2860. Happy Students Medium **Enumeration** Sorting <u>Array</u>

Problem Description

student has to be happy. There are two conditions under which a student will be happy: 1. The student is selected, and the total number of selected students is more than their own specified number (nums[i]). 2. The student is not selected, and the total number of selected students is less than their own specified number (nums[i]).

In this problem, we are dealing with a situation that involves making a group of students happy based on selection criteria. The

array nums represents the number of students in a class, and each element nums[i] implies a certain requirement that the i(th)

- We need to find out how many different ways we can select groups of students so that all students are happy according to their individual requirements.

Intuition The solution to this problem involves sorting the nums array first. This action will group students by their happiness criteria,

allowing us to consider students in sorted order, which is easier to handle.

After sorting the array, we iterate through potential group sizes—from 0 students selected up to all students being selected (hence the range from 0 to n inclusive). During this iteration, we are looking for the positions where the following is true:

• If the current position is i, then all students up to i-1 must require a group size smaller than i (or be included themselves) to be happy, and all students from i onwards must require a group size greater than i to be happy (or not be included). The provided code snippet, however, seems incomplete as it sets up the logic and loop for counting valid ways to select students

but lacks the actual implementation for updating the ans variable, indicating the number of ways to select students. Therefore, the reference solution approach is necessary to understand the complete logic for properly incrementing ans based on valid selections. Without this crucial part of the logic, we cannot deduce the correct number of ways to fulfill the problem statement requirements.

The correct approach will evaluate each position to check whether it satisfies the conditions mentioned above. If a valid position

is found, it should update the count of happiness-satisfying selections (ans). We also need to handle edge cases, for example,

when groups can't be formed because all students have a happiness criteria that's higher or lower than possible group sizes.

To implement the solution for this problem, we need a way to count scenarios where a selection of students makes all students happy. Since the original solution provided is incomplete, let's infer the approach that would lead to a complete solution. After sorting the nums array in non-decreasing order, we perform a sweep across potential group sizes. This requires comparing

the group size i at each step to the numbers in the sorted array and deciding if students are happy. Below is the pattern we can

Initialize a count variable (ans) to store the total number of ways to select students, starting at 0.

follow for a complete solution:

after the i(th) student:

Solution Approach

Sort the array nums to make a linear sweep feasible. Iterate through the array, checking each possible group size from 0 to n, inclusive.

To analyze each group size i, determine if selecting i students will satisfy both happy conditions for students before and

 For students at index greater than or equal to i (not selected), their happiness criterion nums[i] should be strictly greater than i. Only if the current group size satisfies both criteria, it implies a valid configuration. Increment the ans counter for each such case.

• For students at index less than i (selected students), they should all have a happiness criterion nums[i] that is less than or equal to i.

programming languages, giving O(n log n) complexity for sorting the array. Linear Sweep pattern: After sorting, we iterate over each element once, giving us an O(n) complexity for the sweep. The complete solution, therefore, will have an overall time complexity of O(n log n) due to the sorting with an additional O(n)

Let's consider a small example to illustrate the solution approach. Assume we have the following array nums that indicates the

• Sorting algorithm: We use the built-in sort function, which is likely implemented as a variant of quicksort, heapsort, or mergesort in most

Example Walkthrough

sweep, resulting in O(n log n) total time complexity.

Return the ans as the count of all valid selections.

nums = [1, 2, 3, 4]

Student 0 wants more than 1 student to be selected.

Student 1 wants more than 2 students to be selected.

• Student 2 wants more than 3 students to be selected.

• Student 3 wants more than 4 students to be selected.

students' happiness criteria:

In terms of algorithms and data structures:

Following the solution approach: Initialize ans to 0.

All students have a happiness criterion greater than 0, so this configuration makes all students unhappy.

Student 0 is selected. This doesn't satisfy Student 0's happiness criteria (nums [0] = 1) since 1 is not more than 1.

According to the problem, there are 4 students in the class, and they will be happy under the following conditions:

```
Iterate through the possible group sizes:
    With a group size of 0 (i = 0):
0
```

Sort the array nums in non-decreasing order. However, in our example, the array is already sorted, so no action is required.

 Students 0 and 1 are selected. This satisfies both of their criteria since 2 is more than 1 (nums [0]) and 2 is more than 2 (nums [1]). • Students 2 and 3 are not selected. This satisfies their criteria since 2 is less than 3 (nums [2]) and 4 (nums [3]).

Thus, the final ans is 2.

from typing import List

nums.sort()

length = len(nums)

continue

continue

Python

class Solution:

With a group size of 1(i = 1):

With a group size of 2 (i = 2):

With a group size of 3 (i = 3):

def count ways(self, nums: List[int]) -> int:

Get the length of the sorted list

if i > 0 and nums[i - 1] >= i:

if i < length and nums[i] <= i:</pre>

public int countWays(List<Integer> nums) {

Collections.sort(nums);

int n = nums.size();

int answer = 0;

// Get the size of the list

// Initialize answer count to 0

for (int i = 0; $i \le n$; i++) {

return count_distinct_ways;

function countWays(nums: number[]): number {

// Initialize the count of ways to 0.

// Get the length of the nums array.

for (let i = 0; i <= arrayLength; ++i) {</pre>

// Return the total count of valid ways.

def count ways(self, nums: List[int]) -> int:

for i in range(length + 1):

continue

continue

Time and Space Complexity

return answer

const arrayLength = nums.length;

nums.sort((a, b) => a - b);

continue;

++waysCount;

return waysCount;

from typing import List

nums.sort()

let waysCount = 0;

// Sort the input array in ascending order.

// Iterate through the array including a position at the end.

// If conditions are met, increment the count of ways.

Sort the input list to make it easier to process

Iterate over all elements in the sorted list

if i > 0 and nums[i - 1] >= i:

if i < length and nums[i] <= i:</pre>

};

TypeScript

// Sort the list in non-decreasing order

Sort the input list to make it easier to process

This configuration does not make all students happy.

This configuration makes all students happy. Increment ans to 2. With a group size of 4 (i = 4):

Student 3 is not selected. This satisfies their criterion since 3 is less than 4 (nums [3]).

■ This configuration makes all students happy. Increment ans to 1.

■ No student is selected, which means all the students' conditions are checked.

 All students are selected. Student 3's criteria are not satisfied since 4 is not more than 4 (nums [3]). This configuration does not make all students happy.

4. After iterating through all possible group sizes, we find that there are 2 valid selections that make all students happy.

• Students 0, 1, and 2 are selected. This satisfies their criteria since 3 is more than 1 (nums [0]), 2 (nums [1]), and 3 (nums [2]).

- Solution Implementation
 - # Initialize the answer (the number of ways) to zero answer = 0 # Iterate over all elements in the sorted list for i in range(length + 1):

If not the first element, and the previous number is greater than or equal to the index, skip

If not the last element, and the current number is less than or equal to the index, skip

At this point the answer is not changed by the loop, this loop has no effect.

* @return Count of the possible ways numbers can be arranged fulfilling the condition

// Also, consider the cases when the index is at the start or the end of the list

The function currently will always return 0 as it stands.

* Counts the ways in which numbers can be assigned to their indices

* @param nums List of Integer values presumably between 0 and list size

* in the list such that each number is greater than its index.

// Iterate through all possible positions in the list

Java import java.util.Collections; import java.util.List;

class Solution {

/**

return answer

```
if ((i == 0 \mid | nums.qet(i - 1) < i) && (i == n \mid | nums.qet(i) > i)) {
                // Increment the answer if the condition is satisfied
                answer++;
        // Return the total count of valid ways
        return answer;
C++
#include <vector>
#include <algorithm> // For sort()
class Solution {
public:
    // Function to count the number of distinct ways to form index-value pairs.
    int countWays(vector<int>& nums) {
        // Sort the input vector of numbers in non-decreasing order.
        sort(nums.begin(), nums.end());
        // Initialize the answer (count of distinct ways) to 0.
        int count_distinct_ways = 0;
        // Get the number of elements in nums.
        int n = nums.size();
        // Loop through the numbers from 0 to n (inclusive at the beginning and exclusive at the end).
        for (int i = 0: i < n: ++i) {
            // Skip the current number if:
            // - It's not the first number and the previous number is greater or equal to the current index.
            // Or
            // - It's not the last number and the current number is less or equal to the current index.
            if ((i > 0 \& nums[i - 1] >= i) || (i < n - 1 \& nums[i] <= i)) {
                continue;
            // If none of the above conditions are met, increment the count of distinct ways.
            ++count_distinct_ways;
        // Return the total count of distinct ways.
```

// Skip the current iteration if the value at the previous index is greater than or equal to the current index

// OR if the value at the current index is less than or equal to the current index.

if $((i > 0 \& nums[i - 1] >= i) || (i < arrayLength \& nums[i] <= i)) {$

// Check if the current number is greater than it's index (after considering zero-based index adjustment)

Get the length of the sorted list length = len(nums)# Initialize the answer (the number of ways) to zero answer = 0

class Solution:

```
The overall time complexity of the countways function is determined by the sorting operation and the for loop.
   The sort() method applied on nums list is the most costly operation in this snippet. The sort function in Python uses
   Timsort, which has a worst-case time complexity of O(n log n) where 'n' is the length of the list.
   The for loop runs from 0 to n + 1 where 'n' is the length of the nums list. However, the loops body has continue statements
```

If not the first element, and the previous number is greater than or equal to the index, skip

If not the last element, and the current number is less than or equal to the index, skip

At this point the answer is not changed by the loop, this loop has no effect.

not clear from the provided code, and its time and space complexity are discussed as it is.

The function currently will always return 0 as it stands.

case of O(n). **Space Complexity**

The space complexity of the function is determined by the storage requirements that are not directly dependent on the input

Considering the above points, the dominant part in terms of time complexity is the sorting operation. Therefore, the overall time

complexity of the function is 0(n log n) due to the sort, irrespective of the for loop, which has a best case of 0(1) and a worst

The given Python code snippet is meant to count the number of ways or a particular quantity relating to a list of integers.

However, the function countways does not include logic to increment the ans variable, thus the actual purpose of the function is

The nums list is sorted in place, so no additional space is necessary for sorting beyond the space already used to store nums. The variable ans and the loop variable i each take constant space.

Time Complexity

size. In the snippet provided:

which is constant space.

which may terminate the iterations early without performing any additional operations. In the worst case, the loop runs n + 1 times.

As there are no additional data structures used that grow with the size of the input, the space complexity of the function is 0(1),