Previous Next

33. Search in Rotated Sorted Array

Jan. 10, 2019 | 148.2K views

Average Rating: 3.66 (79 votes)

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

You are given a target value to search. If found in the array return its index, otherwise return `-1`.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of $O(\log n)$.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

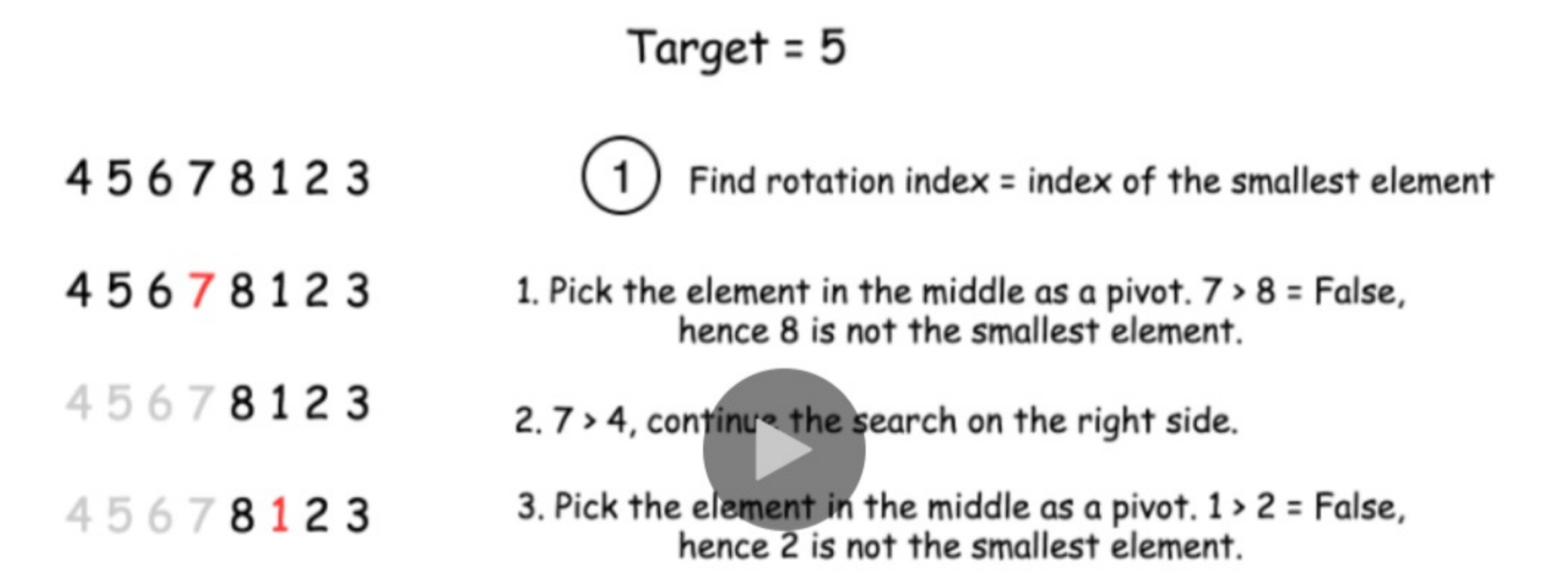
Solution

Approach 1: Binary search

The problem is to implement a search in $\mathcal{O}(\log N)$ time that gives an idea to use a binary search.

The algorithm is quite straightforward :

- Find a rotation index `rotation_index`, *i.e.* index of the smallest element in the array. Binary search works just perfect here.
- `rotation_index` splits array in two parts. Compare `nums[0]` and `target` to identify in which part one has to look for `target`.
- Perform a binary search in the chosen part of the array.



Java | Python | Copy

```python
class Solution:
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        def find_rotate_index(left, right):
            if nums[left] < nums[right]:
                return 0

            while left <= right:
                pivot = (left + right) // 2
                if nums[pivot] > nums[pivot + 1]:
                    return pivot + 1
                else:
                    if nums[pivot] < nums[left]:
                        right = pivot - 1
                    else:
                        left = pivot + 1

        def search(left, right):
            """
            Binary search
            """
            while left <= right:
                pivot = (left + right) // 2
```

Complexity Analysis

- Time complexity : $\mathcal{O}(\log N)$.
- Space complexity : $\mathcal{O}(1)$.

Approach 2: One-pass Binary Search

Instead of going through the input array in two passes, we could achieve the goal in one pass with an *revised* binary search.

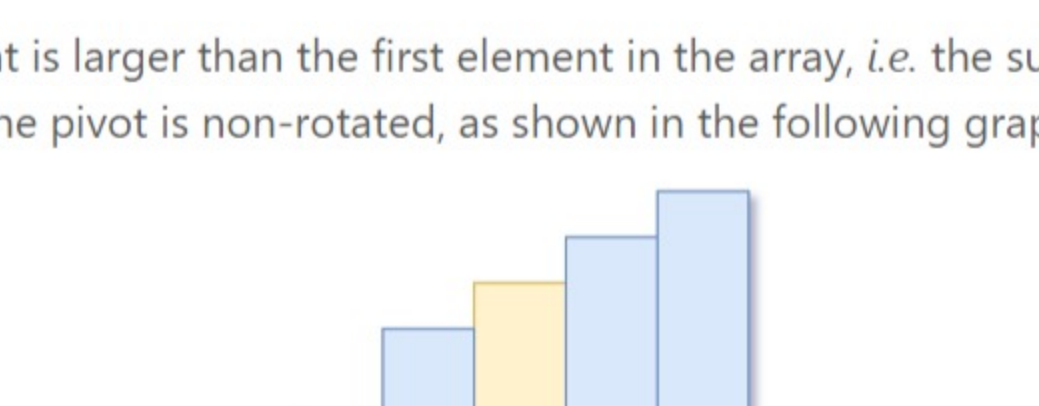
> The idea is that we add some additional *condition checks* in the normal binary search in order to better *narrow down* the scope of the search.

Algorithm

As in the normal binary search, we keep two pointers (*i.e.* `start` and `end`) to track the search scope. At each iteration, we reduce the search scope into half, by moving either the `start` or `end` pointer to the middle (*i.e.* `mid`) of the previous search scope.

Here are the detailed breakdowns of the algorithm:

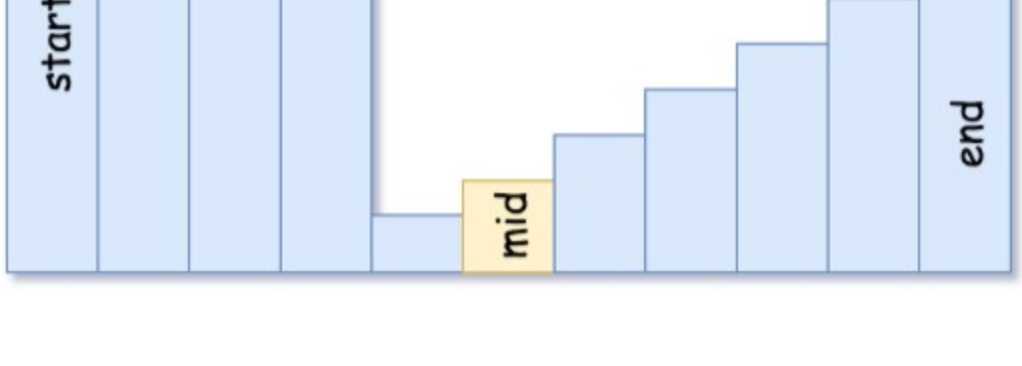
- Initiate the pointer `start` to `0`, and the pointer `end` to `n - 1`.
- Perform standard binary search. While `start <= end`:
    - Take an index in the middle `mid` as a pivot.
    - If `nums[mid] == target`, the job is done, return `mid`.
    - Now there could be two situations:
        - Pivot element is larger than the first element in the array, *i.e.* the subarray from the first element to the pivot is non-rotated, as shown in the following graph.




```
- If the target is located in the non-rotated subarray:
go left: `end = mid - 1`.

- Otherwise: go right: `start = mid + 1`.
```

        - Pivot element is smaller than the first element of the array, *i.e.* the rotation index is somewhere between `0` and `mid`. It implies that the sub-array from the pivot element to the last one is non-rotated, as shown in the following graph.




```
- If the target is located in the non-rotated subarray:
go right: `start = mid + 1`.

- Otherwise: go left: `end = mid - 1`.
```

- We're here because the target is not found. Return -1.

Implementation

Java | Python | Copy

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        start, end = 0, len(nums) - 1
        while start <= end:
            mid = start + (end - start) // 2
            if nums[mid] == target:
                return mid
            elif nums[mid] >= nums[start]:
                if target >= nums[start] and target < nums[mid]:
                    end = mid - 1
                else:
                    start = mid + 1
            else:
                if target <= nums[end] and target > nums[mid]:
                    start = mid + 1
                else:
                    end = mid - 1
        return -1
```

Complexity Analysis

- Time complexity: $\mathcal{O}(\log N)$.
- Space complexity: $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 42 Sort By

Type comment here... (Markdown is supported)

Preview Post

reyou 128 April 27, 2019 7:02 AM Report
Formula: If a sorted array is shifted, if you take the middle, always one side will be sorted. Take the recursion according to that rule.

1- take the middle and compare with target, if matches return.
2- if middle is bigger than left side, it means left is sorted
Read More
51 Share Reply
SHOW 4 REPLIES

haoyangfan 897 January 25, 2019 8:12 AM Report
16-line C solution beats 100% that uses most basic form of binary search

int search(int* nums, int numsSize, int target) {
    int start = 0, end = numsSize - 1;
    while (start <= end) {
Read More
48 Share Reply
SHOW 6 REPLIES

lim142857 33 July 27, 2019 9:51 AM
Python 3 modified binary search (readable)(fast)(short)

        # Initilize two pointers
        begin = 0
        end = len(nums) - 1
Read More
13 Share Reply
SHOW 3 REPLIES

Sithis 11048 January 11, 2019 5:34 PM Report
One pass solution.

public int search(int[] a, int key) {
    int lo = 0;
    int hi = a.length - 1;
Read More
19 Share Reply
SHOW 8 REPLIES

softwareshortcut 424 April 18, 2019 8:04 PM
Creating helper methods is key otherwise you'd easily get lost dealing with indices. Great exercise, thanks for sharing. I see shorter solutions in the comments, but they might be tricky to come up with during an interview.
8 Share Reply

wrg 6 July 12, 2019 7:57 AM Report
In the worst case isn't the time complexity of the solution
O(n + log(n)) ? The worst case is when the pivot is at index n - 1.

So if I understand this correctly when n = 1 000 000 in worst case scenario (pivot is at index n -1))
The runtime of this solution would be 1 000 000 + log(1 000 000) == 1 000 006 (base 10)
Read More
6 Share Reply
SHOW 2 REPLIES

elstestnewway 4 January 13, 2019 9:34 PM Report
Language C
0 ms

int search(int *nums, int numsSize, int target)
{
Read More
4 Share Reply
SHOW 1 REPLY

valentinzhao 140 December 2, 2019 10:02 AM
Best official solution article ever.
3 Share Reply

nish_d 29 January 10, 2019 7:59 PM
find_rotate_index and search are doing almost the same thing. Is there a way to combine them both? We might have come across target in find_rotate_index already, and just need to end the program there.
2 Share Reply
SHOW 1 REPLY

layak 5 June 6, 2020 7:58 AM Report
why do we have equal to sign in this condition? nums[mid] >= nums[start]
1 Share Reply
SHOW 2 REPLIES

‹ 1 2 3 4 5 ›