz;
```

```
        }else if(nums[j]>nums[wz]){
            if(nums[j]>=target&&nums[wz]<target){
                i=wz+1;
            }else{
                j=wz-1;
            }
        }else{
            if(nums[i]<=target&&nums[wz]>target){
                j=wz-1;
            }else{
                i=wz+1;
            }
        }
    }
    return -1;
  }
};
```