Documentation: Search Insert Position Algorithm

Description:

The Search Insert Position algorithm aims to find the index of a target value within a sorted array of distinct integers. If the target value is found within the array, the algorithm returns its index. However, if the target is not present in the array, the algorithm determines the index where the target would be inserted while maintaining the ascending order of the array.

Algorithm Complexity:

The algorithm must achieve a runtime complexity of O(log n), where 'n' represents the number of elements in the input array. Achieving this complexity ensures efficient performance even for large datasets.

Input:

- **nums:** A sorted array of distinct integers.
- **target:** The integer value being searched for or to be inserted into the array.

Output:

- **Index:** The index of the target value within the array if found. If the target is not present, the output indicates the index where the target would be inserted while preserving the array's sorted order.

Algorithm Steps:

1. **Initialization:** Set the left pointer to 0 and the right pointer to the index of the last element in the array.
2. **Binary Search:** Use a binary search approach to continuously narrow down the search range until the target is found or the search range is exhausted.
    a. Calculate the mid index as the average of the left and right pointers.
    b. If the value at the mid index matches the target, return the mid index.
    c. If the value at the mid index is less than the target, update the left pointer to mid + 1, narrowing the search range to the right half.
    d. If the value at the mid index is greater than the target, update the right pointer to mid - 1, narrowing the search range to the left half.
3. **Result Determination:** If the loop terminates without finding the target, return the current position of the left pointer. This indicates the correct index for inserting the target while maintaining the sorted order of the array.

Example Usage:

```
solution = Solution()

nums = [1, 3, 5, 6]

print(solution.searchInsert(nums, 5))  # Output: 2

print(solution.searchInsert(nums, 2))  # Output: 1

print(solution.searchInsert(nums, 7))  # Output: 4
```

Constraints:

- 1 <= nums.length <= 10^4

- -10^4 <= nums[i] <= 10^4

- The input array 'nums' contains distinct values sorted in ascending order.

- -10^4 <= target <= 10^4