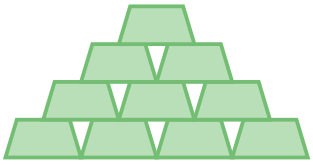
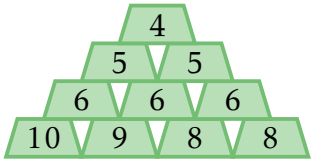
5.2.6 Maximum Amount of Gold

Maximum Amount of Gold Problem
Given a set of gold bars of various weights and a backpack that can hold at most W pounds, place as much gold as possible into the backpack.

Input: A set of n gold bars of integer weights $w_1, \ldots, w_n$ and a backpack that can hold at most W pounds.
Output: A subset of gold bars of maximum total weight not exceeding W.



You found a set of gold bars and your goal is to pack as much gold as possible into your backpack that has capacity W, i.e., it may hold at most W pounds. There is just one copy of each bar and for each bar you can either take it or not (you cannot take a fraction of a bar). Although all bars appear to be identical in the figure above, their weights vary as illustrated in the figure below.



A natural greedy strategy is to grab the heaviest bar that still fits into the remaining capacity of the backpack and iterate. For the set of bars shown above and a backpack of capacity 20, the greedy algorithm would select gold bars of weights 10 and 9. But an optimal solution, containing bars of weights 4, 6, and 10, has a larger weight!

Input format. The first line of the input contains an integer W (capacity of the backpack) and the number n of gold bars. The next line contains n integers $w_1, \ldots, w_n$ defining the weights of the gold bars.

Output format. The maximum weight of gold bars that fits into a backpack of capacity W.

Constraints. $1 \le W \le 10^4$; $1 \le n \le 300$; $0 \le w_1, \ldots, w_n \le 10^5$.

Sample.

Input:

```
10 3
1 4 8
```

Output:

```
9
```

The sum of the weights of the first and the last bar is equal to 9.

Iterative vs Recursive Dynamic Programming Algorithms

In many cases, an iterative approach is preferable since it has no recursion overhead and it may allow saving space by exploiting a regular structure of the table. Still, a recursive approach has its own advantages and may be even preferable for some problems. First, it may be faster if not all the subproblems need to be solved. For example, whereas it is clear that the dynamic programming algorithm for finding the edit distance of two strings needs to solve all subproblems (that is, to find the edit distance of all its prefixes), it is not the case for the Maximum Amount of Gold Problem. Indeed, if the weight of all bars is divisible by, say, ten, then we are just not interested in the values of $pack(w, i)$ when w is not divisible by ten. Still, our iterative algorithm for the problem computes $pack(w, i)$ for all pairs (w, i)! At the same time, the recursive algorithm computes the values of $pack(w, i)$ for only those (w, i) that are needed for computing the final value. For the special case when all weights are divisible by ten, the iterative algorithm is about ten times slower than the recursive one. Second, if recursion overhead is not that important, a recursive algorithm may be easier to implement: one just uses a hash table and converts (almost mechanically) a recurrence relation into a recursive algorithm. When doing this, one does not have to think about data structures for storing the solutions for subproblems and an order of solving subproblems.