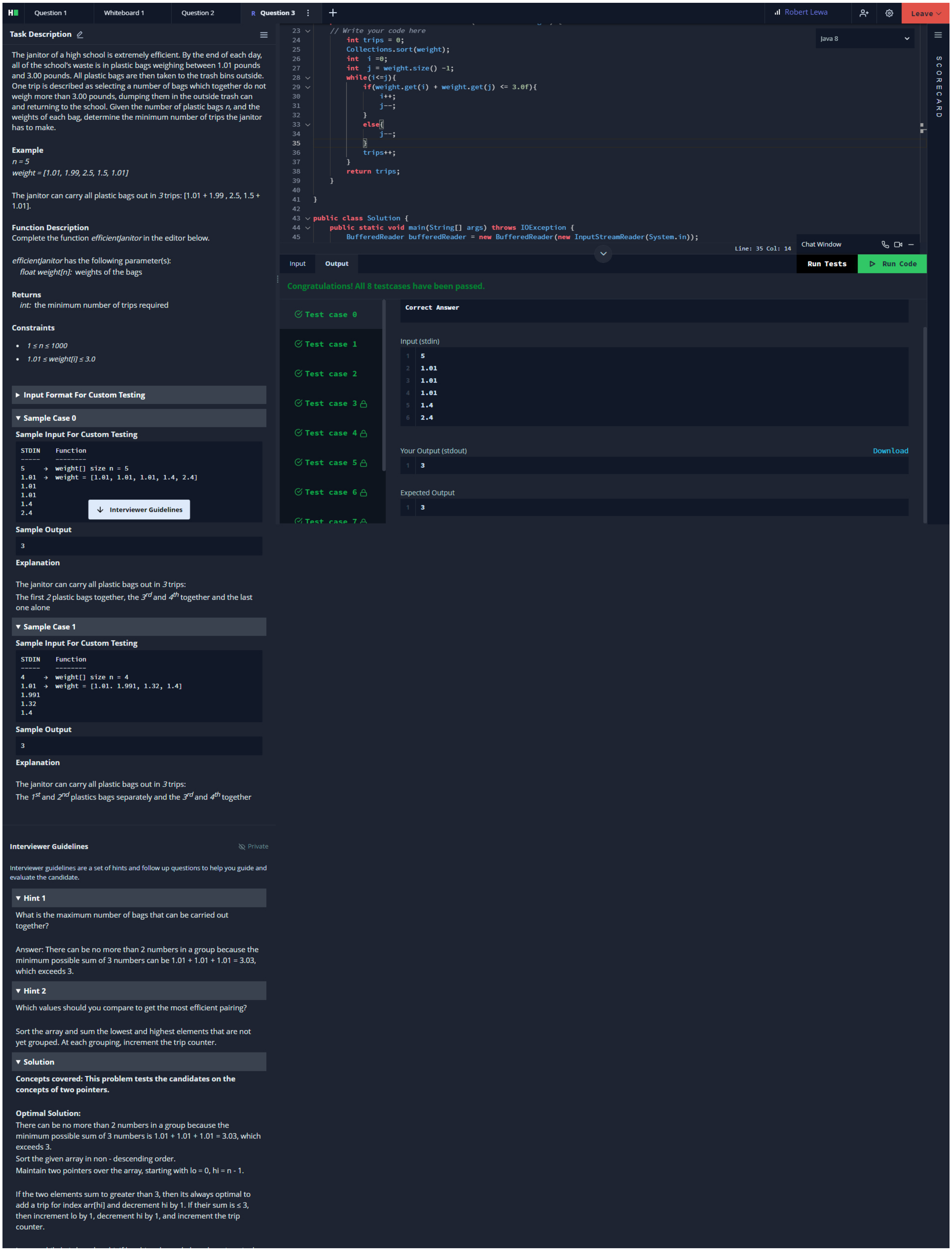

Task Description

The janitor of a high school is extremely efficient. By the end of each day, all of the school's waste is in plastic bags weighing between 1.01 pounds and 3.00 pounds. All plastic bags are then taken to the trash bins outside. One trip is described as selecting a number of bags which together do not weigh more than 3.00 pounds, dumping them in the outside trash can and returning to the school. Given the number of plastic bags *n*, and the weights of each bag, determine the minimum number of trips the janitor has to make.

Example

n = 5

weight = [1.01, 1.99, 2.5, 1.5, 1.01]

The janitor can carry all plastic bags out in *3* trips: [1.01 + 1.99 , 2.5, 1.5 + 1.01].

Function Description

Complete the function *efficientJanitor* in the editor below.

efficientJanitor has the following parameter(s):

float weight[n]: weights of the bags

Returns

int: the minimum number of trips required

Constraints

- *1 ≤ n ≤ 1000*
- *1.01 ≤ weight[i] ≤ 3.0*

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
5     →  weight[] size n = 5
1.01  →  weight = [1.01, 1.01, 1.01, 1.4, 2.4]
1.01
1.01
1.4
2.4
```

Sample Output

```
3
```

Explanation

The janitor can carry all plastic bags out in *3* trips:

The first *2* plastic bags together, the *3rd* and *4th* together and the last one alone

▼ **Sample Case 1**

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
4     →  weight[] size n = 4
1.01  →  weight = [1.01. 1.991, 1.32, 1.4]
1.991
1.32
1.4
```

Sample Output

```
3
```

Explanation

The janitor can carry all plastic bags out in *3* trips:

The *1st* and *2nd* plastics bags separately and the *3rd* and *4th* together

```java
    // Write your code here
        int trips = 0;
        Collections.sort(weight);
        int  i =0;
        int  j = weight.size() -1;
        while(i<=j){
            if(weight.get(i) + weight.get(j) <= 3.0f){
                i++;
                j--;
            }
            else{
                j--;
            }
            trips++;
        }
        return trips;
    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
```

Line: 35 Col: 14

Congratulations! All 8 testcases have been passed.

- ✓ Test case 0
- ✓ Test case 1
- ✓ Test case 2
- ✓ Test case 3
- ✓ Test case 4
- ✓ Test case 5
- ✓ Test case 6
- ✓ Test case 7

Correct Answer

Input (stdin)

```
5
1.01
1.01
1.01
1.4
2.4
```

Your Output (stdout)

```
3
```

Expected Output

```
3
```

Interviewer Guidelines

Private

Interviewer guidelines are a set of hints and follow up questions to help you guide and evaluate the candidate.

▼ **Hint 1**

What is the maximum number of bags that can be carried out together?

Answer: There can be no more than 2 numbers in a group because the minimum possible sum of 3 numbers can be 1.01 + 1.01 + 1.01 = 3.03, which exceeds 3.

▼ **Hint 2**

Which values should you compare to get the most efficient pairing?

Sort the array and sum the lowest and highest elements that are not yet grouped. At each grouping, increment the trip counter.

▼ **Solution**

Concepts covered: This problem tests the candidates on the concepts of two pointers.

Optimal Solution:

There can be no more than 2 numbers in a group because the minimum possible sum of 3 numbers is 1.01 + 1.01 + 1.01 = 3.03, which exceeds 3.

Sort the given array in non - descending order.

Maintain two pointers over the array, starting with lo = 0, hi = n - 1.

If the two elements sum to greater than 3, then its always optimal to add a trip for index arr[hi] and decrement hi by 1. If their sum is ≤ 3, then increment lo by 1, decrement hi by 1, and increment the trip counter.

Iterate while lo is less than hi. If lo = hi at the end, then there is a single element that must be in its own trip.

```
def efficientJanitor(weight):
    weight.sort()
    n = len(weight)
    lo = 0
    hi = n - 1
    ans = 0
    while(lo < hi):
        # too big to pair, so decrement hi
        if(weight[lo] + weight[hi] > 3.0):
            hi = hi - 1
        # small enough to pair, so increment lo, decrement hi
        else:
            lo = lo + 1
            hi = hi - 1
        # increment trip counter in either case
        ans = ans + 1
    # a bag is left, so 1 more trip
    if(lo == hi):
        ans = ans + 1
    return ans
```

Error Handling:

1. It is important to sort the array in non - descending order.
2. Decrement the right pointer by 1 if the current pair forms a sum greater than 3.
3. If both the pointers are equal, there is a bag left so increment the trip counter.

▼ Complexity Analysis

Time Complexity - O(N log N).
Sorting the array takes O(NlogN) time, and then the two pointers traversal take O(N) time only.

Space Complexity - O(1) - No extra space is required.