Problem 1. Checkerboard. [20 points]

You are given a checkerboard with n rows and n columns. This is just a $n \times n$ table V. The (i, j) entry of V has value $v(i, j)$ (can be positive or negative). We want to move a checker from the top-left corner (i.e., position $(1, 1)$) to the bottom-right (i.e., position (n, n)). The only legal moves are to move a checker down by one square, right by one square, or diagonally down-right by one square. The value of a sequence of moves is the sum of the values of the entries in the sequence. We want the maximum value of any legal sequence of moves. Here is an example of a 3×3 checkerboard:

	1	*2*	*3*
1	3	30	12
2	−12	7	−9
3	39	−2	15

The sequence of moves $(1, 1) \to (1, 2) \to (2, 2) \to (3, 3)$ is legal and has total value $3 + 30 + 7 + 15 = 55$.

Give a dynamic programming algorithm that given the values V determines the maximum value of any legal sequence of moves from $(1, 1)$ to (n, n). You just need to output the total value from the optimal solution.

(Faster (and correct) algorithm in $O(\cdot)$ notation is worth more credit.)

(a) Define the entries of your table in words. E.g., $T(i)$ is ..., or $T(i, j)$ is ....

(b) State the recurrence for the entries of your table in terms of smaller subproblems.

(c) Write pseudocode for your algorithm to solve this problem.

(d) Analyze the running time of your algorithm.

Problem 2. Subset-Sum. [20 points]

Given a list $A = [a_1, a_2, \ldots, a_n]$ of positive integers, and a positive integer k, **give a dynamic programming algorithm** that determines if there exists a subset S of indices $\{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} a_i = k$$

You can use each element of A at most once. All elements of A are distinct (i.e., no repeats). You don't need to output the subset.
Example: $a_1 = 5, a_2 = 10, a_3 = 2$. For $k = 12$ there is a solution (using a_2 and a_3). For $k = 14$ there is no solution.
(Faster (and correct) algorithm in $O(\cdot)$ notation is worth more credit.)

(a) Define the entries of your table in words. E.g., $T(i)$ is ..., or $T(i, j)$ is ....
(b) State the recurrence for the entries of your table in terms of smaller subproblems.
(c) Write pseudocode for your algorithm to solve this problem.
(d) Analyze the running time of your algorithm.

Problem 3. [7 parts = 35 points] No explanations necessary. **You need to simplify your answer wherever possible.**

Let $\omega_n = \exp(2\pi i/n)$ denote the n-th root of unity.
In polar coordinates $\omega_n = (1, \frac{2\pi}{n})$.

Part (a):
What is $(\omega_n^2)^{-1}$, in other words, what is the multiplicative inverse of $(\omega_n)^2$?
Let $(\omega_n^2)^{-1} = \omega_n^j$. **What is j?** Specify a j where $0 \leq j \leq n$.

Part (b):

$$\omega_n + \omega_n^2 + \cdots + \omega_n^n = \_\_\_\_\_$$

(You can assume n is even.)

Part (c): The recurrence:

$$T(n) = 2T(n/2) + O(1) \text{ solves to: } \_\_\_\_\_$$

(Simply solve in optimal $O()$ notation.) You don't need to show your work.

Part (d):

$$\text{What is } 14^{96} \bmod 97 \text{ ? } \_\_\_\_\_$$

Note: 97 is prime.

Part (e):

$$\text{What is } 25^{-1} \bmod 625 \text{ ? } \_\_\_\_\_$$

In the RSA protocol, let $p = 11, q = 31$ for the following questions.
Part (f): What is the smallest valid encryption key e where $e > 0$?
Part (g): The decryption key d is chosen so that $d \equiv e^{-1} \bmod X$.
What is X?
(Don't compute d, just tell us X.)

Problem 4. Cartesian Sum. [20 points]

Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the *Cartesian sum* of A and B, defined by:

$$C = x + y : x \in A \text{ and } y \in B.$$

We want to find the set of elements in C and also the number of times each element of C is realized as a sum of elements in A and B. Show that the problem can be solved in $O(n \log n)$ time by reducing it to the polynomial multiplication algorithm. Use the **polynomial multiplication** algorithm as a black-box without modifying it. You need to explain what input you put into the polynomial multiplication algorithm and what you do with the output to get the solution to the Cartesian Sum problem. Don't give code.

Example: for $A = [1, 2, 3]$ and $B = [2, 3]$ then $C = [3, 4, 5, 6]$ and the solution to the Cartesian Sum problem is:

3 appears and is obtainable in 1 way
4 appears and is obtainable in 2 ways
5 appears and is obtainable in 2 ways
6 appears and is obtainable in 1 way.

You don't have to print your output in that form, whatever form that is convenient is OK, e.g., as a vector $c = (0, 0, 0, 1, 2, 2, 1)$. Just make sure to explain the format of your output and how it solves the problem.

You can assume each number between 0 and $10n$ appears at most once in A and at most once in B (although this doesn't matter for the solution, it's almost the same algorithm to solve the problem when the sets A and B might contain elements multiple times).

Let $P(x) = p_0 + p_1x + p_2x^2 + \cdots + p_mx^m$ and let $Q(x) = q_0 + q_1x + q_2x^2 + \cdots + q_mx^m$ be the pair of polynomials that will be used in the polynomial multiplication algorithm.

Part (a): Explain how you define the coefficients of these polynomials $P(x)$ and $Q(x)$ from the input sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$.

Part (b): You run the polynomial multiplication algorithm on $P(x)$ and $Q(x)$. You get the coefficients $r_0, r_1, \ldots, r_{2m}$ of the polynomial $R(x)$ where $R(x) = P(x) \times Q(x)$. Explain how you go from these coefficients to get the solution to the Cartesian sum problem.

Problem Extra Credit. Longest Common Sub*!?*. [15 points]

Given two strings $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_m$ **give a dynamic programming algorithm** to find the **length** k of the longest string $Z = z_1, \ldots, z_k$ where Z appears as a **substring** of X and as a **subsequence** of Y. Recall, a substring is **consecutive** elements.
For example, for the following input:

$$\begin{aligned} X &= a, \mathbf{b}, \mathbf{d}, \mathbf{b}, \mathbf{a}, b, f, g, d \\ Y &= \mathbf{b}, e, t, f, \mathbf{d}, \mathbf{b}, f, \mathbf{a}, f, r \end{aligned}$$

then the answer is 4 (since, b, d, b, a is a substring of X and it is also a subsequence of Y). You do not need to output the actual substring, just its length.
(Faster (and correct) in asymptotic $O(\cdot)$ notation is worth more credit.)

(a) Define the entries of your table in words. E.g., $T(i)$ is ..., or $T(i, j)$ is ....
(b) State the recurrence for the entries of your table in terms of smaller subproblems.
(c) Write pseudocode for your algorithm to solve this problem.
(d) Analyze the running time of your algorithm.