Perfect Square In C++

```cpp
#include <iostream>
#include <vector>
#include <climits>
#include <cmath>
using namespace std;

int main() {
    vector<int> arr = {0, 1, 2, 3, 1, 2, 3, 4, 2, 1, 2, 3};
    int n = arr.size();
    vector<int> dp(n + 1, INT_MAX); // dp array where dp[i] represents the minimum number of perfect squares summing up to i
    //int dp[n+1]={INT_MAX};
    dp[0] = 0; // Base case: 0 requires 0 squares
    dp[1] = 1; // 1 requires 1 square (1)

    for (int i = 2; i <= n; i++) {
        for (int j = 1; j * j <= i; j++) {
            dp[i] = min(dp[i], dp[i - j * j] + 1);
        }
    }

    // Output the dp array
    for (int i = 0; i <= n; i++) {
        cout << dp[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Explanation of the Code:

- **Input Array**: The array you provided is {0, 1, 2, 3, 1, 2, 3, 4, 2, 1, 2, 3}. However, the actual input to the problem is simply the number n, where we want to find the minimum number of perfect squares for all numbers from 0 to n.
- **DP Array (dp)**: The dp array stores the minimum number of perfect squares that sum up to each index value. The array is initialized to INT_MAX to signify that no solution has been found yet, and it is updated with the minimum value as we iterate.
- **Base Cases**:
    - dp[0] = 0: 0 requires no squares.
    - dp[1] = 1: 1 can be represented as a square of 1 (1^2).
- **Recursive Case**:
    - For each value i from 2 to n, the code checks all possible perfect squares j*j that can be subtracted from i. It calculates the minimum value of dp[i] by comparing it with dp[i - j*j] + 1, where +1 accounts for using the square j*j.
- **Output**: The program prints the values in the dp array from index 0 to n.

Example Walkthrough:

The goal is to find the minimum number of perfect squares that sum up to each number from 0 to the length of the array.

DP Table Calculation:

1. **dp[0]** = 0 (Base case: 0 requires 0 squares)
2. **dp[1]** = 1 (Base case: 1 can be written as 1^2)
3. **dp[2]** = 2 (2 can be written as 1^2 + 1^2)
4. **dp[3]** = 3 (3 can be written as 1^2 + 1^2 + 1^2)
5. **dp[4]** = 1 (4 can be written as 2^2)
6. **dp[5]** = 2 (5 can be written as 4 + 1^2)
7. **dp[6]** = 3 (6 can be written as 4 + 1^2 + 1^2)
8. **dp[7]** = 4 (7 can be written as 4 + 1^2 + 1^2 + 1^2)
9. **dp[8]** = 2 (8 can be written as 4 + 4)
10. **dp[9]** = 1 (9 can be written as 3^2)
11. **dp[10]** = 2 (10 can be written as 9 + 1^2)
12. **dp[11]** = 3 (11 can be written as 9 + 1^2 + 1^2)
13. **dp[12]** = 3 (12 can be written as 9 + 1^2 + 1^2

	+ 1^2) **Output:** After the dp array is computed, the output will be: 0 1 2 3 1 2 3 4 2 1 2 3 3
Output:- 0 1 2 3 1 2 3 4 2 1 2 3 3	