Search in Rotated Sorted Array

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

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

class Solution {

int binarySearch(int ar[],int i,int j,int target){

while(i<j){

int m=(i+j)/2;

if(ar[m]==target)

return m;

else if(ar[m]<target)

i=m+1;

else

j=m-1;

}

if(i<ar.length && ar[i]==target)

return i;

else

return -1;

}

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

int i=0;

int j=nums.length-1;

while(i<j){

int m=(i+j)/2;

if((m-1>=0 && m+1<=j) && (nums[m]<nums[m-1] && nums[m]<nums[m+1])){

i=m;

break;

}

else if(nums[m]>=nums[i] && nums[m]>=nums[j])

i=m+1;

else

j=m-1;

}

System.out.println(i);

if(nums[i]==target)

return i;

else{

if(i==0){

return binarySearch(nums,0,nums.length-1,target);

}

else if(target<=nums[nums.length-1]){

return binarySearch(nums,i,nums.length-1,target);

}

else{

return binarySearch(nums,0,i,target);

}

}

}

}