1. Here’s a recap of problem 2 in the handout given in class. Suppose you’re running a consulting business. Each month, you can either run your business from an office in New York (NY) or San Francisco (SF). In month $i$, you’ll incur an operating cost of $N_i$ or $S_i$ if you run the business out of NY or SF respectively. However, if you run the business out of one city in month $i$, and then out of another city in month $i + 1$, you’ll incur a fixed moving cost of $M$ to switch base offices.

Given a sequence of $n$ months, a plan is a sequence of $n$ locations – each equal to NY or SF – where the $i$th location indicates the city in which you will be based on the $i$th month. The cost of the plan is the sum of the operating costs for each of the $n$ months plus a moving cost of $M$ each time you switch cities. The plan can begin in either city.

Given $N_1, N_2, \ldots, N_n$ and $S_1, S_2, \ldots, S_n$ and moving cost $M$, find a plan of minimum cost. Such a plan is called optimal.

a. Some time ago, a student submitted the following solution for the problem: Initialize two arrays CITY[1,...,n] and OPT[1,...,n]. Let CITY[$i$] denote the city that you should be on month $i$ in the optimal plan. Let OPT[$i$] denote the cost of the optimal plan up to week $i$. Here’s the initial step: If $N_1 \leq S_1$, set CITY[1] to NY and OPT[1] to $N_1$; else set CITY[1] to SF and OPT[1] to $S_1$.

For $i = 2$ to $n$ do the following: if CITY[$i - 1$] = NY and if OPT[$i - 1$] + $N_i <$ OPT[$i - 1$] + $S_i + M$ then set OPT[$i$] to OPT[$i - 1$] + $N_i$ and CITY[$i$] to NY; else set OPT[$i$] to OPT[$i - 1$] + $S_i + M$ and CITY[$i$] to SF.

On the other hand if CITY[$i - 1$] = SF and if OPT[$i - 1$] + $N_i + M <$ OPT[$i - 1$] + $S_i$ then set OPT[$i$] to OPT[$i - 1$] + $N_i + M$ and CITY[$i$] to NY; else set OPT[$i$] to OPT[$i - 1$] + $S_i$ and CITY[$i$] to SF.

Once the for loop is done, return the arrays OPT and CITY.

Show that the above algorithm does not work. That is, show that there are inputs for which the output CITY will not be an optimal plan. Explain what is wrong with the algorithm.

b. In contrast, in class you saw a solution that will always find the cost of an optimal plan correctly using dynamic programming. Translate that solution into pseudocode so that not only does it find the optimal cost, it also finds an optimal plan that generated the optimal cost.
2. **The maximum subsequence sum problem – again!** Recall that in the maximum subsequence sum problem, we are given a sequence of $n$ numbers $x_1, x_2, \ldots, x_n$. Our goal is to find a contiguous subsequence whose sum is as large as possible. I've shown you four different ways to solve the problem – brute force (takes $O(n^3)$ time), improved brute force (takes $O(n^2)$ time), and a “clever” algorithm (takes $O(n)$ time).

a. This time around, we want to solve this problem using dynamic programming. For $j = 1, \ldots, n$, define $\text{SUM}[j]$ as the largest sum of a subsequence that ends at position $j$. Find a recursive formula for $\text{SUM}[j]$. Now, write a pseudocode that computes $\text{SUM}[j]$ for $j = 1, \ldots, n$.

b. Build on the part (a)’s pseudocode so that you can also output the subsequence with the best sum. What is the running time of your algorithm?

c. Apply your algorithm to the sequence 5, 15, −30, 10, −5, 40, 10, −8.

3. **Coin change.** Suppose you have an infinite supply of quarters (25 cents), dimes (10 cents), nickels (5 cents) and pennies (1 cent). When you need to make change for $x$ cents, your instinct is to do this greedily: pack in as many quarters as you can, then as many dimes, then as many nickels, then finally as many pennies as needed. For example, if $x = 68$, you make change as follows: $25 + 25 + 10 + 5 + 1 + 1 + 1$. This greedy method will always yield the fewest number of coins. In the case of $x = 68$, you used seven coins altogether and no other combination will yield fewer than seven coins.

It turns out, however, that the greedy method is not always this lucky. That is, if the coin denominations are different, the same greedy algorithm may not use the fewest number of coins. For example, suppose a country’s coins are worth 1, 3 and 4 cents only. If you make change for say 6 cents using the greedy method, you would have given it as $6 = 4 + 1 + 1$ but a better solution is $6 = 3 + 3$ because the latter uses two coins while the former three coins.

Given coin denominations $a_1, a_2, \ldots, a_k$ where $a_1 = 1$ cent and $a_1 < a_2 < \cdots < a_k$, and $x$, our goal is to make change for $x$ with coins whose values are $a_1, \ldots, a_k$ using as few coins as possible. Design a dynamic programming algorithm for this problem. That is, you now have to define the variables yourself and develop a recursive formula for it. Note that the output of the algorithm should be $(n_1, n_2, \ldots, n_k)$ where $n_i$ denotes the number of coins of value $a_i$ that is used to make change for $x$.

4. A **path** $P_n$ on $n$ nodes is just a sequence of $n$ nodes that form a line segment. The example below is a path on 5 nodes. Let us denote the $i$th node on the path as $v_i$. Each $v_i$ has weight $w_i$ associated with it. Now suppose $S$ is a subset of these nodes. The **weight** of $S$ is simply equal to the sum of the weights of the nodes in $S$. 
For this problem, we are interested in independent sets, which are subsets of nodes no two of which are adjacent to each other. Our goal is to find an independent set of $P_n$ whose total weight is as large as possible. In the example below, the set $\{v_1, v_3, v_5\}$ is an independent set whose weight is $1 + 6 + 6 = 12$. The set $\{v_2, v_5\}$ is another independent set and its weight of 14 is the largest among all independent sets of the path.

```
1 8 6 3 6
```

Given $P_n$ and the weights $w_1, w_2, \ldots, w_n$, design a dynamic programming algorithm that finds an independent set of $P_n$ whose weight is as large as possible. Note that your output should not just give the weight of the optimal independent set, it should output the set itself. What is the running time of your algorithm?