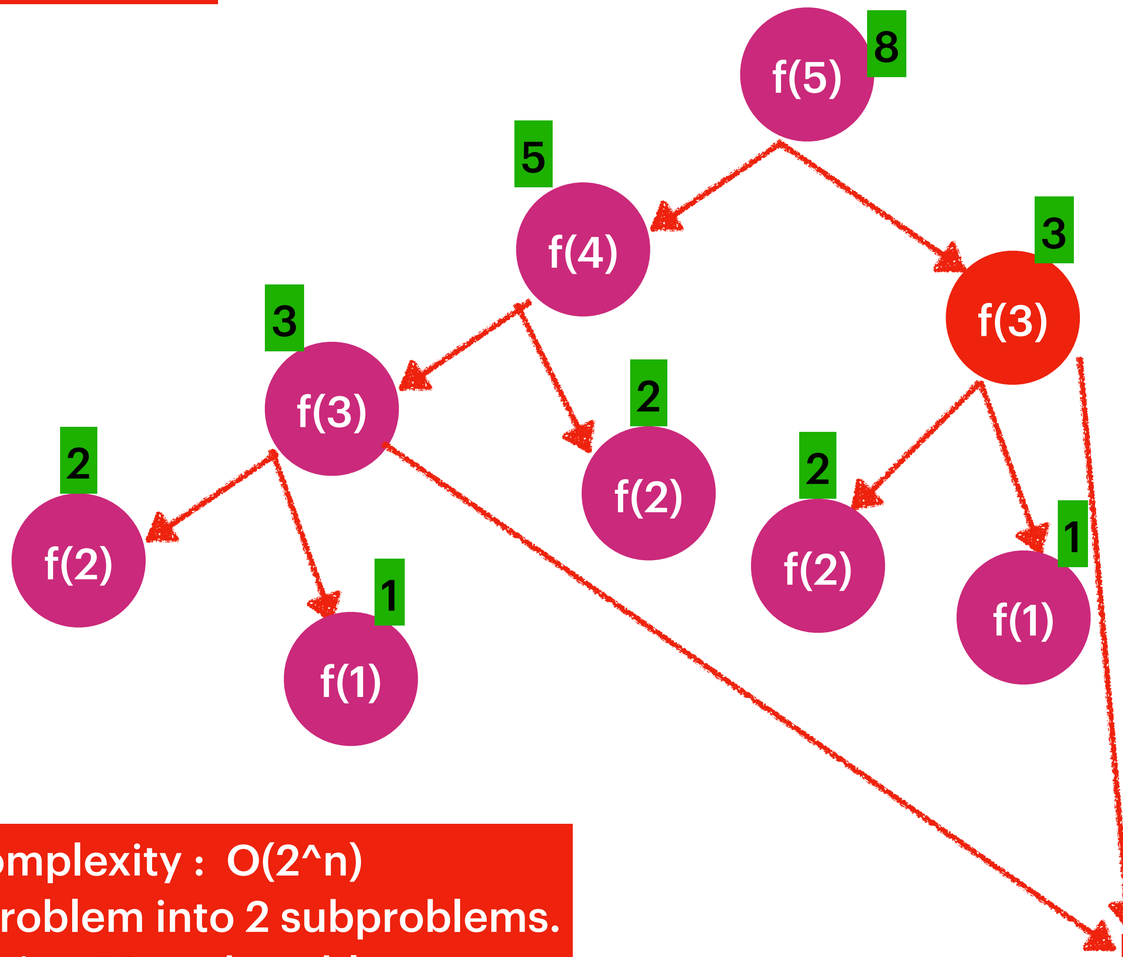
Recursinve Solution

f(n) = f(n-1) + f(n-2);

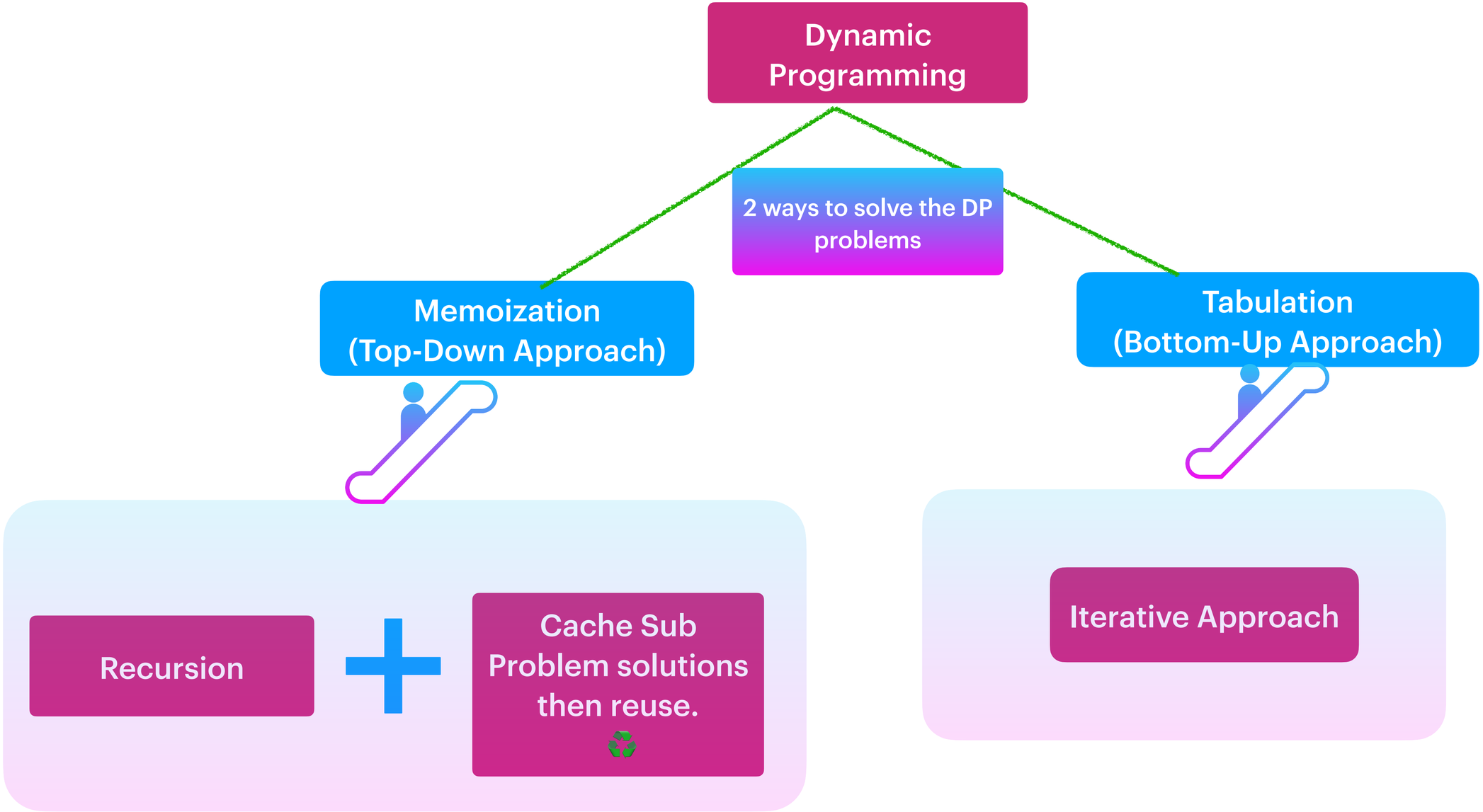


Base Condition —>
if(n <= 2)
{
return n;
}

Time Complexity : O(2^n)
We devide each problem into 2 subproblems.
Leads to solving 2^n subproblems.

Overlapping
Solving Same SubProblem again

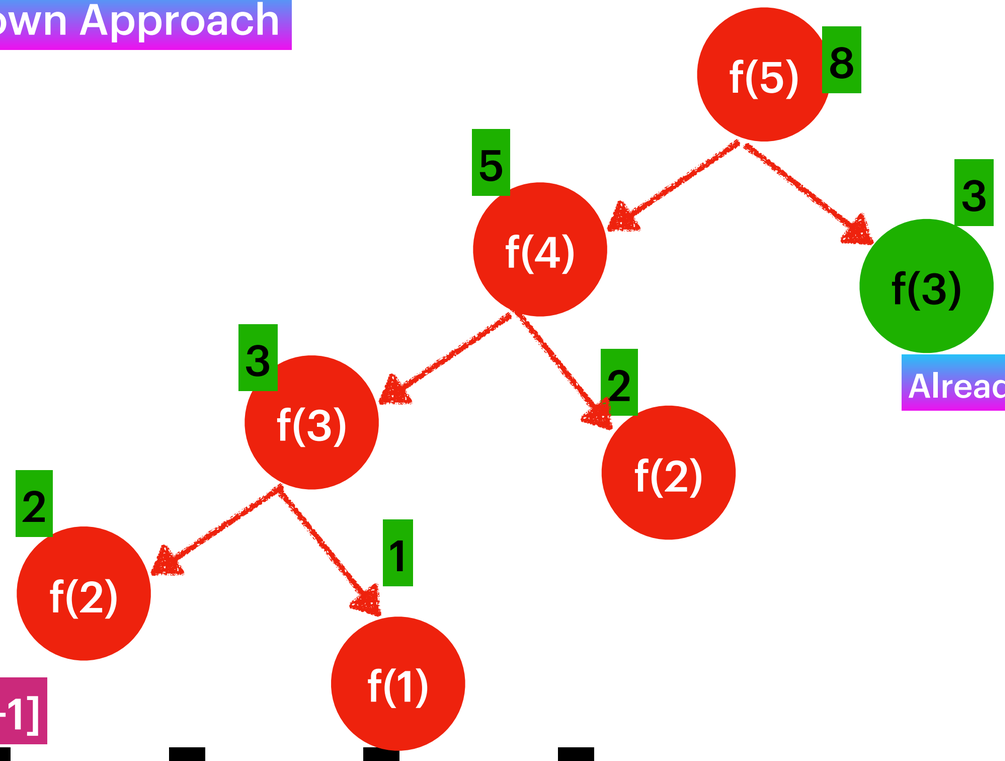
Space Complexity : O(n) :
N StackFrames would be active.


Dynamic Programming
2 ways to solve the DP problems
Memoization (Top-Down Approach)
Tabulation (Bottom-Up Approach)
Recursion
Cache Sub Problem solutions then reuse.
Iterative Approach

Memoization

Top Down Approach

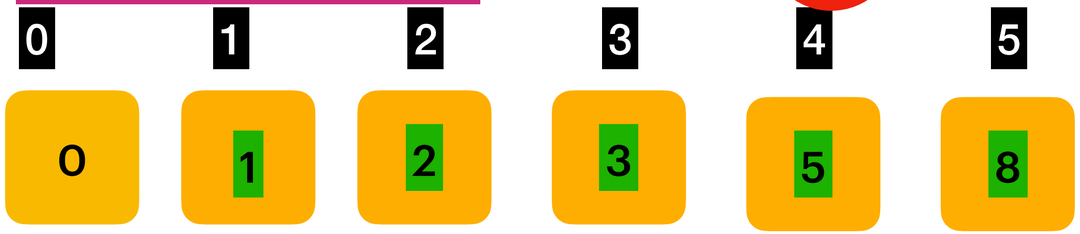
f(n) = f(n-1) + f(n-2);



Base Condition —>
if(n <= 2)
{
return n;
}

Already solved reuse the result

int[] dp = new int[n+1]



Time Complexity : O(n)
At max we solve n subproblems, as we are caching and reusing the Sub problem results.

Space Complexity : O(n) :
N StackFrames would be active.

return dp[n]

746. Min Cost Climbing Stairs

Easy 5736 994 Add to List Share

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.
```

Example 2:

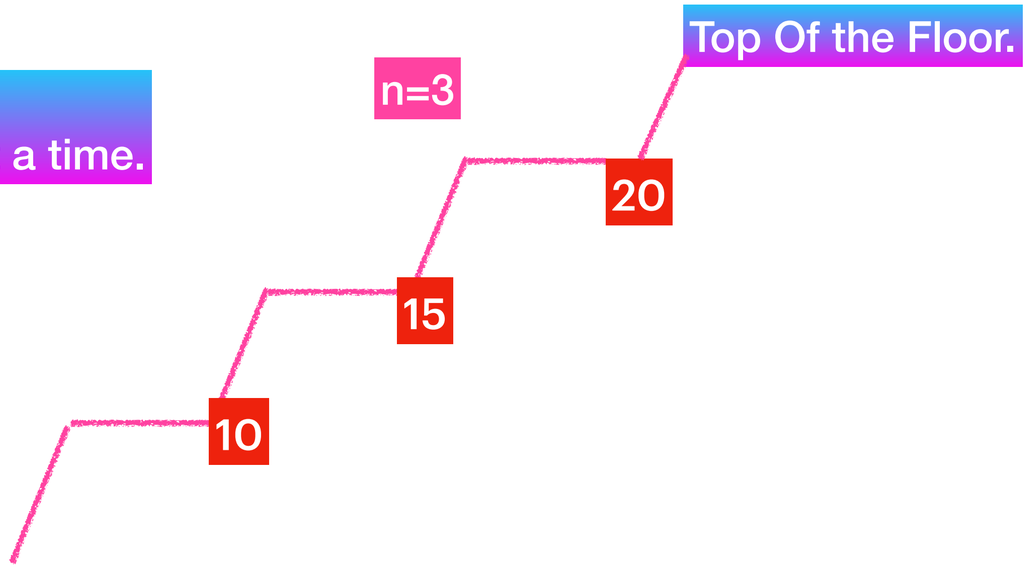
```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```

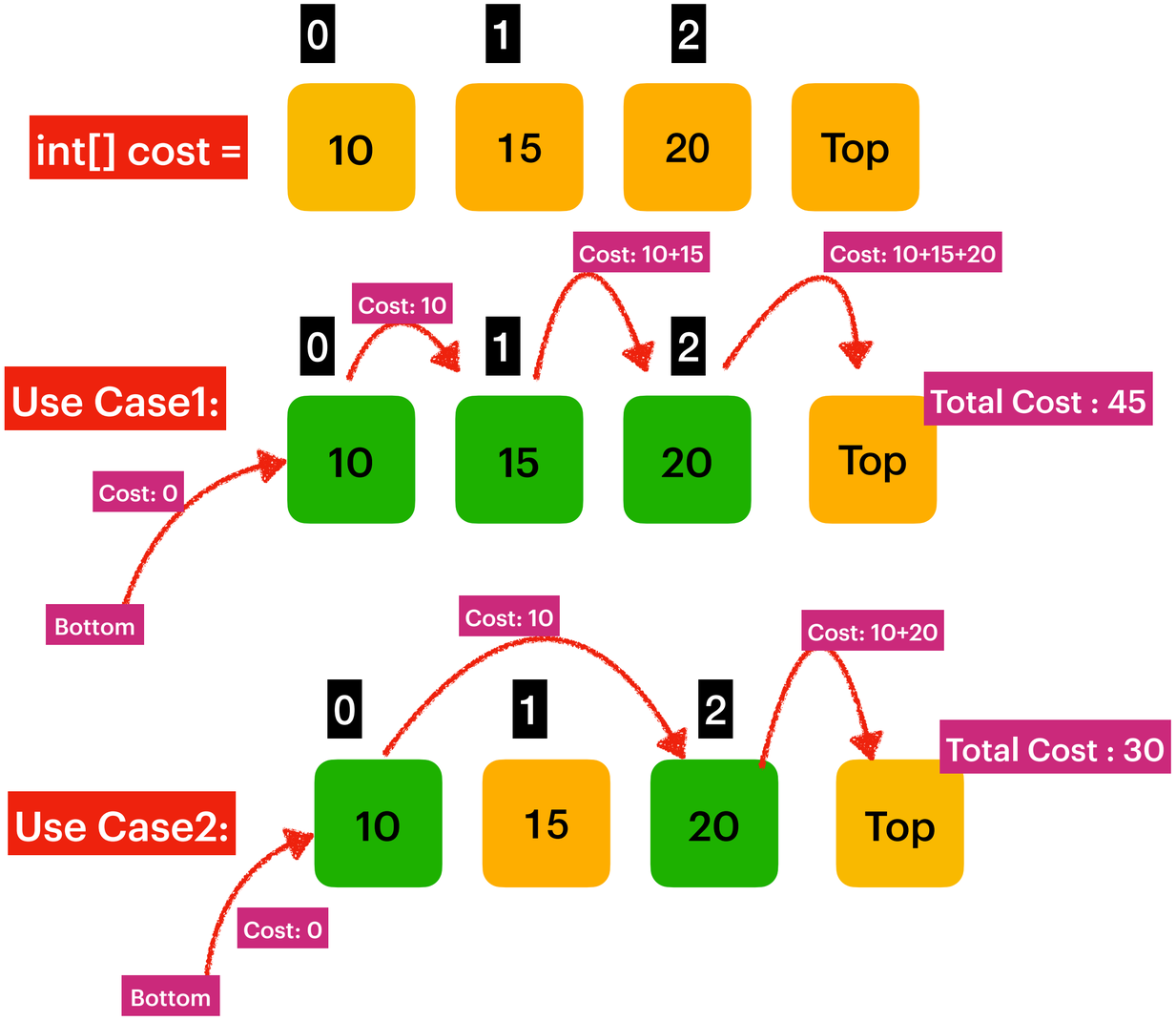
Constraints:

- `2 <= cost.length <= 1000`
- `0 <= cost[i] <= 999`

Find out the minimum cost to reach the top of the floor.

Can take either 1 step or 2 step at a time.

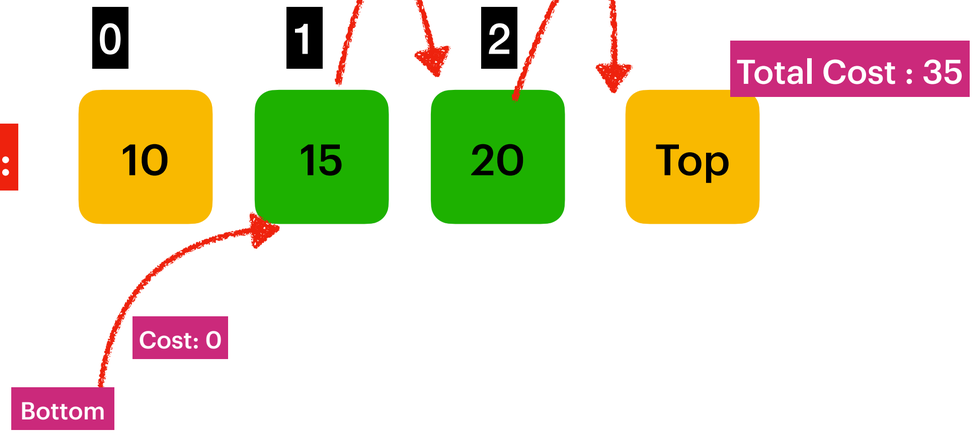


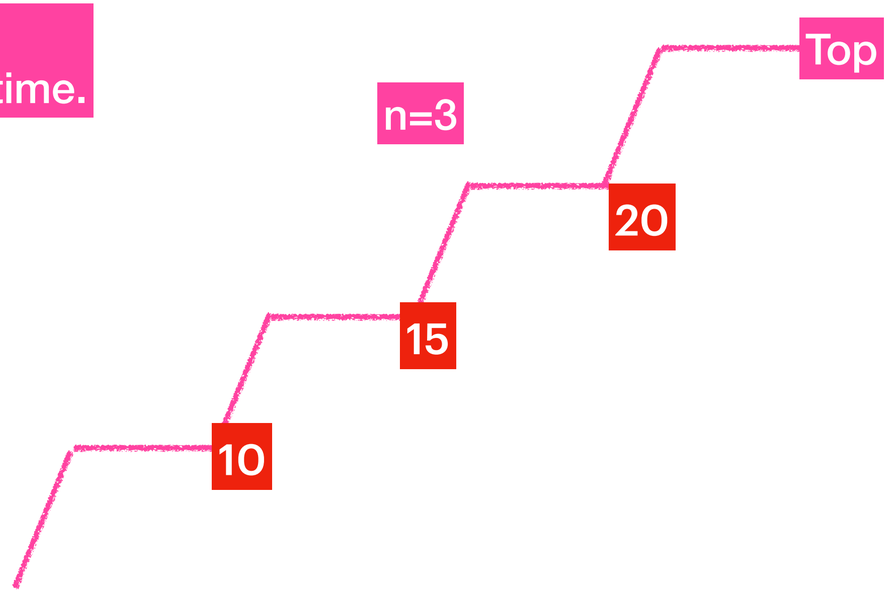


From Bottom either you can climb to step 1 or step2
Then the Cost is 0

Can take
either 1 step or 2 step at a time.

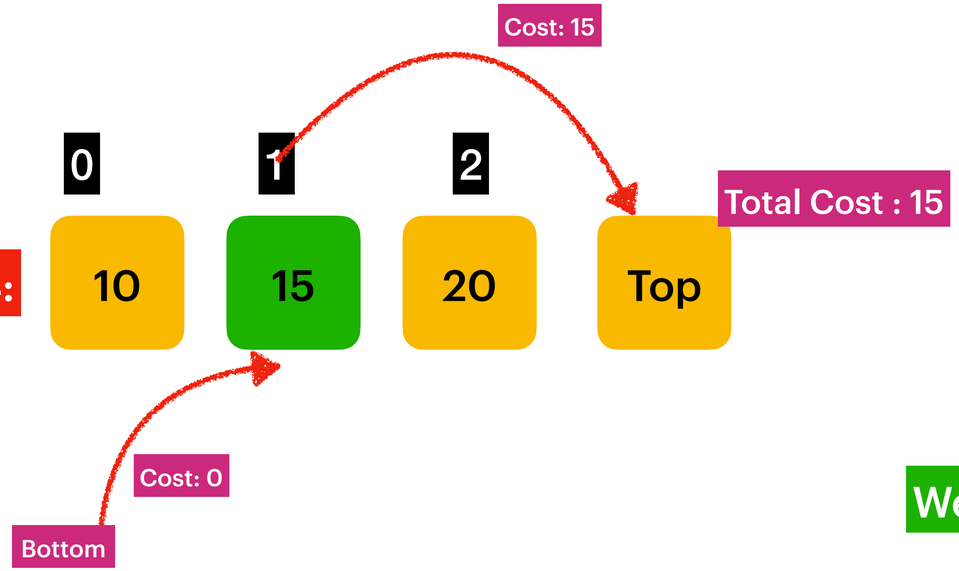
Use Case3:



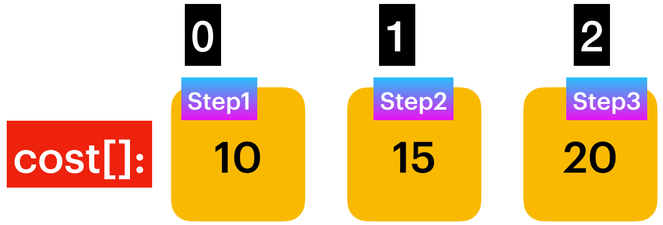


From Bottom either you can climb to step 1 or step2
Then the Cost is 0

Use Case4:



We should return the minimum cost i.e :: 15



State: int[] dp = new int[n+1]

Size should be n+1 as we need to reach the top of the floor
Here the dp[] represents minimum cost to reach each step.

Base Sub Problem Results :

From bottom cost to climb step1 = 0 —> dp[0] = 0

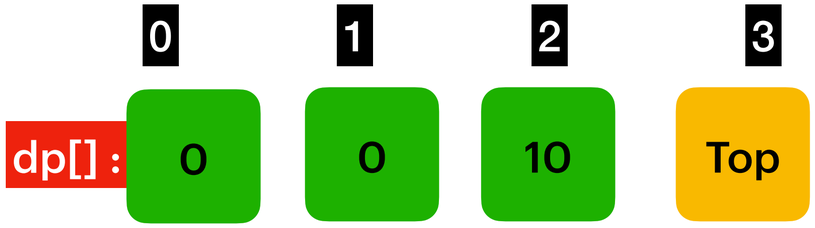
From bottom cost to climb step2 = 0 —> dp[1] = 0

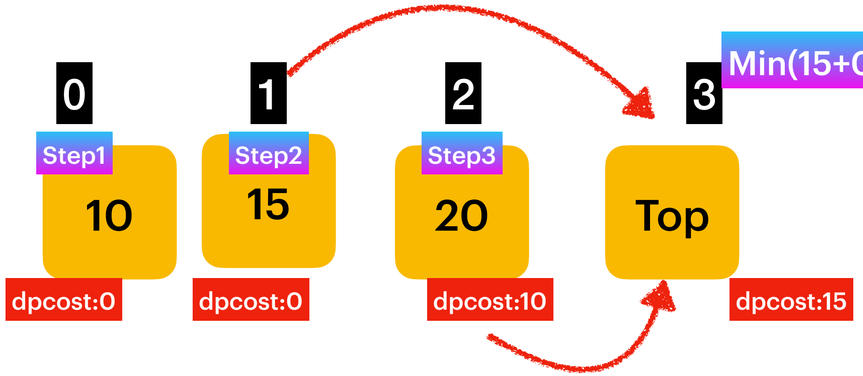
Here the dp[] represents minimum cost to reach each step.

Solved step1 & step2 sub problems.

We should start from index=2 [i.e 3rd step]

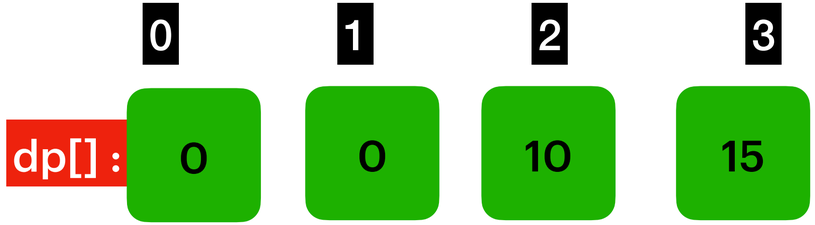
3rd step can be reached either from step1 or step2.
From Step1 we can take 2step climb.
OR
From step2 we can take 1Step climb.
We need to consider minimum cost of these two.





Min(15+0, 20+10) = 15

Now the top can be reached either from step2 or step3.
From Step2 we can take 2step climb.
OR
From step3 we can take 1Step climb.
We need to consider minimum cost of these two.



Recurrence Relation —->

We can reach ith step either from i-1 or i-2 steps.

dp[i] = Math.min(cost[i-1] + dp[i-1] , cost[i -2] + dp[i-2])

Time Complexity : O(n)

Space Complexity : O(n) :

Can we improve on Space, Yes we just need previous two subproblem results to solve current subproblem.
So we can swap between two variables. So that Space would be constant O(1).