# Documentation

The problem revolves around a guessing game that aims to minimize the maximum cost of identifying a hidden number in the worst-case scenario. Unlike a simple guessing game, this task requires strategic planning to guarantee the smallest possible price regardless of the outcome. The challenge is rooted in ensuring optimal guesses that account for all possible results while balancing costs across various scenarios. This makes it an ideal candidate for dynamic programming, a method that breaks problems into overlapping subproblems and solves them efficiently.

The game is set up with a range of numbers from 1 to n, and every incorrect guess costs an amount equal to the guessed number. The goal is not just to guess the number but to devise a strategy that minimizes the highest possible cost. For instance, in the case of n=10, guessing strategically can reduce the worst-case cost to 16 dollars. This requires evaluating all possible guesses within a range and selecting the option that guarantees the least loss.

To solve this, a dynamic programming approach is employed. A 2D table, dp[i][j], is used to store the minimum cost required to guess a number correctly for every possible range [i, j]. The problem is solved iteratively by starting with smaller ranges, such as single numbers or adjacent pairs, for which the cost is straightforward. The solution is then extended to larger ranges by calculating the cost for each possible guess and determining the worst-case scenario for the resulting subranges.

Each number k within the range is considered a potential guess for a range ][i, j]. The cost of choosing k is calculated as k+max(dp[i][k−1],dp[k+1][j]), where the two terms represent the cost of the left and right subranges, respectively. The value of dp[i][j] is updated with the minimum cost obtained across all guesses k. The final result, representing the minimum cost to guarantee a win for numbers from 11 to n, is stored in dp[1][n].

The time complexity of this approach is $O(n^3)$, as each of the $O(n^2)$ ranges involves $O(n)$ computations to evaluate all possible guesses. The space complexity is $O(n^2)$ due to the 2D DP table used to store intermediate results. While this may seem computationally intensive, it is feasible given the constraints, with n up to 200.

This problem demonstrates the practical application of decision-making under uncertainty, a concept often encountered in areas like risk management and game theory. It emphasizes the importance of balancing risks and rewards, even in seemingly straightforward scenarios. The dynamic programming approach not only provides an optimal solution but also serves as a robust example of how to tackle complex problems systematically by leveraging overlapping subproblems and efficient computation. Understanding this solution offers valuable insights into algorithmic strategies and optimization techniques.