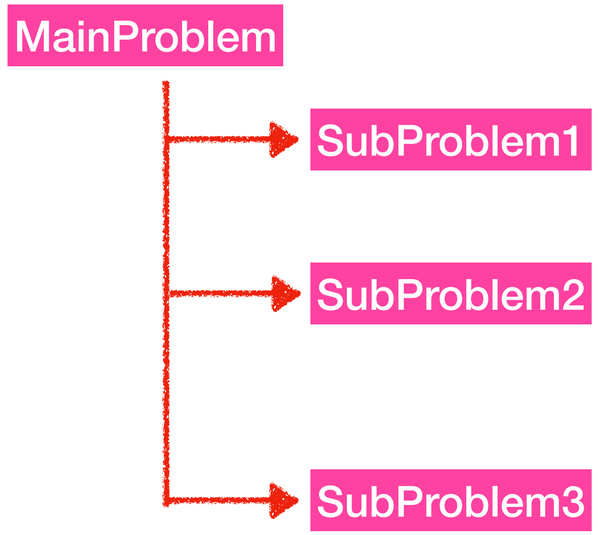
Dynamic Programming [DP]

Divides the main problem into possible subproblems
obtain the optimal solution at each subproblem level.
Reuse the subproblem results to solve the main problem.



Dynamic Programming always
demands Optimal Structure
so that we can achieve optimal solution at
each subproblem.

On solving DP problem we do also focus on
Choosing the choice of elements to achieve
optimal solution.

DP = Choice + Optimal Result

ATM :
ATM Aaways makes sure that
the user receives minimal notes.

Assume Possible Notes in ATM [100,200,500,2000]

Trying to WithDraw : 700 ->
No.Of notes given by ATM is 2 [500(1) + 200(1)]

Optimal Structure is Minimal note count.

Can also have other
below possibilities but which are not optimistic.
100(7) —-> (X)
200(3) + 100(1) —->(X)
100(3) + 200 (2) —> (X)

DP =
Choice [notes] + Optimal [Should return minimal notes]

Google Maps

Source —> Destination

DP = choice [Routes] + Optimistic [ShortestPath]

Amazon Delivery Guy

He should not revist the location twice.

DP =
locations [choice] +
Optimal [should not revisit Twice]

Can be solved in 2 ways

Dynamic Programming [DP]

Tabulation [Bottom Up Approch]

Its a iterative solution

Memoization [Top Down Approch]

Its a Recursive Solution with caching

Each DP problems demands following three steps.

State

We would need to store the sub problem results so that which can be reused to solve main problem.
We generally take a data structure to store subproblem results.
Ex: array, two dimensional array, HashMap etc…

Base SubProblem Results

Each DP solution needs seeding of optimal solution for the base subproblems.
These base subproblem optimal results we can obtain from problem statement.

How many subproblems we would need to feed for DP solution is depends on Recurrence Relation.

Recurrence Relation

Recurrence relation can be the main principle in DP solution.
This is to be derived by understanding the algorithm and the problem statement.
Problem statement should have the optimal structure to derive Recurrence Relation.

70. Climbing Stairs

Easy 11345 353 Add to List Share

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:

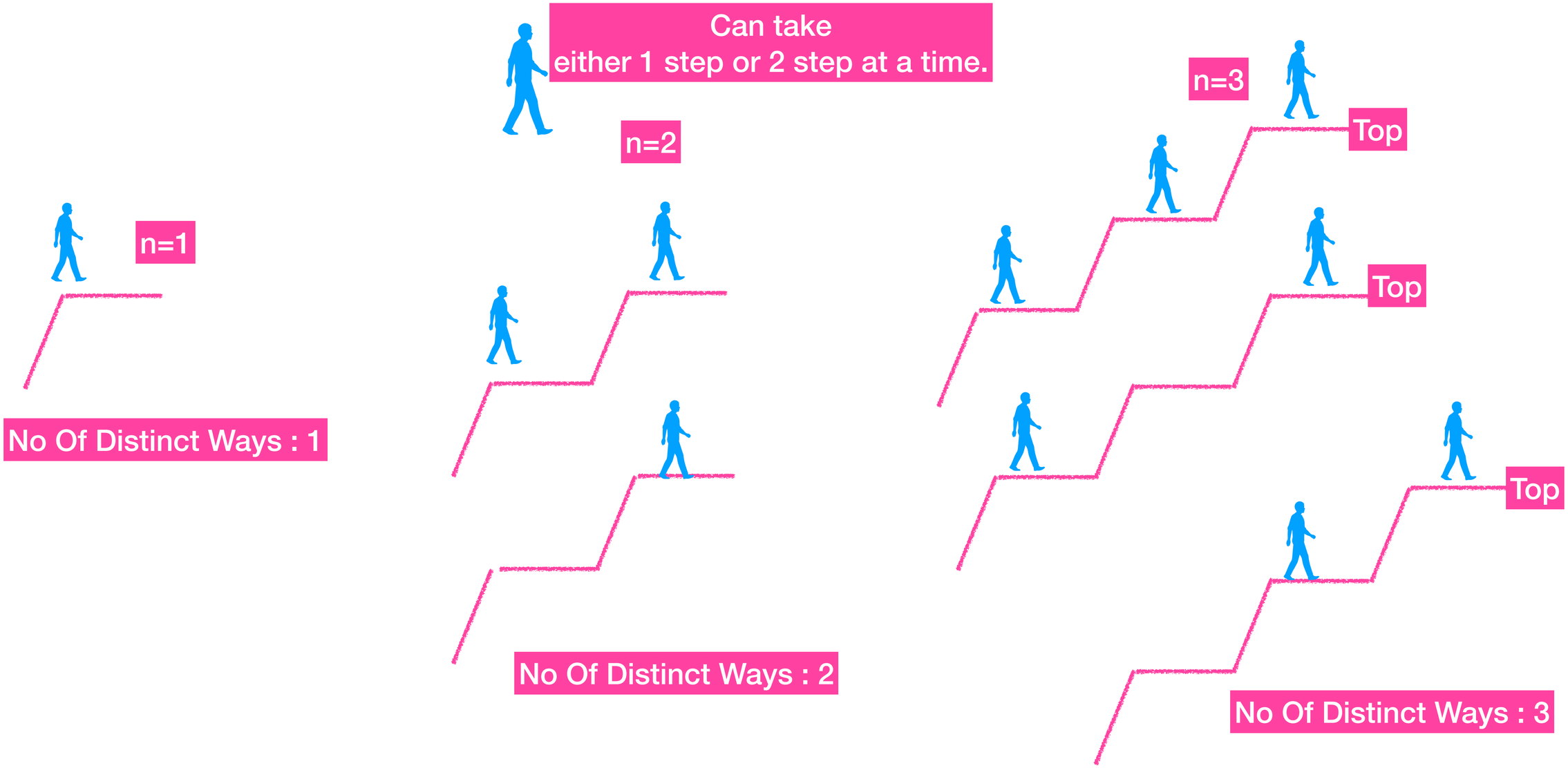
```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

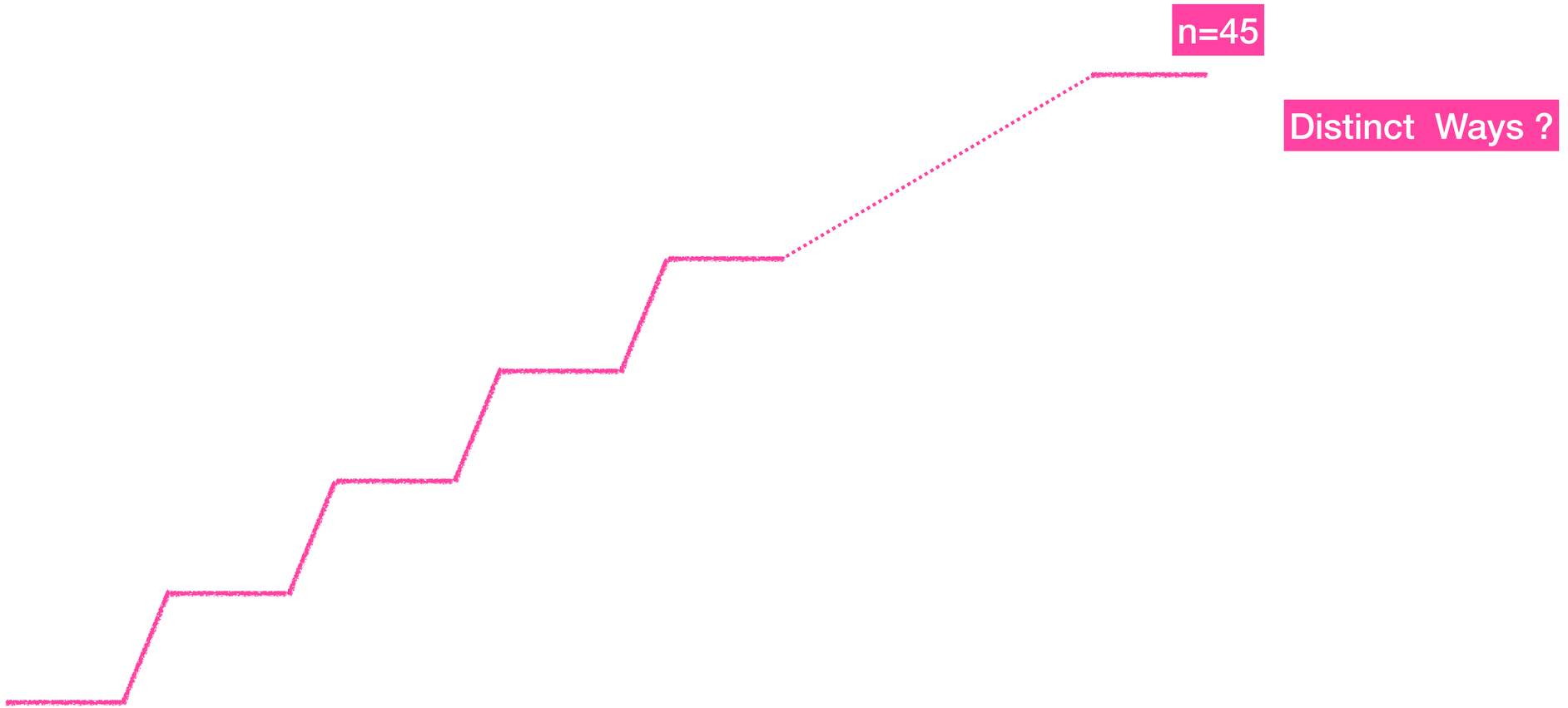
Example 2:

```
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
```

Constraints:

- `1 <= n <= 45`

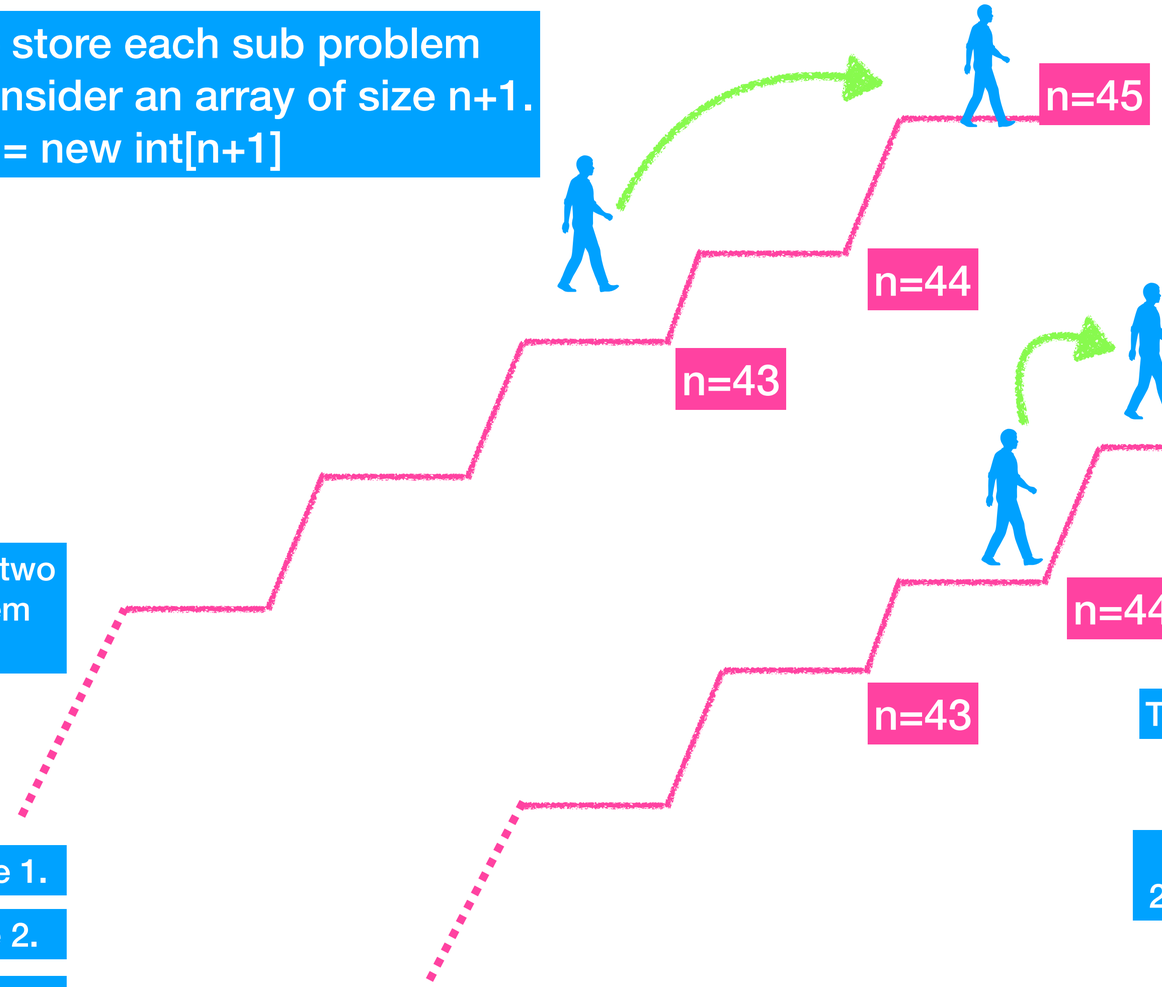

Can take
either 1 step or 2 step at a time.
n=1
No Of Distinct Ways : 1
n=2
No Of Distinct Ways : 2
n=3
Top
Top
Top
No Of Distinct Ways : 3


n=45
Distinct Ways ?

State :

We would need to store each sub problem result so that let's consider an array of size n+1.
int[ ] dp = new int[n+1]

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



Base SubProblem Results

As the Recurrence relation demands two sub problem results, let's derive them from problem statement.

n = 1 then the distinct ways would be 1.

n=2 then the distinct ways would be 2.

So that base subproblem results would be
dp[1] = 1
dp[2] = 2

There are two possibilities to reach 45th step.

We can take
2 Step from 43rd step

We can take
1 Step from 44the step

Recurrence Relation

dp[i] = dp[i-1] + dp[i-2]

Distinct Ways to reach 45th Step =
Distinct Ways to reach 43rd step + Distinct Ways to reach 44th step

As per recurrence relation
dp[3] = dp[2] + dp[1] = 2+1 =3
dp[4] = dp[3] + dp[2] = 3+2 = 5 etc..

N = 4 (Main Problem)

Bottom UP Approach

Solving the Solution started from
n=1 to n=4
Bottom to UP.