Solutions to Quiz 3

Solution 1: See the table below. The times to compute the objective values come straight from the lecture notes. The construction times are explained below.

Algorithm	**Parameters**	**Objective Value**	**Solution**
Longest Common Subsequence	m, n (sequence lengths)	mn	$m+n$
Chain Matrix Multiplication	n (no. of matrices)	n^3	n
Floyd-Warshall (shortest paths)	n (no. of vertices)	n^3	n

LCS: We trace a path through the $m \times n$ helper matrix from entry $[m, n]$ down to $[0, 0]$. Each step decrements either the row index or the column index or both, so it runs in $O(m+n)$ time.

CMM: Each access to the helper matrix splits the sequence. After $n-1$ splits, we have broken the sequence into singletons. Thus, the total time is $O(n)$.

FW: Each access to the helper matrix yields an additional vertex along the shortest path. Since any shortest path can have at most n vertices in total, the overall time to construct any path is $O(n)$.

Solution 2: For $0 \le i \le m$ and $0 \le j \le n$, define $\text{scs}(i, j)$ to be the length of the SCS of X_i and Y_j. As a basis, observe that if one sequence is empty, the SCS is just the other sequence, and hence $\text{scs}(i, 0) = i$ and $\text{scs}(0, j) = j$. If i and j are both strictly positive, we consider the following cases:

- If $x_i = y_j$ then we claim that the SCS must end in this common symbol (since using any other symbol would only make the sequence longer). We add this common symbol into the end of the SCS and recurse on remaining strings X_{i-1} and Y_{j-1}, for a cost of $1 + \text{scs}(i-1, j-1)$.
- If $x_i \neq y_j$, then the SCS must end with either x_i or y_j (since if it were to end in any other symbol, it could be made shorter by removing it). There are two options, and we try both and take the better of the two.
  - The SCS ends in x_i. We add x_i to the end of the SCS and recurse on X_{i-1} and Y_j, for a cost of $1 + \text{scs}(i-1, j)$.
  - The SCS ends in y_j. We add y_j to the end of the SCS and recurse on X_i and Y_{j-1}, for a cost of $1 + \text{scs}(i, j-1)$.

The principal of optimality applies, since each subproblem should be solved optimally. This gives the following DP formulation:

$$\text{scs}(i,j) \;=\; \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ 1 + \text{scs}(i-1, j-1) & \text{if } i,j > 0 \text{ and } x_i = y_j, \\ \min \begin{pmatrix} 1 + \text{scs}(i-1, j), \\ 1 + \text{scs}(i, j-1) \end{pmatrix} & \text{if } i,j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

The final answer is $\text{scs}(m,n)$. If implemented, this would run in $O(mn)$ time, since each of the $O(mn)$ tables entries can be computed in constant time.

By the way, is a rather tricky alternative solution, which is based on using the longest common subsequence. Let $\text{lcs}(X,Y)$ denote the length of the longest common subsequence of X and Y. It can be shown by an inductive argument that each character of the LCS appears in the SCS exactly once, and the remaining characters of X and Y are added individually to the SCS. (We omit the proof, but it would be required for a proper justification.) It follows that $\text{scs}(X,Y) = m + n - \text{lcs}(X,Y)$.

Solution 3: The length of the longest non-increasing sequence (LNS) is based on dynamic programming. We assume that the points have been sorted by the x-coordinates. For $0 \leq i \leq n$, let $\text{lns}(i)$ denote the length of the LNS ending at point p_i. (Thus, the sequence *must* end with p_i.) To avoid subscripting out of bounds, it will be convenient to prepend a "sentinel point" $p_0 = (-\infty, +\infty)$.

For the basis case, we have $\text{lns}(0) = 0$. For $i > 0$, the LNS that ends with p_i can be expressed as an LNS that ends at some point p_j, where $0 \leq j < i$ and $y_j \geq y_i$ and to this we add p_i. Since we do not know what this point is, let us try them all and take the largest. This leads to the following DP formulation

$$\text{lns}(i) \;=\; \begin{cases} 0 & \text{if } i = 0, \\ 1 + \max\limits_{\substack{0 \leq j < i \\ y_j \geq y_i}} \text{lns}(j) & \text{otherwise.} \end{cases}$$

Since we do not know which point the LNS ends with, the overall answer is the maximum over all the possibilities. Thus, we have

$$\text{lns}(P) \;=\; \max_{1 \leq i \leq n} \text{lns}(i).$$

(A common mistake is to return $\text{lns}(n)$. However, this is incorrect, since the overall LNS may not end with p_n, for example, if y_n is very large.) If implemented this would take $O(n^2)$ time, since we are computing a table with $O(n)$ entries, each of which can be computed in $O(n)$ time.

A natural (but incorrect) alternative approach is to define $\text{lns}'(i)$ to be the length of the longest non-increasing subsequence among $\{p_1, \ldots, p_i\}$, but not requiring that the sequence ends at p_i. This suggests the following DP formulation:

$$\text{lns}'(i) \;=\; \begin{cases} 0 & \text{if } i = 0, \\ \max \begin{cases} 1 + \max\limits_{\substack{0 \leq j < i \\ y_j \geq y_i}} \text{lns}'(j) \\ \text{lns}'(i-1) \end{cases} & \text{otherwise.} \end{cases}$$

The final answer would be $\text{lns}'(n)$. Unfortunately, this does not work since it loses the context of y-coordinate of the last point in the sequence. Consider the sequence $\langle (1,2), (2,1), (3,4), (4,3) \rangle$. It

is easy to verify that $\text{lns}'(2) \leftarrow 2$, $\text{lns}'(3) \leftarrow \text{lns}'(2) = 2$, and so $\text{lns}'(4) \leftarrow 1 + \text{lns}'(3) = 3$, but this is incorrect, since the longest nonincreasing subsequence is of length 2.

Solution 4: For $1 \leq i \leq j \leq n$, let $M(i,j)$ denote the minimum cost (number of operations) needed to compute the product $A(i,j) = A_i A_{i+1} \ldots A_j$.

Basis: If $i = j$ then the sequence contains only one matrix, and so the cost is 0. (There is nothing to multiply.) Thus, $M(i,i) = 0$.

General case: If $i < j$, then the product $A(i,j)$ can be split into two groups $A(i,k)$ times $A(k+1,j)$, by considering each k, $i \leq k < j$ and taking the best. The cost of computing $A(i,k)$ is given recursively by $M(i,k)$, and the cost of computing $A(k+1,j)$ is given by $M(k+1,j)$. The time needed for the final product is either p_{i-1}^2 (if these are square matrices, that is, $p_{i-1} = p_j = p_k$) or the standard time, $p_{i-1}p_k p_j$, otherwise.

This yields the following DP formulation:

$$M(i,j) \;=\; \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \leq k \leq j-1} M(i,k) + M(k+1,j) + \begin{cases} p_{i-1}^2 & \text{if } p_{i-1} = p_k = p_j \\ p_{i-1}p_k p_j & \text{otherwise} \end{cases} & \text{if } i < j. \end{cases}$$

The final cost is $M(1,n)$. If implemented, this would take $O(n^3)$ time, since we are building a table of size $O(n^2)$, and each entry can be computed in $O(n)$ time.

Solution 5:

(a) The residual network G_f is shown in Fig. 1.

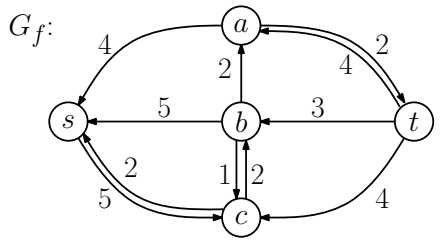

Figure 1: Network flow.

(b) There is one s-t path in G_f, $\langle s, c, b, a, t \rangle$.

(c) The minimum capacity edge on this path is 2, which means that the maximum flow that can be pushed along this path has value 2.