CSCI 570 Fall 2015 HW8

1. Problem 7 from Chapter 6.

   For $j \in \{1, 2, \ldots, n\}$, let $OPT(j)$ denote the maximum possible return an investor can make by selling the stock on day j.

   If the investor held the stock on day $j-1$, then $OPT(j) = OPT(j-1) + (p(j) - p(j-1))$. Otherwise (the share hasn't been bought yet as on day $j-1$), the investor can make at most $p(j) - p(j) = 0$ money by selling it on day j (ie, by buying and selling on day j). Hence we have the recurrence $OPT(j) = max(OPT(j-1) + p(j) - p(j-1), 0)$.

   Compute $OPT(j)$ starting from the initial condition $OPT(1) = 0$ and the optimal day to sell is $argmax_j OPT(j)$. Once we have found the best day to sell, the best day to buy is the day on which the share price is the smallest among all the days occurring on/before the best sell day.

2. Given a sequence $\{a_1, a_2, \ldots, a_n\}$ of n numbers, describe an $\mathcal{O}(n^2)$ algorithm to find the longest monotonically increasing subsequence.

   Let l_i denote the length of the longest monotonically increasing subsequence that ends with a_i. Clearly $l_1 = 1$. Compute the sequences S_i, S_{ij} using the following recurrences.

   Initialize $S_1 = a_1$.

   For $1 \leq j < i$, if $a_j > a_i$ then $S_{ij} = a_i$. Otherwise, S_{ij} is set to $\{S_j$ concatenated with $a_i\}$.

   Now S_i is set to the longest among the sequences $S_{ij}, 1 \leq j < i$.

   Claim: The length of S_i, $l(S_i) = l_i$.

   Assume otherwise. Let k be the smallest index such that $l(S_k) < l_k$. Let O_k be a sequence of length l_i ending with a_k. Let a_j be the last but one element of the sequence O_k. As $j < k$, $l_j = l(S_j)$

   $$l(S_k) < l_k = l_j + 1$$

$$\Rightarrow l(S_j) < l_j$$

This is a contradiction as $j < k$. Thus our claim is true.

Clearly the longest monotonically increasing subsequence is by definition the longest of the sequences $S_i, 1 \leq i \leq n$. The running time is $\Theta(n^2)$.

3. Problem 12 from Chapter 6.

Let $OPT(j)$ denote the minimum cost of a solution for the first j servers (Note that a copy of the file should be in server j for every such solution).

If k (with $k < j$) is the highest index server that contains a copy of the file, then the access cost for the servers $(k+1, \ldots, j)$ is $(j-k-1) + (j-k-2) + \ldots + 2 + 1 + 0 = \binom{j-k}{2}$. For placing a copy at server j, we pay a cost of c_j. Thus we have the following recurrence

$$OPT(j) = c_j + \min_{0 \leq k < j} (OPT(k) + \binom{j-k}{2})$$

with initial conditions $OPT(0) = 0$.

Compute $OPT(j)$ in increasing order of j using the recurrences and at each step record the highest index server where the last but one copy was placed (ie, k that minimized the second term in the recurrence). Then $OPT(n)$ gives the minimum cost for solving the problem and by tracking back the last but one server copy at each step we can find the optimal configuration.

At each step, it takes $\mathcal{O}(j)$ to compute $OPT(j)$. Hence the total running time is $\mathcal{O}(n^2)$.

4. There are a series of volunteering activities lined up one after the other for the next year, $V_1, V_2, , V_n$. For any i^{th} volunteering activity, you are given the happiness H_i associated with it. For any i^{th} activity, you are also given N_i, which is the number of immediately following volunteering activities that you cannot participate in if you participate in that i^{th} activity. Give an efficient dynamic programming solution to maximize the happiness. Also state the runtime of your algorithm.

For $1 \leq i \leq n$, let S_i denote a solution, the maximum possible happiness for the set of events $V_i, ..., V_n$ and let opt(i) denote its happiness. If S_i chooses to volunteer for V_i, then it has to exclude V_{i+1} through V_{i+N_i}. In this case, its happiness is H_i plus the happiness it achieves for the set $V_{i+N_i+1}, ..., V_n$. Since S_i is optimal for the set $V_i, .., V_n$, S_i restricted to the subset $V_{i+N_i+1}, ..., V_n$ has to be optimal. Thus in this case, opt(i) $= H_i$ +opt$(i + N_i + 1)$. If S_i chooses not to volunteer for V_i, then its

happiness is the happiness it achieves for the set $V_{i+1}, ..., V_n$. Since S_i is optimal for the set $V_i, .., V_n$, S_i restricted to the subset $V_{i+1}..., V_n$ has to be optimal. Thus in this case, opt(i) = opt(i+1). We thus have a recurrence for all i, opt(i)= max(opt(i+1), opt$(i + N_i + 1)+H_i$). (We understand that opt(index > n) = 0). The boundary condition is that opt(n)=H_n and we start solving recurrence starting with opt(n), opt(n-1) and so on until opt(1). Runtime complexity of this solution is O(n).

5. (Difficult, optional) You are given n points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ on the real plane. Further they are sorted from left to right and no two points have the same x-coordinate. That is $x_1 < x_2 < \ldots < x_n$.

   A bitonic tour is defined as follows. The tour starts from (x_1, y_1), goes through some intermediate points and reaches (x_n, y_n). Then it goes back to (x_1, y_1) through every one of the rest of the points. All points (except x_1, y_1) are thus visited exactly once. Further from (x_1, y_1) to (x_n, y_n) you have to keep going right at every step. Likewise, you have to keep going left from (x_n, y_n) to (x_1, y_1).

   Describe an $\mathcal{O}(n^2)$ algorithm to compute the shortest bitonic tour.

A bitonic tour is defined as follows. The tour starts from (x_1, y_1), goes through some intermediate points and reaches (x_n, y_n). Then it goes back to (x_1, y_1) through every one of the rest of the points. All points (except x_1, y_1) are thus visited exactly once. Further from (x_1, y_1) to (x_n, y_n) you have to keep going right at every step. Likewise, you have to keep going left from (x_n, y_n) to (x_1, y_1).
Describe an $\mathcal{O}(n^2)$ algorithm to compute the shortest bitonic tour.

Let $p_j = (x_j, y_j)$ denote the j-th point. A shortest bitonic tour can be thought of as a cycle where the vertices are points. Edges connect the points if they are are visited one after another.

Consider the shortest bitonic tour on the first i points. Observe that such a tour must contain an edge (p_k, p_i) with $k < i-1$. For $k < i-1$, a shortest bitonic tour on the points $p_1, \ldots, p_i$ that contains (p_k, p_i) must be a shortest bitonic tour on the points $p_1, \ldots, p_{k+1}$ minus the edge (p_k, p_{k+1}) plus the edge (p_k, p_i) and plus the path $\{(p_{k+1}, p_{k+2}), \ldots, (p_{i-1}, p_i)\}$. Consequently k can be chosen such that we end up with the shortest bitonic tour on $p_1, \ldots, p_i$.

That the subproblem on $p_1, \ldots, p_{k+1}$ exhibits the optimal substructure property can be proved by a typical replacement strategy. Suppose we have an optimal solution on $p_1, \ldots, p_i$ points which uses a solution on $p_1, \ldots, p_{k+1}$ which is not optimal. Now by replacing the non-optimal solution on the $p_1, \ldots, p_{k+1}$ points by an optimal solution into the "$p_1, \ldots, p_i$" problem we obtain a solution which is better than the optimal solution on $p_1, \ldots, p_i$, which is a contradiction.

Let $OPT(i)$ be the length of the shortest bitonic tour on the first i points, we can write the following recursion:

$$OPT(i) = \min_{1 \le k \le i-2}\{OPT(k+1) + D(k+1, i) + d(k, i) - d(k, k+1)\}$$

for all $3 \le i \le n$, where

$$d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$
$$D(i, j) = \sum_{k=i}^{j-1} d(k, k+1)$$

The base cases are: $OPT(1) = 0, OPT(2) = 2d(1, 2)$. $OPT(n)$ is the length of the shortest bitonic tour.
Each step of the iterations costs $O(n)$ and we need to compute n values in total, so the total running time is $O(n^2)$.