Tutorial 5, Design and Analysis of Algorithms, 2019
Use *Dynamic Programming* to solve the following problems:

1. (a) Solve the following instance of the 0/1 *Knapsack Problem:*
   $(I_i)_{i=1}^5 = (I_1, I_2, I_3, I_4, I_5)$
   $(w_i)_{i=1}^5 = (1, 2, 3, 4, 5)$
   $(p_i)_{i=1}^5 = (5, 4, 3, 2, 1)$
   $W = 8$

   (b) Suppose that in a 0/1 *Knapsack Problem*, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value (for example, as given in problem $1(a)$). Give an efficient algorithm (having time complexity $O(n \log n)$) to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.

2. Find *all possible* shortest salseman tours and cost for the following instance of the *Traveling Salseman Problem* for a directed K_4 graph having the following adjacency matrix:

$$\begin{bmatrix} 0 & 8 & 3 & 6 \\ 2 & 0 & 1 & 5 \\ 4 & 7 & 0 & 9 \\ 8 & 2 & 5 & 0 \end{bmatrix}$$

3. There is a tower of floors, and an egg dropper with ideal eggs. The physical properties of the ideal egg is such that it will shatter if it is dropped from floor n^* or above, and will have no damage whatsoever if it is dropped from floor $n^* - 1$ or below. The problem is to find a strategy such that the egg dropper can determine the floor n^* in as few egg drops as possible. Design an efficient algorithm for solving this problem, and find its complexity. Show the working of your algorithm for $n = 10$ floors, and $m = 2$ eggs.

4. Let $G = (V, E)$ be an undirected graph with n nodes. Recall that a subset of the nodes is called an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we will see that it can be done efficiently if the graph is "simple" enough. Call a graph $G = (V, E)$ a *path* if its nodes can be written as $v_1, v_2, ..., v_n$, with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer weight w_i. Consider, for example, the five-node path drawn in Figure 1. The weights are the numbers drawn inside the nodes. The goal in this question is to solve the following problem:
   Find an independent set in a path G whose total weight is as large as possible.
   Design an efficient algorithm that takes an n-node path G with weights and returns an independent set of maximum total weight. The running time of your algorithm should be polynomial in n, independent of the values of the weights. Give a formal correctness proof for your algorithm. Find the time complexity of your algorithm and show its working on the given example (Figure 1).

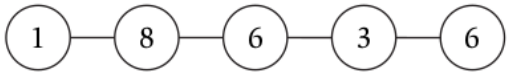

Figure 1: A path with weights on the nodes. The maximum weight of an independent set is 14.

5. Suppose you are managing the construction of billboards on a Highway that runs for M miles. The possible sites for billboards are given by numbers $\{x_1, x_2, ..., x_n\}$, each in the interval $[0, M]$. If you place a billboard at location x_i, you receive a revenue of $r_i > 0$. Regulations imposed by the Highway Department require that no two of the billboards be within less than or equal to 5 miles of each other. You have to place billboards at a subset of the sites so as to maximize your total revenue, subject to this restriction.
   Example: Suppose $M = 20, n = 4$,
   $\{x_1, x_2, x_3, x_4\} = \{6, 7, 12, 14\}$, and
   $\{r_1, r_2, r_3, r_4\} = \{5, 6, 5, 1\}$.
   Then the optimal solution would be to place billboards at x_1 and x_3, for a total revenue of 10. Design an efficient algorithm that takes an instance of this problem as input and returns the maximum total revenue that can be obtained from any valid subset of sites. The running time of the algorithm should be polynomial in n. Show the working of your algorithm on the given example.

6. We are looking at the price of a given stock over n consecutive days, numbered $i = 1, 2, ..., n$. For each day i, we have a price $p(i)$ per share for the stock on that day. (We will assume for simplicity that the price was fixed during each day.) We would like to know: How should we choose a day i on which to buy the stock and a later day $j > i$ on which to sell it, if we want to maximize the profit per share, $p(j) - p(i)$? (If there is no way to make money during the n days, we should conclude this instead). Design an efficient algorithm to find the optimal numbers i and j in time $O(n)$. Show the working of your algorithm on the following problem instances:
   $(p(i))_{i=1}^{10} = (83, 18, 34, 2, 71, 26, 44, 46, 48, 75)$
   $(p(i))_{i=1}^{10} = (83, 75, 71, 48, 46, 44, 34, 26, 18, 2)$

7. The owners of an independently operated gas station are faced with the following situation. They have a large underground tank in which they store gas; the tank can hold up to L gallons at one time. Ordering gas is quite expensive, so they want to order relatively rarely. For each order, they need to pay a fixed price P for delivery in addition to the cost of the gas ordered. However, it costs c to store a gallon of gas for an extra day (l gallons remaining at the end of a day will give extra cost of lc), so ordering too much ahead increases the storage cost. They are planning to close for a week in the winter, and they want their tank to be empty by the time they close. Luckily, based on years of experience, they have accurate projections for how much gas they will need each day until this point in time. Assume that there are n days left until they close, and they need g_i ($g_i \leq L$) gallons of gas for each of the days $i = 1, ..., n$. Assume that the tank is empty at the end of day 0. Design an efficient algorithm to decide on which days they should place orders, and how much to order so as to minimize their total cost. Also find the time complexity of the algorithm.