4634 !Overflow

You are responsible for implementing a driver to communicate telemetry data to an external device. The particular external device in question happens to use a very basic microprocessor that only has an 8-bit accumulator register. During normal operations, your program will send a short sequence of signed 8-bit values to the external device (between 1 and 10 such numbers in any given sequence).

The external device will compute the sum of all the values in a given sequence, by starting with the first one, then adding the second to it, then adding the next one to the result, and so on. Unfortunately, you have discovered (after extensive debugging after midnight) that the microprocessor of the external device crashes if you send it particular sequences of values. More debugging revealed that any sequence that produces a result outside of the range $-128..127$ during the calculation of the sum will cause the microprocessor of the external device to crash. (Your favourite hypothesis on this behaviour is that the developer of the software of the external device forgot to turn off the processor exception caused by the overflow, and that no interrupt handler was set up.)

Since you can not fix the external device, you have to code a workaround on your side. The simplest solution seems to be to reorder the sequence to guarantee that all intermediate values stored in the accumulator while computing the sum are in the range $-128..127$. But first you have to write a program to count how many valid orderings of the sequence can be formed.

Input

Your input will consist of an arbitrary number of records (fewer than 20), presented as one record per line. Each record will start with the sequence length n, followed by n integers in the range $-128..127$. No integer will appear twice in the same sequence (record).

The end of the input is indicated by a line starting with the value '0' (i.e., $n = 0$. For other input records the value of n will be in the range 1..10.

Output

For each input record, your program must produce the appropriate output. An permutation of the input numbers is considered valid if the value stored in the accumulator never exceeds the range $-128..127$ during any step of the calculation.

If one or more valid permutations of the input sequence exist, your program must output a message of the form

k `valid permutations found.`

where k is the actual number of valid permutations found for this input sequence.

If no valid permutations were found your program must output the message

`No valid permutations found.`

Sample Input

```
4 10 126 -30 20
4 10 126 2 3
1 20
0
```

Sample Output

```
14 valid permutations found.
No valid permutations found.
1 valid permutations found.
```