

[Previous \(/articles/n-ary-tree-level-order-traversal/\)](/articles/n-ary-tree-level-order-traversal/) Next [\(/articles/russian-doll-envelopes/\)](/articles/russian-doll-envelopes/)

154. Find Minimum in Rotated Sorted Array II

(/problems/find-minimum-in-rotated-sorted-array-ii/)

Nov. 10, 2019 | 11.8K views

Average Rating: 4.80 (20 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]`).

Find the minimum element.

The array may contain duplicates.

Example 1:

Input: `[1,3,5]`**Output:** `1`

Example 2:

Input: `[2,2,2,0,1]`**Output:** `0`

Note:

- This is a follow up problem to Find Minimum in Rotated Sorted Array (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/>).
- Would allow duplicates affect the run-time complexity? How and why?

Solution

Approach 1: Variant of Binary Search

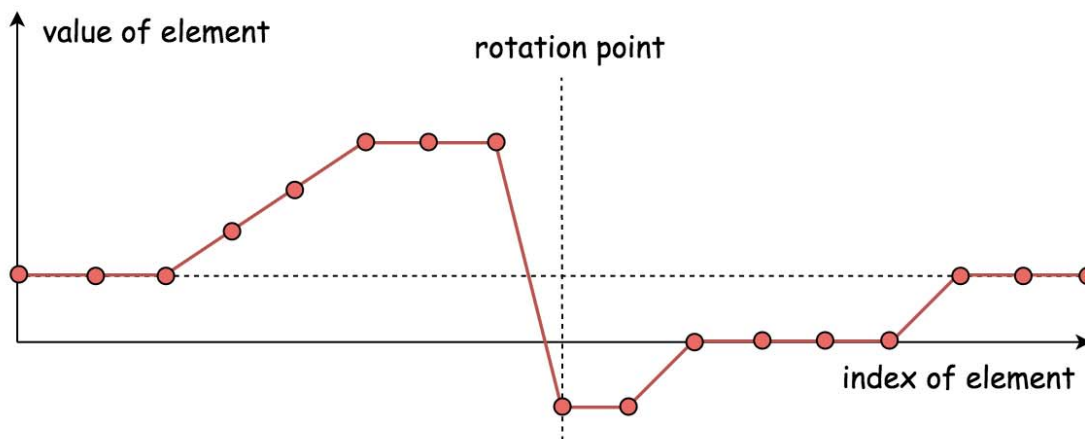
Intuition

Given a sorted array in ascending order (denoted as $L[i]$), the array is then rotated over certain *unknown pivot*, (denoted as $L'[i]$). We are asked to find the *minimum value* of this sorted and rotated array, which is to find the value of the first element in the original array, i.e. $L[0]$.

The problem resembles a common problem of *finding a given value from a sorted array*, to which problem one could apply the **binary search** algorithm. Intuitively, one might wonder if we could apply a variant of binary search algorithm to solve our problem here.

Indeed, this is the right intuition, though the tricky part is to figure out a **concise solution** that could work for all cases.

To illustrate the algorithm, we draw the array in a 2D dimension in the following graph, where the X axis indicates the index of each element in the array and the Y axis indicates the value of the element.



The main structure of our algorithm remains the same as the classical binary search algorithm. As a reminder, we summarize it briefly as follows:

- We keep two pointers, i.e. low , $high$ which point to the lowest and highest boundary of our search scope.
- We then reduce the search scope by moving either of pointers, according to various situations. Usually we shift one of pointers to the mid point between low and $high$, (i.e. $pivot = (low+high)/2$), which reduces the search scope down to half. This is also where the name of

the algorithm comes from.

Articles > 154. Find Mi

- The reduction of the search scope would stop, either we find the desired element or the two pointers converge (*i.e.* `low == high`).

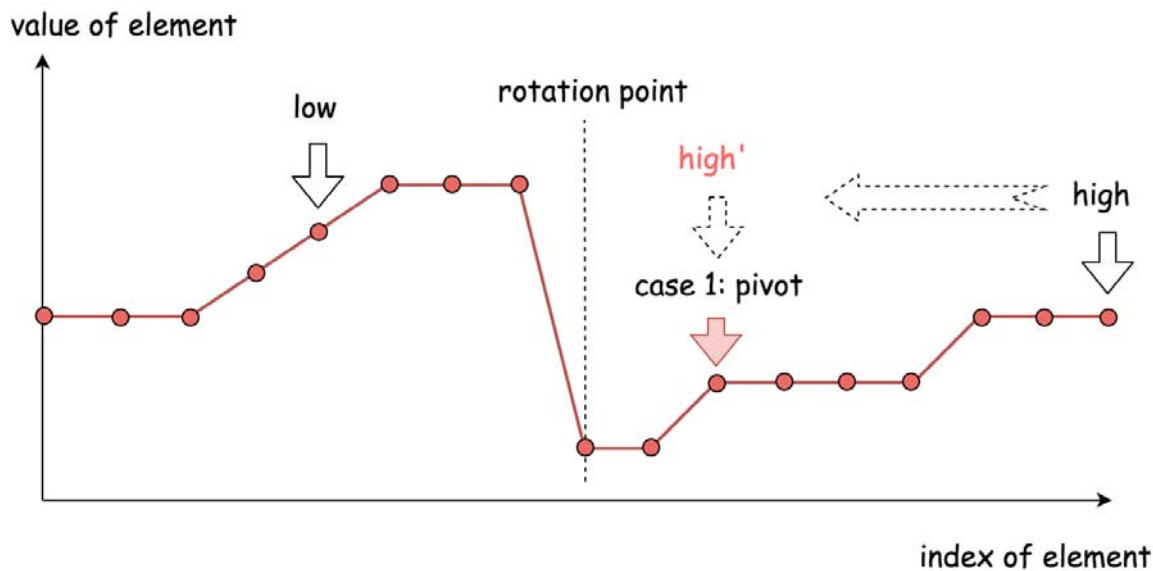
Algorithm

In the classical binary search algorithm, we would compare the pivot element (*i.e.* `nums[pivot]`) with the value that we would like to locate. In our case, however, we would compare the pivot element to the element pointed by the upper bound pointer (*i.e.* `nums[high]`).

Following the structure of the binary search algorithm, the essential part remained is to design the cases on how to update the two pointers.

Here we give one example on how we can break it down **concisely** into three cases. Note that given the array, we consider the element pointed by the `low` index to be on the left-hand side of the array, and the element pointed by the `high` index to be on the right-hand side.

Case 1). `nums[pivot] < nums[high]`

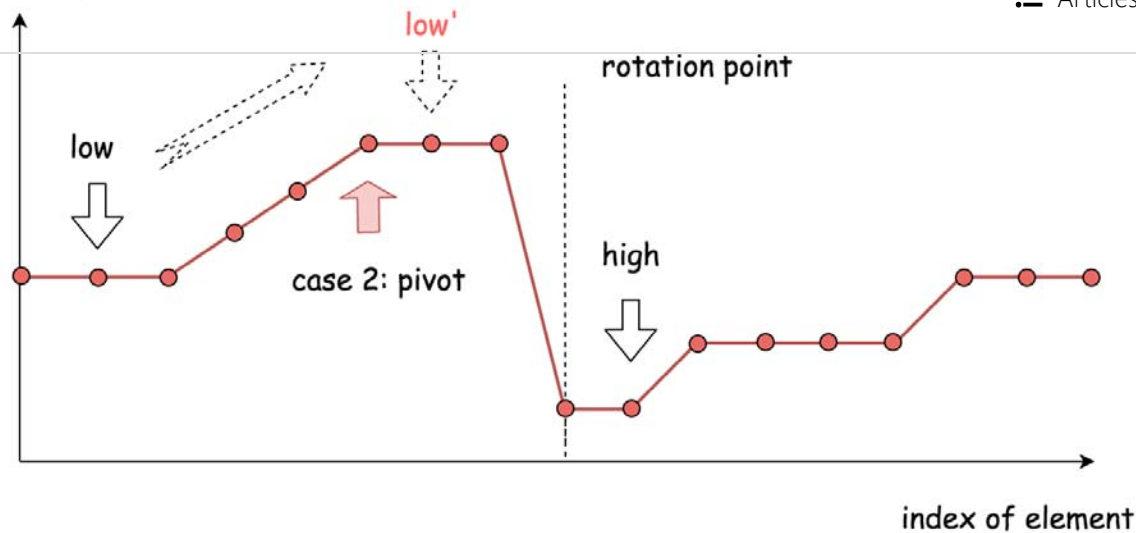


- The pivot element resides in *the same half* as the upper bound element.
- Therefore, the desired minimum element should reside to the **left-hand side** of pivot element. As a result, we then move the upper bound down to the pivot index, *i.e.* `high = pivot`.

Case 2). `nums[pivot] > nums[high]`

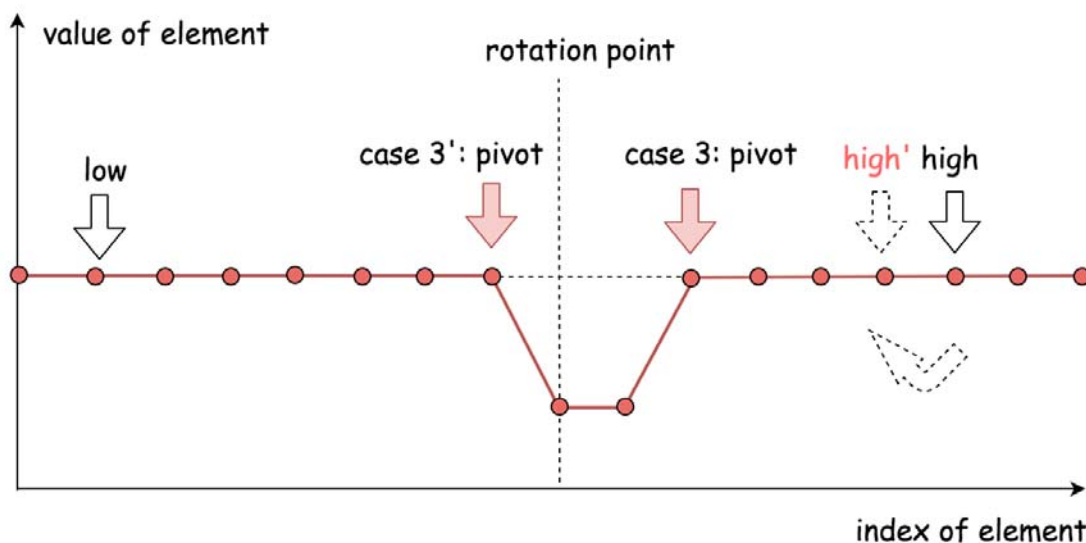
value of element

Articles > 154. Find Mi



- The pivot element resides in *the different half* of array as the upper bound element.
- Therefore, the desired minimum element should reside to the **right-hand side** of the pivot element. As a result, we then move the lower bound up next to the pivot index, *i.e.* $low = pivot + 1$.

Case 3). $nums[pivot] == nums[high]$



- In this case, we are not sure which side of the pivot that the desired minimum element would reside.
- To further reduce the search scope, a safe measure would be to reduce the upper bound by one (*i.e.* $high = high - 1$), rather than moving *aggressively* to the pivot point.
- The above strategy would prevent the algorithm from stagnating (*i.e.* endless loop). More importantly, it maintains the **correctness** of the procedure, *i.e.* we would not end up with

skipping the desired element.

Articles > 154. Find Mi

To summarize, this algorithm differs to the classical binary search algorithm in two parts:

- We use the upper bound of search scope as the reference for the comparison with the pivot element, while in the classical binary search the reference would be the desired value.
- When the result of comparison is equal (*i.e.* Case #3), we further move the upper bound, while in the classical binary search normally we would return the value immediately.

Here are some sample implementations based on the above algorithm. *Note:* the idea is inspired by the post from sheehan (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-ii/discuss/48808/My-pretty-simple-code-to-solve-it>) in the discussion forum.

Java

Python

Copy

```

1 class Solution:
2     def findMin(self, nums: List[int]) -> int:
3         low = 0
4         high = len(nums)-1
5         while high > low:
6             pivot = low + (high - low) // 2
7             # risk of overflow: pivot = (low + high) // 2
8             # Case 1):
9             if nums[pivot] < nums[high]:
10                 high = pivot
11                 # alternative: high = pivot - 1
12                 # too aggressive to move the `high` index,
13                 # it won't work for the test case of [3, 1, 3]
14             # Case 2):
15             elif nums[pivot] > nums[high]:
16                 low = pivot + 1
17             # Case 3):
18             else:
19                 high -= 1
20             # the 'low' and 'high' index converge to the inflection point.
21         return nums[low]

```

Complexity Analysis

- Time complexity: on average $\mathcal{O}(\log_2 N)$ where N is the length of the array, since in general it is a binary search algorithm. However, in the worst case where the array contains identical elements (*i.e.* case #3 $\text{nums}[\text{pivot}] == \text{nums}[\text{high}]$), the algorithm would deteriorate to iterating each element, as a result, the time complexity becomes $\mathcal{O}(N)$.
- Space complexity : $\mathcal{O}(1)$, it's a constant space solution.

Discussion

The problem is a follow-up to the problem of 153. Find Minimum in Rotated Sorted Array (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/>). The difference is that in this problem the array can contain duplicates. *So the question is "Would allow duplicates affect the run-time complexity? How and why?"*

First of all, the problem of 153. Find Minimum in Rotated Sorted Array (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/>) can be considered as a specific case of this problem, where it just happens that the array does not contain any duplicate. As a result, the very solutions of this problem would work for the problem of #153 (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/>) as well. It is just that we would never come across the case #3 (i.e. `nums[pivot] == nums[high]`) in the problem of #153 (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/>).

It is due to the fact that there might exist some duplicates in the array, that we come up with the case #3 which eventually renders the time complexity of the algorithm to be linear $\mathcal{O}(N)$, rather than $\mathcal{O}(\log_2 N)$.

One might wonder that whether it works in case #3 if we move the lower boundary (i.e. `low += 1`), rather than the upper boundary (i.e. `high -= 1`).

The short answer is that it could work for some cases, but not for all. For instance, given the input `[1, 3, 3]`, by moving the lower boundary, we would skip the correct answer.

While we do `low = pivot + 1` to reduce the search scope, then why not do `high = pivot - 1` instead of `high = pivot`? Or a similar question would be "why don't we do *check of `low <= high` rather than `low < high`*"?

As a matter of fact, the binary search algorithm has several forms of implementation (https://en.wikipedia.org/wiki/Binary_search_algorithm), regarding how we set the boundaries and the loop conditions. One can refer to the Explore card of Binary Search (<https://leetcode.com/explore/learn/card/binary-search/>) in LeetCode for more details. As simple as the idea of binary search might seem to be, it is tricky to make it work for all cases.

As one would discover from the card, the above implementation of binary search complies with the template II (<https://leetcode.com/explore/learn/card/binary-search/126/template-ii/937/>) of binary search. And by replacing `high = pivot` with `high = pivot - 1`, the algorithm will not work.

As subtle as it looks like, the update of the pointers should be consistent with the conditions of the

loop. As a rule of thumb, it is advised to stick with one form of binary search, and **not to mix them up**.
Articles > 154. Find Minimum in Rotated Sorted Array II

One might notice that we are calculating the pivot with the formula of `pivot = low + (high-low)/2`, rather than the more intuitive term `pivot = (high+low)/2`.

Actually, this is done intentionally to prevent the numeric overflow issue, since the sum of two integers could exceed the limit of the integer number. As a fun fact, the above mistake prevails in many implementations of binary search, as revealed from a post titled "*Nearly All Binary Searches and Mergesorts are Broken*" (<https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html>) from googleblog in 2006.

Rate this article:

Previous (/articles/n-ary-tree-level-order-traversal/)

Next (/articles/russian-doll-envelopes/)

Comments: 6

Sort By ▼



Type comment here... (Markdown is supported)

Preview

Post



(/lenchen1112)

lenchen1112 (/lenchen1112) ★ 664 🕒 December 18, 2019 3:50 AM

Just use the same idea of #153 (<https://leetcode.com/problems/3sum>) but with one more comparison.

```
# Python3
class Solution:
```

Read More

2 ^ v | Share | Reply



(/wunan96nj)

wunan96nj (/wunan96nj) ★ 1 🕒 December 30, 2019 11:25 PM

I think the solution is not correct enough, while the list is [1,1,1,1,3,1], although the result is the same, the rotation point is 0 instead of 5.

The better one I think is sited from brightwang from <https://leetcode.com/problems/search-in-rotated-sorted-array-ii/discuss/284013/Double-Binary-Search%3A-First-find-pivot-then-find-target> (<https://leetcode.com/problems/search-in-rotated-sorted-array-ii/discuss/284013/Double-Binary-Search%3A-First-find-pivot-then-find-target>)

Read More

1 ^ v | Share | Reply

SHOW 1 REPLY



(/toma_ofinger)

toma_ofinger (/toma_ofinger) ★ 1 🕒 May 12, 2020 12:09 PM

[Articles](#) > 154. Find Minimum in Rotated Sorted Array II

The solution is correct but not optimal. The thing is that they hurry up to go in linear direction (hi=1). You should always remove one of the halves, when possible. The only time you should be agnostic is when all num[low] == num[pivot] == num[high]. In the code, in the last else branch, you should still check if num[low] == num[pivot]. If it's not, then the minimum has to be in the left half of the array, between num[low] and

[Read More](#)0 ^ v | [Share](#) | [Reply](#)

(/algorithmimplementer)

AlgorithmImplementer (/algorithmimplementer) ★ 387 🕒 May 9, 2020 2:44 AM

Both iterative and recursive solution here (<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-ii/discuss/621899/Java-Clean-Code-with-Comments>)

0 ^ v | [Share](#) | [Reply](#)

(/robotbandit)

RobotBandit (/robotbandit) ★ 4 🕒 January 23, 2020 11:45 AM

https://youtu.be/g_zBbdUiOxM (https://youtu.be/g_zBbdUiOxM)[Read More](#)-1 ^ v | [Share](#) | [Reply](#)

(/itsthemakattack)

itsthemakattack (/itsthemakattack) ★ 65 🕒 February 14, 2020 12:11 PM

Faster than 72%, Python 3.

```
return min(nums)
```

-5 ^ v | [Share](#) | [Reply](#)