In a given integer array A, we must move every element of A to either list B or list C. (B and C initially start empty.)

Return true if and only if after such a move, it is possible that the average value of B is equal to
the average value of C, and B and C are both non-empty.

Note:
- 1 <= len(A) <= 30
- 0 <= A[i] <= 10 000

First let's do some notations:
A: the initial array
sum(X): the sum of the elements in array X
len(X): the number of elements in array X
B and C: the arrays in which we splitted A

We have:
sum(A) = sum(B) + sum(C) and len(A) = len(B) + len(C)

We also know that the averages of B and C are equal:
sum(B) / len(B) = sum(C) / len(C) (replace the RHS from formulas above)
sum(B) / len(B) = (sum(A) - sum(B)) / (len(A) - (len(B)) (multiply over diagonals)
sum(B) * (len(A) - len(B)) = (sum(A) - sum(B)) * len(B)
sum(B) * len(A) - sum(B) * len(B) = sum(A) * len(B) - sum(B) * len(B)
sum(B) * len(A) = sum(A) * len(B)
sum(B) = sum(A) * len(B) / len(A) (property of a partition with respect to our initial array A)

Our problem is reduced to finding one partition B in which the sum of the elements is equal to sum(A) * len(B) / len(A)

As a first step, we can check only for one partition, and the possible lengths for that partition are in range [1, n / 2], because after exceeding n / 2 we have the other partition in range [1, n / 2]

As a second step, we need a data structure to store all the possible sums of a subarray up to a certain point.
We can use a list of sets, in which l[i] represents all the possible sums of i elements.
In the end, we need to check if the sum sum(A) * len(B) / len(A) is in a valid group i.

Ex: [1, 2, 3, 4]

0							
1	2	3	4				
3	4	5	6	7			