Find out how many times the array has been rotated

Problem Statement: Given an integer array **arr** of size **N**, sorted in ascending order (**with distinct values**). Now the array is rotated between 1 to N times which is unknown. Find how many times the array has been rotated.

Pre-requisites: Find minimum in Rotated Sorted Array, Search in Rotated Sorted Array II & Binary Search algorithm

▼ **Examples**

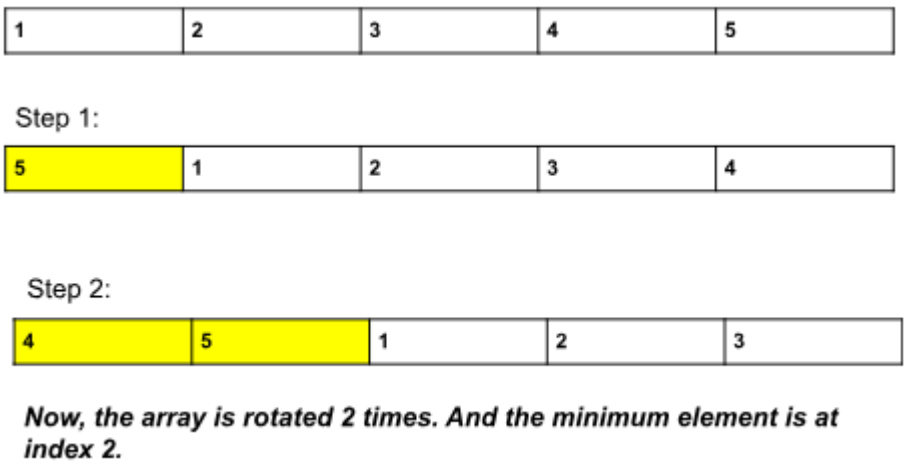
Example 1:
Input Format: arr = [4,5,6,7,0,1,2,3]
Result: 4
Explanation: The original array should be [0,1,2,3,4,5,6,7]. So, we can notice that the array has been rotated 4 times.

Example 2:
Input Format: arr = [3,4,5,1,2]
Result: 3
Explanation: The original array should be [1,2,3,4,5]. So, we can notice that the array has been rotated 3 times.

Solution:

How does the rotation occur in a sorted array?

Let's consider a sorted array: {1, 2, 3, 4, 5}. If we rotate this array 2 times, it will become: {4, 5, 1, 2, 3}. In essence, we moved the element at the last index to the front, while shifting the remaining elements to the right. We performed this process twice.



Now, the array is rotated 2 times. And the minimum element is at index 2.

Observation:

- We can easily observe that ***the number of rotations in an array is equal to the index(0-based index) of its minimum element.***

- *So, **in order to solve this problem, we have to find the index of the minimum element.***

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

▼ Brute Force Approach
▼ Algorithm / Intuition ›

Algorithm:

- First, we will declare an 'ans' and an 'index' variable initialized with a large number and -1 respectively.
- Next, we will iterate through the array and compare each element with the variable called 'ans'. Whenever we encounter an element 'arr[i]' that is smaller than 'ans', we will update 'ans' with the value of 'arr[i]' and also update the 'index' variable with the corresponding index value, 'i'.
- Finally, we will return 'index' as our answer.

▼ Code ›

```
import sys
def findKRotation(arr : [int]) -> int:
    n = len(arr)  # Size of array
    ans = float('inf')
    index = -1
    for i in range(n):
        if arr[i] < ans:
            ans = arr[i]
            index = i
    return index

if __name__ == "__main__":
    arr = [4, 5, 6, 7, 0, 1, 2, 3]
    ans = findKRotation(arr)
    print("The array is rotated", ans, "times.")
```

Output: The array is rotated 4 times.

▼ Complexity Analysis ›
Time Complexity: O(N), N = size of the given array.**Reason:** We have to iterate through the entire array to check if the target is present in the array.

Space Complexity: O(1)**Reason:** We have not used any extra data structures, this makes space complexity, even in the worst case as O(1).

▼ Optimal Approach

▼ Algorithm / Intuition ›

Optimal Approach(Using Binary Search):

We are going to use the binary search algorithm to solve this problem.

The primary objective of the Binary Search algorithm is to efficiently determine the appropriate half to eliminate, thereby reducing the search space by half. It does this by determining a specific condition that ensures that the target is not present in that half.

In the previous article, find the minimum in a rotated sorted array, we have discussed how to find the minimum element in a rotated and sorted array using Binary search. In this problem, we will employ the same algorithm to determine the index of the minimum element. In the previous problem, we only stored the minimum element itself. However, in this updated approach, we will additionally keep track of the index. By making this small adjustment, we can effectively solve the problem using the existing algorithm.

Algorithm:

The steps are as follows:

To begin, we will declare the variable 'ans' and initialize it with the largest possible value. Additionally, we will have two pointers, 'low' and 'high', as usual. In this case, we will also introduce an 'index' variable and initialize it with -1.

1. **Place the 2 pointers i.e. low and high:** Initially, we will place the pointers like this: low will point to the first index and high will point to the last index.
2. **Calculate the 'mid':** Now, inside a loop, we will calculate the value of 'mid' using the following formula:
   mid = (low+high) // 2 ('//' refers to integer division)
3. **If arr[low] <= arr[high]:** In this case, the array from index low to high is completely sorted. Therefore, we can select the minimum element, arr[low].
   Now, ***if arr[low] < ans, we will update 'ans' with the value arr[low] and 'index' with the corresponding index low.***
   Once this is done, there is no need to continue with the binary search algorithm. So, we will **break** from this step.
4. Identify the sorted half, and after picking the leftmost element, eliminate that half.
    1. **If arr[low] <= arr[mid]:** This condition ensures that the left part is sorted. So, we will pick the leftmost element i.e. **arr[low]**.
       Now, ***if arr[low] < ans, we will update 'ans' with the value arr[low] and 'index' with the corresponding index low.***
       After that, we will eliminate this left half(i.e. low = mid+1).
    2. **Otherwise, if the right half is sorted:** This condition ensures that the right half is sorted. So, we will pick the leftmost element i.e. **arr[mid]**.
       Now, ***if arr[mid] < ans, we will update 'ans' with the value arr[mid] and 'index' with the corresponding index mid.***
       After that, we will eliminate this right half(i.e. high = mid-1).

5. This process will be inside a loop and the loop will continue until low crosses high. Finally, we will return the ‘index’ variable that stores the index of the minimum element.

Dry-run: *Please refer to the video for a detailed explanation.*

▼ Code ›

```
import sys
def findKRotation(arr : [int]) -> int:
    low = 0
    high = len(arr) - 1
    ans = float('inf')
    index = -1
    while low <= high:
        mid = (low + high) // 2

        # If search space is already sorted,
        # then arr[low] will always be
        # the minimum in that search space
        if arr[low] <= arr[high]:
            if arr[low] < ans:
                index = low
                ans = arr[low]
            break

        # If left part is sorted
        if arr[low] <= arr[mid]:
            # Keep the minimum
            if arr[low] < ans:
                index = low
                ans = arr[low]

            # Eliminate left half
            low = mid + 1
        else:  # If right part is sorted
            # Keep the minimum
            if arr[mid] < ans:
                index = mid
                ans = arr[mid]

            # Eliminate right half
            high = mid - 1

    return index

if __name__ == "__main__":
    arr = [4, 5, 6, 7, 0, 1, 2, 3]
    ans = findKRotation(arr)
    print("The array is rotated", ans, "times.")
```

Output: The array is rotated 4 times.

▼ Complexity Analysis ›
Time Complexity: O(logN), N = size of the given array.
Reason: We are basically using binary search to find the minimum.

Space Complexity: O(1)
Reason: We have not used any extra data structures, this makes space complexity, even in the worst case as O(1).