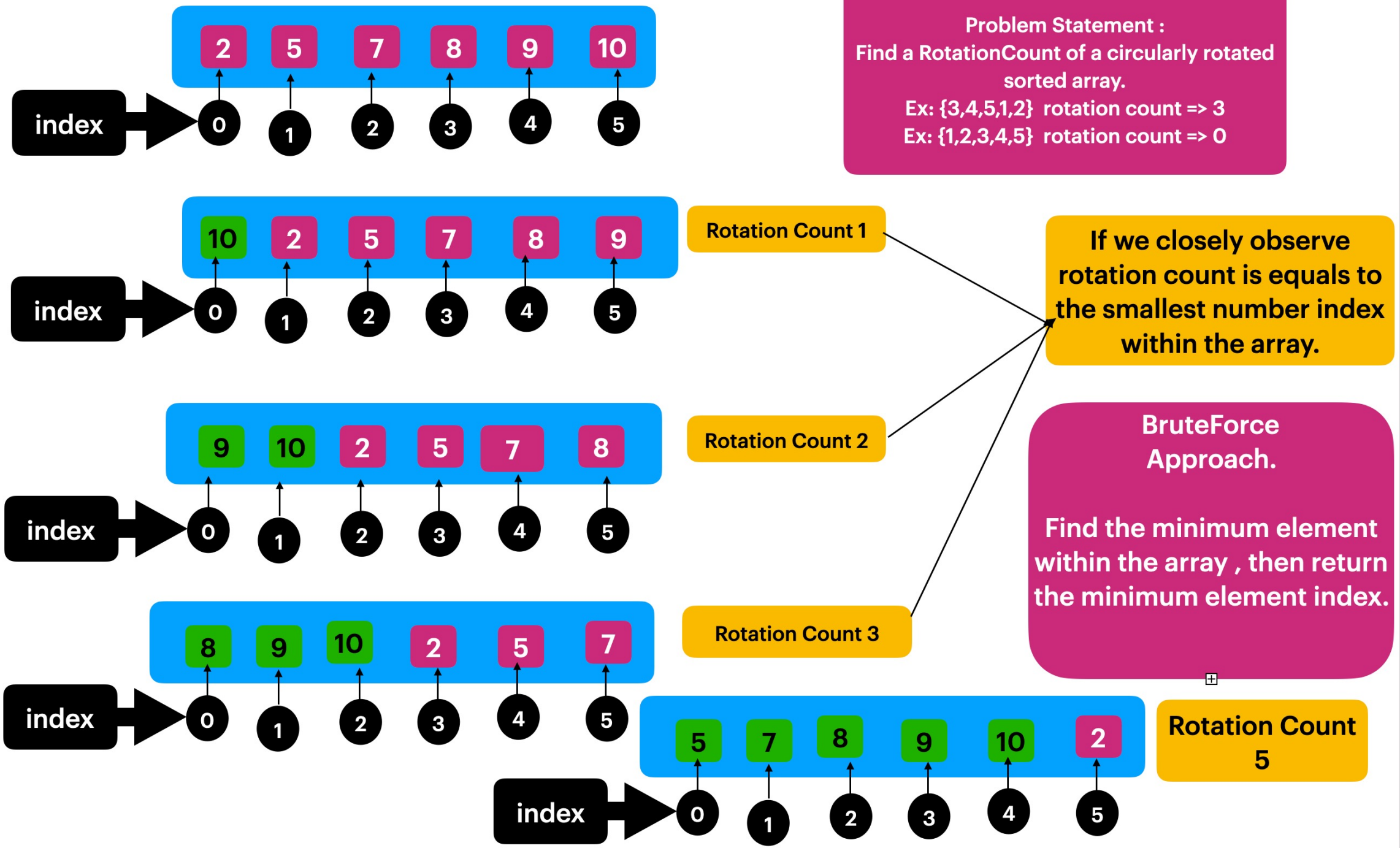
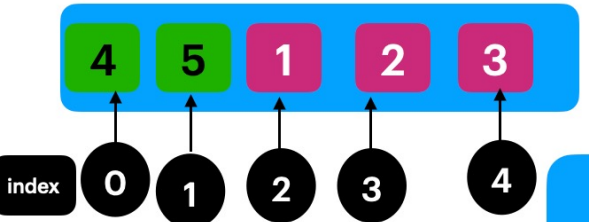

Problem Statement :
Find a RotationCount of a circularly rotated sorted array.
Ex: {3,4,5,1,2} rotation count => 3
Ex: {1,2,3,4,5} rotation count => 0
index
2 5 7 8 9 10
0 1 2 3 4 5
Rotation Count 1
10 2 5 7 8 9
Rotation Count 2
9 10 2 5 7 8
Rotation Count 3
8 9 10 2 5 7
If we closely observe rotation count is equals to the smallest number index within the array.
BruteForce Approach.
Find the minimum element within the array , then return the minimum element index.
5 7 8 9 10 2
Rotation Count 5



Rotation count = 2

start = 0
end = 4
mid = 2

arr

As arr[mid-1] greater than arr[mid] so return mid.

return mid i.e 2

Let's track the solution with Binary Search

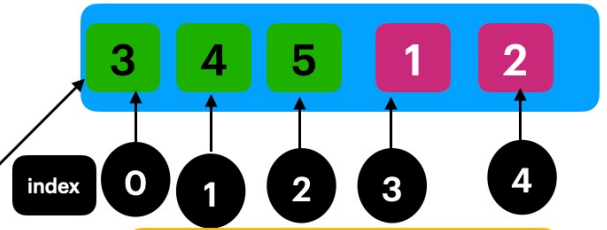
rotationIndex → Index of minimum element in an array.

When we work on BinarySearch we always figure out the solution through mid element.

2 possibilities

arr[mid-1] > arr[mid]
Here "mid" is the rotation index

arr[mid] > arr[mid+1]
Here "mid+1" is the rotation index



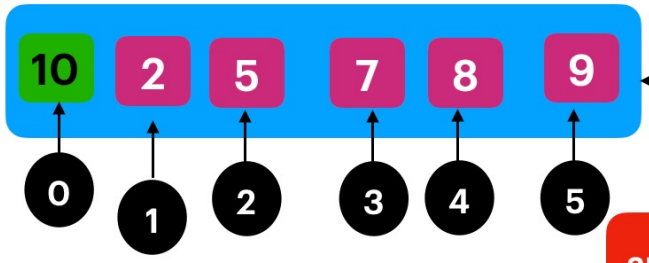
Rotation count = 3

As arr[mid] greater than arr[mid+1] so return mid+1.

arr

start = 0
end = 5
mid = 2

return mid i.e 3

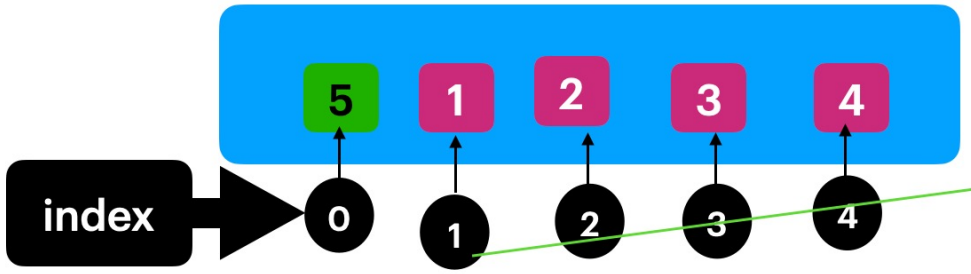


What if you don't find rotationIndex in current iteration. Then we need to think about considering either left part or right part .

arr[mid] > arr[0] , It means left part is sorted. Move to right
start = mid + 1 ;

2 possibilities

arr[mid] < arr[end] , It means right part is sorted. Move to left
end = mid - 1 ;



arr[mid] < arr[end] so move to left
end = mid-1

Step1

arr

start = 0
end = 5
mid = 2

arr

Step2

start = 0
end = 1
mid = 0

arr[mid] > arr[mid+1] so return mid+1

Return mid+1 i.e 1

Rotation count = 1

Algorithm to find number of rotation in a circularly sorted array with BinarySearch

1. Initialise start = 0 , end = n-1 , mid = start+ (end-start)/2

2. If the arr[mid] > arr[mid+1] then return mid+1

3. If the arr[mid-1] > arr[mid] then return mid

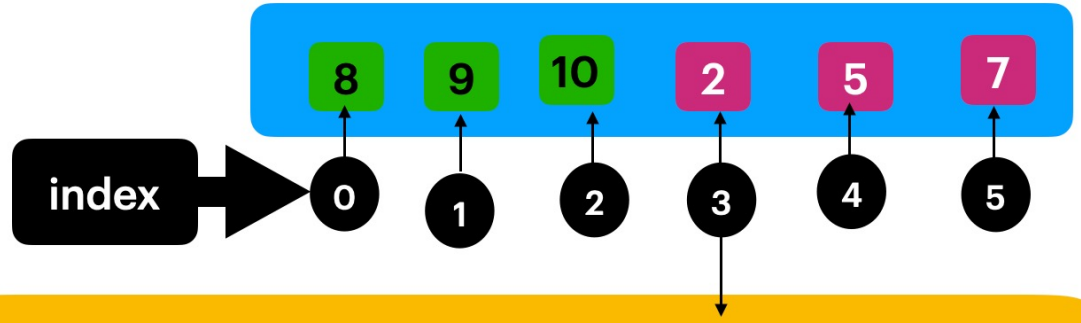
4. If arr[mid] > arr[0] it means left array is sorted so move to right

i.e start = mid + 1;

5. If arr[mid] < arr[end] , it means right array is sorted then move to left

i.e end = mid - 1;

TimeComplexity = O(logn)
SpaceComplexity (Iteration) = O(1)
SpaceComplexity (Recursive) = O(logn)



Problem Statement :

Find index of an element in a rotated sorted array.

Ex: {3,4,5,1,2}

targetElement = 5

output = 2 (Index of the target 5)

Ex: {3,4,5,1,2}

target = 8

output = -1

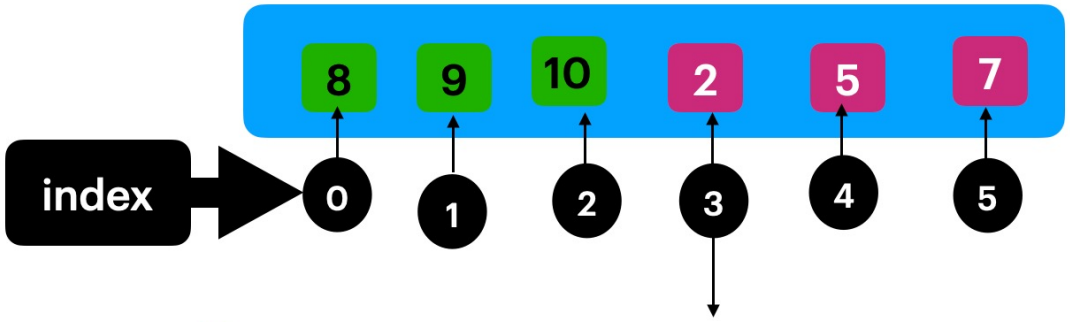
(No element found with the target 8 so return -1)

BruteForce Algorithm :

LinearSearch, start from 0 repeat till n-1 , if the element found return index.

Time Complexity: O(n)

Space Complexity: O(1)



Find a rotationIndex

rotationIndex == 0
search(arr, 0, n-1, target)

rotationIndex != 0

if(target >= arr[0])
start = 0
end = rotationIndex-1
search(arr,start , end , target)

if(target <= arr[0])
start = rotationIndex
end = arr.length-1
search(arr,start , end , target)

Problem Statement :
Find an element in a rotated sorted array.
Ex: {3,4,5,1,2}
targetElement = 5
output = 2 (Index of the target 5)
Ex: {3,4,5,1,2}
target = 8
output = -1
(No element found with the target 8 so return -1)

BinarySearch Algorithm

1. Find the rotationIndex.

2. If the rotationIndex is '0' then search form index 0 to n-1

3. If the rotationIndex != 0 then we need to choose either leftPart or RightPart

If the tartget >= arr[0], we need to search in left part.
i.e search(arr, 0, rotationIndex-1 , target)

If the tartget <= arr[0], we need to search in right part.
i.e search(arr, 0, rotationIndex , n-1, target)

TimeComplexity = O(logn)
SpaceComplexity (Iteration) = O(1)
SpaceComplexity (Recursive) = O(logn)