Assignment Date	18 November 2022
Student Name	K.Ananthavalli
Student Roll Number	815819104004
Maximum Marks	

Utilization of algorithm dynamic programming,optimization

Example:

Given an array **arr[]** of **N** elements, the task is to find the minimum cost for reducing the array to a single element in **N-1** operations where in each operation:

- Delete the elements at indices **i** and **i+1** for some valid index **i**, replacing them with their sum.
- The cost of doing so is **arr[i] + arr[i+1]**, where arr[] is the array state just before the operation.
- This cost will be added to the final cost.

Examples:

Input: *arr[] = {3, 4, 2, 1, 7}*

Output: *37*

Explanation:

Remove the elements at 0th and 1st index. arr[] = {7, 2, 1, 7},

Cost = 3 + 4 = 7

Remove 1st and 2nd index elements. arr[] = {7, 3, 7}, Cost = 2 + 1 = 3

Remove 1st and 2nd index elements, arr[] = {7, 10}, Cost = 3 + 7 = 10

Remove the last two elements. arr[] = {17}, Cost = = 7 + 10 = 17

Total cost = 7 + 3 + 10 + 17 = 37

This is the minimum possible total cost for this array.

Input: *arr[] = {1, 2, 3, 4}*

Output: *19*

Explanation:

Remove the 0th and 1st index elements. arr[] = {3, 3, 4}. Cost = 1 + 2 = 3

Remove the 0th and 1st index elements. arr[] = {6, 4}. Cost = 3 + 3 = 6

Remove the 0th and 1st index elements. arr[] = {10}. Cost = 6 + 4 = 10

Total cost = 3 + 6 + 10 = 19.

This is the minimi=um possible cost.

Sub-optimal solution (using Range DP): The problem can be solved using the following idea:

- *Let **arr[]** be the original array before any modifications are made.*
- *For an element in the array that has been derived from indices i to j of a[], the cost of the final operation to form this single element will be the sum **arr[i] + arr[i+1] + . . . + arr[j]**. Let this value be denoted by the function cost(i, j).*
- *To find the minimum cost for the section **arr[i, i+1, … j]**, consider the cost of converting the pairs of sub-arrays **arr[i, i+1 . . . k] & arr[k+1, k+2 . . . j]** into single elements, and choose the minimum over all possible values of **k** from **i to j-1** (both inclusive).*

For implementing the above idea:

- The cost function can be calculated in constant time with preprocessing, using a prefix sum array:
  - Calculate prefix sum (say stored in pref[] array).
  - So cost(i, j) can be calculated as (pref[j] – pref[i-1]).
- Traverse from **i = 0 to N-1**:
  - Traverse **j = i+1 to N-1** to generate all the subarray of the main array:
    - Solve this problem for all these possible subarrays with the following dp transition – **dp[i][j] = cost(i, j) + min$_{i \le k \le j-1}$(dp[i][k] + dp[k+1][j])** as explained in the above idea.
- Here **dp[i][j]** is the minimum cost of applying **(j – i)**

operations on the sub-array **arr[i, i+1, . . . j]** to convert it to a single element.

cost(i, j) denotes the cost of the final operation i.e. the cost of adding the last two values to convert **arr[i, i+1, . . ., j]** to a single element.

- The final answer will be stored on **dp[0][N-1]**.

Below is the implementation of the above approach.

- C++
- Java
- C#
- Javascript

```
// C++ code to implement the approach

#include <bits/stdc++.h>
using namespace std;

// Function to find the minimum cst
int minCost(int arr[], int N)
{
    // Creating the prefix sum array
    int pref[N+1], dp[N][N];
    pref[0] = 0;
    memset(dp, 0, sizeof(dp));

    // Loop to calculate the prefix
sum
    for (int i = 0; i < N; i++) {
        pref[i + 1] = pref[i] +
arr[i];
    }

    // Iterating through all subarrays
    // of length 2 or greater
    for (int i = N - 2; i >= 0; i--) {
        for (int j = i + 1; j < N;
j++) {

            // Cost function = sum of
            // all elements in the
subarray
            int cost = pref[j + 1] -
pref[i];
            dp[i][j] = INT_MAX;
            for (int k = i; k < j;
```

```
k++) {

                    // dp transition
                    dp[i][j]
                    = min(dp[i][j],
dp[i][k]
                          + dp[k + 1][j] +
cost);
               }
          }
     }

     // Return answer
     return dp[0][N - 1];
}

// Driver code
int main()
{
     int arr[] = { 3, 4, 2, 1, 7 };
     int N = sizeof(arr) /
sizeof(arr[0]);

     // Function call
     cout << minCost(arr, N);
     return 0;
}
```

Output