Knapsack Problem
CSC-014
Prof. Ostheimer

This project is designed to introduce you to algorithm design. The particular algorithm you will be studying will be important when we study cryptography. Have fun. Get to know each other. Think.

Our goal today is to write an algorithm for a computer program to solve a problem known as the *Knapsack Problem* and to analyze how long it will take such a computer program to run.

We can describe this problem by describing the desired input and output:

- **Input**: a list of positive integers $x_1, x_2, \ldots, x_n$, and another positive integer y.
- **Output**:
  - "Yes" if some of the x's can be summed to y (using a given x at most once)
  - "No" if not

This problem is called the *Knapsack Problem* for the following reason. Suppose you have a knapsack that has capacity y, and you have n objects with volume $x_1, x_2, \ldots, x_n$ respectively. Your goal is to pick which objects to put in your knapsack so as to fill the knapsack completely.

Part I: Understanding the problem specification

1. Give several different examples of input which produce "Yes" for output.
2. Give several different examples of input which produce "No" for output.
3. Give several different examples of illegal input.

Part II: Developing an algorithm Describe a step-by-step recipe (this is what we call an *algorithm*) for solving the Knapsack Problem. You don't need to write this down, but you do need to be able to describe the process completely. Imagine that your 8 year-old sister is going to take a quiz tomorrow in school in which her teacher gives her values for $x_1, x_2, \ldots, x_5$ and y and asks her if the Knapsack Problem would yield a "Yes" or "No" answer for those particular x's and y. Your little sister should be able to get the right answer on the quiz by following your instructions.

Let me know when you have completed this stage of the group work. We need to talk as a class some before you are ready to move on to the next part.

Part III: Analyzing the speed of the algorithm Here we will analyze the algorithm known as the Exhaustive Algorithm. By "analyze" we mean that we will figure out how long it will take to run.

1. Come up with 3 numbers x_1, x_2, x_3 and another number y that would produce the output "No".
2. List all the possibilities that must be checked in the case above to verify that the output should be "No".
3. How many possibilities are there in this case?
4. How many additions are required to find the sum in *each* of those possibilities? What is the maximum number of additions needed for one of the possibilities?
5. In the example you came up with, $n = 3$. How many possibilities need to be examined when $n = 4$? $n = 5$?
6. Come up with a formula using n that describes the total number of possibilities in general.
7. At most how many additions are required to find the sum of *one* of those possibilities? Give a formula in terms of n.
8. If a computer can do a billion additions per second, at most how long will it take to sum up *one* of those possibilities? Give a formula in terms of n.
9. How long will it take to do the additions for *all* of the possibilities? Again, come up with a formula in terms of n. It's OK to overestimate here, but be sure you understand the sense in which you are overestimating.
10. If $n = 100$, how long will the Exhaustive Algorithm take? Be sure to choose a sensible measure of time here. For example, if the number of seconds is very large, translate to minutes. If that's still large, translate to hours, and so forth.