□ Predict the Winner – Documentation

1. Problem Statement

You are given an array of integers nums, representing scores on cards arranged in a row. Two players take turns picking either the first or last card, adding the chosen card's value to their score.

- Player 1 starts first.
- Both players play optimally.
- The game ends when all cards are picked.

Goal: Return True if Player 1 can win or draw, otherwise return False.

2. Intuition

Instead of tracking each player’s score separately, we track the score difference (Player 1 score − Player 2 score). At each move, a player wants to maximize their advantage while minimizing the opponent’s.

This naturally fits a recursive minimax strategy with memoization.

3. Key Observations

- Both players know the entire array and make decisions optimally.
- At each turn, the player has only two choices: pick from either end.
- The optimal strategy can be framed as a maximizing vs. minimizing problem, similar to game theory.
- We only need to know if Player 1 can tie or beat Player 2.

4. Approach

Step-by-step:

i. Define a recursive function dp(left, right) that returns the maximum score difference the current player can achieve from subarray nums[left:right+1].
ii. If only one element is left, return that element (nums[left]).
iii. For every choice, subtract the opponent's best move:
    a. Choosing left: nums[left] - dp(left + 1, right)
    b. Choosing right: nums[right] - dp(left, right - 1)
iv. Take the maximum of these two choices.
v. Use lru_cache to store already computed results and avoid recomputation.
vi. Player 1 can win or draw if the final difference dp(0, n-1) is >= 0.

5. Edge Cases

- nums has only one element → Player 1 wins by default.
- All elements are equal → Game will always result in a draw.
- The array is already sorted in descending order → Player 1 usually has an advantage.

6. Complexity Analysis

🔢 Time Complexity:

- $O(n^2)$ because we are solving each (left, right) subproblem once using memoization.

💾 Space Complexity:

- $O(n^2)$ for memoization table stored in the recursion cache.
- O(n) recursion stack depth in worst case.

7. **Alternative Approaches**
    - Pure Recursion
        - Without memoization, leads to exponential time complexity ($O(2^n)$).
    - Bottom-Up DP (Tabulation)
        - Can be implemented using a 2D DP table.
        - Slightly faster due to iterative control but more complex to implement cleanly.

8. **Test Cases**

Test Case	Input	Output	Explanation
Basic Win	[1, 5, 233, 7]	True	Player 1 picks 1, then gets 233.
Basic Loss	[1, 5, 2]	False	Player 2 will end with a higher score.
Single Element	[5]	True	Player 1 picks the only number.
All Equal Elements	[3, 3, 3, 3]	True	Game ends in a draw; Player 1 wins by rule.
Descending Order	[9, 7, 5, 1]	True	Player 1 always picks highest.

9. **Final Thoughts**

    - This problem teaches optimal decision-making in games.
    - Using recursive minimax with memoization is a powerful technique in game-related problems.
    - You can also try the bottom-up solution for learning tabular DP.
    - The problem is great practice for understanding how score difference tracking simplifies multi-player problems.