Knapsack

You are very lucky to win a free shopping voucher. You quickly grab the catalog of that shop to check what items are sold in that shop. Intuitively, you want to go back home with the best combination of items that you can get. Unfortunately, you do not know the price of each item; what you know is just the weight of all items and you begin to wonder what kind of catalog is this.

Being the avid shopper as you always have been, you want to list down all possible sets of items that can be put into your limited sack, i.e. the total weight of a set of items should not exceed the capacity of your sack, otherwise your sack will tear down and you will get nothing. You do not really care what items you take, but you just want to know how many different sets of items you can put in your precious knapsack.

Input

The first line of the input contains 2 integers N (1 <= N <= 25), denoting the number of items in the shop and K (1 <= K <= 100), denoting the capacity of your sack. The next line contains N integers, with the i^{th} integer denoting the weight of the i^{th} item. The weight of an item will not exceed 500.

Output

Print the number of different sets of items that can be put in your sack.

Sample Input 1

```
5 6
3 1 6 4 5
```

Sample Output 1

```
9
```

Sample Input 2

```
3 1
4 3 7
```

Sample Output 2

```
1
```

Explanation

For Sample Input 1, please refer to the following table.

NO.	Items in Sack (denoted by weight)	Total Weight
1	Empty Sack	0
2	3	3
3	3, 1	4
4	1	1
5	1, 4	5
6	1, 5	6
7	6	6
8	4	4
9	5	5

For Sample Input 2, the only possible way is to leave your sack empty. How unfortunate!

Skeleton

You are given the skeleton file `Knapsack.java`.

Notes

1. You must use **recursion** to solve this problem.