The problem exhibits optimal substructure and overlapping subproblems.

Dynamic Programming (DP) efficiently solves this by storing intermediate results.

Steps:

1. Dp Table Definition:

Dp [i , j] : Maximum net score differences the current player can achieve from boxes i to j.

2. Base Case:

For a single box (i == j), the player takes it:

Dp [i , i] = boxes[i]

Dp [i] [j] = max

(

boxes[i] - Dp [i + 1] [j], // Take left, opponent faces i+1, j

boxes[j] - Dp [i] [j - 1] // Take right, opponent faces i , j – 1 .

)

Example Input: [2, 7, 3]

1. Base Cases:

Dp [0] [0] = 2 , Dp [1] [1] = 7 , Dp [2] [2] = 32.

Length 2 Intervals:

[0,1] : max(2 – 7 , 7 – 2) = max (- 5 , 5) = 5

[1 , 2] : max(7 – 3 , 3 – 7) = 43.

Length 3 Interval ([0 , 2]) :

max(2 – 4 , 3 – 5) = max(-2 , -2) = -2

Output: -2(Alex's net score).

Time Complexity :

Time: O (n^2) due to nested loops over intervals.

Space: O (n^2) for the Dp table.

This DP approach ensures both players' optimal strategies are evaluated without redundancy. By solving smaller subproblems first, the algorithm efficiently computes the final result for the entire array.