Answer the questions below through the Udacity site.

Note that for some questions it is not sufficient to simply pass the tests on the site. Your code must have a certain running time, and in some cases you must also answer additional questions in the comments of your code.

1. Suppose there is a procedure f that decides a language A in $O(g(n))$ time, where n is the length of the input string. Use dynamic programming (not memoization) to create an algorithm that decides A^* in $O(n^2 g(n))$ time. Implement your strategy in python here.

https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/e-3160768618/m-3234618854

2. Consider an arbitrary string $X = x_0 \ldots x_{n-1}$. A *subsequence* of X is a string of the form $x_{i_1} x_{i_2} \ldots x_{i_k}$ where $1 \leq i_1 < \ldots < i_k < n$. For example 'ape' is a subsequence of 'apple.' A string is *palindromic* if it is equal to its own reverse (i.e. the string is the same whether read backwards or forwards).

   We will use dynamic programming to find the length of the longest palindromic subsequence of X. For $i \leq j$, let $L(i,j)$ be the length of the longest palindromic subsequence of the substring $x[i:j]$ (the substring going from the ith character to the jth character of x, not including the jth character).
   a. Derive a recurrence for L(i, j), remembering to include the base case?
   b. Why is your recurrence correct?
   c. Write a dynamic programming algorithm based on the recurrence for L(i, j) in python. Use backtracking to construct a longest palindromic subsequence.
   d. What is its running time?

   Answer the above questions in the comments of your submission to Udacity at https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-2549628613

3. Let $A(x) = a_0 + a_1 x + \ldots + a_n x^n$ and let $B(x) = b_0 + b_1 x + \ldots + b_n x^n$ n be two polynomials. Recall that $C(x) = A(B(x)) = a_0 + a_1 B(x) + a_2 B^2(x) + \ldots a_n B^n(x) = c_0 + c_1 x + \ldots c_{n^2} x^{n^2}$. Implement an $O(n^3 \log n)$ for finding the coefficients of C at

   https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-2697878553

   Explain your running time analysis in your comments. It's okay if its faster than $\Omega(n^3 \log n)$.

(Bonus 10pts) Make your algorithm $O(n^3)$. Explain your running time analysis in your comments.

4. Given two sets A and B of integers, their sum set C is defined to be $C = \{a+b : a \in A, b \in B\}$. For each $c \in C$, let $m(c)$ denote the number of ways in which c can be obtained, i.e., $m(c) = |\{(a,b) : a \in A, b \in B, a+b = c\}|$. For example, if $A = \{1,2,3\}$ and $B = \{0,1,4\}$, then their sum set is $C = \{1,2,5,3,6,4,7\}$ and $m(1) = 1$, $m(2) = 2$, $m(3) = 2$, $m(4) = 1$, $m(5) = 1$, $m(6) = 1$, and $m(7)$ = 1.
    a. For two arbitrary sets A and B of integers, what is the worst-case complexity of computing their sum set?
    b. Suppose now that A, B consist of integers in the range $[0, n)$. Give an upper bound on the number of distinct elements in their sum set.
    c. With the above assumption, give an $O(n \log n)$ to find $(c, m(c))$ for each $c \in C$.

Answer the above questions in the comments of your submission to Udacity at

https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-2569679056