

■ ■ Minimum in Rotated Sorted Array

Find the pivot — the smallest element after rotation

● Binary Search

■ MEDIUM

■ What You Will Learn

- What the 'pivot point' in a rotated array looks like
- Why the minimum is always at the rotation point
- How to eliminate halves based on `arr[mid]` vs `arr[right]`
- A cleaner binary search pattern using `right` as reference

■ Intuition First

After rotation, the array looks like two ascending slopes joined at a valley. The minimum is at that valley. Key insight: compare `arr[mid]` with `arr[right]`. If `arr[mid] > arr[right]`, the minimum is in the RIGHT half (the valley is to the right). If `arr[mid] < arr[right]`, the minimum is in the LEFT half (including `mid` itself).

Rotated array visualized:

```

    7
   6
  5
 4   ← rotation point
    3
   2
  1   ← minimum

[4, 5, 6, 7, 0, 1, 2, 3]
  ^valley here^
```

■ Visual — Decision Making

Array: [4, 5, 6, 7, 0, 1, 2]
 L M R

arr[mid]=7 > arr[right]=2
→ minimum is in RIGHT half → L = mid+1

Array: [4, 5, 6, 7, 0, 1, 2]
 L M R

arr[mid]=1 < arr[right]=2
→ minimum is in LEFT half (could be mid itself) → R = mid

Array: [4, 5, 6, 7, 0, 1, 2]
 LR

L == R → nums[L] = 0 is the minimum ✓

■ Java Code

```
1  class Solution {
2  public int findMin(int[] nums) {
3      int left = 0, right = nums.length - 1;
4
5      while (left < right) { // Note: strict < here
6          int mid = left + (right - left) / 2;
7
8          if (nums[mid] > nums[right]) {
9              // Min is in RIGHT half (past mid)
10             left = mid + 1;
11         } else {
12             // Min is in LEFT half (possibly mid itself)
13             right = mid; // Don't do mid-1! mid could be answer
14         }
15     }
16     // left == right, pointing to minimum
17     return nums[left];
18 }
19 }
```

■■ Complexity

Complexity	Value	What it means
Time	$O(\log n)$	Halving each step until convergence
Space	$O(1)$	Constant extra variables only

■ Common Mistakes

■ Mistake 1

Using `right=mid-1` when `arr[mid]<arr[right]`. Since `mid` could be the answer, use `right=mid` to preserve it.

■ Mistake 2

Using `left<=right` loop. This version converges when `left==right`, so use strict `left<right` to avoid infinite loop.

■ Quick Revision

✓	Compare <code>arr[mid]</code> with <code>arr[right]</code> (not <code>arr[left]</code>)
✓	<code>arr[mid] > arr[right]</code> → min is in right half → <code>left = mid+1</code>
✓	<code>arr[mid] < arr[right]</code> → min is in left half (include mid) → <code>right = mid</code>
✓	Loop exits when <code>left == right</code> → <code>nums[left]</code> is the answer
✓	Time $O(\log n)$, Space $O(1)$

■ Visual Thinking

Picture the rotated array as a mountain range: high peaks on the left side, then a sudden cliff down to the valley (minimum), then gentle upward slope on the right. You're looking for the valley floor. At each mid point, if you're still on high ground (`arr[mid] > arr[right]`), the valley is ahead to the right. Otherwise, you're already past the cliff and heading toward or at the valley.