Use case - Finding the winning strategy in a card game in

Problem Description:- Imagine a card game where each p
receives a hand of cards with values. The objective is to
the best way to maximize the score for a player take
from drawing cards - Each player can either pick the
first or last card from the remaining pile.

Assumptions:

- Each player tries to maximize their score.

Cards are represented by integers, which include their value

· Two player alternate turns, and each player picks a card from
either the begining or the end of the list.

You need to design an algorithm that helps a player
find the optimal strategy to guarantee the highest possible
score given that the opponent is also playing optimally.

plan:

We can solve this problem using Dynamic programming
By Calculating The optimal Score for every possible scenario,
taking into account the Best choices for both players.

Steps:-

1. Define The Game: Represent the pile of cards as a
list of integers.

2. Recursive Strategy:- A function will recursively
determine the Best score a player can achieve

3. Dynamic programming:- Stor intermediate results
to avoid recalculating them.

4. Base cases:- when only one card is left, the
current player takes it.

Program:-

```
def find-optimal - strategy (cards):
    n= len (cards)
#create a memoization table to store subproblem results.
dp=[10]*n for-in range (n)]
#Fill the table for subproblems of increasing sizes ~~for leng~~
for length in range (1,n+1):
    for i in range (n length +1):
    J = i+length -1
      #If only one card is left, the player takes it
       if i==J
      dp [i][J] = cards [i]
     else:
       #Choose the Best of two choices
     #1. Take the left card, and the opponent plays
     optimally on the remaining (i+1,J)
   # 2. Take the right card, and the opponent plays
    optimally on the remaining (i, J-1)
    take-left = cards [i] - dp [i+1] [J]
    take-right = cards [J] - dp [i] [J-1]
    dp [i] [J] = max [take-left, take-right]
  #dp [0] [n-1] will have the optimal score difference
    for the first player
    return [dp [0] [n-1] + sum (cards))
   #Example case
    cards = [3, 9, 1, 2]
print ("First player's optimal score :", find_optimal
                                   - strategy (cards)).
```

Explanation:-

Consider the array of cards: [3, 9, 1, 2].

1. First player (you) can choose two
   * taking the left most card (3), leaving the cards
   * taking the right most card (2), leaving the cards [3,
2. The opponent will then take their turn, playing optim
to minimize the first player's score. This program
Computes the Best possible outcomes for the first
player.

First player's optimal score : 5

First player, if playing optimally can guarantee
a score of 5 regardless of how the opponent plays.

Optimizing strategy:-

By using Dynamic programming, we ensure
that the solution is computed efficiently, avoiding
redundant Calculations, This approach ensures both
players play optimally, and the first gets the
highest Score possible given the opponent's best
move.

VELTECH	
EX No.	13
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	[illegible] 15/10