Describe and analyze ***dynamic programming*** algorithms for the following problems. Remember, the solution to a dynamic programming problem has three things:

- A recurrence relation
- A *brief* description of what your recurrence function represents and what each case represents.
- A *brief* description of the memory element/storage and how it's filled in.

For DP problems and other algorithmic problems as a whole, I want to emphasize that we will be grading clarity and conciseness which admittedly is a subjective measure, but it is necessary. If you need pages of text to describe your algorithm, you're doing something wrong. You should be editing and rewriting your solution multiple times before submission. **Figures can do more in a smaller space than pages of text, use them.**

1. **Coin Change** Let's say you have at your disposable a wide assortment of (not neccessarily dollar) coins and you need to make change for a particular value x. As you're doing so, you wonder to yourself *how many different ways can I make chnage for* x. Well I think that's a excellent question so let's figure it out.

   Problem: You are give an integer value x and an array A where each element of the array represents a coin denomination:

   Example: $A = [1,2,3]$ and $x = 5$. Output is 5 ($\{1,1,1,1,1\}, \{1,1,1,2\}, \{1,1,3\}, \{1,2,2\}, \{2,3\}$).

2. **Subset sum**: You are given an array A of size **n** and a number **m** and we have to find whether there exists a subset with sum divisible by **m**.

   Example: $A = [7,4,6,3]$.

   - There exists no subset divisible by 12
   - There exists a subset that is divisible by 8 ($\{7,3,6\}$)

3. **Largest Square of 1's** You are given a $n \times n$ bitonic array A and the goal is to find the set of elements within that array that form a square filled with only **1**'s.

$$\begin{array}{cc} & i \rightarrow \\ j \downarrow & \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Figure 1. Example: The output is the sidelength of the largest square of 1's (4 in the case of the graph above, yes there can be multiple squares of the greatest size).

4. **KnapSack**: This problem describes a situation where you have a bunch of items that have a corresponding weight and value and your goal is to fit a collection of items with the greatest value into a "knapsack" with a finite capacity.

   So let's formalize this problem: you are given:

   - a array of values V where each element corresponds to item i with value $V[i]$
   - an array of integer weights W where each elements corresponds to item i with weight $W[i]$
   - a integer X which corresponds to the capacity of the knapsack.

   Problem: Find maximum value of items that can be fit into knapsack of the defined capacity.

5. **Maximum rectangle**: You are given a 2D array A that contains positive and negative integer values. You need to find the rectangle that has the largest sum of elements.

$$\begin{array}{cc} & i \rightarrow \\ j \downarrow & \begin{bmatrix} 2 & 1 & -10 & 3 & -4 \\ 10 & -6 & 6 & 5 & 4 \\ -1 & 0 & 9 & -5 & -9 \\ -8 & -2 & 7 & 8 & -3 \\ -7 & -2 & 6 & 0 & 4 \end{bmatrix} \end{array}$$

Figure 2. Example: The output is the sum of the greatest rectangle sum (*30* in the case of the array above.).

6. **Rod cutting:** The rod cutting problem assumes you have some rod of length n that you need to sell. The issue is that the market is illogical and rod price is not linearly proportional with rod length.

   Problem: You are given an integer x that represents the length of rod you have and an array A where i corresponds to a rod length and $A[i]$ corresponds to the price a rod of that length would fetch. You need to determine the maximum value you can fetch from the rod assuming you cut it optimally.

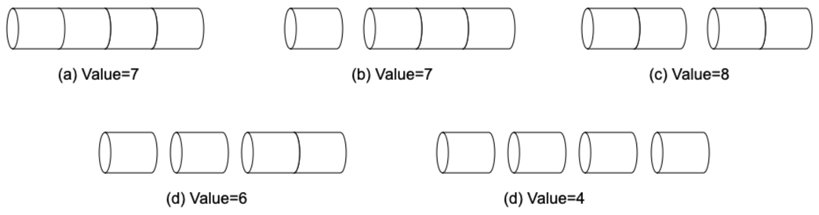

Figure 3. Example: $A = [1, 4, 6, 7]$ and $x = 4$, output should be **8**.

7. In lecture we discussed the following two problems:

    - **Longest increasing subsequence (LIS)** - Given an array ($A[1..n]$) of n integers find the longest increasing subsequence.
    - **Longest common subsequence (LCS)** - Given two arrays ($A[1..n]$ and $B[1..n]$), what is the length of the longest subsequence present in both (for the sake of simplicity let's assume both arrays are of size n).

    Now I want the Longest Common Increasing Sub-sequence: given two arrays (A and B) each containing a sequence of n integers, what is the length of the longest subsequence that is present in both arrays.