

Search in Rotated Sorted Array

This problem requires $O(\log n)$ runtime, which implies a modified binary search.

Key Idea: Even though the array is rotated, one of the halves is always sorted: - Either $\text{nums}[\text{left} \dots \text{mid}]$ is sorted, or
- $\text{nums}[\text{mid} \dots \text{right}]$ is sorted.

We determine which half is sorted and check if the target lies in that half. If yes, search that half; else search the other half.

Algorithm Steps:

1. Initialize two pointers:

- $\text{left} = 0$, $\text{right} = \text{nums.size}() - 1$

2. While $\text{left} \leq \text{right}$:

- Find $\text{mid} = (\text{left} + \text{right}) / 2$

- If $\text{nums}[\text{mid}] == \text{target} \rightarrow \text{return mid}$

- Determine which half is sorted:

- If $\text{nums}[\text{left}] \leq \text{nums}[\text{mid}] \rightarrow \text{Left half sorted}$

- If $\text{target} \in [\text{nums}[\text{left}], \text{nums}[\text{mid}]]$ search left half;
else search right half

- Else \rightarrow Right half sorted

- If $\text{target} \in [\text{nums}[\text{mid}], \text{nums}[\text{right}]]$, search right half; else search left half

3. Return -1 if not found.

Complexity: Time: $O(\log n)$ (binary search)
Space: $O(1)$