Day 9

03 December 2022 15:50

$n = 2$

0	1	1
0	1	2

Answer

$n=0$	$n=1$	$n=2$
$(0000)_2$	$(0001)_2$	$(0010)_2$

$n=5$

0	1	2	3	4	5
$(0000)_2$	$(0001)_2$	$(0010)_2$	$(0011)_2$	$(0100)_2$	$(0101)_2$

0	1	1	2	1	2
0	1	2	3	4	5

Qustion

$n=5$

```
for(int i=0; i<=n; i++){
    (1) Binary Representation
    (2) Count No of 1's
```

$$0 \leq n \leq 10^5$$

$$n = 10^5$$

0 10^5

count no's of 1's

count no's of 1's

$n=5$

$5 \% 2 \Rightarrow 1$ $\quad 5/2 = 2$

$2 \% 2 \Rightarrow 0$ $\quad 2/2 = 1$

$1 \% 2 \Rightarrow 1$ $\quad 1/2 = 0$

$(101)_2$

$n=7$

$7 \% 2 \Rightarrow 1$ $\quad 7/2 = 3$

$3 \% 2 \Rightarrow 1$ $\quad 3/2 = 1$

$1 \% 2 \Rightarrow 1$ $\quad 1/2 = 0$

$(111)_2$

```
for(int i=0; i<=n; i++)
{
ans[i] = no_of_1s(i);
}
```

(Binary Representation of 1's bit count)

Dynamic Programming

$n=10$

$n=0$ $\quad n=1$ $\quad n=2$ $\quad n=3$ $\quad n=4,$

$(0000)_2$ $(0001)_2$ $(0010)_2$ $(0011)_2$ $(0100)_2$

0 1 1 2 1

$n=5$ $(0101)_2$ 2

$n=6$ $(0110)_2$ 2

$n=7$ $(0111)_2$ 3

$n=8$ $(1000)_2$ 1

$n=9$ $(1001)_2$ 2

$n=10$ $(1010)_2$ 2

$n=0$ (0)
$n=1$ (1)
$n=2$ (1)
$n=3$ (2)
$n=4$ (1)
$n=5$ (2)

$n=6$ (2)
$n=7$ (3)
$n=8$ (1)
$n=9$ (2)
$n=10$ (2)

n is even $\rightarrow$ $\lfloor n/2 \rfloor$

n is even

$n=2$ (1) (1)
$n=4$ (2) (1)
$n=6$ (3) (2)
$n=8$ (4) (1)
$n=10$ (5) 2

$n/2$

$n=20$ $(10) \rightarrow (2)$

$16+4$

$(10100)_2$

n is even

no of 1's bit in n
=
(no of 1's bit in $n/2$)

n is odd

n=1 (1)

n=3 (2)

n=5 (2)

n=7 (3)

n=11 (3)

(8+3) $(1011)_2$

n = 23

n= 9(2)

n=1 (0) (0) +1 =(1)

n= 3 (1) (1) + 1 =(2)

n = 5 (2) (1) + 1 =(2)

n= 7 (3) (2) + 1 =(3)

n= 9 (4) (1) +1 =(2)

n/2 (11) (3)+1 = 4

23 16 + 7

$(10111)_2$ (4)

if n is even

no of 1's bit in n = no of 1's bit in n/2

if n is odd

no of 1's bit in n = (no of 1's bit in n/2)+1

Recursion

n=0 (0)

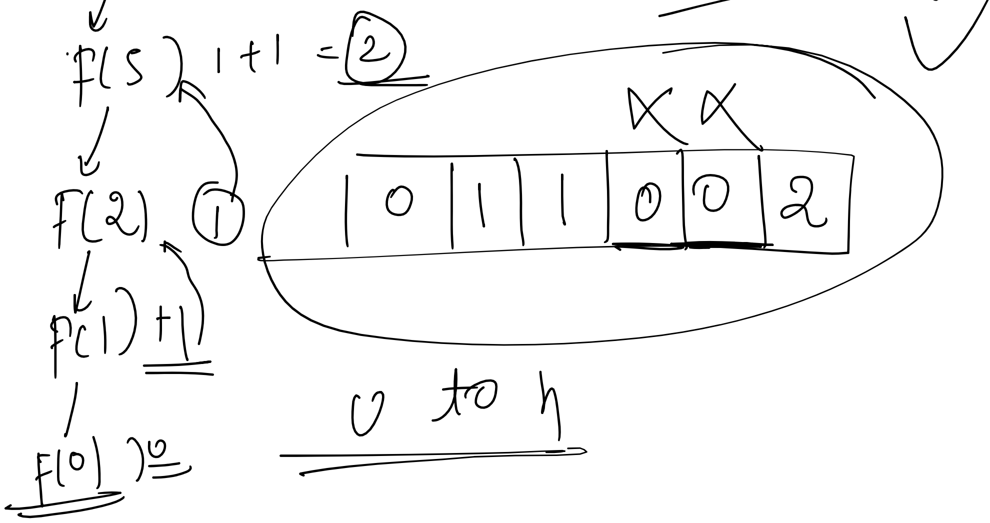
int no_of_1sbit(int n, vector<int> &ans){

```
    if (n == 0) return 0;  ✓
  [ if (n%2 == 0) //even
    { ans[n] = no_of_1sbit(n/2, ans);
    }else{
        ans[n] = no_of_1sbit(n/2, ans)+1;
    }
    no_of_1sbit(n-1, ans); //
    return ans[n];
}
```

T.C — O(2^N)

S.C — O(N) ✓

n = 5



TLE

Recursion (80%) ★

↓ Memoization ★ (Top to Bottom) (10%)

↓ Tabulation (Bottom to Top)

Memoization

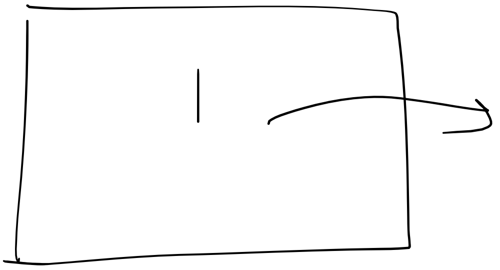
0	-1	-1	-1	-1	-1
0	1	2	3	4	5

T.C — O(N)
S.C — O(N)

① Auxillary Array (n+1)

② Base Case → Auxillary Array Initialize

③

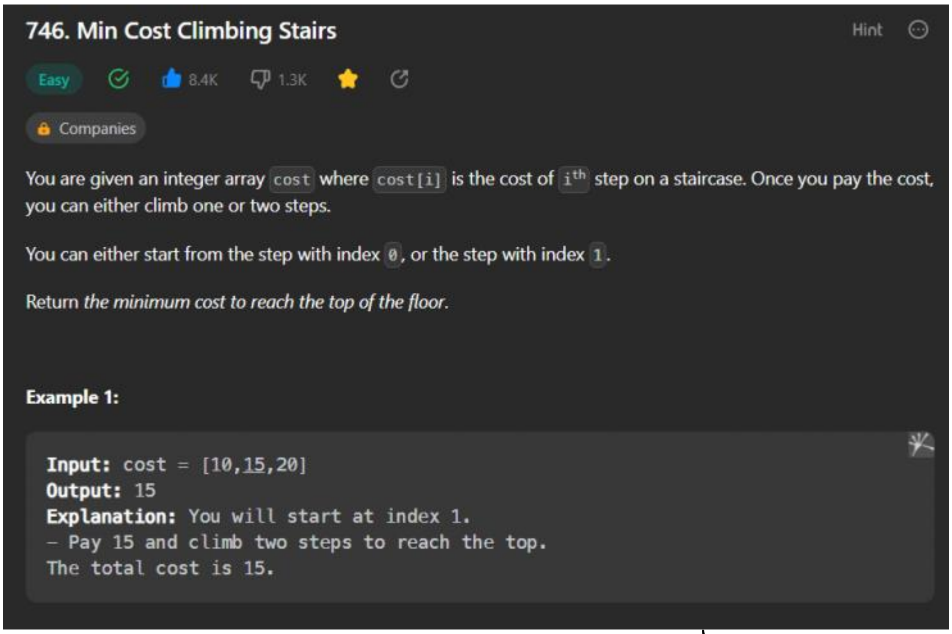


Recursive

Auxillary Array

T.C — O(N)
S.C — O(1)

Recursion	Memoization	Tabulation
T.C — O(2^N)	T.C — O(N)	T.C — O(N)
S.C — O(N	S.C — O(N)	S.C — O(1)



746. Min Cost Climbing Stairs

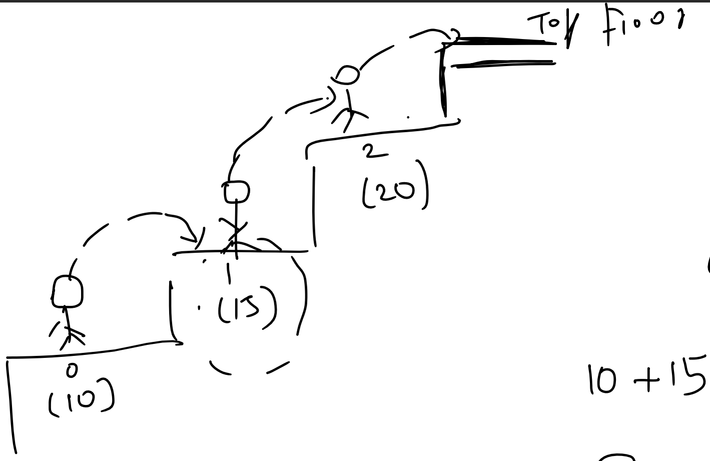
You are given an integer array `cost` where `cost[i]` is the cost of i^{th} step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor.*

Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```



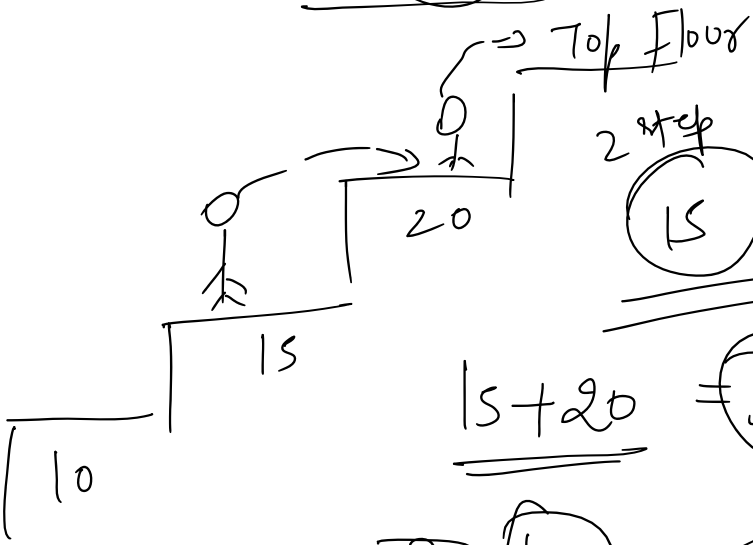
Can|

10 + 15 = $\boxed{25}$

0th index starting

10 + 15 + 20 = $\boxed{45}$

$\geq \boxed{25}$ ✓



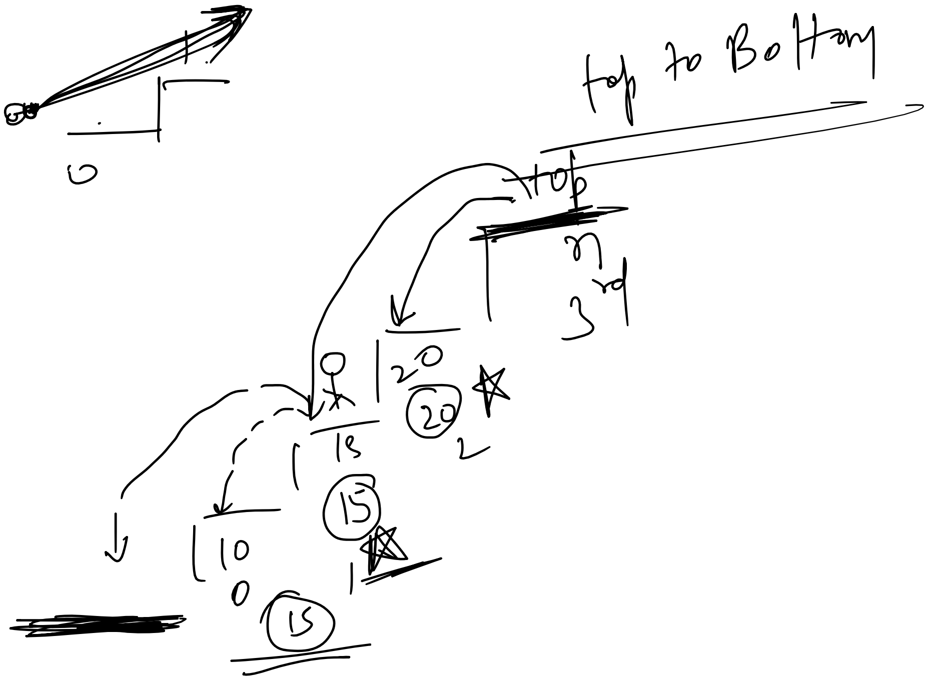
2 step $\boxed{15}$

15+20 = $\boxed{35}$

$\geq \boxed{15}$ ✓

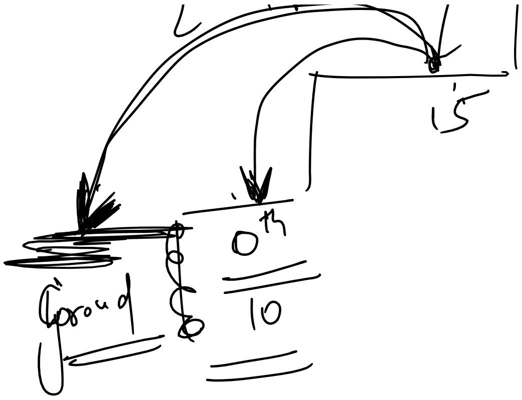
₹ 15

Recursion (top to bottom)

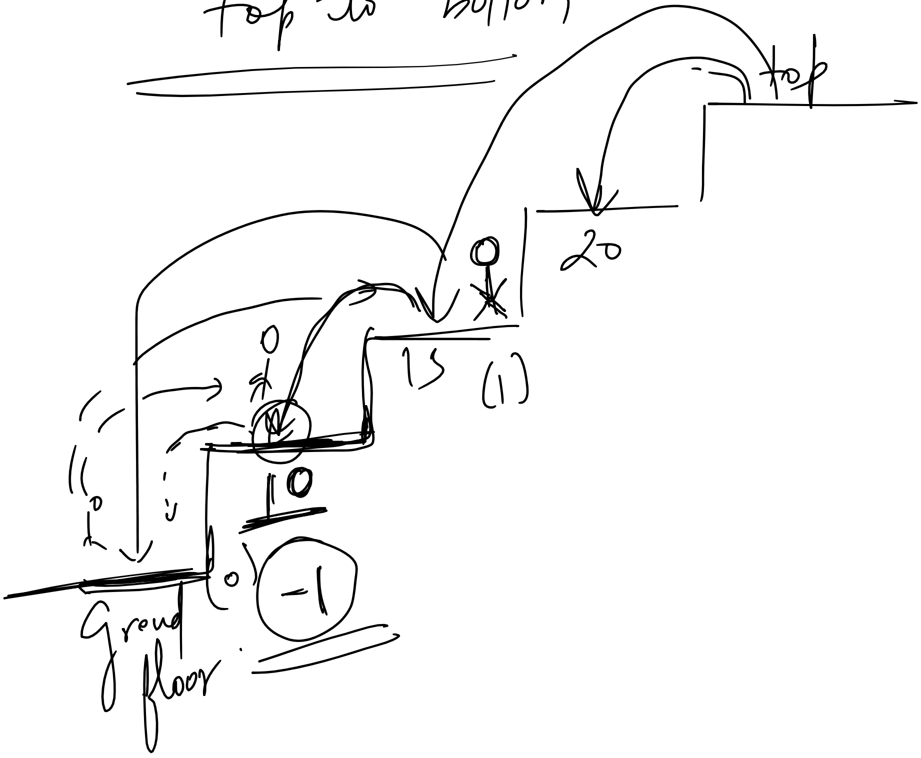


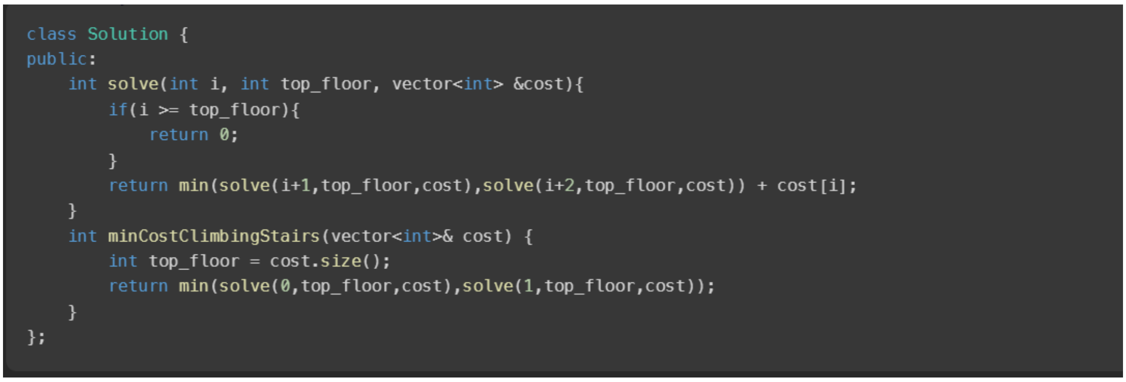
Recursion (top to Bottom)

```
int solve(int n, vector<int> & cost){
        //Base case  if(n<0) return 0;
return min( solve(n-1, cost) + cost[n-1],
            solve(n-2, cost) + cost[n-2] );
}
```

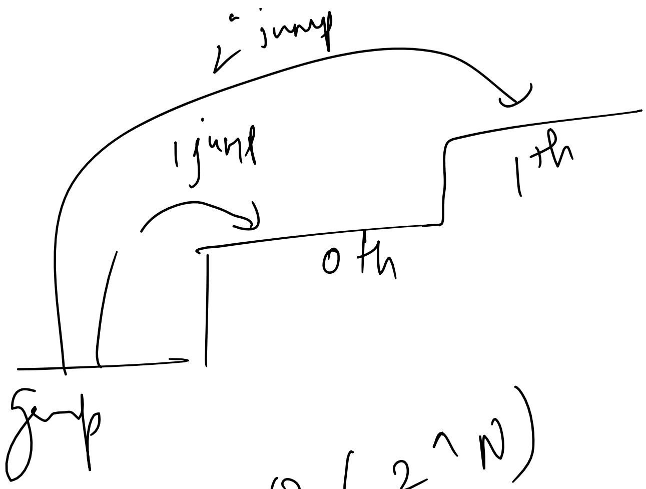


top to bottom



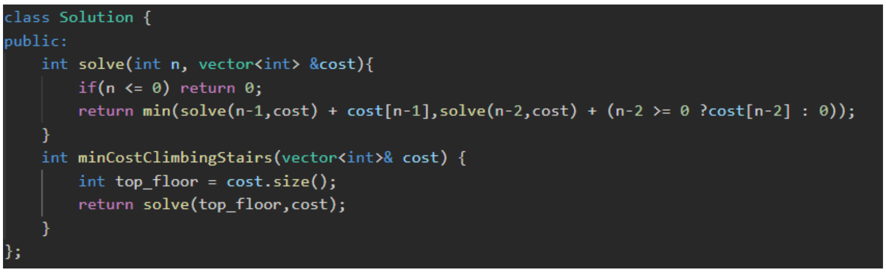


```
class Solution {
public:
    int solve(int i, int top_floor, vector<int> &cost){
        if(i >= top_floor){
            return 0;
        }
        return min(solve(i+1,top_floor,cost),solve(i+2,top_floor,cost)) + cost[i];
    }
    int minCostClimbingStairs(vector<int>& cost) {
        int top_floor = cost.size();
        return min(solve(0,top_floor,cost),solve(1,top_floor,cost));
    }
};
```

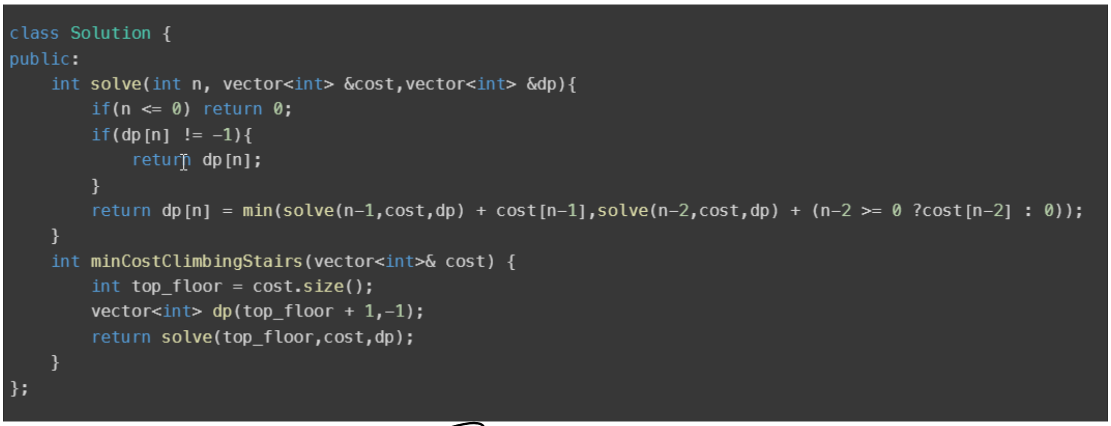


T.C — O(2^N)

S.C — O(N)



```
class Solution {
public:
    int solve(int n, vector<int> &cost){
        if(n <= 0) return 0;
        return min(solve(n-1,cost) + cost[n-1],solve(n-2,cost) + (n-2 >= 0 ?cost[n-2] : 0));
    }
    int minCostClimbingStairs(vector<int>& cost) {
        int top_floor = cost.size();
        return solve(top_floor,cost);
    }
};
```

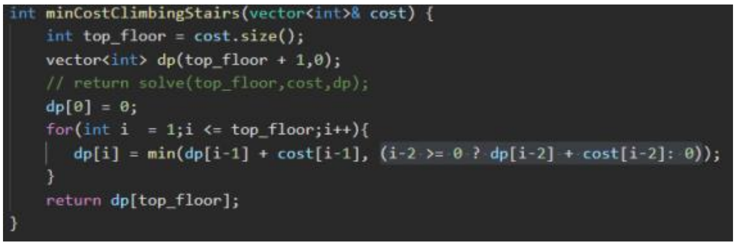


```
class Solution {
public:
    int solve(int n, vector<int> &cost,vector<int> &dp){
        if(n <= 0) return 0;
        if(dp[n] != -1){
            return dp[n];
        }
        return dp[n] = min(solve(n-1,cost,dp) + cost[n-1],solve(n-2,cost,dp) + (n-2 >= 0 ?cost[n-2] : 0));
    }
    int minCostClimbingStairs(vector<int>& cost) {
        int top_floor = cost.size();
        vector<int> dp(top_floor + 1,-1);
        return solve(top_floor,cost,dp);
    }
};
```

T.C — O(N)

S.C — O(N)

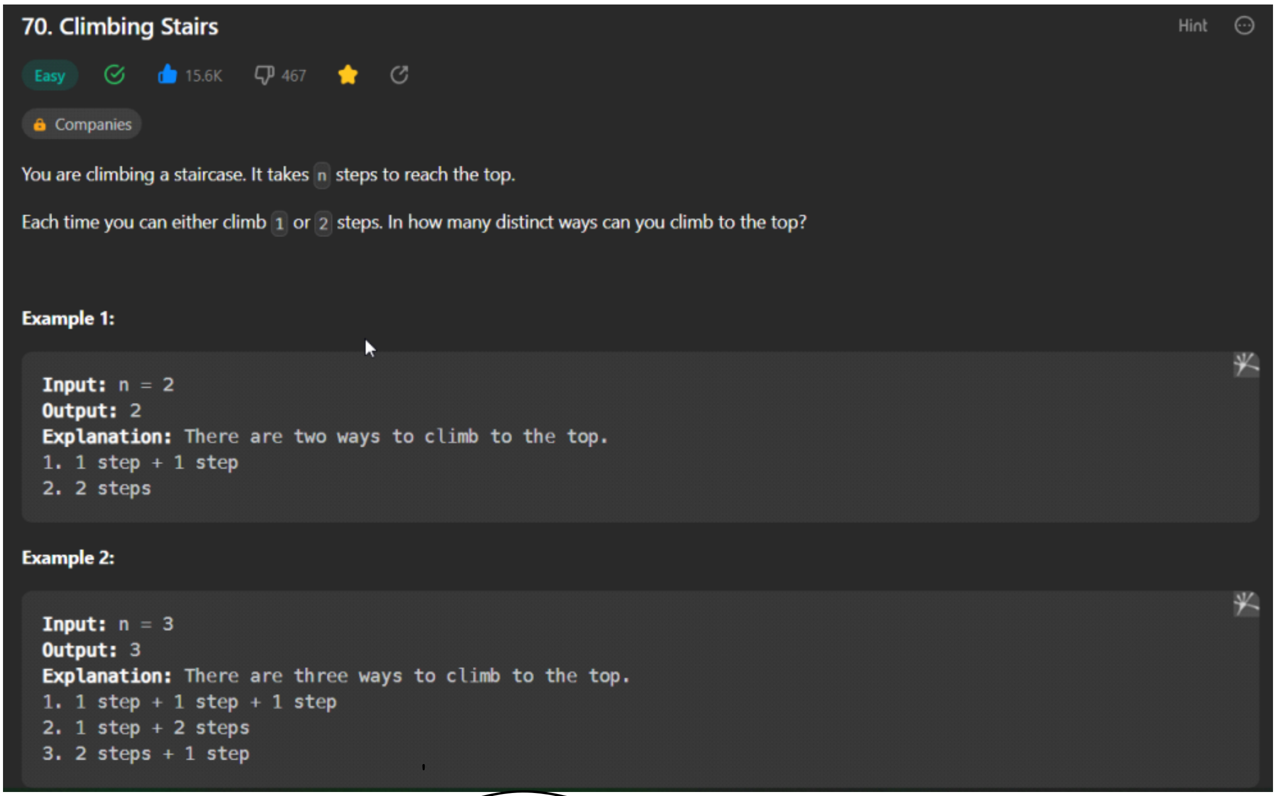
Tabulation



```
int minCostClimbingStairs(vector<int>& cost) {
    int top_floor = cost.size();
    vector<int> dp(top_floor + 1,0);
    // return solve(top_floor,cost,dp);
    dp[0] = 0;
    for(int i  = 1;i <= top_floor;i++){
        dp[i] = min(dp[i-1] + cost[i-1], (i-2 >= 0 ? dp[i-2] + cost[i-2]: 0));
    }
    return dp[top_floor];
}
```

T.C - O(N)

S.C - O(1)



70. Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

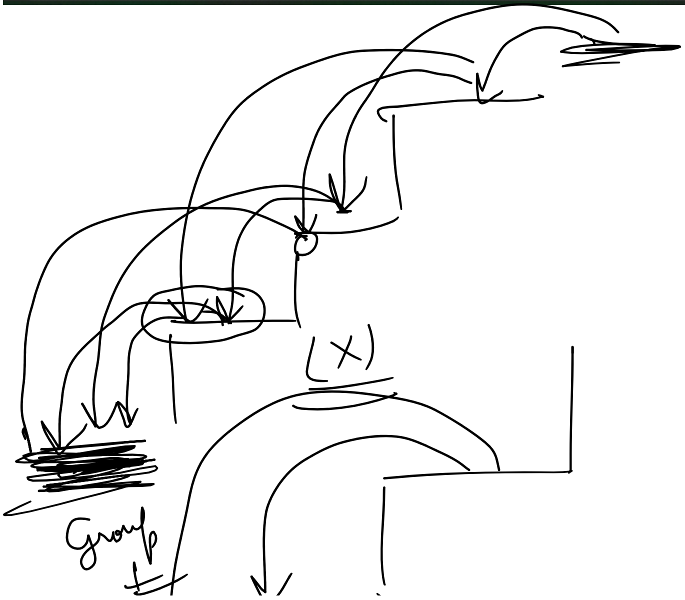
Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

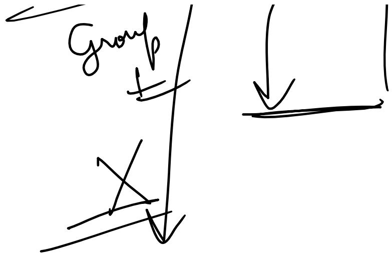
```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

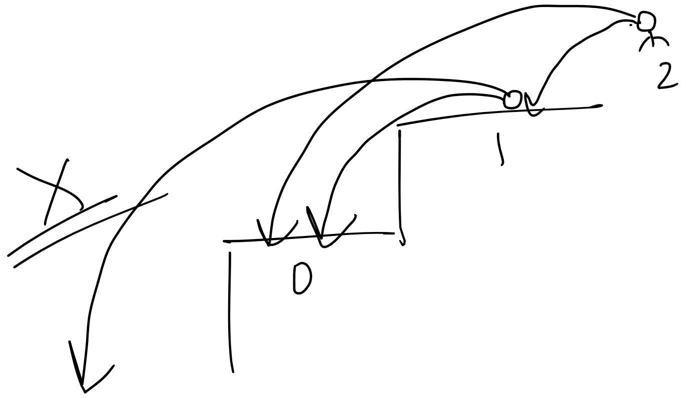
Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```



1 or 2 steps





Distinct

Recursive-calls को Add करनी होती है॥

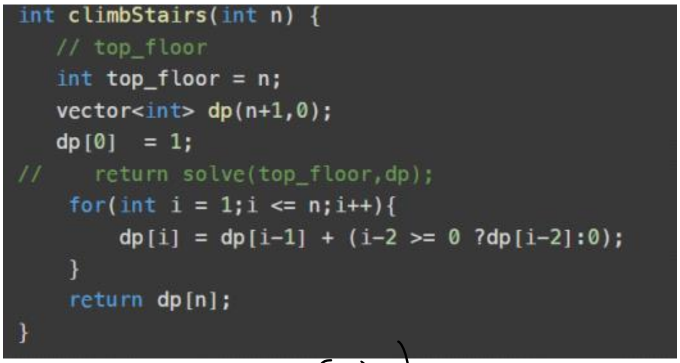
T.C — $O(2^N)$

S.C — $O(N)$

Memoization

T.C — $O(N)$

S.C — $O(N)$



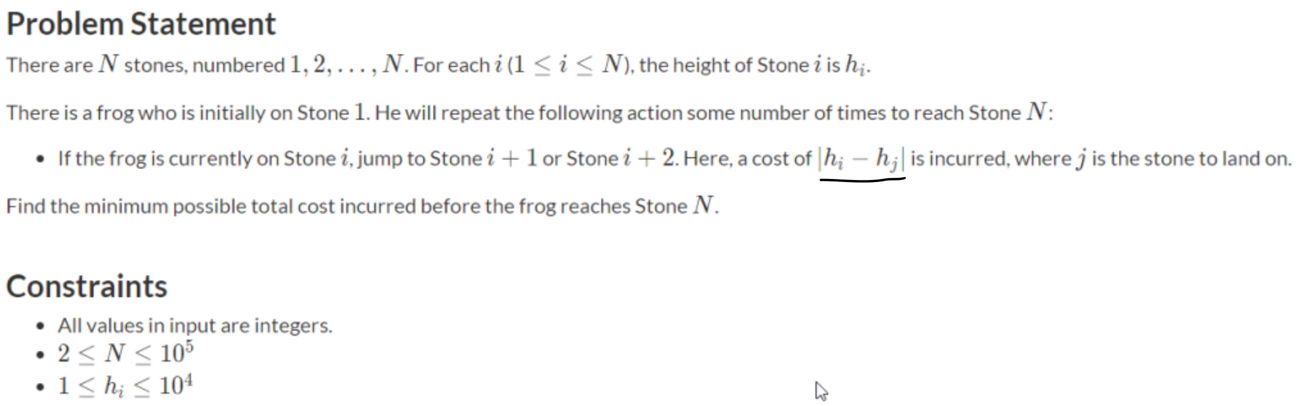
```
int climbStairs(int n) {
   // top_floor
   int top_floor = n;
   vector<int> dp(n+1,0);
   dp[0]  = 1;
//     return solve(top_floor,dp);
    for(int i = 1;i <= n;i++){
        dp[i] = dp[i-1] + (i-2 >= 0 ?dp[i-2]:0);
    }
    return dp[n];
}
```

— (n)

[redacted]

T.C — O(N)

S.C — O(1)



Problem Statement

There are N stones, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), the height of Stone i is h_i.

There is a frog who is initially on Stone 1. He will repeat the following action some number of times to reach Stone N:

- If the frog is currently on Stone i, jump to Stone $i + 1$ or Stone $i + 2$. Here, a cost of $\underline{|h_i - h_j|}$ is incurred, where j is the stone to land on.

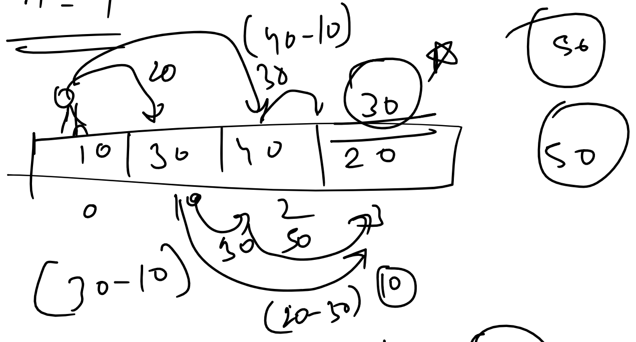
Find the minimum possible total cost incurred before the frog reaches Stone N.

Constraints

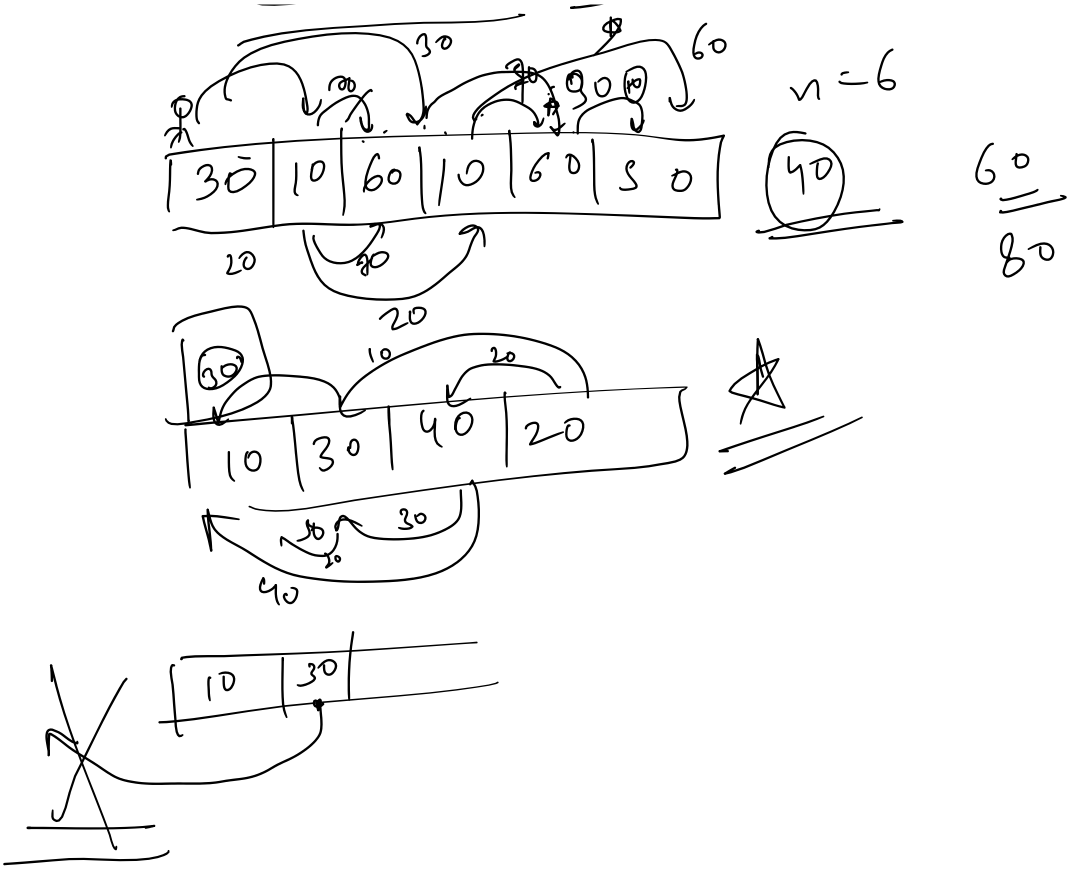
- All values in input are integers.
- $2 \leq N \leq 10^5$
- $1 \leq h_i \leq 10^4$

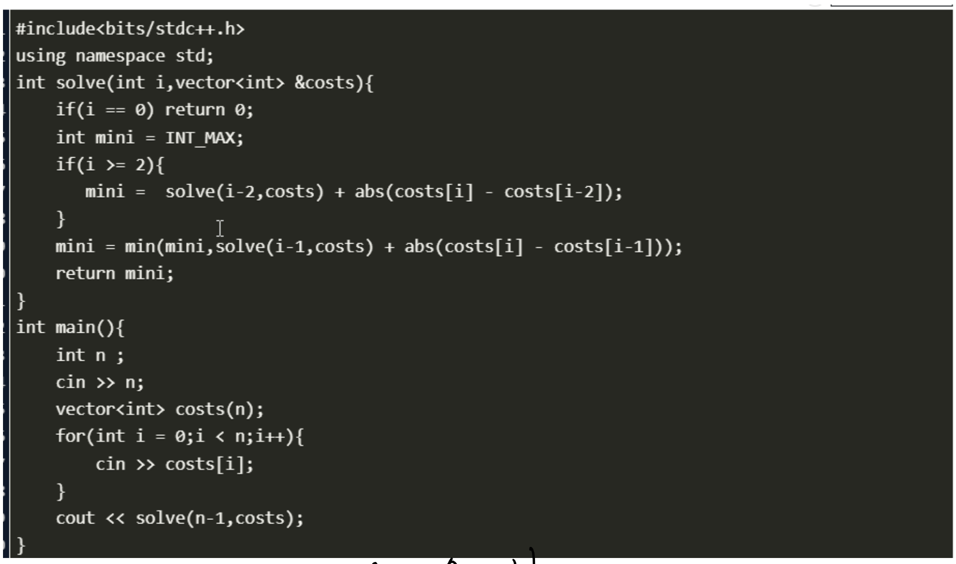
abs(cost[2] - cost[1])) abs(cost[3]- cost[1]))

n = 4



1 → 2 → 4   Ojha 30

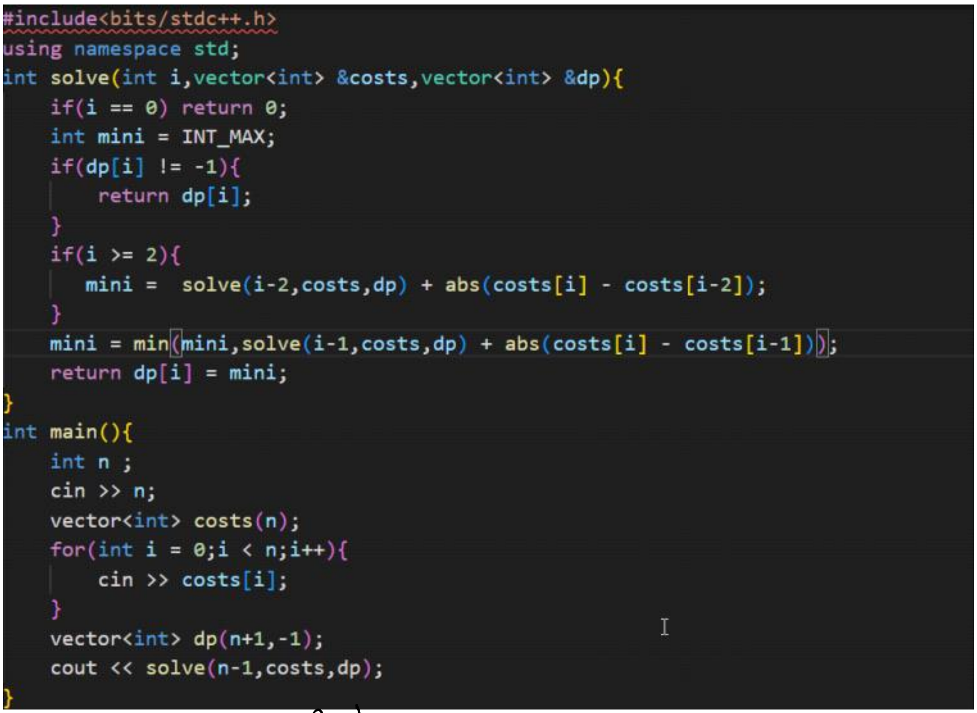




```
#include<bits/stdc++.h>
using namespace std;
int solve(int i,vector<int> &costs){
    if(i == 0) return 0;
    int mini = INT_MAX;
    if(i >= 2){
       mini =  solve(i-2,costs) + abs(costs[i] - costs[i-2]);
    }
    mini = min(mini,solve(i-1,costs) + abs(costs[i] - costs[i-1]));
    return mini;
}
int main(){
    int n ;
    cin >> n;
    vector<int> costs(n);
    for(int i = 0;i < n;i++){
        cin >> costs[i];
    }
    cout << solve(n-1,costs);
}
```

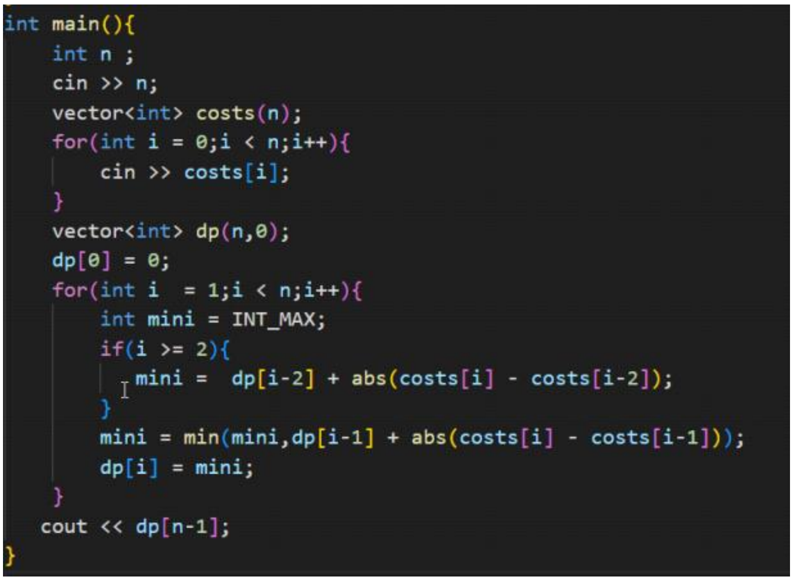
T.C - O(2^N)

S.C - O(N)



```
#include<bits/stdc++.h>
using namespace std;
int solve(int i,vector<int> &costs,vector<int> &dp){
    if(i == 0) return 0;
    int mini = INT_MAX;
    if(dp[i] != -1){
        return dp[i];
    }
    if(i >= 2){
       mini =  solve(i-2,costs,dp) + abs(costs[i] - costs[i-2]);
    }
    mini = min(mini,solve(i-1,costs,dp) + abs(costs[i] - costs[i-1]));
    return dp[i] = mini;
}
int main(){
    int n ;
    cin >> n;
    vector<int> costs(n);
    for(int i = 0;i < n;i++){
        cin >> costs[i];
    }
    vector<int> dp(n+1,-1);
    cout << solve(n-1,costs,dp);
}
```

T.C - O(N)
S.C - O(N)



```
int main(){
    int n ;
    cin >> n;
    vector<int> costs(n);
    for(int i = 0;i < n;i++){
        cin >> costs[i];
    }
    vector<int> dp(n,0);
    dp[0] = 0;
    for(int i  = 1;i < n;i++){
        int mini = INT_MAX;
        if(i >= 2){
           mini =  dp[i-2] + abs(costs[i] - costs[i-2]);
        }
        mini = min(mini,dp[i-1] + abs(costs[i] - costs[i-1]));
        dp[i] = mini;
    }
   cout << dp[n-1];
}
```

T.C - O(N)
S.C - O(1)