The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

In our case however, we’re not working with weight but rather with volume, given that we have to fit certain parcels into the container the volume is our restriction instead of the weight.

The knapsack problem can be defined in 3 ways: the 0-1 knapsack problem, the bounded knapsack problem and the unbounded knapsack problem.

The unbounded knapsack problem is the one we’ll need for this project.

It is defined such that we:

Maximize with the value of a certain parcel, and the amount of said parcel. (The sum of all the values in the container)

Subject to with the weight of each object, in our case we’d replace this with volume, so:

However it’s a bit more complicated than that, because not only does the sum of the total volume of our parcels need to be smaller than the volume of our container (obviously), we’d have to make sure it fits (because we can’t saw any boxes in half).

There is no polynomial-time algorithm for this problem.

An algorithm is said to be solvable in polynomial time if the number of steps required to complete the algorithm for a given input is for some nonnegative integer , where is the complexity of the input. Polynomial-time algorithms are said to be "fast."

is defined as all algorithms which solve a problem in steps or less. So a problem is solvable in polynomial time if it’s solvable in less than steps (I am not entirely sure what the k stands for, because that could obviously drastically change the number of steps, so I assume it can’t be random).

There is a [pseudo-polynomial time](https://en.wikipedia.org/wiki/Pseudo-polynomial_time) algorithm using [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming).