There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the possible rotation and an integer target, return *the index of*target*if it is in*nums*, or*-1*if it is not in*nums.

You must write an algorithm with O(log n) runtime complexity.

**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

**Example 3:**

**Input:** nums = [1], target = 0

**Output:** -1

class Solution {

public int search(int[] nums, int target)

{

int start=0;

int end=nums.length-1;

while(start<end)

{

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

if(nums[mid]>nums[end])

{

start=mid+1;

}

else

{

end=mid;

}

}

int pivot=start;

start=0;

end=nums.length-1;

if(target>=nums[pivot]&&target<=nums[end])

{

start=pivot;

}

else

{

end=pivot;

}

while(start<=end)

{

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

if(nums[mid]==target)

{

return mid;

}

else if(nums[mid]<target)

{

start=mid+1;

}else

{

end=mid-1;

}

}

return -1;

}

}

