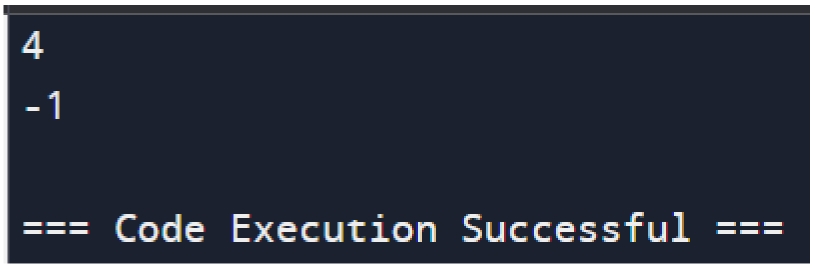
44. Search in Rotated Sorted Array There is an integer array nums sorted in ascending order (with distinct values). Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n1], 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

PROGRAM:-

```
def search(nums, target):
   left, right = 0, len(nums) - 1

   while left <= right:
      mid = (left + right) // 2

      if nums[mid] == target:
         return mid

      # Determine which part is sorted
      if nums[left] <= nums[mid]:  # Left part is sorted
         if nums[left] <= target < nums[mid]:
            right = mid - 1  # Target is in the left part
         else:
            left = mid + 1  # Target is in the right part
      else:  # Right part is sorted
         if nums[mid] < target <= nums[right]:
            left = mid + 1  # Target is in the right part
         else:
            right = mid - 1  # Target is in the left part

   return -1  # Target is not found

# Example usage:
nums = [4,5,6,7,0,1,2]
target = 0
index = search(nums, target)
print(index)  # Output: 4

# Another example:
nums = [4,5,6,7,0,1,2]
target = 3
index = search(nums, target)
print(index)  # Output: -1
```

OUTPUT:-



```
4
-1

=== Code Execution Successful ===
```

TIME COMPLEXITY:-O(log n)