Algorithms Activity 11: 2D Dynamic Programming

Model 1: Some sets

$$A = \{1, 2, 3, 5, 7\}$$
$$B = \{4, 16, 19, 23, 25, 72, 103\}$$
$$C = \{3, 34, 4, 12, 5, 2, 99\}$$

1 For each number below, say whether each set has some subset which adds up to the given number. For example, A and C have subsets which add up to 7 ($\{7\}$ and $\{5, 2\}$ respectively), but B does not.

(a) 149

(b) 148

(c) 9

(d) 16

In general, consider the following problem, called the Subset Sum problem:

- **Input**:
  - a set $\{x_1, \dots, x_n\}$ of n positive integers, and
  - a positive integer S.
- **Output**: is there a subset of $\{x_1, \dots, x_n\}$ whose sum is exactly S?

2 Describe a brute-force algorithm for solving this problem.

3 What is the running time of your brute-force algorithm?

Let's see how to attack this problem using dynamic programming.
Step 1: Break the problem into subproblems and make a recurrence.

- We can make the problem simpler by restricting ourselves to only using *some* of the x_i. For example, a subproblem might look like "Can we find a subset of only $\{x_1, \dots, x_k\}$ that adds up to S?" for some $k \le n$.
- However, by itself this doesn't help: just knowing whether we can add up to S using only $x_1, \dots, x_k$ doesn't tell us whether we can add up to S using $x_1, \dots, x_n$. In particular, in order to add up to S we might need to use some of the elements from $x_1, \dots, x_k$ in addition to some of the other elements. We can fix this by generalizing along another dimension as well: we need to know whether we can add up not just to S itself, but to *any* sum $0 \le s \le S$. That is, a subproblem now looks like "Can we find a subset of only $\{x_1, \dots, x_k\}$ that adds up to s?" for some $k \le n$ and $s \le S$.

Define $\mathit{canAddTo}(k, s)$ to be a true or false value which is the answer to the above question.

4 Fill in base cases for *canAddTo*:

- $\mathit{canAddTo}(k, 0) =$ ______________________________ for all $k \ge 0$,

  because ____.

- $\mathit{canAddTo}(0, s) =$ ______________________________ for all $s > 0$,

  because ____.

5 Now consider $\mathit{canAddTo}(k, s)$ in the general case, when $k > 0$ and $s > 0$. That is, we are trying to find whether we can add up to exactly s using only some subset of $x_1, \dots, x_k$. In order to break this problem down into subproblems, we would need to decrease k and/or s. Fill in the following steps.

- If _______________, then we definitely cannot use x_k as part of a

  subset adding to s, because it is too _________________________.

In this case, we would get the same result if we only allowed ourselves to use $\{x_1, \dots, x_{k-1}\}$, that is, *canAddTo*(k, s) is the

same as ____.

- Otherwise, we have two choices: we can try to use _________ as part of our subset or not. If we don't use it, it is the same as the previous case. If we do use it, then in order to complete the

  subset we have to make a subset using only _________________

  which adds up to ______________________________________.

6 Use your reasoning above to write down a complete recursive definition of *canAddTo*.

Step 2: Memoize.

7 Explain why it would be extremely slow to directly evaluate *canAddTo*(n, S) as a recursive function.

8 How big of an array do we need to store all the intermediate results from *canAddTo*(n, S)?

9 What order can we fill in the array, so that we never try to fill in an array slot before filling in other slots it depends on?

10 How long does it take to fill in each slot in the array (assuming we have already computed any other necessary array slots)?

11 Therefore, what is the running time of this dynamic programming algorithm?

12 Is this faster than your answer to Question 3?