

The task is to implement an algorithm to find the position of a target number in a sorted list of unique integers or where it would need to be inserted. The required time complexity is $O(\log n)$, which suggests using a binary search algorithm.

Python Solution

```
def find_target_position(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        if nums[mid] == target:
            return mid # Target found
        elif nums[mid] < target:
            left = mid + 1 # Search in the right half
        else:
            right = mid - 1 # Search in the left half

    return left # Position where the target should be inserted

# Example usage
nums = [1, 3, 5, 6]
target = 5
print(find_target_position(nums, target)) # Output: 2

target = 2
print(find_target_position(nums, target)) # Output: 1

target = 7
print(find_target_position(nums, target)) # Output: 4
```

Explanation

1. Binary Search:
 - Start with two pointers: left at the beginning and right at the end of the list.
 - Calculate the middle index mid of the current range.
 - Compare the value at nums[mid] with target:
 - If equal, return mid (target found).
 - If nums[mid] < target, move left to mid + 1.
 - If nums[mid] > target, move right to mid - 1.
2. Insertion Point:
 - If the loop ends without finding the target, left will point to the position where the target should be inserted to maintain the sorted order.

Time Complexity

- Time Complexity: due to the binary search.
- Space Complexity: as no additional space is used.

Example Outputs

Input:

nums = [1, 3, 5, 6], target = 5

Output:

2

Input:

nums = [1, 3, 5, 6], target = 2

Output:

1

Input:

```
nums = [1, 3, 5, 6], target = 7
```

Output:

4

Let me know if you need further clarification or enhancements!