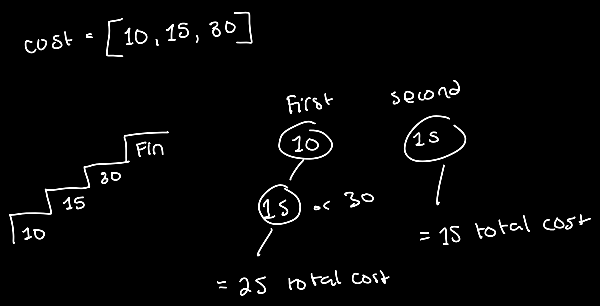
<u>Problem:</u> For staircase, the i-th step is assigned a non-negative cost indiccated by a cost array.

When pay cost for step -> climb ONE or TWO steps
Find minimum cost to reach top of stairase

first step can be FIRST or SECOND step



How to recognize dp problem?

optimization -> min
-> max

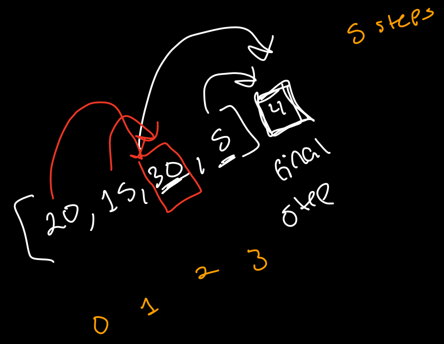
① Based in Recursion -> base case

Generate all paths to pick minimum cost

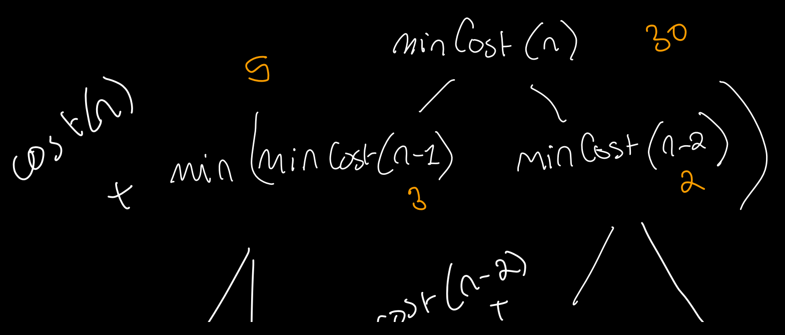
② Recurrence relation

formula for basis of recursive sol'n

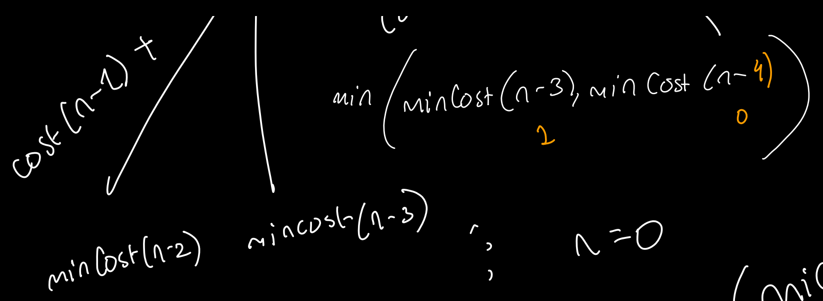
func ( ) :
# base case
func ( )



n=4



Top Down



recurrence relation

$$\text{minCost}(i) = \text{cost}[i] + \min(\text{mincost}(i-1), \text{mincost}(i-2))$$

$\rightarrow$ base case
$$\begin{cases} i < 0 : \text{return } 0 \\ i == 0 : \text{return cost}[0] \\ i == 1 : \text{return cost}[1] \end{cases}$$

```
const   minCostClimbingStairs = function (cost) {
            const  n= cost.length;
            return Math.min (min Cost (n-1, cost),
                              min Cost (n-2, cost)):
}

const  minCost = (i, cost)= {
              if  i < 0:  return 0
              if  i== 0 or i== 1 : return cost[i]

                         (min Cost (i-1, cost),
```

return $\underbrace{cost[i] + min(min\ldots \quad minCost(i-2, cost))}_{\text{recurrence relation}}$

Time: 2^n

For every call, call ② recursive functions

Space: n

call stack only contains of single branch down to bottom of binary tree AT WORST

– stack contains max height of tree

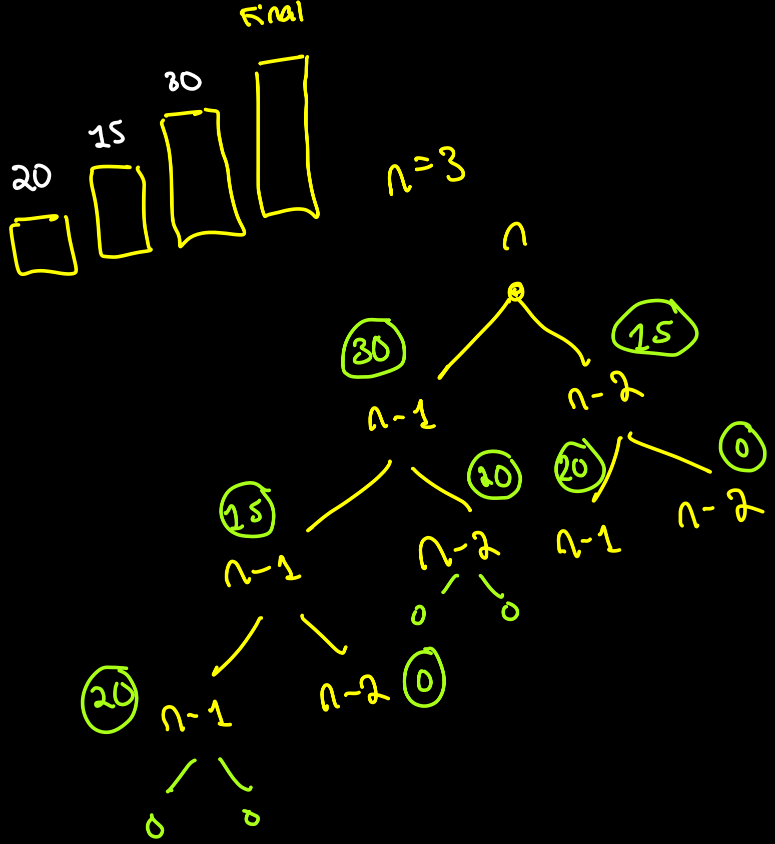
$$\frac{n}{}$$

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

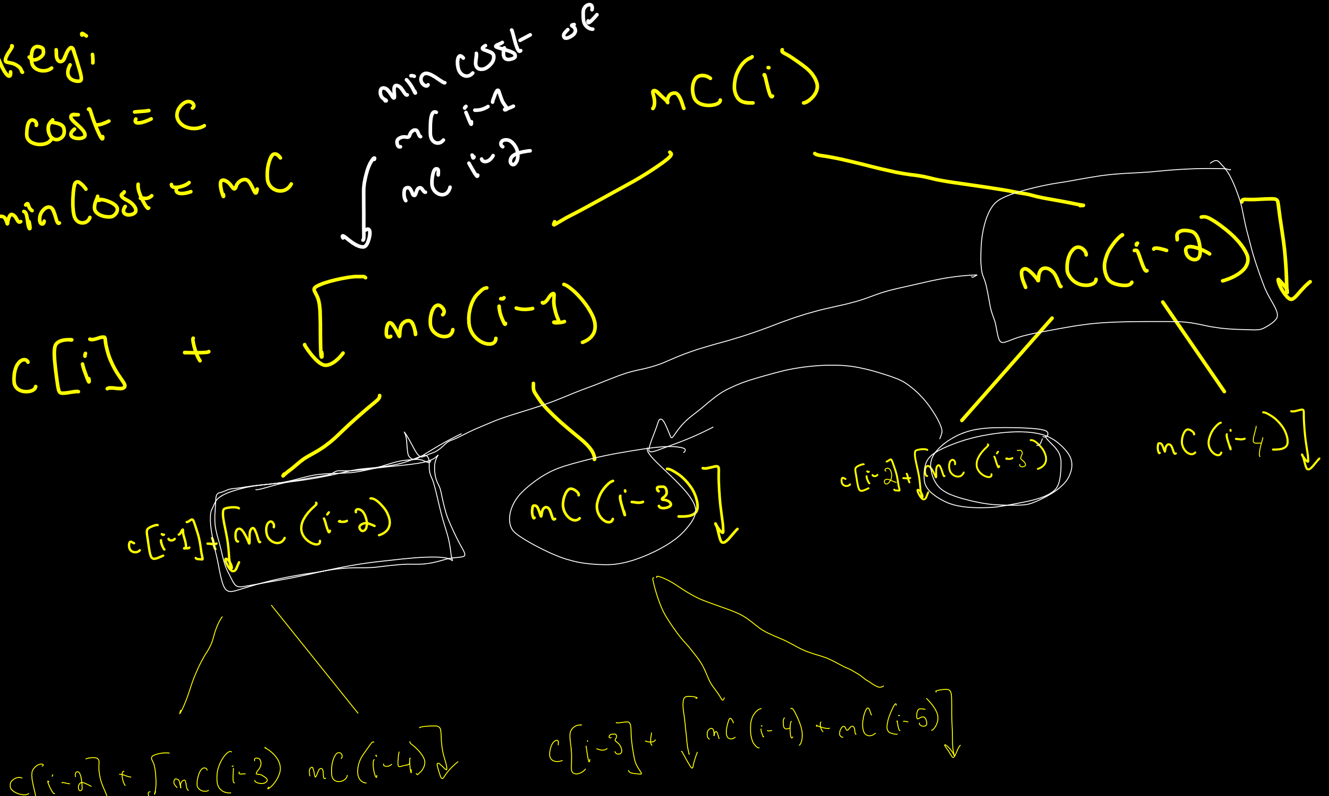


double the calls for

every value of n

1. define recurrence relation
2. create (brute force) recursive function
3. memoize — state based tree

Key:
cost = C
min Cost = mC



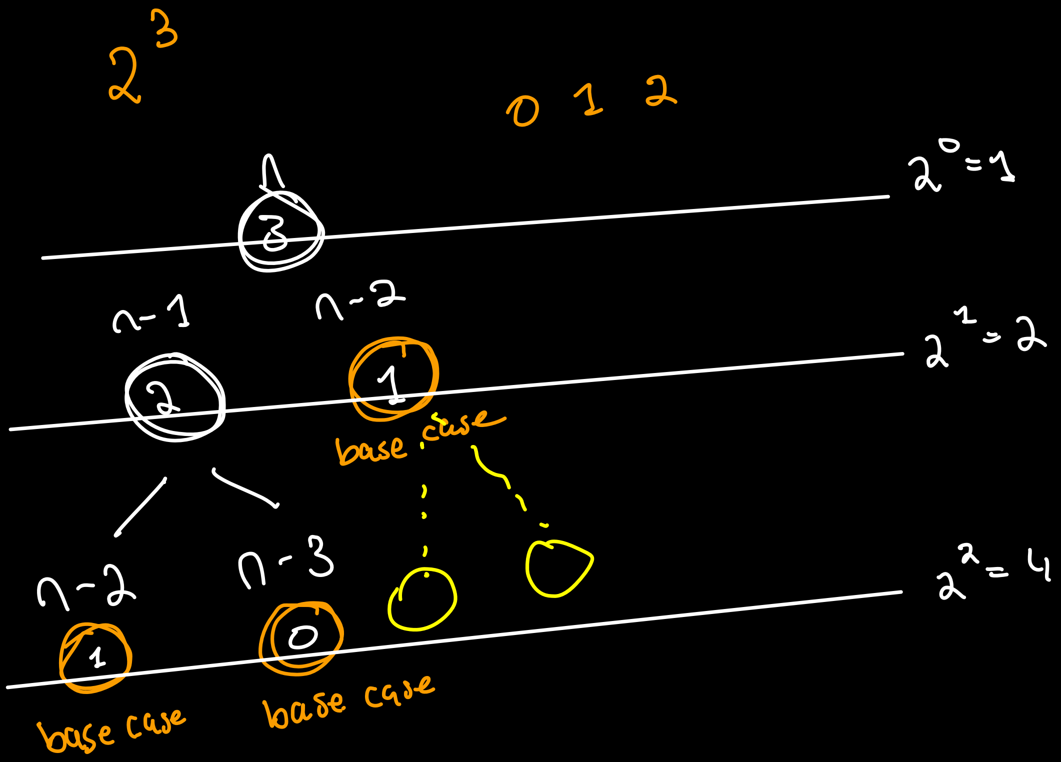
☆ Note: Data structure used to memoize == Data structure iterated thru

(in this case: array)

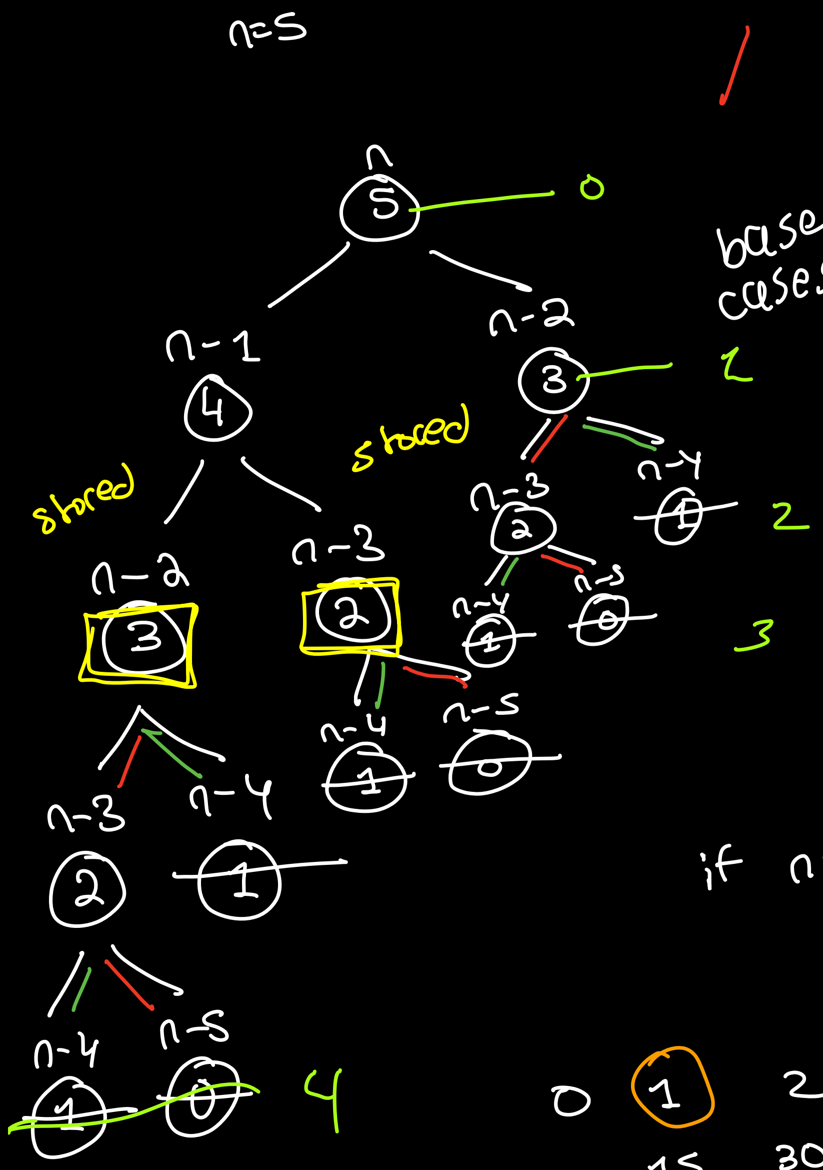
? which side will hit the base case first?

→ the right most branch

memoize the right most branch, then if traveversing thru - if the value hasn't been seen before - include it in the array!



the last 2 levels of the tree will contain BASE CASES



array

base cases { 0 : cost [0] = 20
1 : cost [1] = 15

2 : 45

3 : 55

4 : 110

i

if n== i+1 :

cost[1] + cost[3] = 55

0	1	2	3	4
20	15	30	5	10

0 → 20

1 → 15

0 1 1→2

2 → min(20 , 15) = 15

15+ cost[2] = 45

1 , 2

3 → min(15, 45) = 15 1→3

15+ cost [3] = 20

2 , 3

4 → min (45 , 20) = 20   1 → 2 → 4

20 + cost[4] = 30

5 → min (20 , 30) = $\boxed{30+0}$ (the 20 and 30 are marked 3 , 4)

initialize array size n w -1 values

```
let array = -1 · n
array[0] = cost[0]
array[1] = cost[1]

const minCostClimbingStairs = function (cost) {

    const n = cost.length;
    return Math.min(minCost (n-1, cost),
                    minCost (n-2, cost));
}

const minCost = (i, cost, array) => {
        if i < 0:  return 0
        if i == 0 or i == 1 : return cost[i]

    right = (array[i-1] == -1) ? minCost (i-1, cost)
                                : array[i-1]

    left = (array[i-2] == -1) ? minCost (i-2, cost)
                               : array[i-2]

    array[i] = cost[i] + min(left, right)

    return array[i]
```

}   )

leetcode

Bottom-Up (Iterative)

minCost (0) = cost[0]

minCost (1) = cost[1]

$$2: \quad \min\left(\overset{i-2}{mC(0)}, \overset{i-1}{mC(1)}\right) + cost[2]$$

$$3: \quad \min\left(\overset{i-2}{mC(1)}, \overset{i-1}{mC(2)}\right) + cost[3]$$

$$4: \quad \min\left(\overset{i-2}{mC(2)}, \overset{i-1}{mC(3)}\right) + cost[4]$$

Intro:

- Verify Constraints
- Create Testcases

Brute Force:

- Brainstorming & Pattern Observations
- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity

Optimal:

- Brainstorming & Pattern Observations
- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity

1 → recurrence relation

|

used to
build recursive func

2 → recursive func

Top
↓
down

determine if
optimization using
memoization

convert

3 Top ↓ ⇒ ↑ Bottom

4 Derive iterative
soln for ↑ Bottom

check for wasted space

S minimize
space complexity