[Click on the logo to find **Problem Statement**]

Intuition

The code implements a binary search algorithm to efficiently find the index of a target element in a sorted list (nums). The algorithm maintains two pointers, low and high, which represent the current search range. In each iteration of the while loop, it calculates the middle index (mid) and compares the element at that index with the target. If they are equal, the index is returned. If the element at mid is less than the target, the search is narrowed to the upper half by updating low to mid + 1. Conversely, if the element is greater than the target, the search is restricted to the lower half by updating high to mid - 1. This process continues until the target is found or the search range is exhausted. The algorithm's time complexity is O(log n), as it efficiently halves the search space in each iteration.

Approach

1. **Initialize Pointers:**
    - Set `low` to the beginning of the list (`0`) and `high` to the end of the list (`len(nums) - 1`).
2. **Binary Search Iteration:**
    - Enter a while loop that continues as long as `low` is less than or equal to `high` .
    - Calculate the middle index `mid` using `(high - low) // 2 + low` .
    - Compare the element at index `mid` with the target:
        - If they are equal, return the index `mid` as the target is found.
        - If the element at `mid` is less than the target, update `low` to `mid + 1` to search the upper half.
        - If the element is greater than the target, update `high` to `mid - 1` to search the lower half.
3. **Target Not Found:**
    - If the while loop exits without finding the target, return `-1` to indicate that the target is not present in the list.

Complexity

- Time complexity: O(log n)
- Space complexity: O(1)

Code

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

If you want to see more solutions to coding problems, you can visit: