Binary Search

Created	@2025年5月10日 下午1:14
Question Type	Binary Search
Difficulty	Easy
Need to Redo?	No

1. Question Self-understanding:

1.1 Description:

We have a sorted array and want to find the index of a specific target number in that array.

1.2 Input:

A list of integers.

1.3 Input Assumption

A sorted array.

1.4 Output:

An integer representing the index of the target. If the target does not exist, return -1.

1.5 Example:

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

1.6 Other Q&A:

- Could it be an empty list?
    - No, the list is guaranteed to have at least one element.

2. Attempt 1:

2.1 Thought:

Since the array is in ascending order, we can compare the middle element of the current search range with our target:

- If the middle element is less than the target, then the target must be in the right side of the array.
- If the middle element is greater than the target, then the target must be in the left side of the array.
- If the middle element equals the target, we have found our answer.

By repeatedly halving the search range, we can efficiently locate the target's index (or conclude that it doesn't exist).

2.2 Pseudo-Code: (Ignore this part. It's a draft for brainstorming.)

```
BINARY-SEARCH(A, target)
1  left ← 0
2  right ← length(A) - 1
```

```
3  while left ≤ right
4      mid ← ⌊(left + right) / 2⌋
5      if A[mid] = target
6          return mid
7      else if A[mid] < target
8          left ← mid + 1
9      else
10         right ← mid - 1
11 return -1
```

2.3 Implementation through python:

```
class Solution:
    def search(self, nums: List[int], target: int) → int:
        # There is at least one element and if the len is 1, return 0 if the element is e
        if nums[0] == target:
            return 0



        left, right = 0, len(nums) - 1



        while left <= right:

            mid = (left + right) // 2
            if nums[mid] == target:
                return mid

            if nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
```

```
    return -1
```

2.4 Time Complexity and Space Complexity

2.4.1 Time Complexity:

- Each iteration of the loop halves the search space, so the time complexity is $O(log_2(n))$.

2.4.2 Space Complexity:

- We only use a few variables, so the space complexity is $O(1)$.