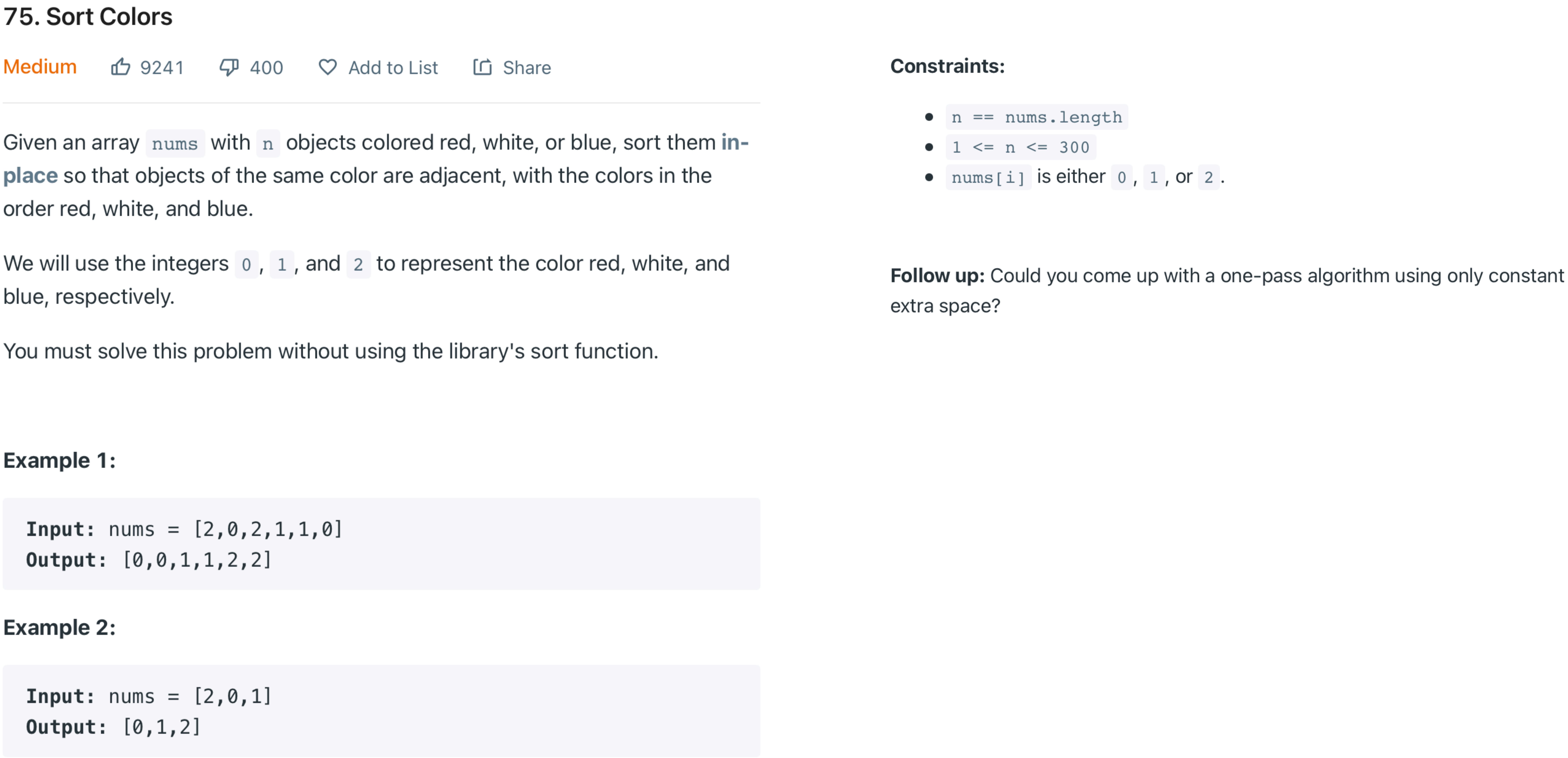


75. Sort Colors

Medium 👍 9241 👎 400 ♡ Add to List Share

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

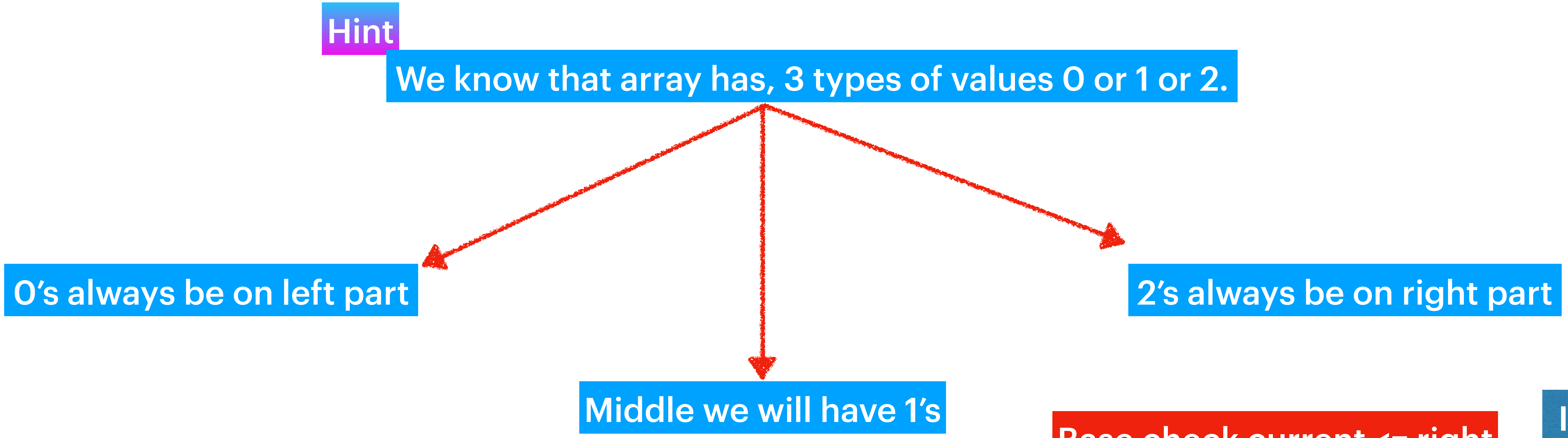
Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either `0`, `1`, or `2`.

Follow up: Could you come up with a one-pass algorithm using only constant extra space?



Hint

We know that array has, 3 types of values 0 or 1 or 2.

0's always be on left part

Middle we will have 1's

2's always be on right part

Base check current <= right

Algoritm

-> Take three pointers, left, right, current.

left = 0;

right = n-1

current = 0

-> Move current pointer till [current <= right]

-> When current points to value '2' then swap with right pointer, decrement the right pointer.

-> When current points to value '0' then swap with left pointer, increment the current & left pointer's.

-> When current points to value '1' just no swap, increment the current pointer.

Time Complexity : O(n)

Space Complexity : O(1)

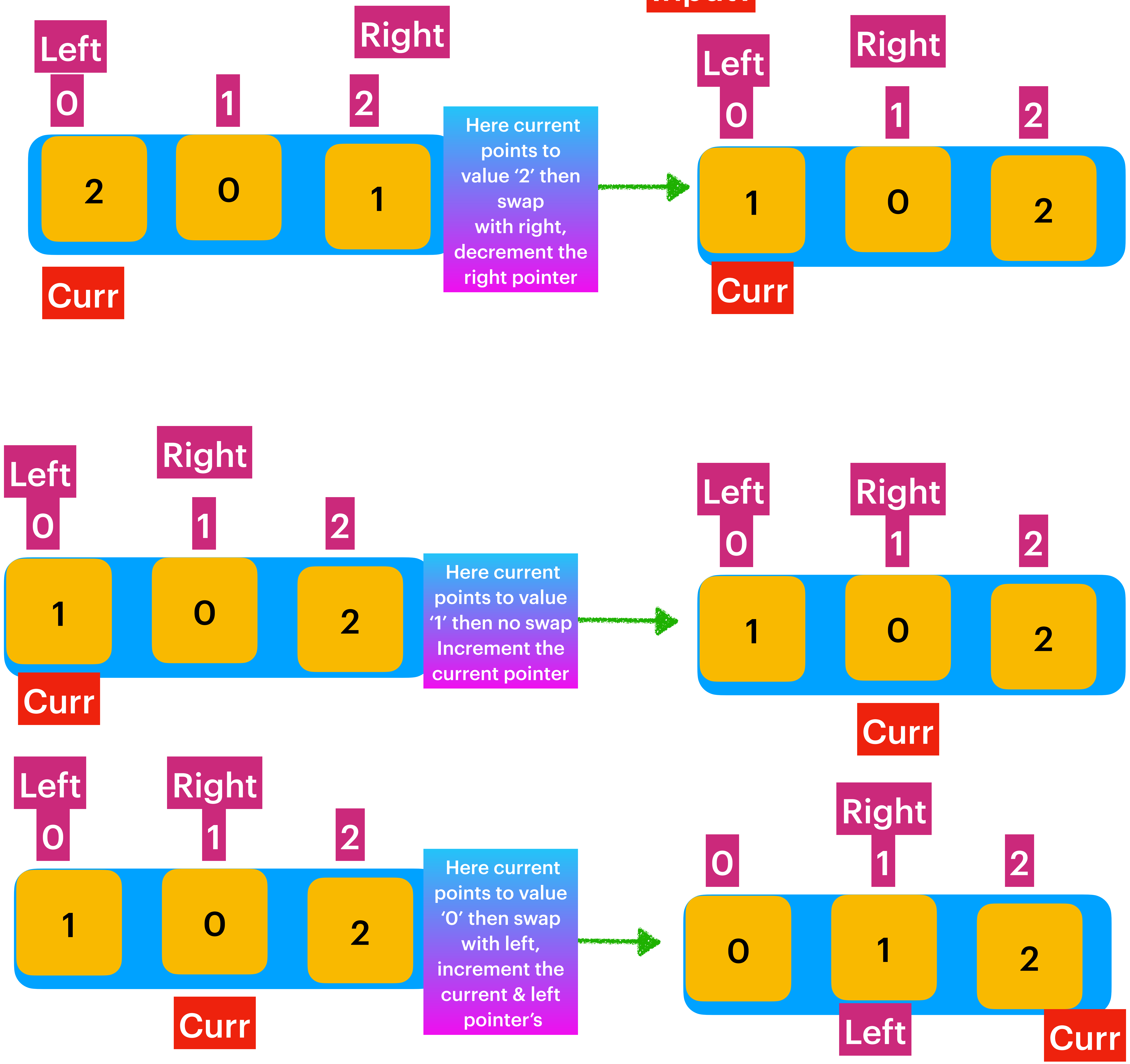
Input= [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

input: [0,0,0]
output: [0,0,0]

[1]
output: [1]

input: [1,2,0]
output: [0,1,2]

Input1



Here current points to value '2' then swap with right, decrement the right pointer

Here current points to value '1' then no swap Increment the current pointer

Here current points to value '0' then swap with left, increment the current & left pointer's

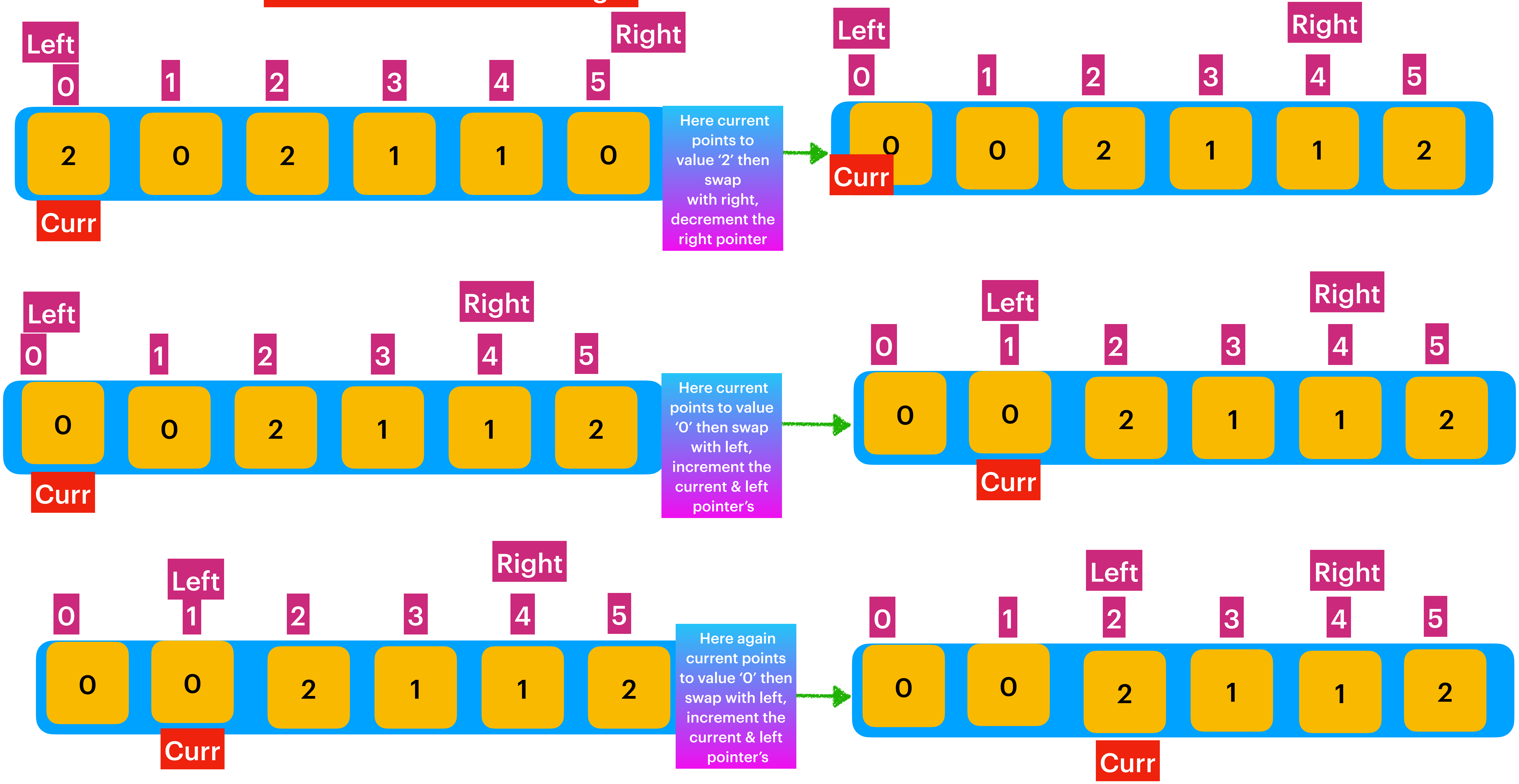
Base Check :: current <= right

Time Complexity : O(n)
Space Complexity : O(1)

As the current > right then we break the process. Array is sorted .

Base Check :: current <= right

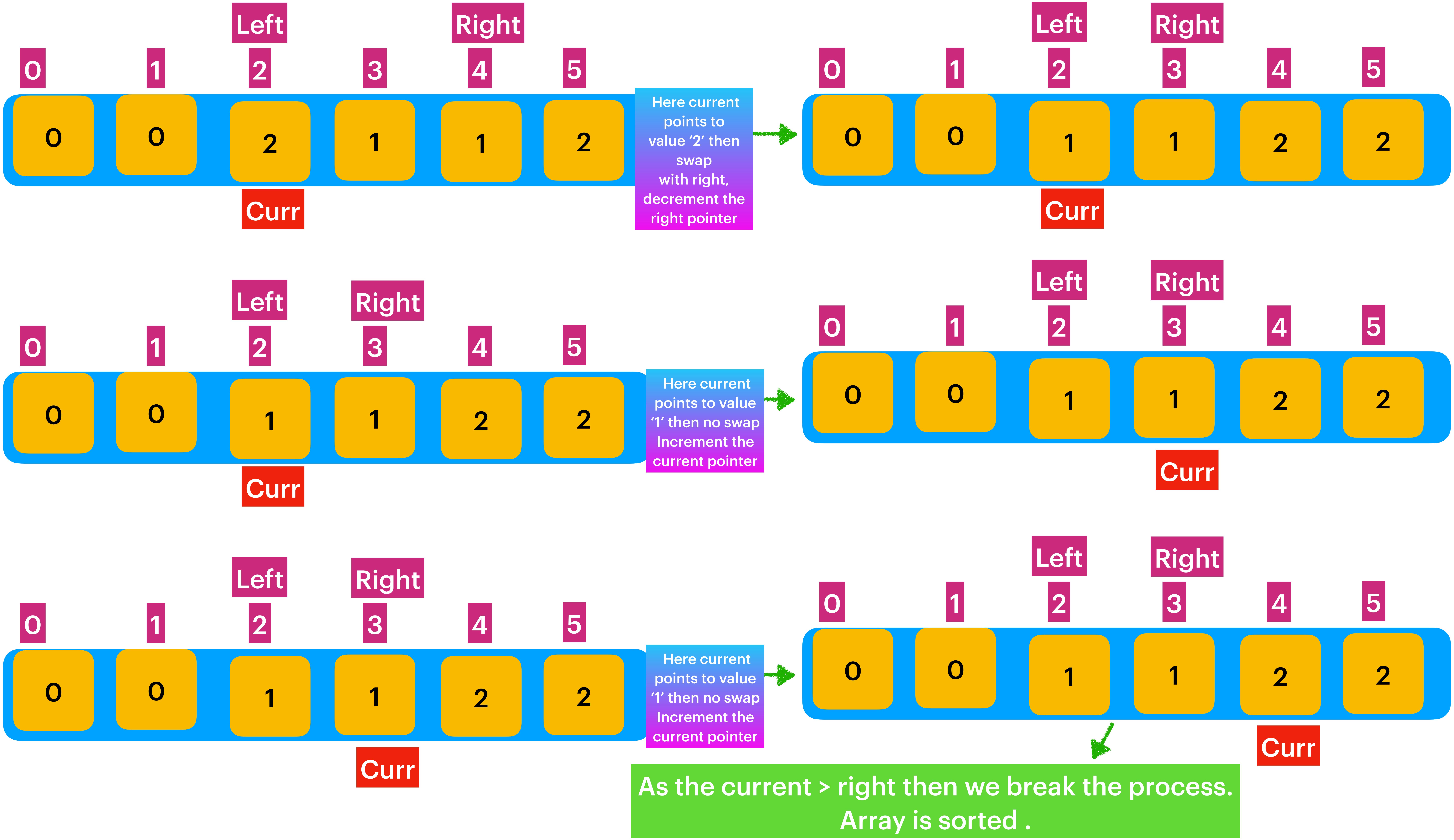
Input : 2



Here current points to value '2' then swap with right, decrement the right pointer

Here current points to value '0' then swap with left, increment the current & left pointer's

Here again current points to value '0' then swap with left, increment the current & left pointer's

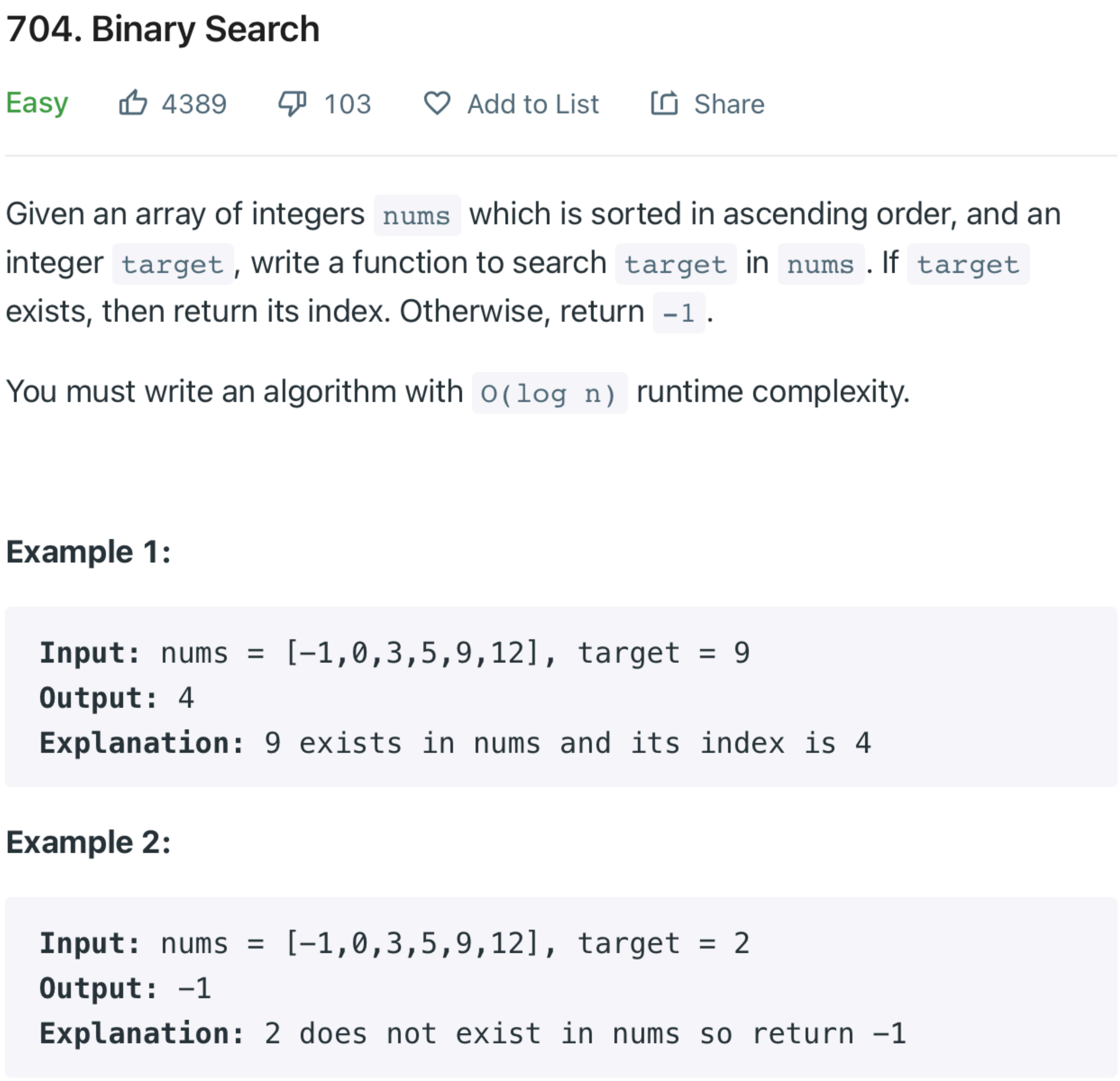


Here current points to value ‘2’ then swap with right, decrement the right pointer

Here current points to value ‘1’ then no swap Increment the current pointer

Here current points to value ‘1’ then no swap Increment the current pointer

As the current > right then we break the process.
Array is sorted .



704. Binary Search

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return `-1`.

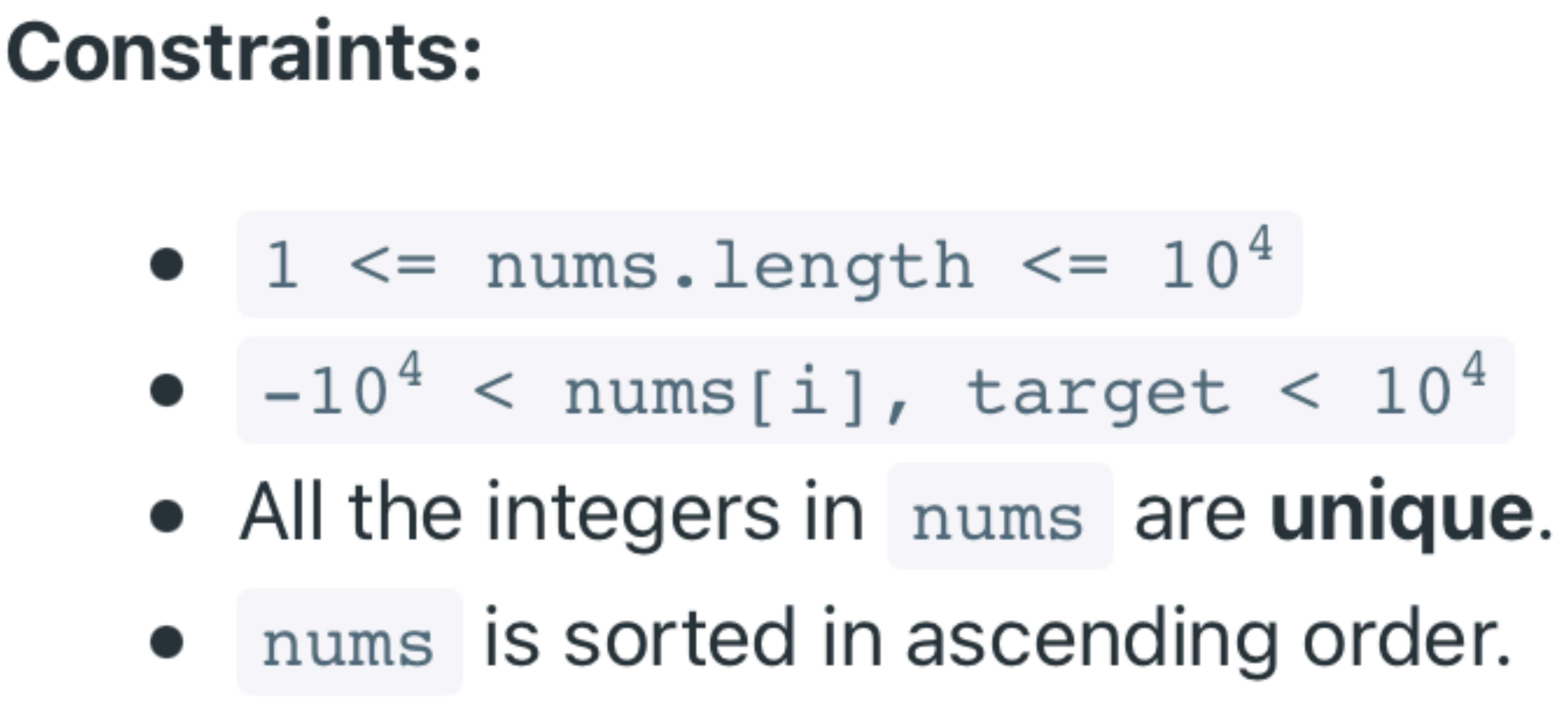
You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

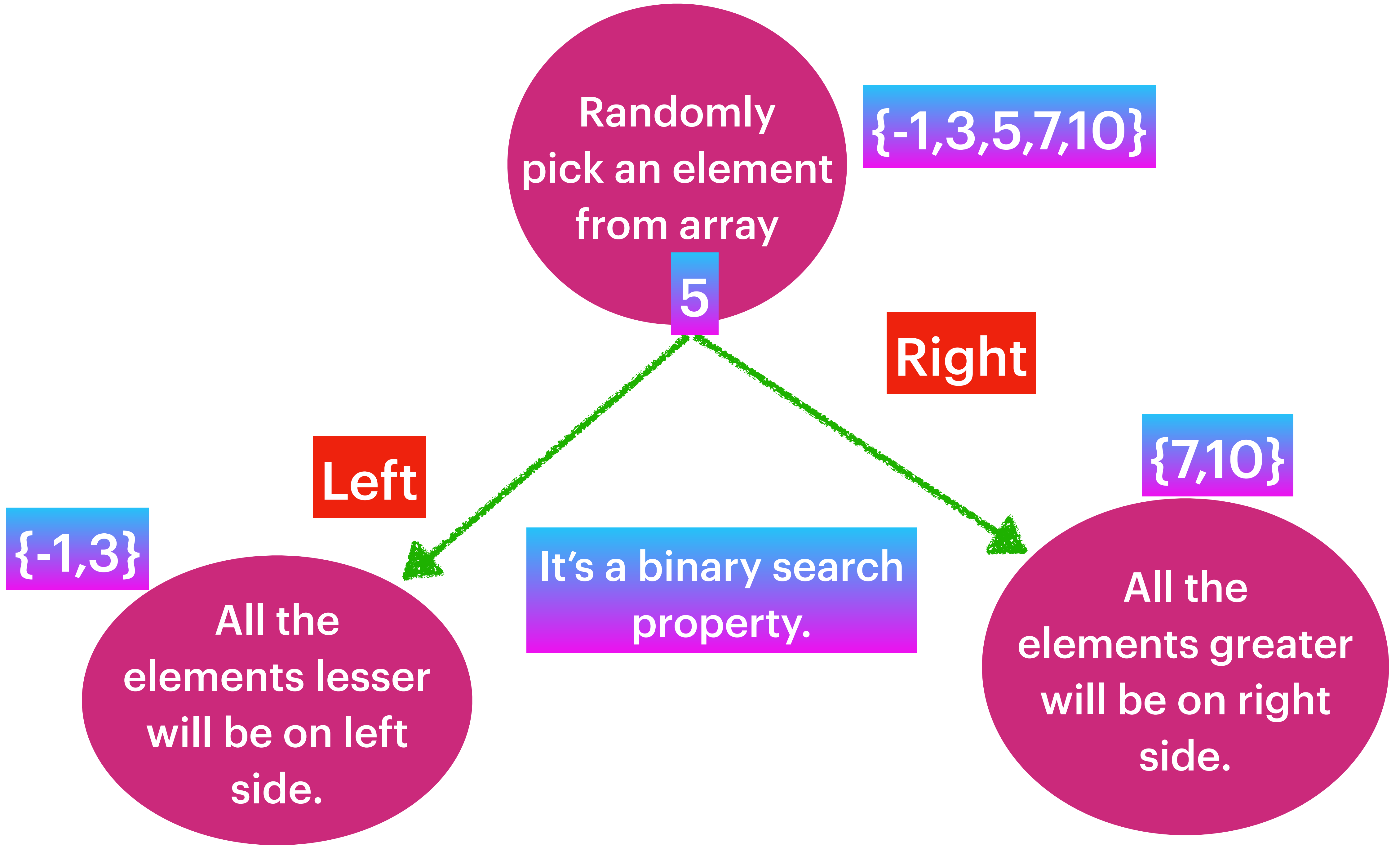


Constraints:

- `1 <= nums.length <= 10`^{4}
- `-10`^{4}` < nums[i], target < 10`^{4}
- All the integers in `nums` are **unique**.
- `nums` is sorted in ascending order.

Hint

Clue in a Sorted Array



Randomly pick an element from array

5

{-1,3,5,7,10}

Right

{7,10}

Left

{-1,3}

It's a binary search property.

All the elements lesser will be on left side.

All the elements greater will be on right side.

Base check left <= right

Time Complexity : O(logn)
Space Complexity : O(1)

Algoritm

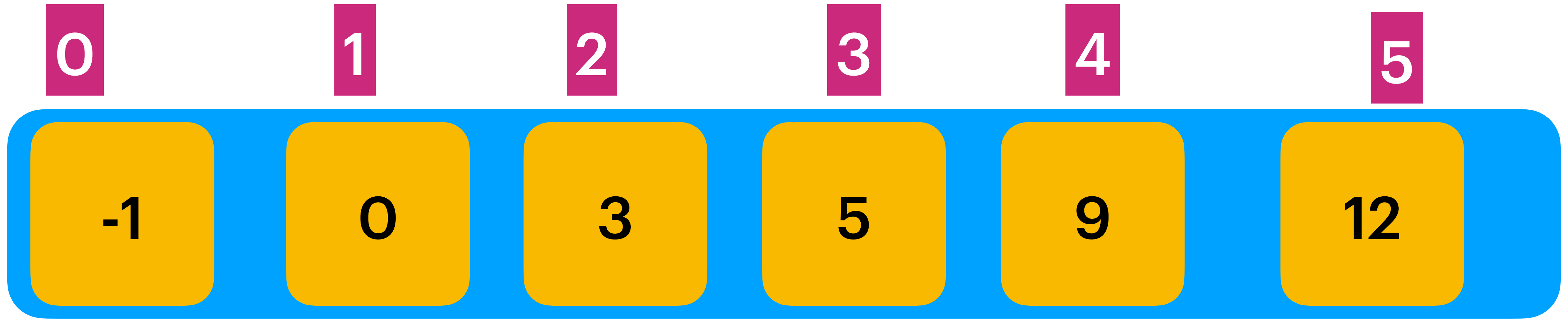
—> Take three pointers , left, right and mid.
left = 0
right = n-1
mid= (left+right)/2 ;
Preferable principle is for calculating mid is
mid = left + (right-left)/2 ;
[avoids lossy compression for higher inputs].

—> When mid == target then return the 'mid' index.

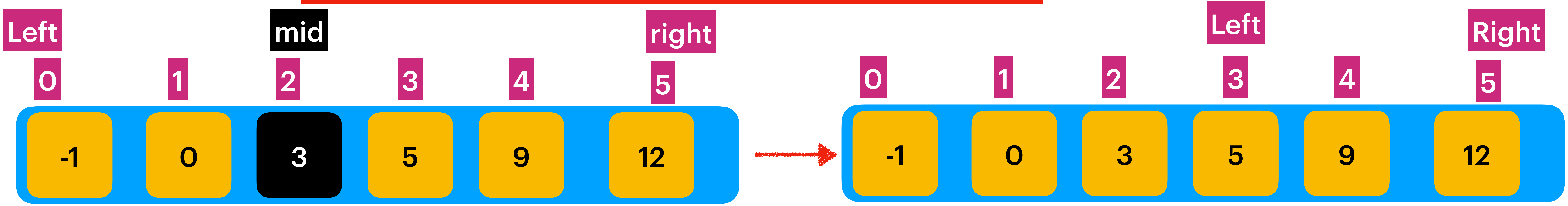
—> When mid > target then element would be on left side so move —> right = mid-1
—> When mid < target then element would be on right the move —> left = mid+1

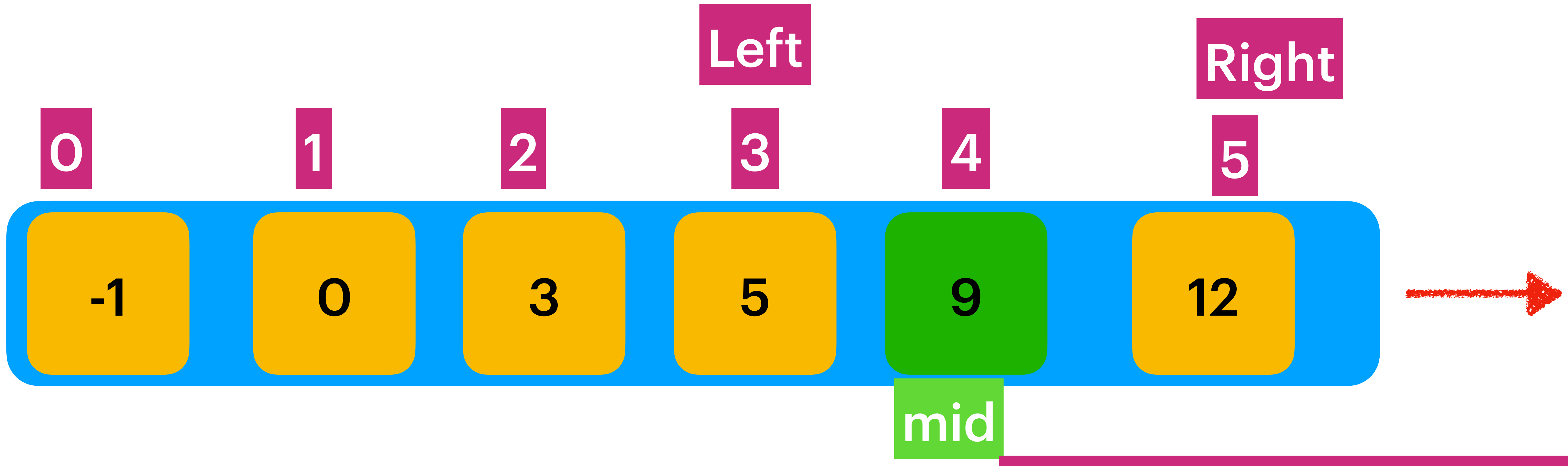
int[] nuts = {-1,0,3,5,9,12}

Target : 9



nums[mid] < target so move to RightPart —> low = mid+1



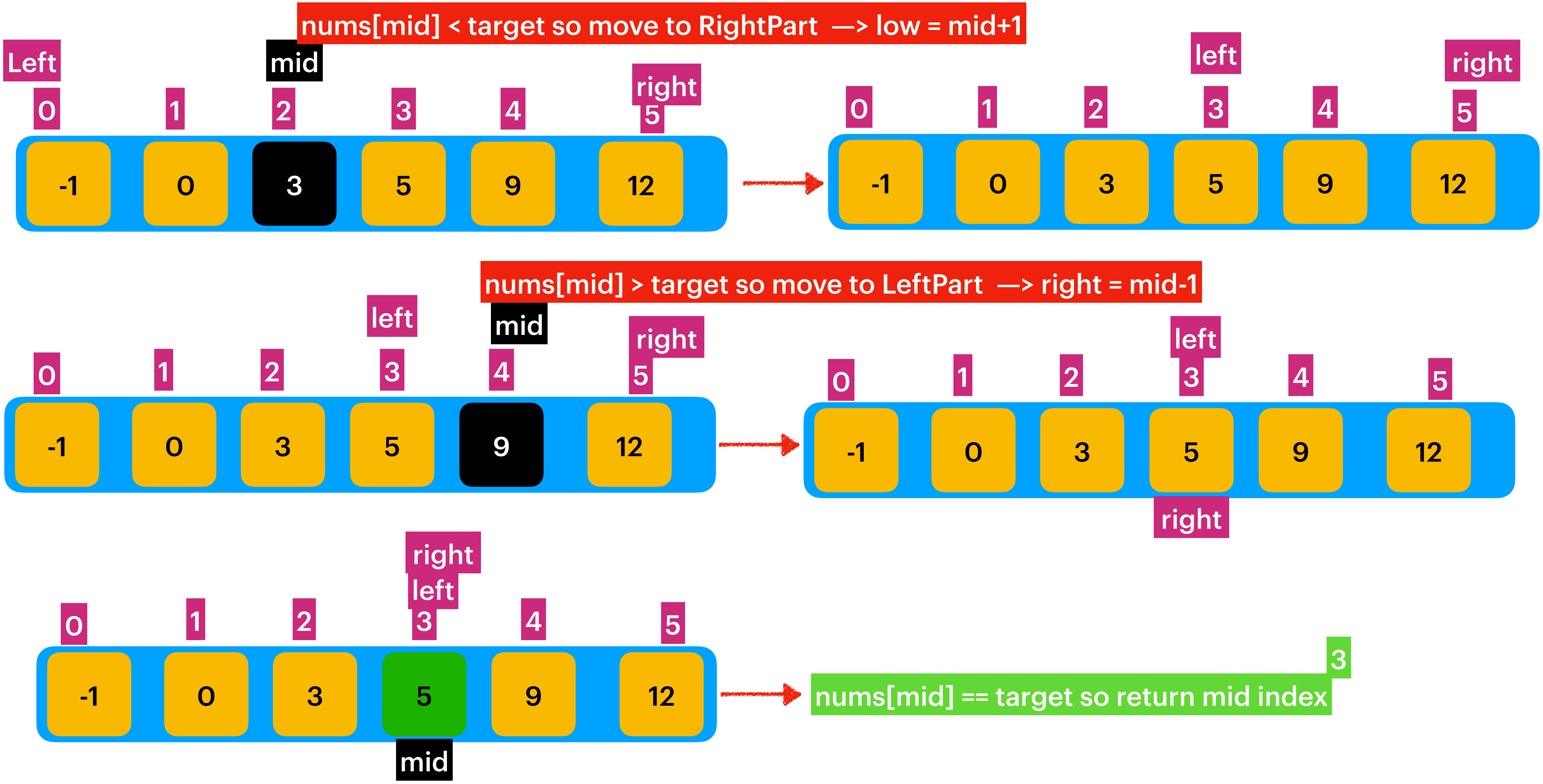


Return mid Index: 4

nums[mid] == target so return mid index

int[] nuts = {-1,0,3,5,9,12}

Target : 5



nums[mid] < target so move to RightPart —> low = mid+1

Left mid right

0 1 2 3 4 5

-1 0 3 5 9 12

left right

0 1 2 3 4 5

-1 0 3 5 9 12

nums[mid] > target so move to LeftPart —> right = mid-1

left mid right

0 1 2 3 4 5

-1 0 3 5 9 12

left

0 1 2 3 4 5

-1 0 3 5 9 12

right

right

left

0 1 2 3 4 5

-1 0 3 5 9 12

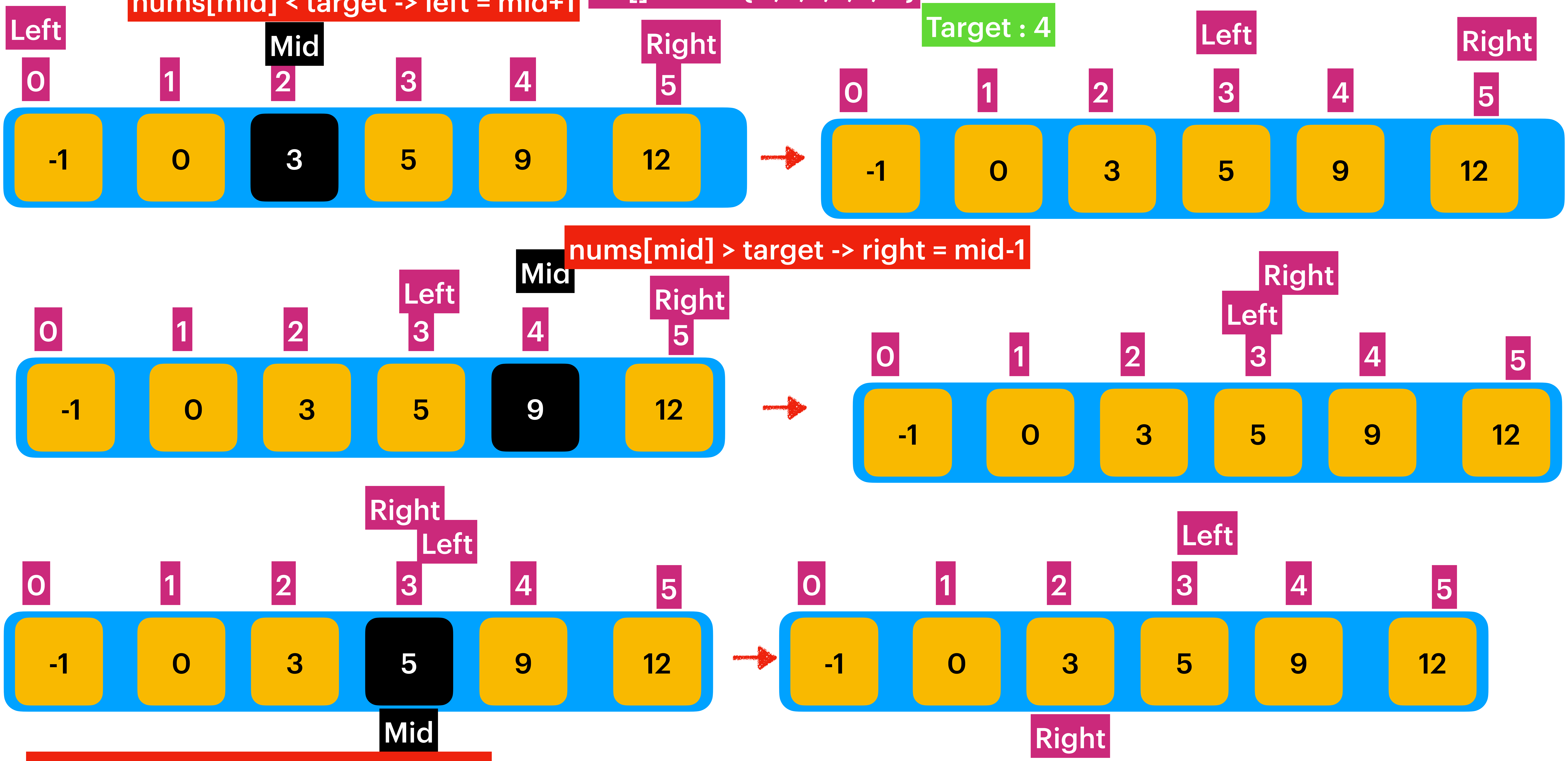
mid

3

nums[mid] == target so return mid index

int[] nuts = {-1,0,3,5,9,12}

Target : 4



nums[mid] < target -> left = mid+1

nums[mid] > target -> right = mid-1

nums[mid] > target -> right = mid-1

Base Check is failed as the left > right

return -1;

Time Complexity Analysis

Assume there are 16 elements
in worst case we take 4 iterations.

Binary Search :

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16} n:16 :: target = -1

{1,2,3,4,5,6,7,8} n:8

{1,2,3,4} n:4

{1,2} n:2

{1}

log2^(16) = 4

log2^(n) ->
Time Complexity : log(n)
Space Complexity : O(1)