1. Perfect Squares (Leetcode-279)

Problem Statement:

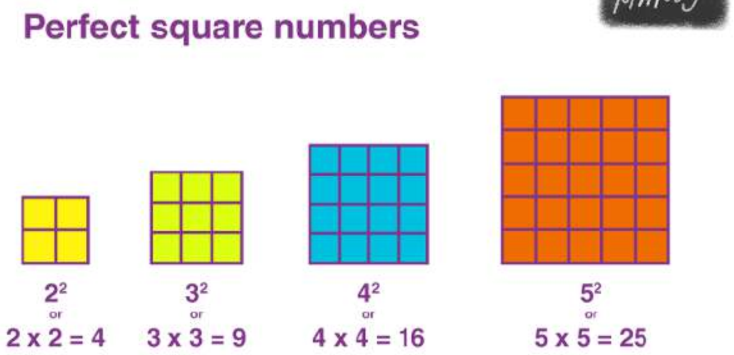
*Given an integer **n**, return the least number of perfect square numbers that **sum to n**.*
A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself.

For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Example 1:
Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.

Example 2:
Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.

Constraints:
1 <= n <= 10^4



PERFECT SQUARE NUMBER OR NOT

$\sqrt{1} = 1 \longrightarrow 1 \times 1 = 1$ P.S.N.

$\sqrt{2} = 1.xx$

$\sqrt{3} = 1.xx$

$\sqrt{4} = 2 \longrightarrow 2 \times 2 = 4$ P.S.N.

$\sqrt{5} = 2.xx$

$\sqrt{6} = 2.xx$

$\sqrt{7} = 2.xx$

$\sqrt{8} = 2.xx$

$\sqrt{9} = 3 \longrightarrow 3 \times 3 = 9$ P.S.N.

Ex1 $N = 12$

output = 3

$i = 1 \times 1$ $1+1+1+ --- +1+1 = 12$ 12 Times 1

$i = 2 \times 2$ $4+4+4 = 12$ 3 Times 4

$i = 3 \times 3$ $9+9 > 12$ ×

$i = 4 \times 4$ $16 > 12$ ×

$\min(3, 12) = 3$

↳ output

Ex2 $N = 13$

output = 2

$i = 1 \times 1$

$1 + 1 + 1 + \dots + 1 = 13$ 13 Times 1

$i = 2 \times 2$ $4 + 4 + 4 + 1 = 13$ 3 Times 4 and 1 Time 1

$3 + 1 = 4$

$i = 3 \times 3$ $9 + 4 = 13$ 1 Time 9 and 1 Time 4

$1 + 1 = 2$

$\min(1, 4, 2) = 2$

↳ output

Logic Build

Ex N=12 output=3

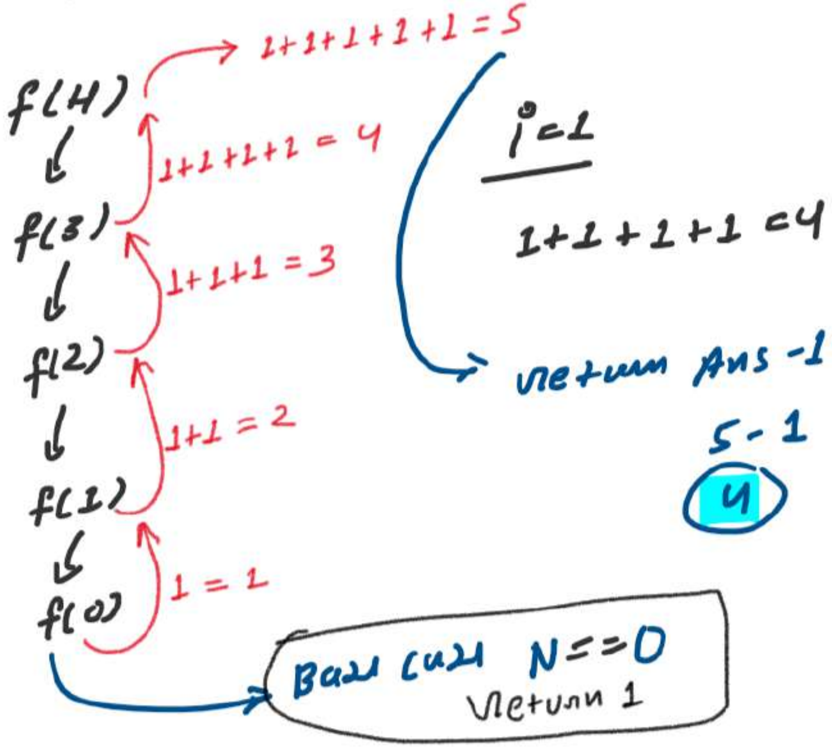
First perfect square numbers

1×1 = 1
2×2 = 4
3×3 = 9
4×4 = 16
5×5 = 25
⋮
$i \times i = i^2$

- $i = 1$
- perfectSqr = $i \times i$
- End = sqrt(N)
  $= \sqrt{12}$
  $= 3$

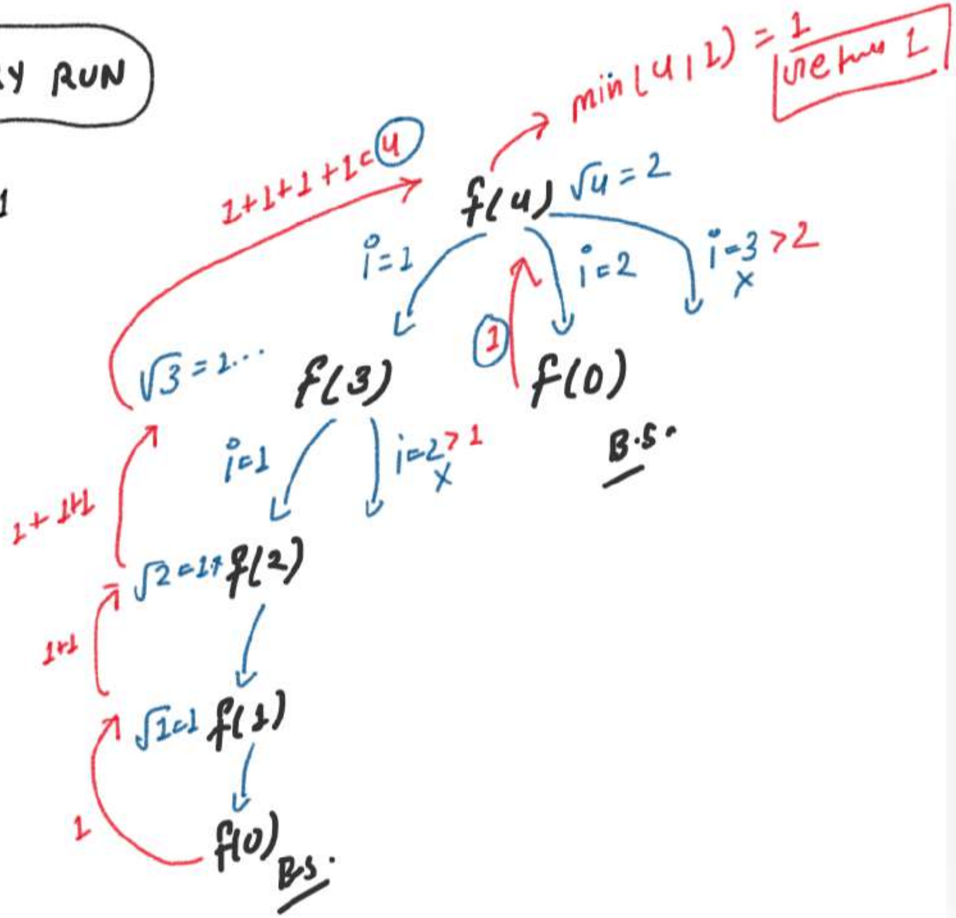
Solve for only 1 using Rec

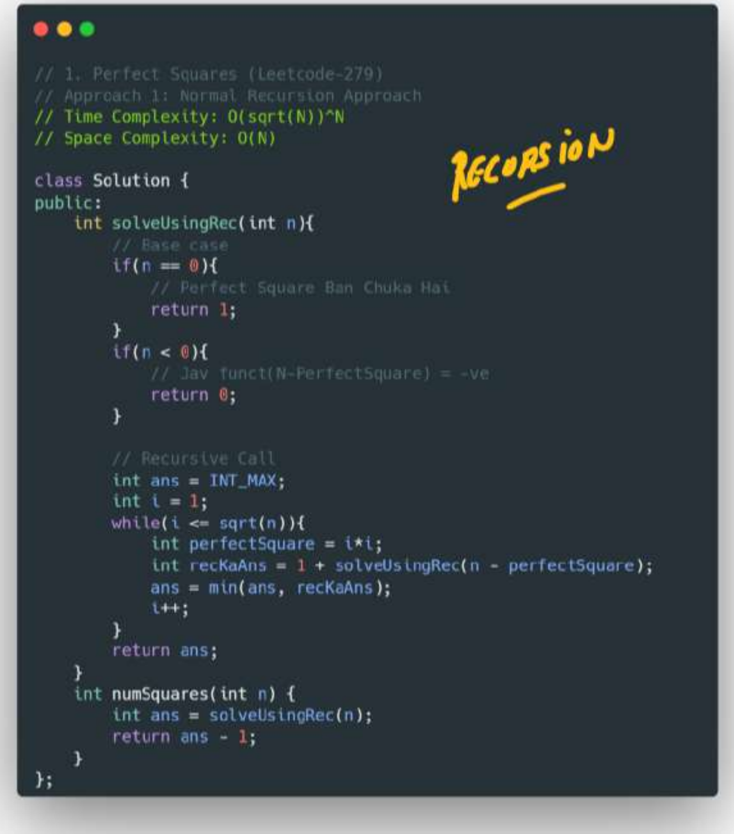
Ex N=4 output=1



Complete DRY RUN

Ex N=4 O/p = 1

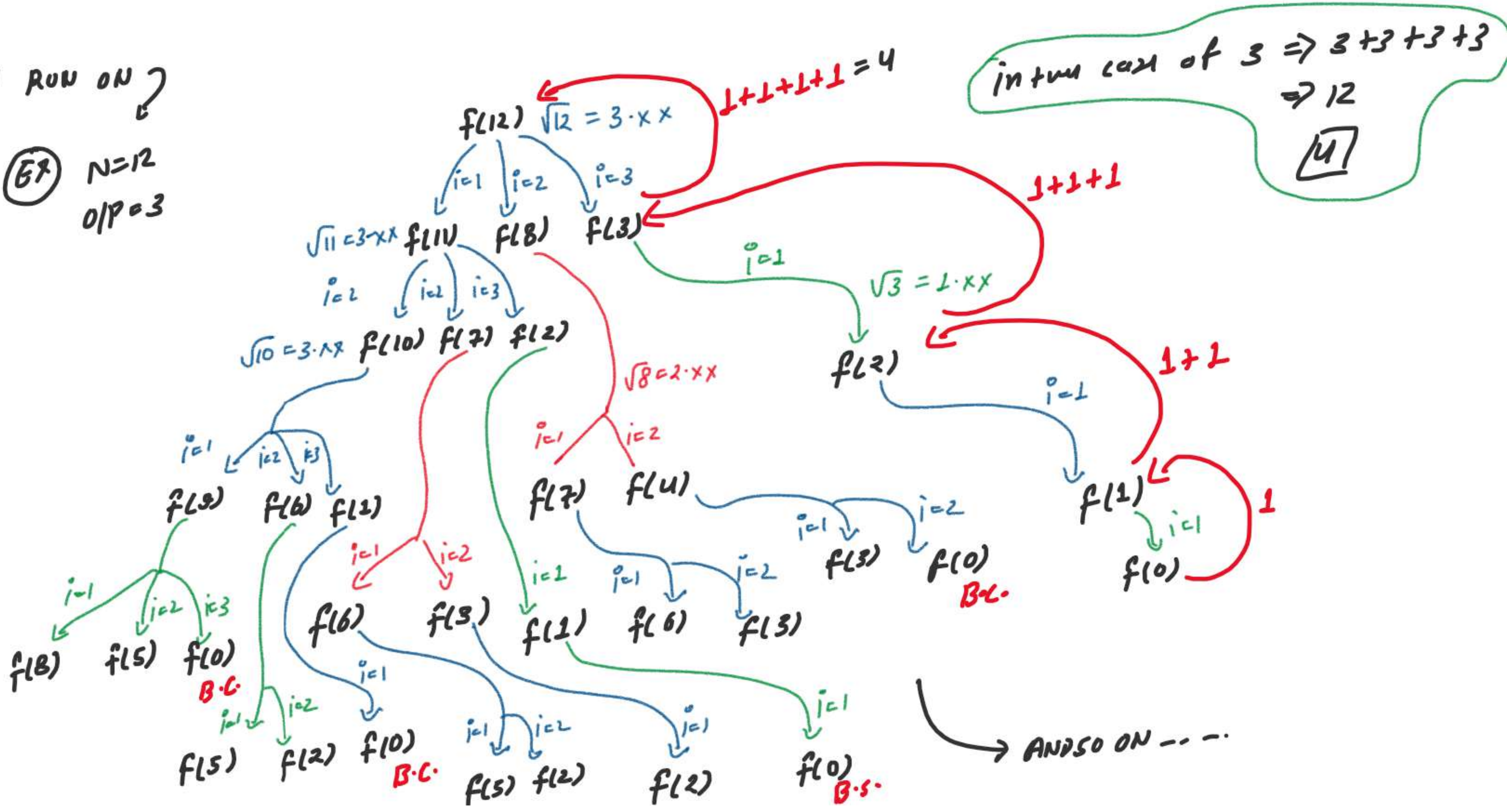




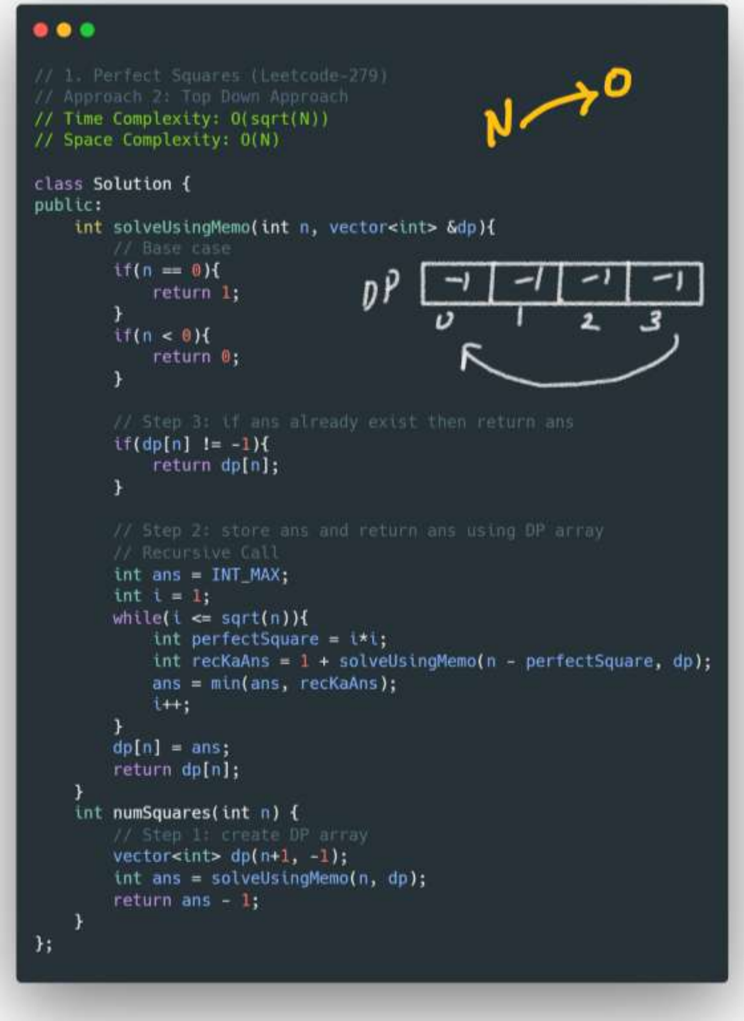
```
// 1. Perfect Squares (Leetcode-279)
// Approach 1: Normal Recursion Approach
// Time Complexity: O(sqrt(N))^N
// Space Complexity: O(N)

class Solution {
public:
    int solveUsingRec(int n){
        // Base case
        if(n == 0){
            // Perfect Square Ban Chuka Hai
            return 1;
        }
        if(n < 0){
            // Jav funct(N-PerfectSquare) = -ve
            return 0;
        }

        // Recursive Call
        int ans = INT_MAX;
        int i = 1;
        while(i <= sqrt(n)){
            int perfectSquare = i*i;
            int recKaAns = 1 + solveUsingRec(n - perfectSquare);
            ans = min(ans, recKaAns);
            i++;
        }
        return ans;
    }
    int numSquares(int n) {
        int ans = solveUsingRec(n);
        return ans - 1;
    }
};
```

DRY RUN ON ↲

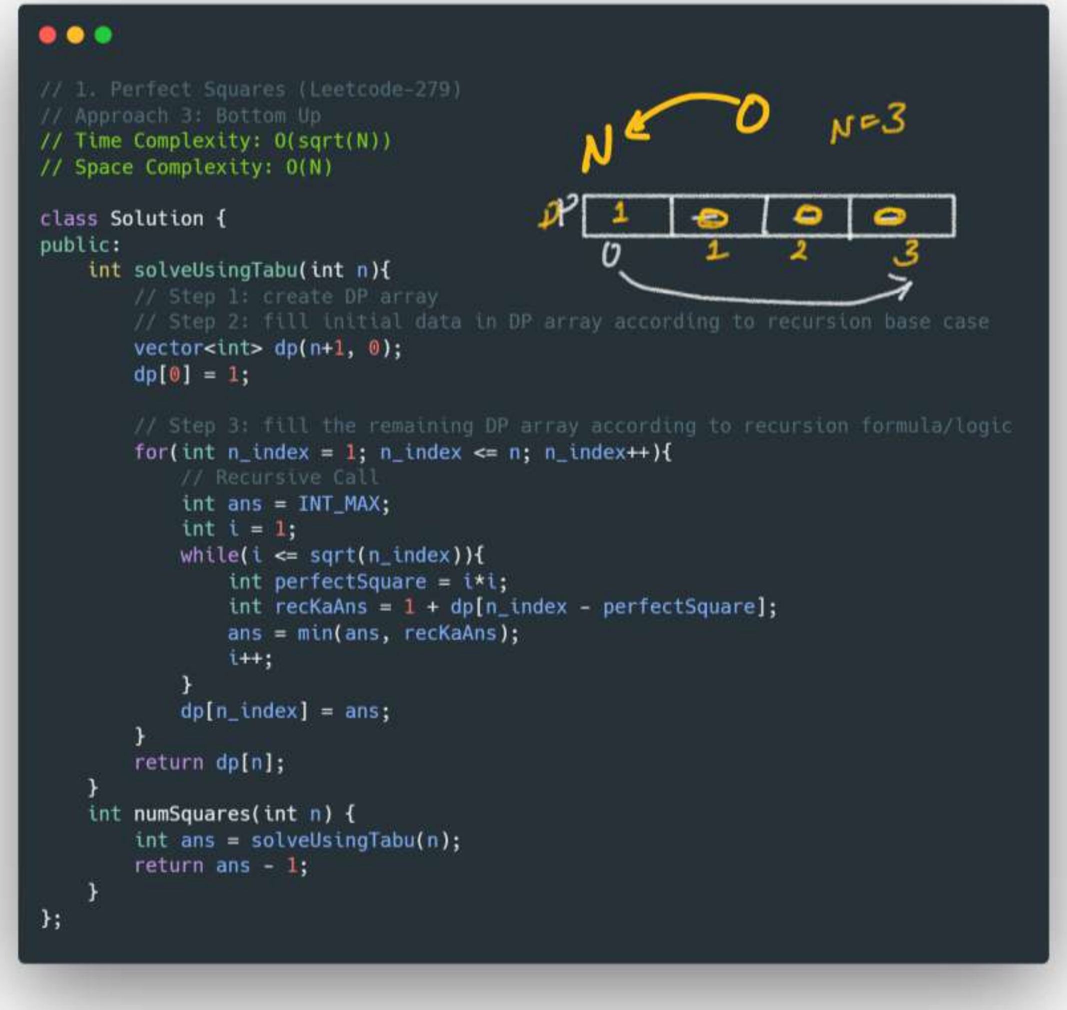
EX N=12
O/P=3



in that case of 3 ⇒ 3+3+3+3
⇒ 12
$\boxed{4}$



```
// 1. Perfect Squares (Leetcode-279)
// Approach 2: Top Down Approach
// Time Complexity: O(sqrt(N))
// Space Complexity: O(N)

class Solution {
public:
    int solveUsingMemo(int n, vector<int> &dp){
        // Base case
        if(n == 0){
            return 1;
        }
        if(n < 0){
            return 0;
        }

        // Step 3: if ans already exist then return ans
        if(dp[n] != -1){
            return dp[n];
        }

        // Step 2: store ans and return ans using DP array
        // Recursive Call
        int ans = INT_MAX;
        int i = 1;
        while(i <= sqrt(n)){
            int perfectSquare = i*i;
            int recKaAns = 1 + solveUsingMemo(n - perfectSquare, dp);
            ans = min(ans, recKaAns);
            i++;
        }
        dp[n] = ans;
        return dp[n];
    }
    int numSquares(int n) {
        // Step 1: create DP array
        vector<int> dp(n+1, -1);
        int ans = solveUsingMemo(n, dp);
        return ans - 1;
    }
};
```



```
// 1. Perfect Squares (Leetcode-279)
// Approach 3: Bottom Up
// Time Complexity: O(sqrt(N))
// Space Complexity: O(N)

class Solution {
public:
    int solveUsingTabu(int n){
        // Step 1: create DP array
        // Step 2: fill initial data in DP array according to recursion base case
        vector<int> dp(n+1, 0);
        dp[0] = 1;

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int n_index = 1; n_index <= n; n_index++){
            // Recursive Call
            int ans = INT_MAX;
            int i = 1;
            while(i <= sqrt(n_index)){
                int perfectSquare = i*i;
                int recKaAns = 1 + dp[n_index - perfectSquare];
                ans = min(ans, recKaAns);
                i++;
            }
            dp[n_index] = ans;
        }
        return dp[n];
    }
    int numSquares(int n) {
        int ans = solveUsingTabu(n);
        return ans - 1;
    }
};
```