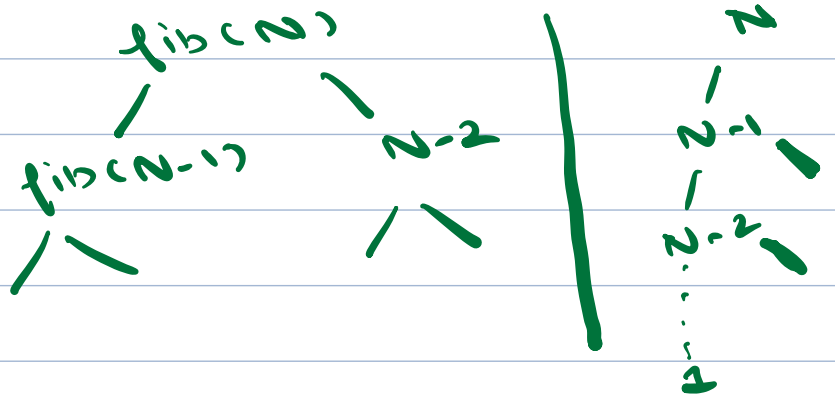
Agenda

1. House Robber } 1D DP
2. Count Unique Paths } 2D DP
3. N digit numbers
4. Catalan Numbers
5. Unique BSTs

DP : store already solved subproblems

Fibonacci : Recursion to DP



TC : $O(2^N)$ ⟶ $O(N)$

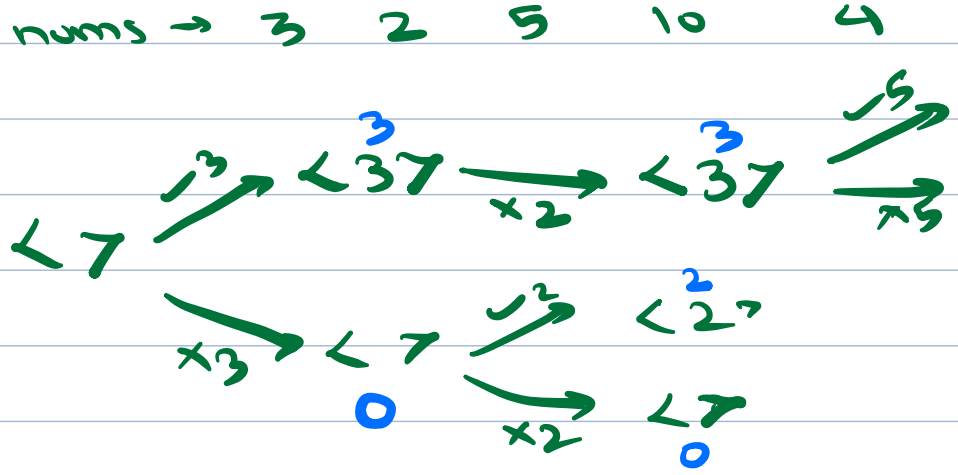
Top Down	Bottom up
1. Recursive	Iterative
2. Memoization	Tabulation
3. Largest → smallest	Smallest → Largest

1. Given an array nums, where each element represents the amt of money in a house, determine max amt of money you can rob without triggering an alarm. The constraint is you cannot rob adjacent houses.

Ex: [1 2 3 1]
ans: 4

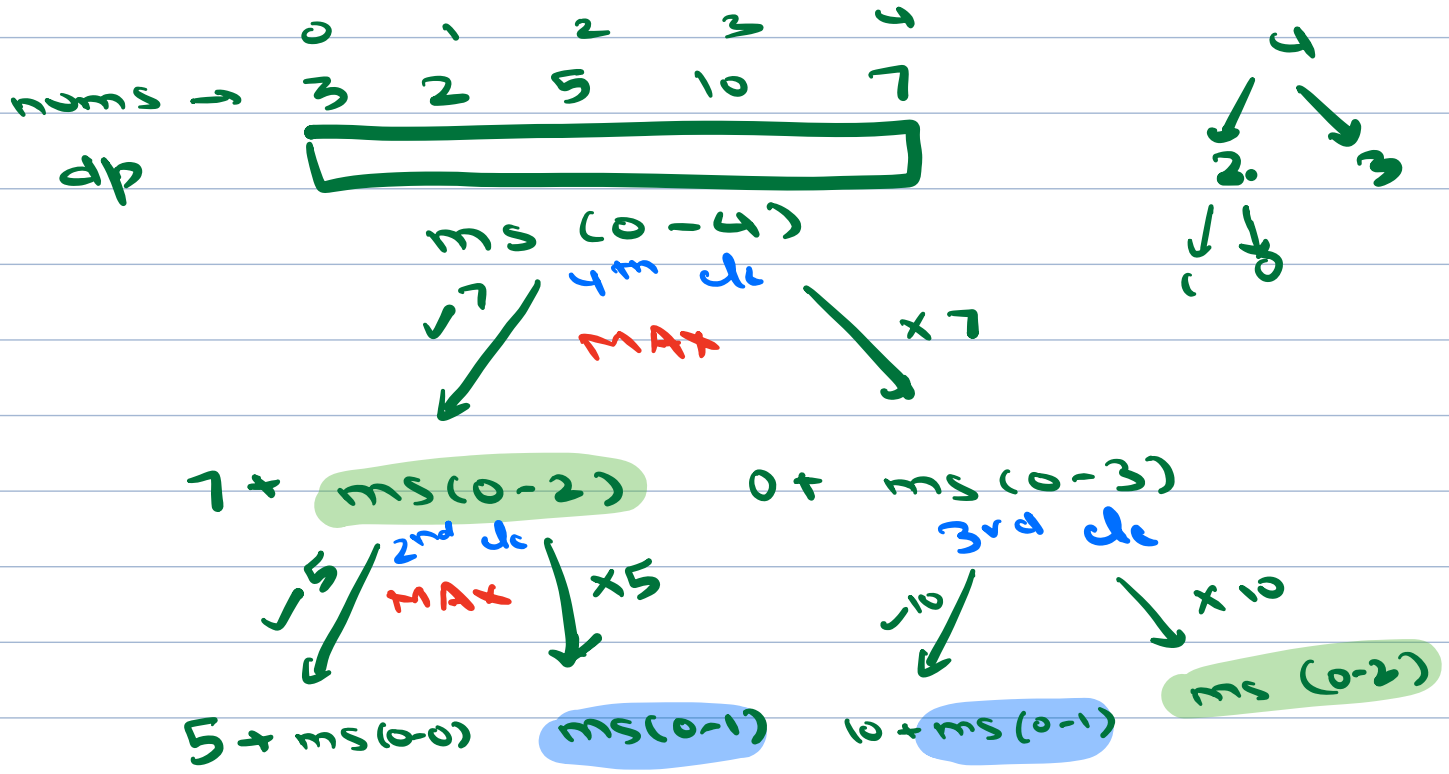
Ex. [15 8 7 20]
ans: 35

Brute Force: Recursively explore all subsets of houses by either robbing / skipping each house, calculate max amt you can rob.



TC: $O(2^N)$
SC: $O(N)$ Recursive stack space

DP { Subset
No. of ways —
Min / Max



How many inputs you need to uniquely define a problem ?

ms (e) → max subset sum from 0 → e

```
int dp[N] = {-1}

int maxSubsetSum (int nums[], int e) {
    if (e == 0)    return nums[0]
    if (e == 1)    return max (nums[0], nums[1])
    if (dp[e] != -1)    return dp[e]
    int include = nums[e] + maxSubsetSum (nums, e-2)
    int exclude = 0 + maxSubsetSum (nums, e-1)
    dp[e] = max (include, exclude)
    return dp[e]
}
```

TC: O(N)

SC: O(N + N) = O(N)

Stack DP

ms(i) = MAx(nums[i] + ms(i-2), ms(i-1))

0 -i

Botrom up Approach

ms(4)

ms(2) ms(3)

// nums

int dp[N] = <-1>

dp[0] = nums[0]

dp[1] = max(nums[0], nums[1])

for (e=2 ; e < N ; e++) <

dp[e] = max (nums[e] + dp[e-2], dp[e-1])

max subsct

sum from 0 → e

return dp[N-1]

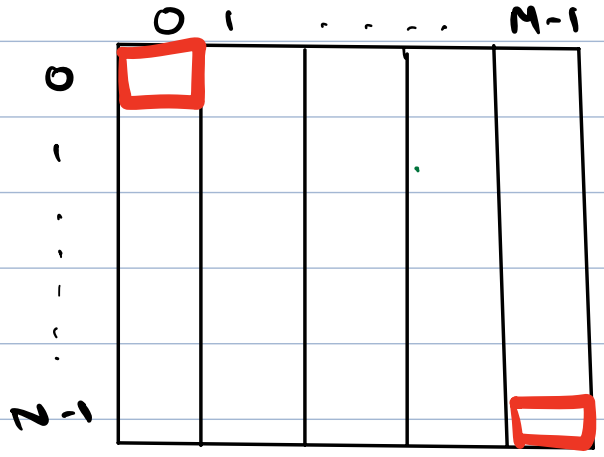
TC: O(N)

SC: O(N)

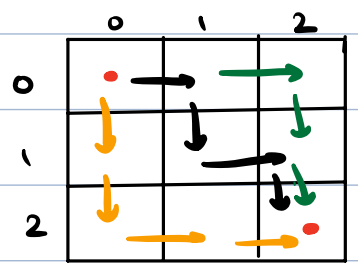
SC: can be optimised to O(1) as dp[i] only depends on dp[i-1], dp[i-2]. Main 2 prev variables (a → dp[i-1], b → dp[i-2])

Given mat[N][M], find total no. of ways from (0,0) to (N-1, M-1). We can move 1 step down or right.

rows cols

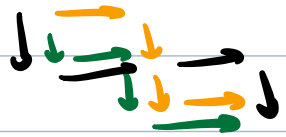


mat[3][3]

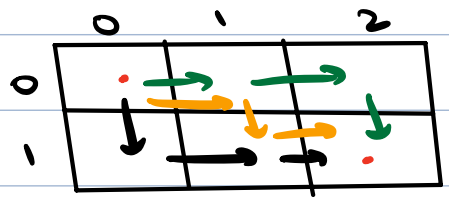


ans = 6

RRDD
RDDR
RDRD
DDRR
DRDR
DRRD



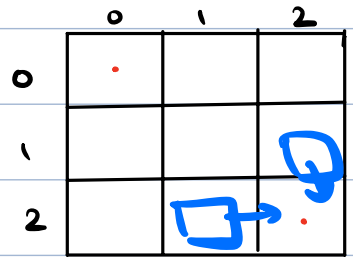
mat[2][3]



ans = 3

RRD
DRR
RDR

mat[3][3]



ways (2,2)

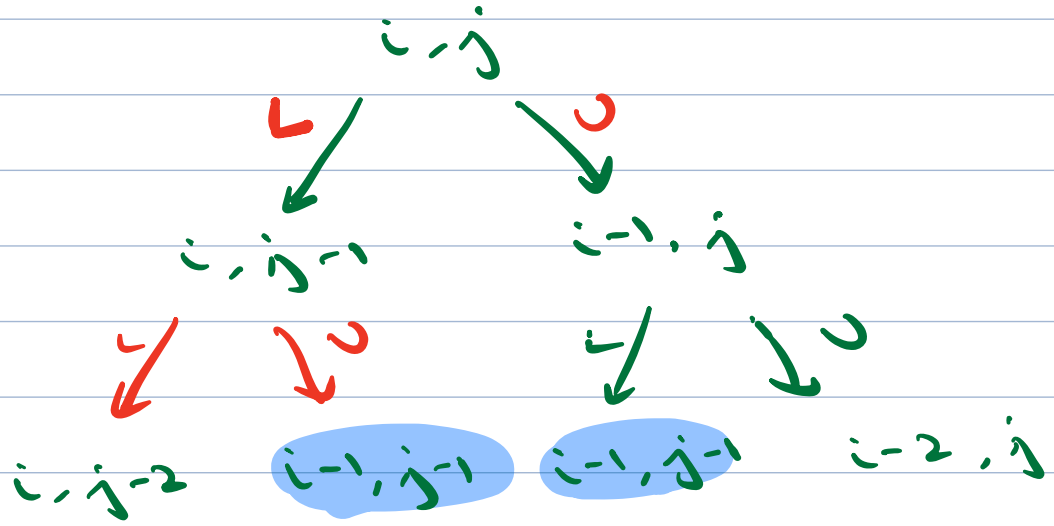
R → 2,1 → DDR, RDD, DRD

OR

D → 1,2 → RRD, DRR, RDR

ways(2,2) = ways(2,1) + ways(1,2)

ways(i,j) = ways(i,j-1) + ways(i-1,j)



// dp[i][j] → No. of ways to reach (i,j) from (0,0)

```
int dp[N][M] = <-1>

int ways(i,j)<
    if (i==0 || j==0)
        return 1

    if (dp[i][j] != -1) return dp[i][j]
    dp[i][j] = ways(i-1,j) + ways(i,j-1)
    return dp[i][j]
>

main()<
    return ways(N-1, M-1)
>
```

TC: O(N×M)

SC: O(NM) + N+M) = O(NM)

DP Stack

ways(0,0) = No. of ways to reach (0,0) from (0,0)
= 1 way (do nothing)

N-1 Down steps + M-1 Right steps

Bottom Up Approach

L → R
T → B

```
int dp[N][M] = {-1}

for (r = 0 ; r < N ; r++) {
    for (c = 0 ; c < M ; c++) {
        if (r == 0 || c == 0)
            dp[r][c] = 1
        else {
            dp[r][c] = dp[r][c-1] +
                       dp[r-1][c]
        }
    }
}
return dp[N-1][M-1]
```

mat[3][3]

	0	1	2
0	1	1	1
1	1	2	3
2	1	3	6

TC: O(NM)
SC: O(NM)

SC : O(M)

2 rows at a time (prev and cur)

```
int prev[M] = {0}, cur[M] = {0};
for (r = 0; r < N; r++) {
    for (c = 0; c < M; c++) {
        if (r == 0 || c == 0)
            cur[c] = 1;
        else
            cur[c] = cur[c-1] + prev[c];
    }
    prev = cur;
}
return cur[M-1];
```

TC : O(NM)

SC : O(M)

10:40

Find how many A digit nos. exist such that their digit sum is B. A valid no. doesn't contain leading zeroes. Returns ans % $10^9 + 7$

A = 2, B = 4 ans = 4

sum = 4

2 2

3 1

1 3

4 0

A = 1 B = 3

ans = 1

3

(Inefficient)

BF : Generate all possible A digit nos. Calculate sum of every num and compare with B

Max A digit no.

A = 2 → 99 ≈ $10^2 - 1$

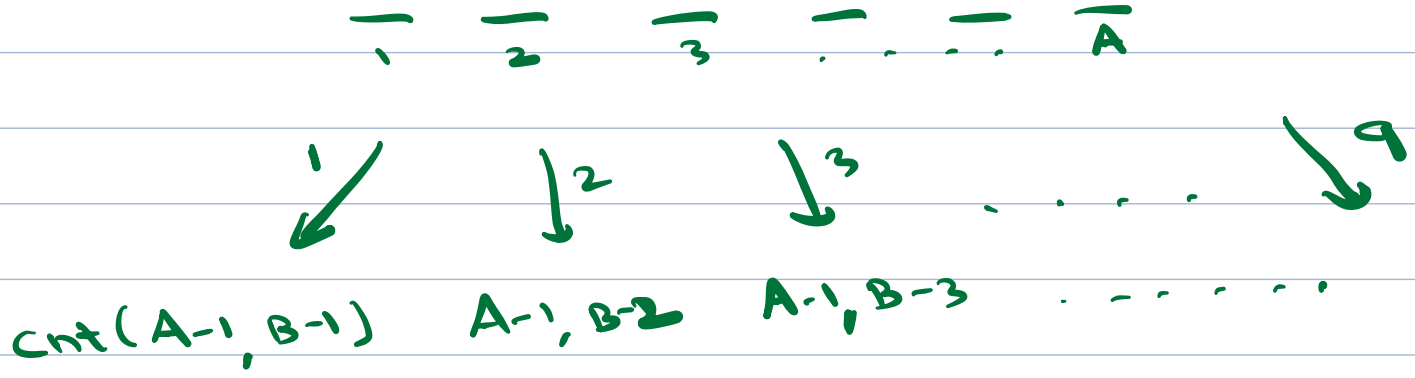
A = 3 → 999 ≈ $10^3 - 1$

A → $10^A - 1$

TC : $O(10^A)$

digits sum

cnt(A, B)



cnt(4, 10)

— — — 1

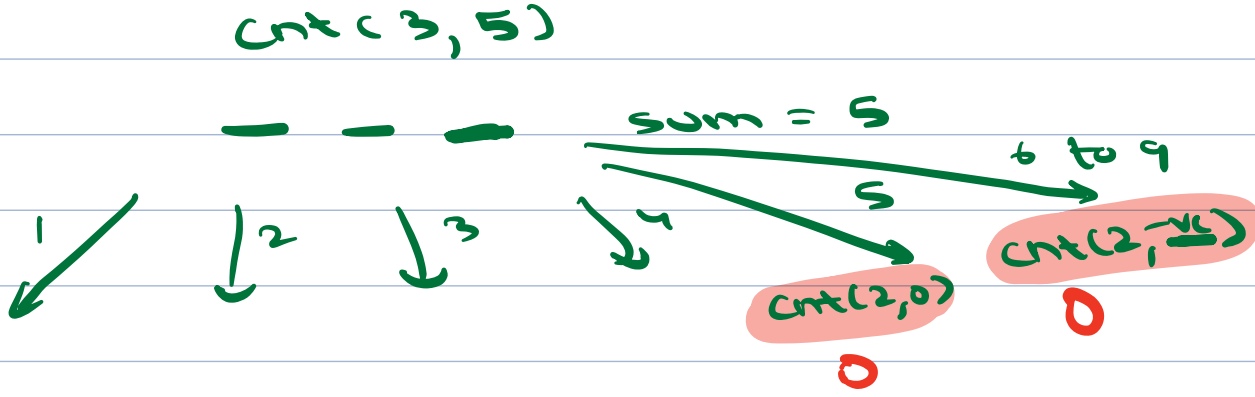
cnt(3, 9)

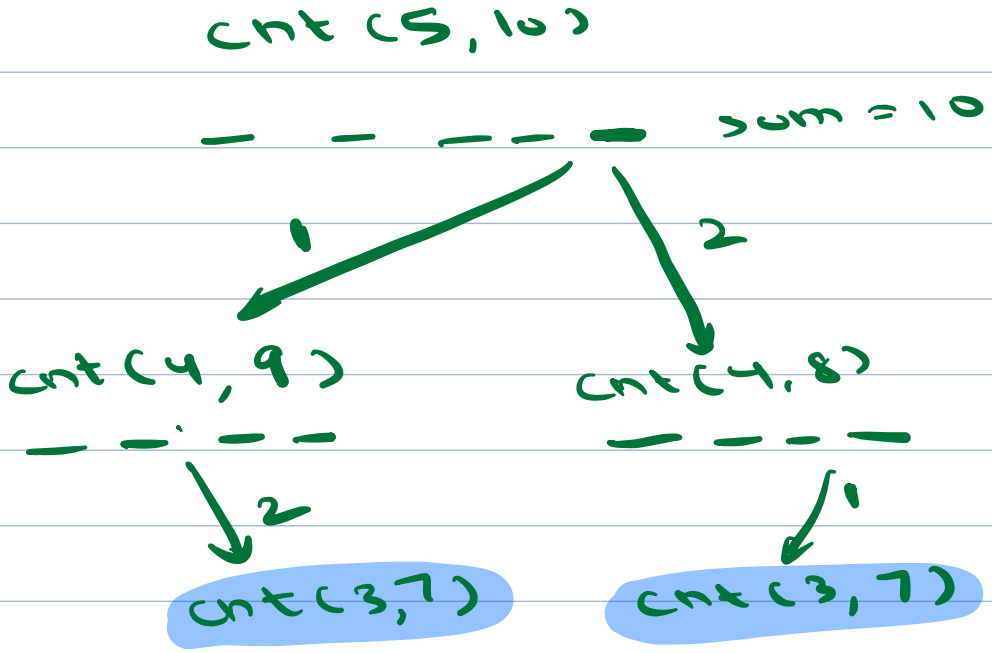
$cnt(A,B) = cnt(A-1,B-1) + cnt(A-1,B-2) + cnt(A-1,B-3) \ldots$

$$cnt(A,B) = \sum_{i=0}^{9} cnt(A-1, B-i)$$

↓

Cnt of A digit no. with sum = B





```
int dp[A+1][B+1] = {-1}

int cnt(int A, int B) {
    if (B <= 0) return 0
    if (A == 1) {
        if (B >= 1 && B <= 9) return 1
        else return 0
    }
    if (dp[A][B] != -1) return dp[A][B]

    int sum = 0
    for (d = 0; d <= 9; d++) {
        sum = (sum + cnt(A-1, B-d)) % M
    }
    dp[A][B] = sum % M
    return dp[A][B]
}
```

cnt(1, 5) = 1

cnt(1, 20) = 0

cnt(1, 0) = 0

TC : O(A*B)

SC : O(AB + A+B) = O(AB)

↓ DP[] ↓ stack

$x = y + z$

$x \% m = (y \% m + z \% m) \% m$

Catalan Numbers

$C_0 = 1$

$C_1 = 1$

$C_2 = C_0 C_1 + C_1 C_0 = 1 + 1 = 2$

$C_3 = C_0 C_2 + C_1 C_1 + C_2 C_0$

$= 2 + 1 + 2 = 5$

$C_4 = C_0 C_3 + C_1 C_2 + C_2 C_1 + C_3 C_0$

$= 5 + 2 + 2 + 5 = 14$

$C_N \rightarrow C_0$ to C_{N-1}

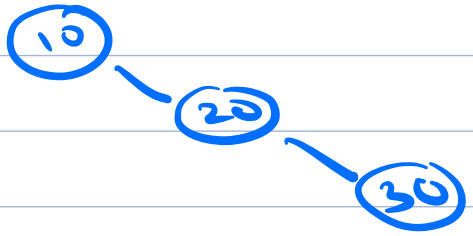
```
int C[N+1]
C[0] = 1      C[1] = 1
for (num = 2 ; num <= N ; num++) {
    int sum = 0 , j = num - 1
    for (i = 0 ; i <= num - 1 ; i++) {
        sum += C[i] * C[j]
        j--
    }
    C[num] = sum
}
return C[n]
```

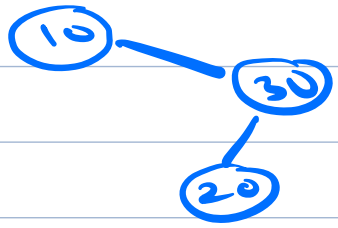
TC : $O(N^2)$

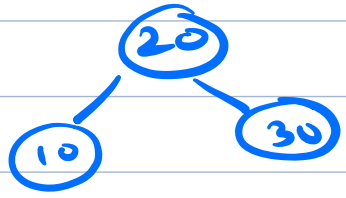
SC : $O(N)$

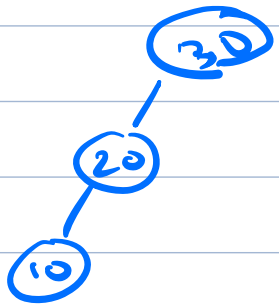
Given N, count total no. of unique BSTs that can be formed using N distinct numbers.

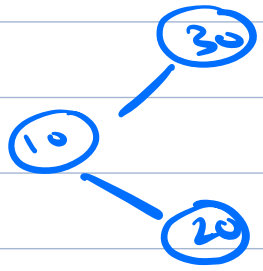
N = 3 10 20 30 ans = 5



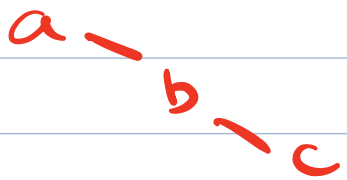


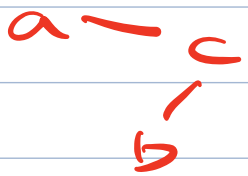


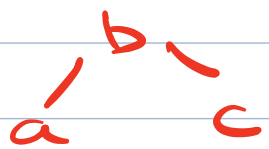


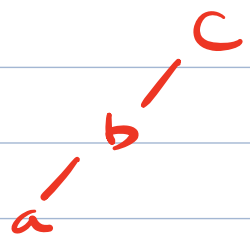


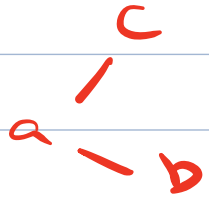
a < b < c





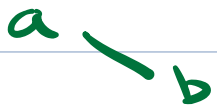


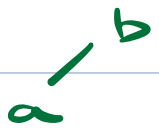




N = 2 a b a < b

ans = 2

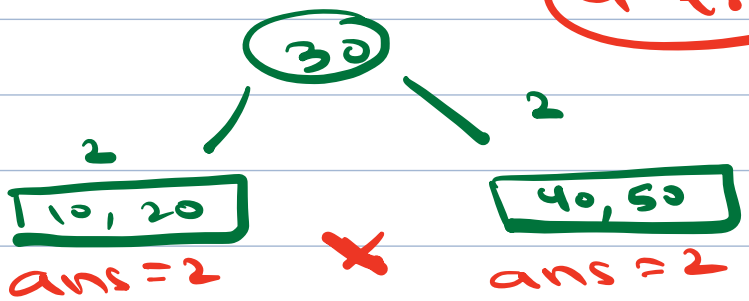


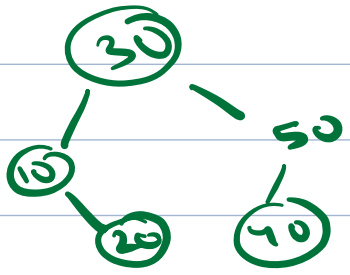


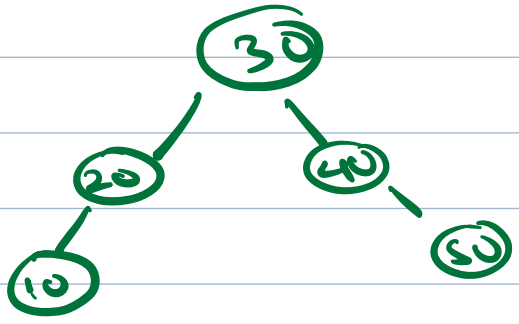
N = 5 10 20 30 40 50

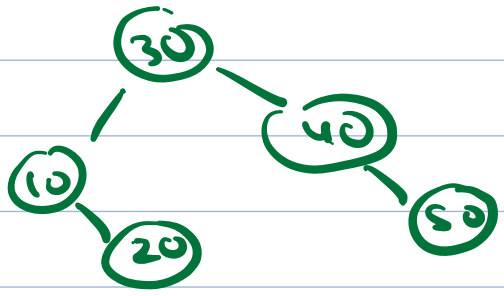
① 30 as root

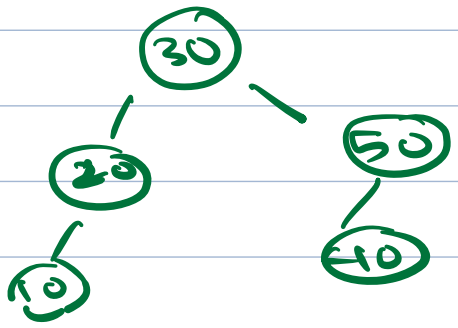
4 trees



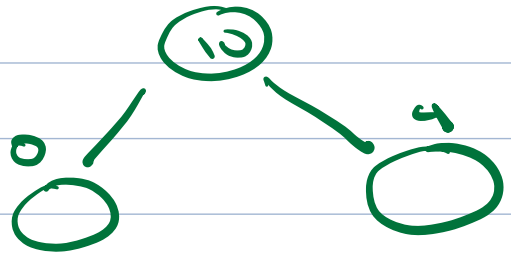






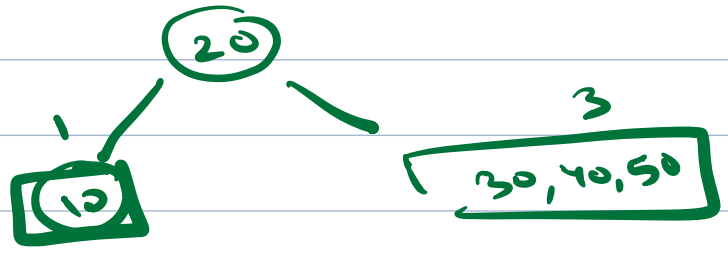


② 10 as root



ways(0) x ways(4)

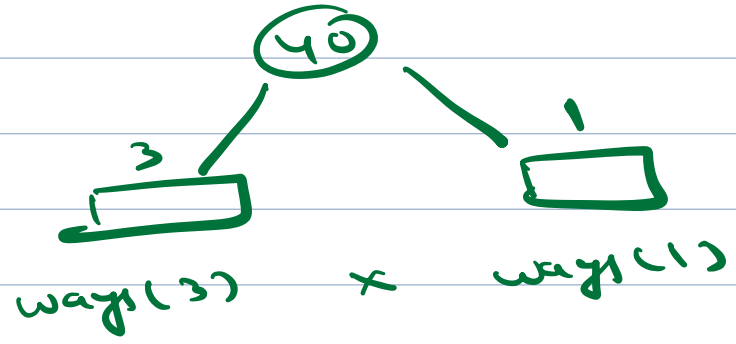
③ 20 as root



ways(1) × ways(3)

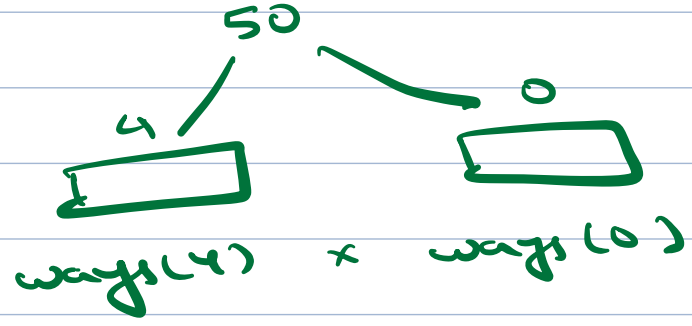
1

④ 40 as root



way(3) × way(1)

⑤ 50 as root



way(4) × way(0)

ways(5) = 10 as root + 20 as root + 30 as root + 40 as root + 50 as root

$= ways(0) \times ways(4) + ways(1) \times ways(3)$

$+ ways(2) \times ways(2) +$

$ways(3) \times ways(1) + ways(4) \times ways(0)$

$C(5) = C_0 C_4 + C_1 C_3 + C_2 C_2 +$

$C_3 C_1 + C_4 C_0$

No. of unique BSTs with N nodes = N^{th} catalan no.

---

10 20 30 40 50 60