Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, and if applicable, efficiency, as well as correctness. Out of 80 points.

**Backtracking and Dynamic Programming**

For the problems below, a correct backtracking recursive algorithm is given, with an example that illustrates how the back-tracking algorithm works.

You need to: A: Give a time analysis for the backtracking algorithm, as an upper bound in \(O\) notation (20 points), B: Convert the BT algorithm to a fast iterative dynamic programming algorithm (DP) (30 points), C: give a time analysis for your DP algorithm (20 points), and D: illustrate the array or matrix produced by your DP algorithm on the example given (10 points).

**Grade maximization**

We are taking a class with \(k\) projects. We have \(H\) hours to divide among the projects, and will spend an integer amount of time on each project. For every project \(i\), we are given an array \(G_i[0..H]\) so that, if we spend \(h\) hours on project \(i\), our grade for that project will be \(G_i[h]\). Naturally, \(G_i[0] \leq G_i[1] \leq G_i[2] \leq \ldots G_i[H]\).

We need to allocate \(H\) hours among projects 1...n, i.e., find non-negative integers \(h_1, \ldots h_k\) with \(\sum h_i = H\) in order to maximize

\[
\sum_i G_i[h_i].
\]

The following recursive algorithm for the general grade maximization problem is based on the possible answers to the question: How many hours do I devote to project 1? The possible answers are 0..\(H\). The recursion just returns the best achievable total grade, not the assignment that achieves it.

\[
\text{BTGM}(H, G_1[0..H], \ldots G_k[0..H]);
\]

1. IF \(n = 1\) return \(G_1(H)\).
2. \(\text{BestGrade} \leftarrow 0\).
3. FOR \(h = 0\) to \(H\) do:
4. \(\text{ThisGrade} \leftarrow G_1[h] + \text{BTGM}(H - h, G_2, \ldots G_k)\)
5. IF \(\text{ThisGrade} > \text{BestGrade}\) THEN \(\text{BestGrade} \leftarrow \text{ThisGrade}\).
6. Return \(\text{BestGrade}\).

The tree of recursive calls for the example \(k = 3, H = 3, G_1[0..3] = [0, 70, 80, 90], G_2[0..3] = [0, 30, 100, 100], G_3[0..3] = [0, 35, 75, 100]\) is on a separate page.

Each recursion explores up to \(H + 1\) cases and since \(k\) is reduced by 1 at every level, the depth of the recursion is \(k\). This gives an upper bound of \((H + 1)^k\) for the total number of recursive calls. Since the recursion is bottom-heavy, this means the total time is \(O((H + 1)^k)\).

\(i\) levels down in the recursion, we are considering the assignments \(G_1...G_k\) and have some number of hours \(0 \leq H' \leq H\) left to divide. Thus, we need a \(k \times H + 1\) matrix \(MG[I, H']\) to store the solution to the problem when the assignments are \(G_1...G_k\) and we have \(H'\) hours to distribute. Since \(I\) increases in the recursive calls, a bottom-up order is in decreasing value of \(I\), and any order on \(H'\). Translating the recursion gives:

\[
\text{DPGM}(H, G_1[0..H], \ldots G_k[0..H]);
\]

1. Initialize \(MG[1..k, 0..H]\)
2. FOR \(H' = 0\) TO \(H\) do:
3. \(MG[k, H] \leftarrow G_k[H]\).
4. FOR \(I = k - 1\) downto 1 do:
5. FOR \(H' = 0\) to \(H\) do:
6. \(\text{BestGrade} \leftarrow 0\).
7. FOR \(h = 0\) to \(H'\) do:
8. \( \text{ThisGrade} \leftarrow G_I[h] + MG(H' - h, I + 1] \)

9. IF \( \text{ThisGrade} > \text{BestGrade} \) THEN \( \text{BestGrade} \leftarrow \text{ThisGrade} \).

10. \( MG[I, H''] \leftarrow \text{BestGrade} \).

11. Return \( MG[1, H] \).

This algorithm takes time \( O(H^2k) \), since there are \( Hk \) entries in the array and each one takes up to \( O(H) \) time to fill.

On the example above, the matrix will have last row \( MG[3, H] = (0, 35, 75, 100) \) equal to \( G_3 \). Then for \( I = 2 \), we will have the row \((0, 35, 100, 135)\). Then the row \( I = 1 \) will be \((0, 70, 105, 170)\). The final answer returned is 170. For doing 1 hour of assignments 1, and 2 hours of assignment 2, and no hours of assignment 3.