Beiðni um endurgjöf (feedback) - tímabil 2 (9. sept - 27. sept)

Code Feedback: Problem J | Star Wars Sorting
Who started the project: Donna
Brief explanation of the task:
The task was to rearrange a list of numbers in a specific order, referred to as “star wars-order.” The list is divided into three parts: the first third, the center third, and the last third. The challenge was to move the center third to the front, place the first third in the center, and leave the last third in place.
Brief explanation of the solution idea:
The solution was to use list slicing to divide the input list into three parts. Then, the parts were rearranged according to the specified order and printed. Specifically:

1. The list is divided into three equal parts.
2. The center third is moved to the front.
3. The first third is placed after the center, and the last third remains at the end.

If any problems arose:

- **What was the problem?**

Initially, I was concerned about how to handle dividing the list correctly, especially ensuring that the list was split into thirds properly,
given the constraint that the number of elements is always a multiple of three.

- **How was it solved?**

I solved this by using Python’s list slicing feature, which makes it easy to divide the list into segments. After calculating the size of
one-third of the list, I used slicing to split it and then rearranged the parts to match the specified order.

Solution #1 (Unsuccessful on Kattis)

```
n = int(input())  # Number of elements in the list
numer = list(map(int, input().split()))  # List of numbers


third = n // 3


# Slice the list into three parts: center, first, and last
first_third = numer[:third]
center_third = numer[third:2*third]
last_third = numer[2*third:]
```

```
# Rearrange them: center + first + last
result = center_third + first_third + last_third

# Print the rearranged list
print(" ".join(map(str, result)))
```

Issue:
I thought I might be handling the slicing and rearranging of the list incorrectly.

- **My output:** 6 55 8 34 1 3
- **Reference output:** 6 8 1 3 34 55

Upon reviewing my code, I realized that the initial implementation was correct in terms of slicing and rearranging. However, I needed to modify the code to first sort the input list before splitting it into thirds and then rearranging the segments.

Solution #2 (Accepted on Kattis)

```
n = int(input())  # Number of elements in the list
numer = list(map(int, input().split()))  # List of numbers

numer.sort()

third = n // 3

# Slice the list into three parts: center, first, and last
first_third = numer[:third]
center_third = numer[third:2*third]
last_third = numer[2*third:]

# Rearrange them: center + first + last
result = center_third + first_third + last_third

# Print the rearranged list
print(" ".join(map(str, result)))
```

Do I think this solution is good?
Yes, I think the solution is good because it's simple and works well.
Simple and efficient:
The solution uses Python's built-in tools like list slicing and sorting, which are easy to use and do the job without making things
Too complicated. Sorting the list before dividing it into thirds ensures everything is in the right order, which is the key to making
it work.
Good use of list slicing:
The code correctly splits the list into three equal parts and then rearranges them as the problem asks. The slicing part is done
Really well and it's straightforward.
Sorting before splitting:
The most important change was realizing that the list had to be sorted first before rearranging the parts. Without this step, the
output wouldn't match the expected result. Adding the sort step fixed the problem and made the solution correct.
What I'd highlight as good:
The sorting step was the key fix. By sorting the list first, everything else worked smoothly and gave the right output.

I'd like a bit more clarification on how slicing works with different list sizes or if the number of elements wasn't always a multiple of three. Would the solution still work in that case?
I'm also open to other suggestions! If there's a more efficient or alternative way to solve this problem, I'd be interested in learning about it.