Problem 1: Operation Sea Evac — The Last Boats from Seagull Island
A sudden tropical cyclone hits Seagull Island, where a music festival was in full swing. The tide is rising fast, and the ferry system is down. You're Captain Arjun, the emergency rescue chief, and your mission is clear:
Evacuate every remaining person before the floodwater cuts off the last dock.

You have a limited number of small rescue boats, each capable of carrying up to 2 people, but there's a catch — the boats can't carry more than a certain weight limit.

The islanders vary in size, and not everyone can be paired together. Time is ticking. You must calculate the minimum number of boats required to rescue everyone safely and swiftly.

Input:

- N: total number of stranded people
- limit: maximum weight a boat can carry
- W[i]: list of weights of the people

Constraints:

- Each boat can carry at most two people
- The combined weight must not exceed limit
- No person can be left behind

Input:

N = 4

limit = 100

W = [70, 50, 80, 50]

Output:

3

Solution :-

To solve this problem , we need to determine the minimum number of boats required to evacuate all people from the island :-

Sort the list of weights -Initialize two pointers-pair people-count boats-Terminate when all are

Pseudocode in Java :-

```
Public class BoatCalculator {
Public static int minBoats(int N, int limit,
                           int[] w){
Arrays.sort(w);
int left = 0;
int right = N-1;
int boats = 0;

while (left <= right) {
if (w[left] + w[right] <= limit){
left++;
}
right--;
boats++;
}
return boats;
}
Public static void main (String[] args){
int N = 4;
int limit = 100;
int[] w = {70, 50, 80, 50};
System.out.println(minBoats(N, limit,
                            w));
}
}
```

Dry Run :-

:- N = 4

:- limit = 100

:- w = [70, 50, 80, 50].

(1.) Sort the weights :- [50, 50, 70, 80].

(2.) Initialize pointer :- l = 0, r = 3, b = 0.

(3.) 1st Iteration (l = 0, r = 3):

if w[0] + w[3] <= 100 → 50 + 80 = 130 > 100

right → 2, boats++ → 1

(4.) 2nd Iteration (l = 0, r = 2);

if w[0] + w[2] <= 100 → 50 + 70 = 120 > 100

(5.) 3rd (l = 0, r = 1);

if w[0] + w[1] <= 100 → 50 + 50 = 100 <= 100

→ can pair.

→ left++ → 1, right-- → 0, boats++ → 3.

:- left(1) > right(0) → loop's end.

Total boats used = 3.

Problem 2: Festival Fallout — Saving Seats in the Rain

After the flood scare, the Seagull Island Festival has resumed in the island's community hall on higher ground. The weather has calmed, but now you face a new challenge.

You're now Event Manager Meera, responsible for rearranging the audience seating. Groups of attendees are arriving — friends, families, music fans — and each group refuses to split up.

The hall has rows of fixed size, each with exactly C chairs. You must assign one row per group, without breaking them apart. Your goal is to make the best use of space and minimize the number of empty chairs left unused.

Input:

- N: number of groups
- C: number of chairs per row
- G[i]: array of group sizes

Constraints:

- A group must be assigned a full row
- Groups cannot be split
- -

Input:

N = 3

C = 5

G = [3, 5, 4]

Output:

3

Solution :-

To solve this problem , we need to assign each group to a seprate row such that the number of empty chairs left unused is minimized.

:-sort the group sizes in descending order – assign each group sizes to a separate Row – Calculate the total empty chairs

Pseudocode :

```
Import java.util.Arrays;
Import java.util.Collections;

Public class FestivalSeating {
Public static int minEmptyChairs(int N, int C,
                                 int[] G){

Integer[] groups = new Integer [N];
For (int i = 0; i < N; i++) {
 groups[i] = G[i];
}.
Arrays.sort(groups, Collection.reverseOrder());
int emptyChairs = 0;
for (int i = 0; i < N; i++) {
emptyChairs += C - groups[i];
}
return emptyChairs;
}.
Public static void main (String[]args){
int N = 3;
 int C = 5;
 int[] G = {3, 5, 4};
op (minEmptyChairs(N, C, G));
}}
```

Dry Run :-

Input :- N = 3

C = 5

G = [3, 5, 4]

(1.) Sort in Descending Order

:- [5, 4, 3]

(2.) Assign Separate row.

(i) 1st group (5) :-

Empty Chairs = 5 - 5 = 0

(ii.) 2nd group (4) :- 5 - 4 = 1

(iii.) 3rd group (3) :- 5 - 3 = 2

(3.) Calculate Total Empty Chairs -

0 + 1 + 2 = 3

Result = 3.