This list of problems goes approximately from easier problems to harder problems. The first few problems are for warmup and easier than anything you might see on the exam, but don't panic if you can't do the last few problems.

Solutions to these problems, as with most DP problems, take the form of a recurrence relation, a short correctness proof of the recurrence relation (if it's not immediately obvious) and a running time analysis.

1. **Maximum value contiguous subsequence**: Given a sequence of n real numbers, $a_1, a_2, ..., a_n$, give an algorithm for finding a contiguous subsequence for which the value of the sum of the elements is maximized.

   Solution:

   Recursion: We recurse on the maximum value subsequence ending at j:

   $$M(j) = \begin{cases} a_j & \text{if } j = 0 \\ \max(M(j-1) + a_j, a_j) & \text{else} \end{cases} \tag{1}$$

   With each element of M, you also keep the starting element of the sum (the same as for $M(j-1)$ or j if you restart). At the end, you scan M for the maximum value and return it and the starting and ending indexes. Alternatively, you could keep track of the maximum value as you create M.

   Running Time: M is size n and evaluating each element of M takes $O(1)$ time for $O(n)$ time to create M. Scanning M also takes $O(n)$ time for a total time of $O(n)$.

2. **Car Assembly** (see Figure 15.1 in CLRS): A factory has two assembly lines, each with n stations. A station is denoted by $S_{i,j}$ where i is either 1 or 2 and indicates the assembly line the station is on, and j indicates the number of the station. The time taken per station is denoted by $a_{i,j}$. A car chasis must pass through each of the n stations in order before exiting the factory. After it passes through station $S_{i,j}$ it will continue to station $S_{i,j+1}$ unless it decides to transfer to the other line. Continuing on the same line incurs no extra cost, but transfering from line i at station $j-1$ to station j on the other line takes time $t_{i,j}$. There is also an entry time e_i and exit time x_i which may be different for the two lines. Give an algorithm for computing the minimum time it will take to build a car chasis.

   Solution:

   Recursion Relation: We recurse on $T_1(j)$ and $T_2(j)$, the fastest way to get to station j on line 1 and the fastest way to get to station j on line 2 respectively. The relation is:

   $$\begin{aligned} T_1(j) &= \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min(T_1(j-1) + a_{1,j}, T_2(j-1) + t_{2,j} + a_{1,j}) & \text{else} \end{cases} \\ T_2(j) &= \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min(T_2(j-1) + a_{2,j}, T_1(j-1) + t_{1,j} + a_{2,j}) & \text{else} \end{cases} \end{aligned} \tag{2}$$

   The solution is then $\min(T_1(n) + x_1, T_2(n) + x_2)$.

   Running Time: Each T is of size n and each entry takes $O(1)$ time to compute for a total running time of $O(n)$.

3. **Making change**: You are given n types of coins with values $v_1, ..., v_n$ and a cost C. You may assume $v_1 = 1$ so that it is always possible to make any cost. Give an algorithm for finding the smallest number of coins required to sum to C exactly.

   For example, assume you coins of values 1, 5, and 10. Then the smallest number of coins to make 26 is 4: 2 coins of value 10, 1 coin of value 5, and 1 coin of value 1.

   Solution:

Recursion: We recurse on $M(j)$, the minimum number of coins required to make change for cost j.

$$M(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min_{v_i \in n}(M(j - v_i)) + 1 & \text{else} \end{cases} \tag{3}$$

Running Time: M has C elements and computing each element takes $O(n)$ time so the total running time is $O(nC)$.

4. **Box stacking**: You are given a set of boxes $\{b_1, ..., b_n\}$. Each box b_j has an associated width w_j, height h_j and depth d_j. Give an algorithm for creating the highest possible stack of boxes with the constraint that if box b_j is stacked on box b_i, the 2D base of b_i must be larger in both dimensions than the base of b_j. You can of course, rotate the boxes to decide which face is the base, but you can use each box only once.

   For example, given two boxes with $h_1 = 5, w_1 = 5, d_1 = 1$ and $h_2 = 4, w_2 = 5, h_2 = 2$, you should orient box 1 so that it has a base of 5x5 and a height of 1 and stack box 2 on top of it oriented so that it has a height of 5 for a total stack height of 6.

   Solution:

   Recursion: Memoize over $H(j, R)$, the tallest stack of boxes with j on top with rotation R.

$$H(j, R) = \begin{cases} 0 & \text{if } j = 0 \\ \max_{i<j \text{ with } w_i > w_j, d_i > d_j}(H(i, R) + h_j) & \text{if } j > 0 \end{cases} \tag{4}$$

   Running Time: The size of H is $O(n|R|)$ where R is the number of possible rotations for a box. For our purposes, $|R| = 3$ (since we only care about which dimension we designate as the "height") so $|H| = O(n)$. Filling in each element of H is also $O(n)$ for a total running time of $O(n^2)$.

5. **Balanced Paritions**: Suppose you are given an array of n integers $\{a_1, ..., a_n\}$ between 0 and M. Give an algorithm for dividing these integers into two sets x and y such that $|\sum_{x_i \in x} x_i - \sum_{y_i \in y} y_i|$, the difference of the sum of the integers in each set, is minimized. For example, given the set $\{2, 3, 2, 7, 9\}$, you can divide it into $\{2, 2, 7\}$ (sums to 11) and $\{3, 9\}$ (sums to 12) for a difference of 1.

   Solution:

   Recursion Relation: Consider just the set of the numbers $\{a_1, ..., a_j\}$. What sums can we make with that set or subsets of it? We can make

   - Any sums we could make with a subset of $\{a_1, .., a_{j-1}\}$
   - Any sums we could make with a subset of $\{a_1, ..., a_{j-1}\} + a_j$

   So: Let c_{ij} be 1 if a subset of $\{a_1, ..., a_i\}$ adds to j and 0 otherwise. The recursion relation for c_{ij} is

$$c_{ij} = \begin{cases} 1 & \text{if } i = 0 \text{ and } j = 0 \\ \max\left[c_{i-1,j}, c_{i-1,j-a_j}\right] \end{cases} \tag{5}$$

   We find the value of j, let it be b, closest to $T = \left(\sum_j a_j\right)/2$ such that $c_{nj} = 1$. The minimum difference is $2(T - b)$.

   Running Time: We need only let j go to nM since the integers are bounded. Therefore, the size of c is n^2M and filling it in takes $O(1)$ per entry for a total running time of $O(n^2M)$.

6. **Boolean parenthesizations**: You are given a boolean expression consisting of a string of the symbols TRUE, FALSE, AND, OR, and XOR. Give an algorithm for finding the number of ways to parenthesize the expression such that it will evaluate to TRUE. For example, there is only 1

way to parenthesize FALSE AND TRUE XOR TRUE such that it evaluates to TRUE: (FALSE AND TRUE) XOR TRUE. (XOR is exclusive or: F XOR F = F, T XOR F = T, F XOR T = T, T XOR T = F)

Solution:

Recursion: Let our expression consist of n atomic elements (TRUE, FALSE) and $n-1$ operators (AND, OR, XOR). The ith atomic element is a_i and the ith operator is o_i so that our expression is $a_1 o_1 a_2 o_2 ... a_{n-1} o_{n-1} a_n$. We keep $T(i,j)$, the number of parenthesizations that makes the expression between a_i and a_j true, and $F(i,j)$, the number of parenthesizations that makes the expression between a_i and a_j false. The recursion for each is:

$$T(i,j) = \sum_{k=i}^{j} \begin{cases} T(i,k)T(k+1,j) & \text{if } o_k = \text{and} \\ (T(i,k)+F(i,k))(T(k+1,j)+F(k+1,j)) - F(i,k)F(k+1,j) & \text{if } o_k = \text{or} \\ T(i,k)F(k+1,j)+F(i,k)T(k+1,j) & \text{if } o_k = \text{xor} \end{cases} \tag{6}$$

$$F(i,j) = \sum_{k=i}^{j} \begin{cases} (T(i,k)+F(i,k))(T(k+1,j)+F(k+1,j)) - T(i,k)T(k+1,j) & \text{if } o_k = \text{and} \\ F(i,k)F(k+1,j) & \text{if } o_k = \text{or} \\ F(i,k)F(k+1,j)+T(i,k)T(k+1,j) & \text{if } o_k = \text{xor} \end{cases}$$

Running Time: There are $O(n^2)$ elements in each of T and F and filling in each element takes $O(n)$ time for a total running time of $O(n^3)$.

7. **Edit Distance**: Given two text strings $A = a_1 a_2 ... a_n$ of length n and $B = b_1 b_2 ... b_m$ of length m, you want to transform A into B with a minimum number of operations of the following types: delete a character from A, insert a character into A, or change some character in A into a new character. The minimal number of such operations required to transform A into B is called the edit distance between A and B. Give an algorithm for finding the edit distance from A to B.

   Solution:

   Recursion Relation: We recurse on $m(i,j)$, the minimum number of operations to change $A(1:i)$ into $B(1:j)$. The relation is

$$m(i,j) = \min\left(m(i-1,j)+1, m(i,j-1)+1, \begin{cases} m(i-1,j-1) & \text{if } a_i = b_j \\ m(i-1,j-1)+1 & \text{else} \end{cases} \right) \tag{7}$$

   Running Time: m has nm elements and evaluating each element takes $O(1)$ time for a total running time of $O(nm)$.

8. **Building Bridges**: Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates $a_1 ... a_n$ and n cities on the northern bank with x-coordinates $b_1 ... b_n$. The cities on each bank are also numbered 1 through n and these numbers do *not* correspond to the ordering of the x-coordinates. You can only build a bridge from a city on the south bank to a city on the north bank with the same number. No two bridges may cross each other. An example of a valid bridge building is shown in Figure 1. Give an algorithm for finding the maximum number of bridges that can be built.

   Solution:

   Consider the sequences $A = N(a_1), ..., N(a_n)$ and $B = N(b_1), ..., N(b_n)$ where $N(a_i)$ is the number of the city with x-coordinate a_i. The length of the longest common subsequence of A and B is the maximum number of bridges. Since A and B are non-repeating, you showed in problem set 6 that the length of the LCS for A and B can be calculated in $O(n \log n)$ time. Make sure you understand why that is the case!

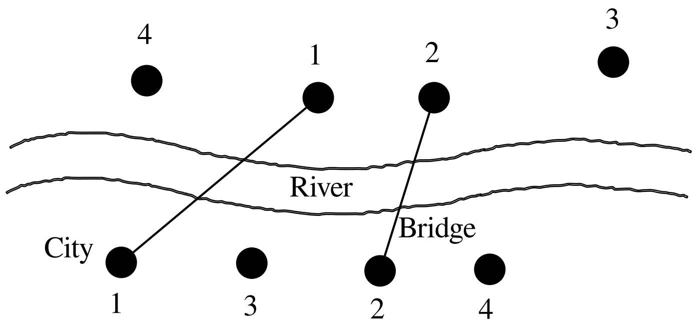

Figure 1: An example of a valid bridge building.

Proof that length of the LCS is the maximum number of bridges: We show that the maximum number of bridges cannot be more than the length of the LCS and that the maximum number of bridges cannot be less than the length of the LCS.

Firstly, assume the length of the LCS is m. Let $c_1, ..., c_m$ be a longest common subsequence of A and B, corresponding to cities $a_{i_1}, ..., a_{i_m}$ in A and $b_{j_1}, ..., b_{j_m}$ in B. Then for $0 < k \leq m$, we can draw a bridge from a_{i_k} to b_{i_k}. None of these bridges intersect. Therefore, we can draw at least as many bridges as the length of the LCS.

Now assume we can draw at most m bridges from cities $C_A = a_{i_1}, ..., a_{i_m}$ to cities $C_B = b_{j_1}, ..., b_{j_m}$ and WLOG assume C_A is ordered by increasing x-coordinate. Then $N(a_{i_k}) = N(b_{j_k})$ since we can draw a bridge between them. Moreover, b_{j_k} must have a higher x-coordinate than any of $b_{j_1}, ...b_{j_{k-1}}$ and a lower x-coordinate than any of $b_{j_{k+1}}, ..., b_{j_m}$ so that none of the bridges cross. Therefore C_A is a subsequence of A and C_B is a subsequence of B and we have found a common subsequence. Thus, the length of the LCS is at least the maxmimum number of bridges.

9. **Function Approximation**: Assume a function $f(x)$ generated a sequence of n points in the plane (x_i, y_i). Given an integer k, we choose a subset of $k + 1$ points and order them according to increasing x-coordinate. This subset must include the first point (x_1, y_1) (smallest x-coordinate) and the last point (x_n, y_n) (largest x-coordinate). We define the function $g(x)$ as the straight line segments connecting each point to the next point in the set so $g(x)$ consists of k line segments. The error is then defined as

$$e = \sum_{i=1}^{n} (y_i - g(x_i))^2. \tag{8}$$

Give an algorithm that takes the n points and k as input and minimizes the error.

Solution:

Recursion Relation: We recurse on $E(i, j)$, the error of the best approximation g_i of the first i points using at most j line segments. Note that as before, we must have a line segment starting at (x_1, y_1) and a line segment ending at (x_i, y_i). The recurrence is

$$E(i,j) = \begin{cases} 0 & \text{if } i = 1 \\ \infty & \text{if } i > 1 \text{ and } j = 0 \\ \min_{0<l<i}(E(l, j-1) + e(l,i)) & \end{cases} \tag{9}$$

where $e(l, i) = \sum_{k=l}^{i} (y_l - g_i(x_l))^2$ assuming we draw the last line segment of g_i from p_l to p_i.

Running Time: E has n^2 entries, each of which take time $O(n)$ to calculate for a total running time of $O(n^3)$.

10. **Two-Person Traversal of a Sequence of Cities**: You are given an ordered sequence of n cities, and the distances between every pair of cities. Design an algorithm to partition the cities into two subsequences (not necessarily contiguous) such that person A visits all cities in the first subsequence (in order), person B visits all cities in the second subsequence (in order), and the sum of the total distances travelled by A and B is minimized. Assume that person A and person B start initially at the first city in their respective subsequences.

    Solution:

    Recursion Relation: We recurse on $C(i,j)$, the minimum distance traveled if person A ends at city i and person B ends at city j. Assume WLOG $i < j$. The relation is:

    $$C(i,j) = \begin{cases} \sum_{k=1}^{j-1} d(k,k+1) & \text{if } i = 0 \\ \min_{0<k<i}(C(k,j) + d(k,i)) & \text{else} \end{cases} \tag{10}$$

    where $d(i,j)$ is the distance between cities i and j.

    Running Time: There are n^2 entries in $C(i,j)$ and filling in each entry takes $O(n)$ for a total of $O(n^3)$.

11. **Bin Packing**: You have n_1 items of size s_1, n_2 items of size s_2, and n_3 items of size s_3. Design an algorithm to pack all of these items into bins each of capacity C, with $C \geq s_3 \geq s_2 \geq s_1$, such that the total number of bins used is minimized.

    Solution:

    Recursion Relation: We recurse on $B(i,j,k)$, the minimum number of bins of capacity C required to pack i items of size s_1, j items of size s_2 and k items of size s_3. The relation is:

    $$B(i,j,k) = \min_{0\leq i'\leq i, 0\leq j'\leq j, 0\leq k'\leq k}(B(i',j',k') + B(i-i',j-j',k-k')) \tag{11}$$

    where at least one of i', j', or k' must be greater than 0 and one of i', j', or k' must be less than i, j or k respectively.

    Running Time: B is size $O(n^3)$ and computing each element of B requires time $O(n^3)$ for a total time of $O(n^6)$.

12. **Scheduling**: Suppose you have one machine and n jobs, $a_1, ..., a_n$. Each job a_j has processing time t_j, profit p_j, and deadline d_j. The machine can only process one job at a time and that job must run uninterruptedly until completion. If job a_j is completed by deadline d_j, you receive profit p_j, but if it is completed after, you receive nothing. Assuming all processing times are integers between 1 and n and $d_j \geq t_j$ for all jobs, give an algorithm for computing the maximum profit you can make.

    Solution:

    Recursion Relation: We first order the jobs by increasing deadline so $a_1, ..., a_n$ are ordered by deadline. We then recurse on $P(i,j)$, the maximum profit we can make using jobs $1...i$ in time j. The relation is:

    $$P(i,j) = \begin{cases} 0 & \text{if } i = 0 \\ \max(P(i-1,j), P(i-1,j-t_i) + p_i) & \text{if } j \leq d_i \\ P(i-1,j) & \text{else} \end{cases} \tag{12}$$

    Running Time: Since all processing times are no more than n, we need only compute j to n^2. Therefore, P is size $O(n^3)$. Filling in each square of P requires $O(1)$ time for a total running time of $O(n^3)$. Note that the $O(n \log n)$ sorting cost has been absorbed.

13. **Bitonic Tours**: You are given a set of points (x_i, y_i) in a two dimensional plane. You start at the point with the lowest x coordinate and must move strictly right until you reach the point with the highest x coordinate, after which you must move strictly left until you return to the starting point. Give an algorithm for determining the shortest distance you can travel.

    Solution:

    Recursion Relation: We first order the points $p_1 = (x_1, y_1), ..., p_n = (x_n, y_n)$ by increasing x coordinate. We then iterate over $D(i,j)$ with $i < j$, where $D(i,j)$ is the length of the optimal bitonic path containing all points less than p_i. This path begins at p_i, goes strictly left until it reaches p_1 and then goes strictly right until reaches p_j. Let $|p_i - p_j|$ be the Euclidean distance between points i and j. The recursion is:

$$D(i,j) = \begin{cases} |p_1 - p_2| & \text{if } i = 1 \text{ and } j = 2 \\ \min(D(i,j-1) + |p_j - p_{j-1}|, D(j,j-1) + |p_{j-1} - p_{j+1}| + \sum_{k=j+1}^{i} |p_k - p_{k+1}|) & \text{else} \end{cases} \tag{13}$$

    The minimum tour is $\min_{j<n} D(n,j) + |p_n - p_j|$.

    Intuitively, either $j-1$ is the point before j on the path from 1 to j or it is on the path from i to 1. You can (and should) prove correctness formally by induction.

    Running Time: D is size n^2. If we precompute $\sum_{k=j+1}^{i} |p_k - p_{k+1}|$ for all i and j in $O(n^2)$ time, we can compute each entry of D in constant time for a total running time of $O(n^2)$.