## Python Use Case:

**Problem Description:**

Two players play a card game with the following rules:

1. Cards are laid out in a row, each with a numerical value.
2. Players take turns picking either the leftmost or rightmost card.
3. The goal is to maximize total points.
4. Both players play optimally, meaning each tries to maximize their own score and minimize the opponent's future score.

**Input:**

- Array `cards[0..n-1]` of integers.

**Output:**

- Maximum score first player can achieve.
- Sequence of moves: `"left"` or `"right"`

```
def optimal_strategy_with_moves(cards):
    n = len(cards)
    dp = [[0]*n for _ in range(n)]
    move = [[""]*n for _ in range(n)]

    # Base cases
    for i in range(n):
        dp[i][i] = cards[i]
        move[i][i] = "left"
```

```
for i in range(n-1):
    if cards[i] >= cards[i+1]:
        dp[i][i+1] = cards[i]
        move[i][i+1] = "left"
    else:
        dp[i][i+1] = cards[i+1]
        move[i][i+1] = "right"

# Fill DP table for subarrays of length 3 to n
for length in range(3, n+1):
    for i in range(n-length+1):
        j = i + length - 1
        pick_left = cards[i] + min(dp[i+2][j] if i+2 <= j else 0,
                                   dp[i+1][j-1] if i+1 <= j-1 else 0)
        pick_right = cards[j] + min(dp[i+1][j-1] if i <= j-2 else 0,
                                    dp[i][j-2] if i <= j-2 else 0)
        if pick_left >= pick_right:
            dp[i][j] = pick_left
            move[i][j] = "left"
        else:
            dp[i][j] = pick_right
            move[i][j] = "right"

# Reconstruct moves
i, j = 0, n-1
moves = []
while i <= j:
    if move[i][j] == "left":
        moves.append("left")
        # Simulate opponent's optimal choice
        if i+1 <= j and dp[i+1][j] <= dp[i][j-1 if j-1 >= i else i]:
            i += 1
        else:
            j -= 1
        i += 0
```

```
        else:
            moves.append("right")
            if i <= j-1 and dp[i][j-1] <= dp[i+1][j]:
                j -= 1
            else:
                i += 1
            j -= 0

    return dp[0][n-1], moves

# Example usage
cards = [3, 9, 1, 2]
score, moves = optimal_strategy_with_moves(cards)
print("Maximum score:", score)
print("Winning moves sequence:", moves)
```

**Explanation:**

1. First player picks **2 (rightmost)**.
2. Opponent picks **3 (leftmost)**.
3. First player picks **9 (leftmost)**.
4. Game ends. Total score for first player = 2 + 9 = 11.