Catalan Numbers

Nth Catalan Number

1. Unique Binary Search Tree
2. Dyk Paths
3. Dyk Words
4. Intersecting chords in Circle
5. Count Balanced Parantheses String
6. Count No of ways of Triangulation
7. Count No of Handshakes
8. Count Mountain Ranges

Nth Catalan Numbers

$C_0 = 1$

$C_1 = 1$

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

$C_3 = C_0 * C_2 + C_1 * C_1 + C_2 * C_0 = 1*2 + 1*1 + 2*1 = 5$

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

$= 1*5 + 1*2 + 2*1 + 5*1 = 14$

$C_5 = C_0 * C_4 + C_1 * C_3 + C_2 * C_2 + C_3 * C_1 + C_4 * C_0$

$= 1*14 + 1*5 + 2*2 + 5*1 + 14*1$

$= 42$

n	C_n	n	C_n	n	C_n
1	1	11	58,786	21	24,466,267,020
2	2	12	208,012	22	91,482,563,640
3	5	13	742,900	23	343,059,613,650
4	14	14	2,674,440	24	1,289,904,147,324
5	42	15	9,694,845	25	4,861,946,401,452
6	132	16	35,357,670	26	18,367,353,072,152
7	429	17	129,644,790	27	69,533,550,916,004
8	1,430	18	477,638,700	28	263,747,951,750,360
9	4,862	19	1,767,263,190	29	1,002,242,216,651,368
10	16,796	20	6,564,120,420	30	3,814,986,502,092,304

C_N will overflow integer range when $N \geq 20$

Nth Catalan No → using Dynamic Programming

$$C_N = C_0 * C_{N-1} + C_1 * C_{N-2} + C_2 * C_{N-3} + \cdots C_{N-1} * C_0$$

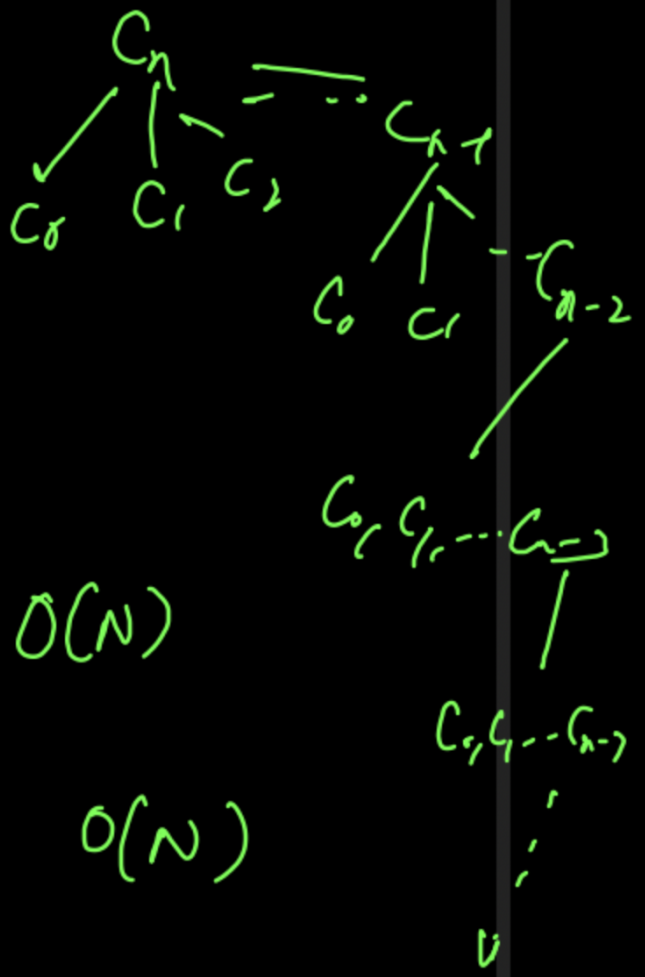
$$C_N = \sum_{j=0}^{N-1} C_j * C_{N-1-j}$$

DP

C_0	C_1	C_2	C_3	C_4	C_5	C_6
1	1	1+1 = 2	1*2 + 1*1 + 2*1 = 5	1*5 + 1*2 + 2*1 + 5*1 = 14	1*14 + 1*5 + 2*2 + 5*1 + 14*1 = 42	132
		$C_0 * C_1$ + $C_1 * C_0$	$C_0 * C_2$ + $C_1 * C_1$ + $C_2 * C_0$	$C_0 * C_3$ + $C_1 * C_2$ + $C_2 * C_1$ + $C_3 * C_0$	$C_0 * C_4$ + $C_1 * C_3$ + $C_2 * C_2$ + $C_3 * C_1$ + $C_4 * C_0$	

Recursion

```
public int catalan(int n){
    if(n == 0 || n == 1) return 1;

    int ans = 0;
    for(int i=0; i<n; i++){
        ans = ans + catalan(i) * catalan(n - 1 - i);
    }
    return ans;
}
```



Recursion → Height → $O(N)$

Recursion → Breadth (calls) → $O(N)$

Total TC = $O(N^N)$ Exponential

Memoization

Recursive

```
public int catalan(int n, int[] dp){
    if(n == 0 || n == 1) return 1;
    if(dp[n] != -1) return dp[n];

    int ans = 0;
    for(int i=0; i<n; i++){
        ans = ans + catalan(i, dp) * catalan(n - 1 - i, dp);
    }

    return dp[n] = ans;
}
public int numTrees(int n) {
    int[] dp = new int[n + 1];
    Arrays.fill(dp, -1);
    return catalan(n, dp);
}
```

Recursion

for loop

Time → O(N * N)

Space → O(N) 1D DP

O(N) R.C.S.

Tabulation

Iterative

```
int[] dp = new int[n + 1];
dp[0] = dp[1] = 1;
for(int i=2; i<=n; i++)
    for(int j=0; j<i; j++)
        dp[i] += dp[j] * dp[i - j - 1];
return dp[n];
```

Time → O(N * N)

Space → O(N) 1D DP

Nth catalan No using Binomial coeff

$$N^{th}\ Catalan = \frac{{}^{2n}C_n}{(n+1)} = \frac{\frac{2n!}{n!\ n!}}{n+1} = \frac{2n!}{(n+1)!\ n!}$$

Combination

$$f(0) = \frac{{}^{0}C_0}{1} = \frac{1}{1} = 1$$

$$f(1) = \frac{{}^{2}C_1}{2} = \frac{\frac{2!}{1!\ 1!}}{2} = 1$$

$$f(2) = \frac{{}^{4}C_2}{3} = \frac{4*3*2*1}{2*1*1*3} = 2$$

Recurrence Relation ${}^{n}C_r = {}^{n}C_{r-1} + {}^{n-1}C_{r-1}$

→ 2D DP

Perks

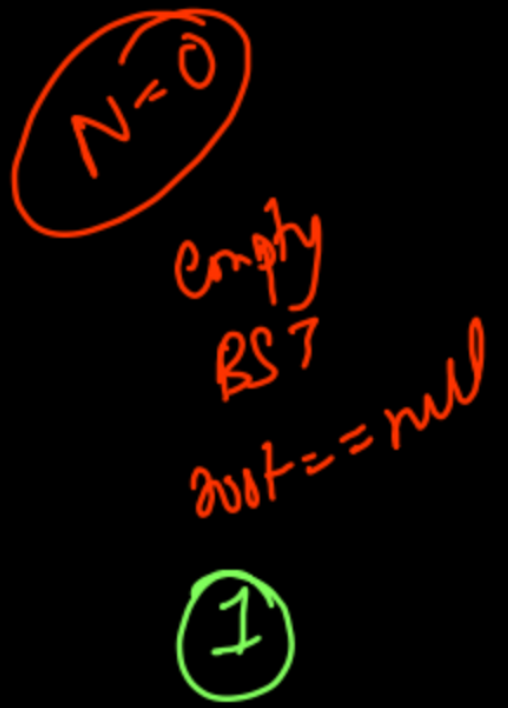
① Gadgets → 5 Gadgets
- Headset → 13k
- Mouse, Keyboard
- Apple multiport Adapter
- 15k smartwatch

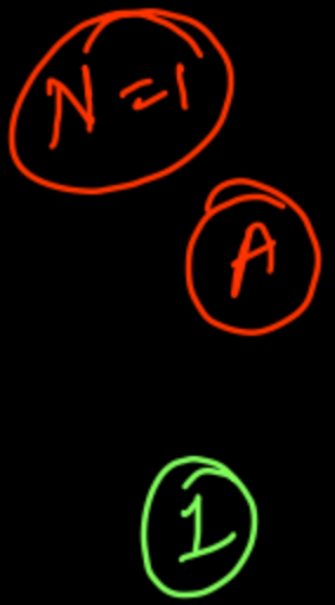
② Food
- Breakfast
- \+
- Lunch
- \+
- Snacks

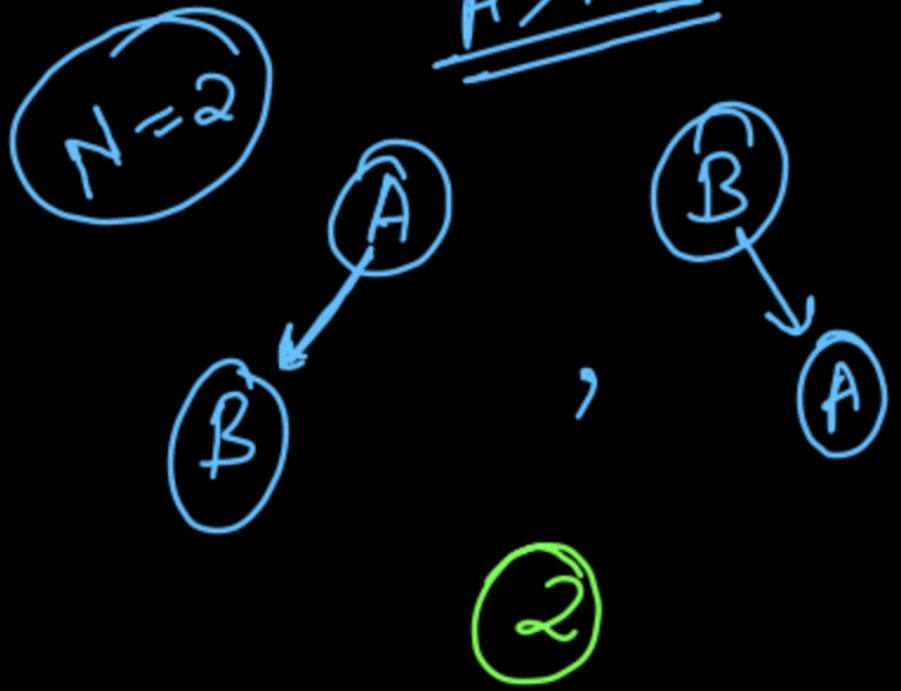
\+ Barista

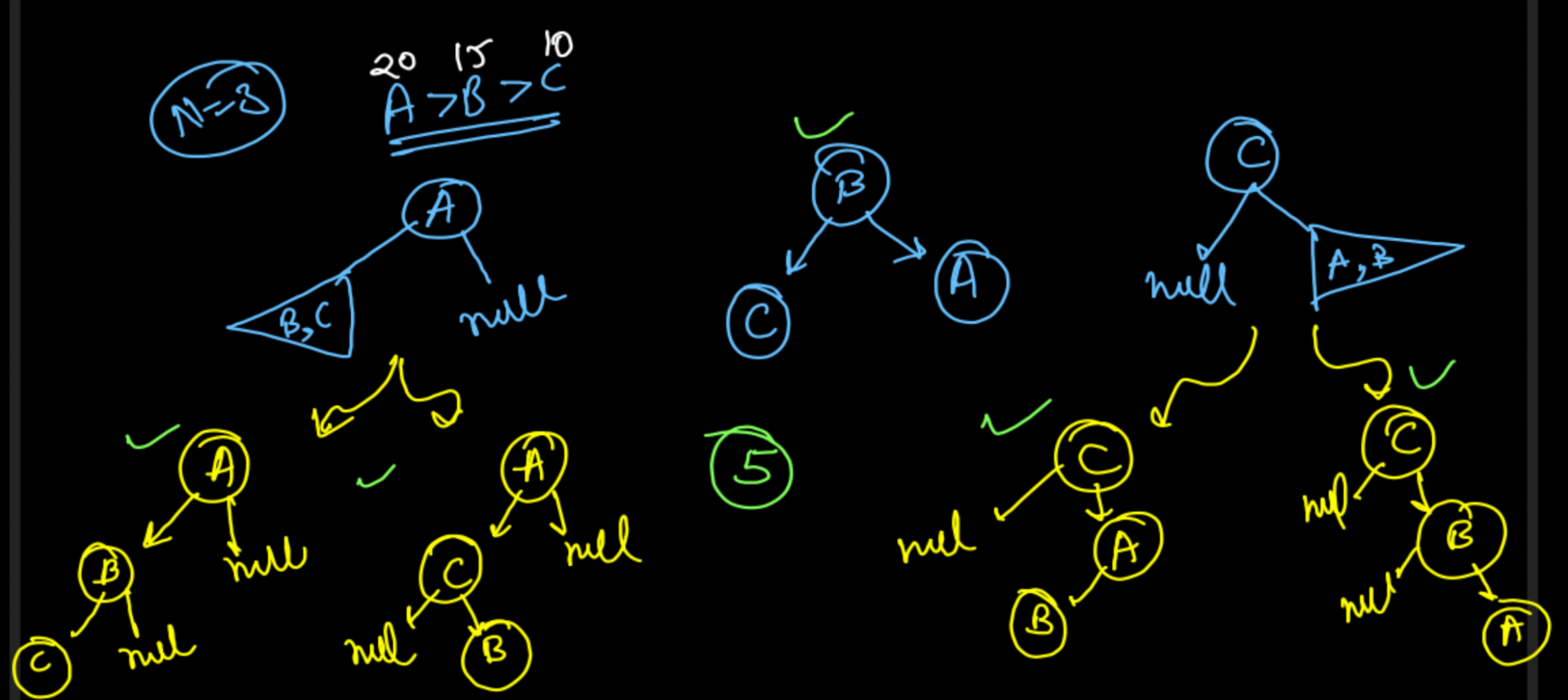
③ Uber → 20 dollars → 1500 Rs

Unique BST



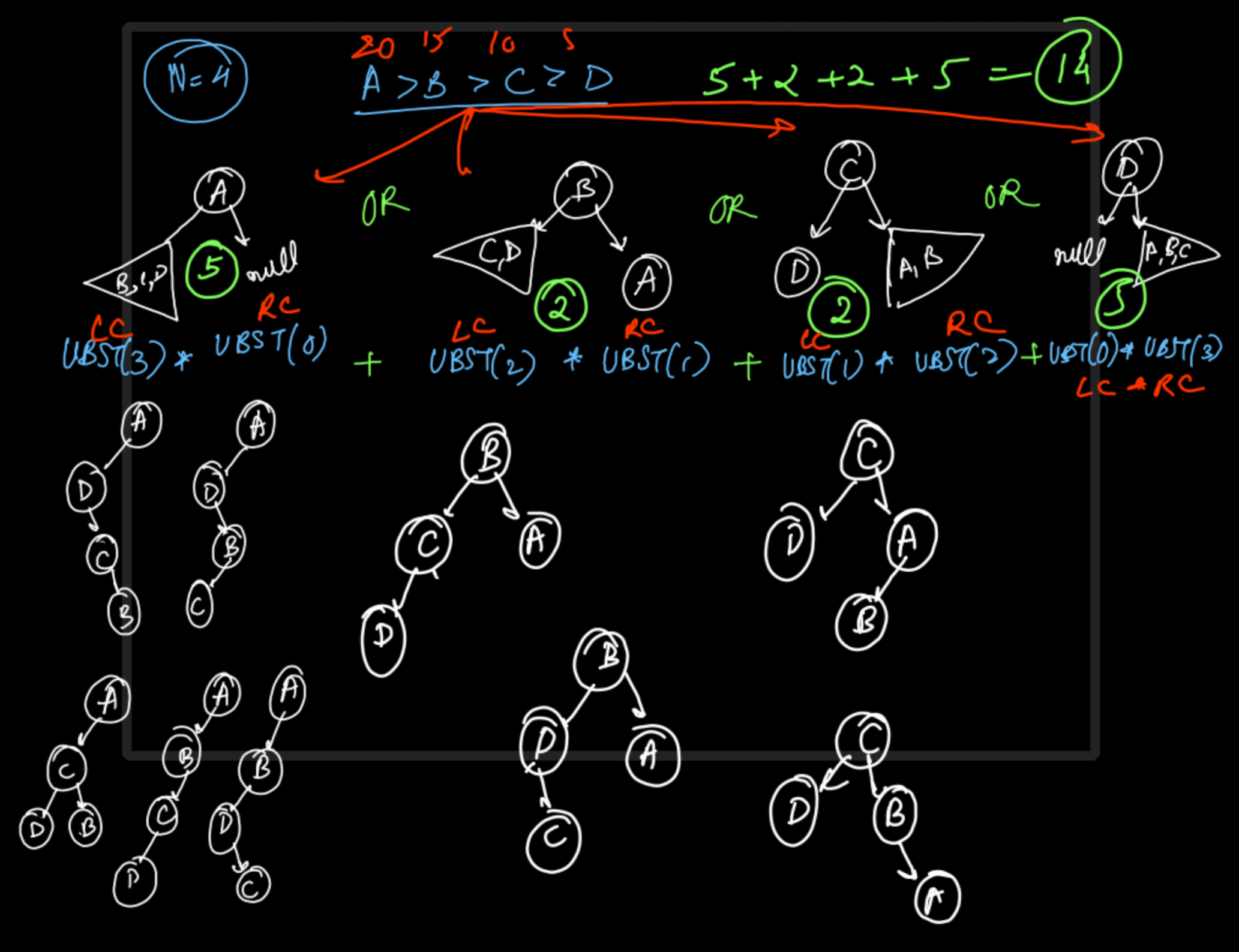






Unique BST(n) = UBST(n) with k_1 as root + k_2 as root

+ k_3 as root + -- k_n as root


N=4
20 15 10 5
A > B > C > D
5+2+2+5 = 14
OR
OR
OR
LC
RC
UBST(3) * UBST(0)
+
UBST(2) * UBST(1)
+
UBST(1) * UBST(2)
+
UBST(0) * UBST(3)
LC * RC

② Count Balanced Parantheses

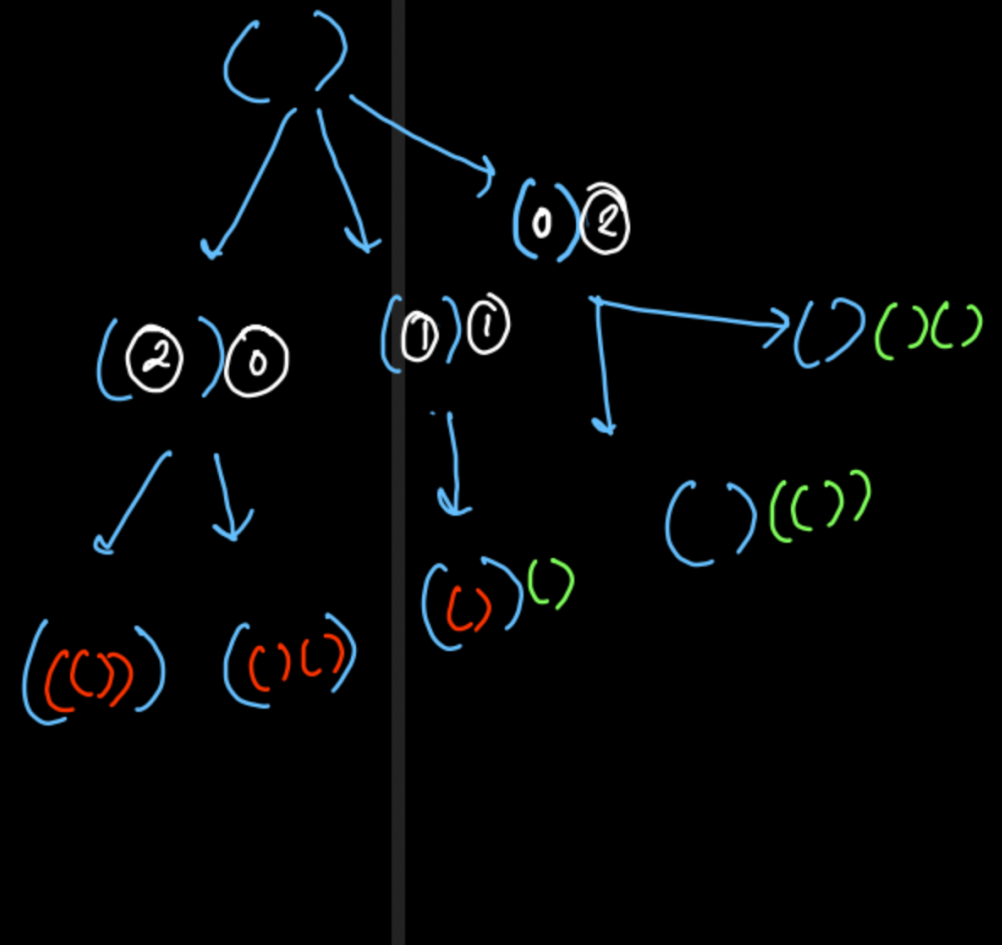
N → opening brackets, closing brackets } pairs

N=0
""

N=1
"()"
1

N=2
"(())"
"()()"
2

N=3
((()))
(()())
(())()
()(())
()()()
5



N=4

( )

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

(3)0 + (2)1 + (1)2 + (0)3

5 2 2 5 = 14

(()())

(())()

()(())

()()()

()((()))

(()()())

((()))()

((())())

(()(()))

((()()))

()()()()

()()(())

()(())()

()(()())

()((()))

(())()()

(())(())

((()))()

((())())

(()(()))

((()()))

③ Count Mountain Ranges

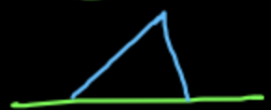
n uphills, n downhills ⇒ above or on the sea level

uphill → (

downhill →)

Count balanced strings

N=1

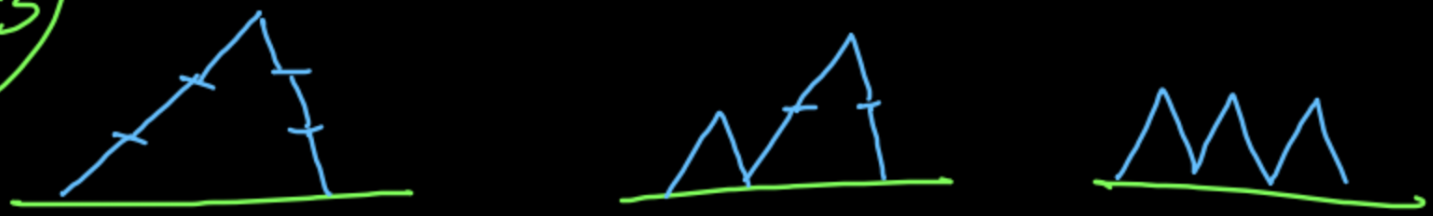


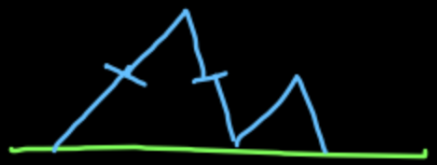
①

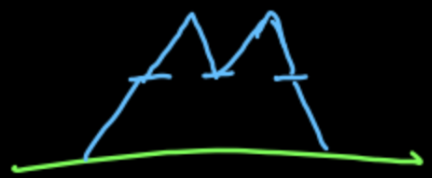
N=2

②

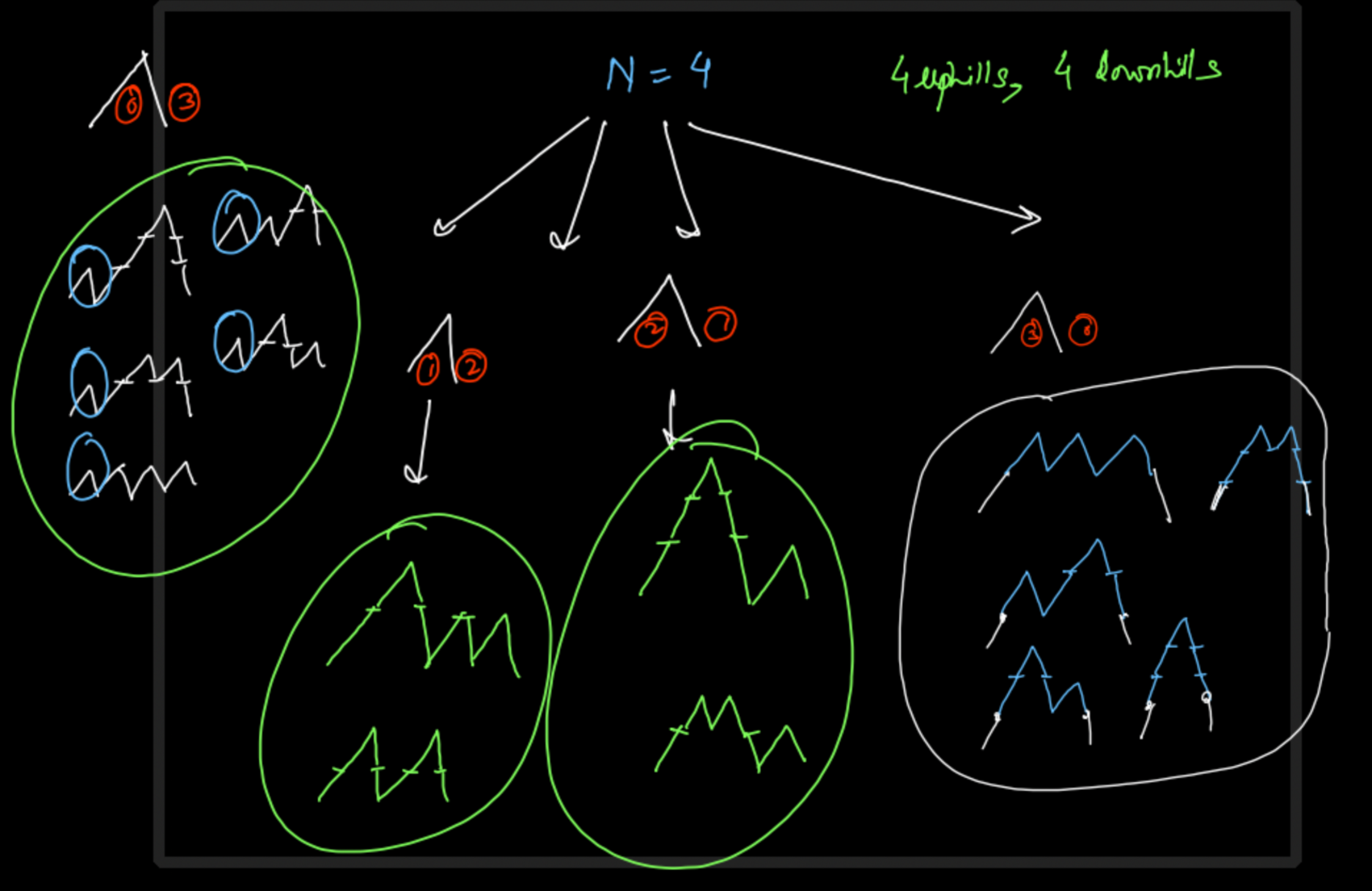
N=3







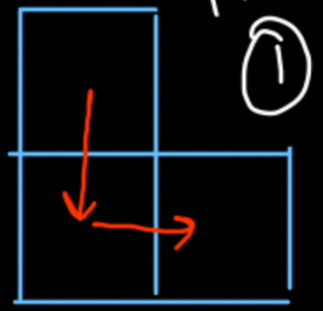
⑤

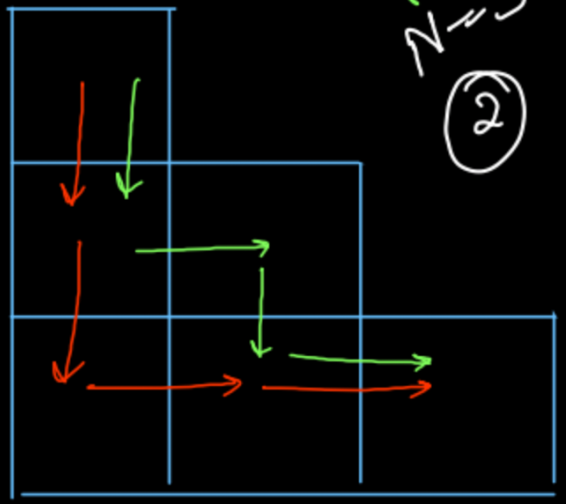

N = 4
4 uphills, 4 downhills

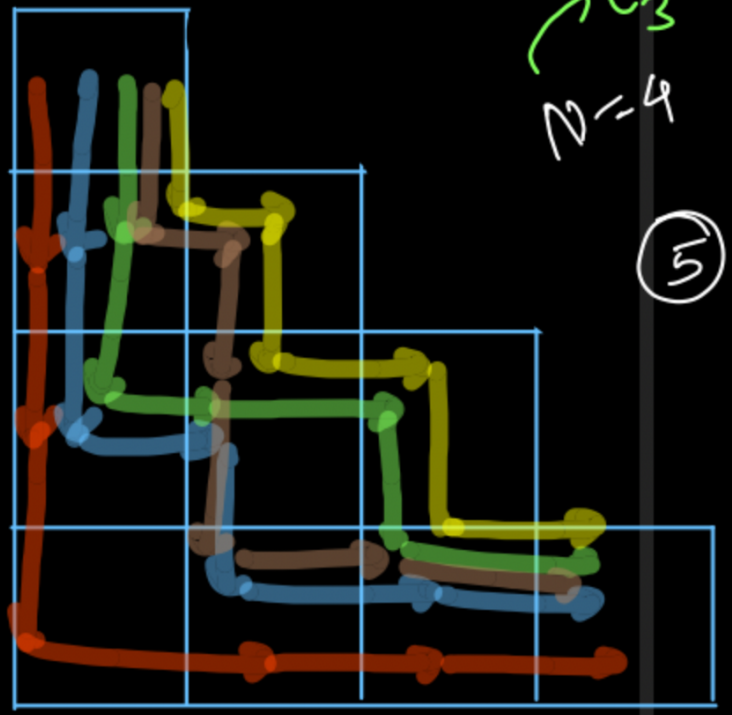
If entire matrix
↳ Unique Paths-1
↳ $\frac{(n+m-2)!}{(n-1)!(m-1)!}$

(4) Count Paths in Upper-Right Triangle or Lower-Left Triangle

2 moves: Down, Right







(N rows, N cols) ⇒ C_{N-1} ⇒ (N-1) uphills + (N-1) downhills

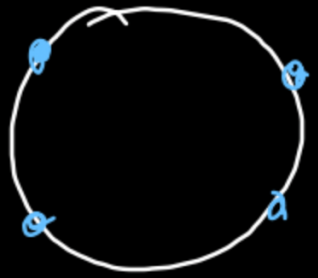
(5) Handshakes / Non-intersecting chords

Every person should do a handshake, and no person can do 2 handshakes

N = 2

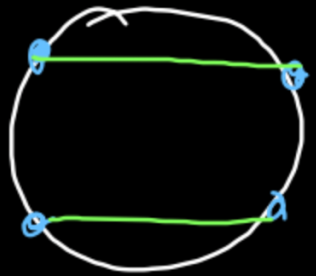
(1)

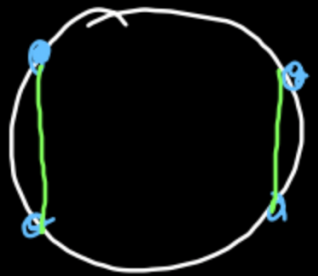
(4)



N = 4

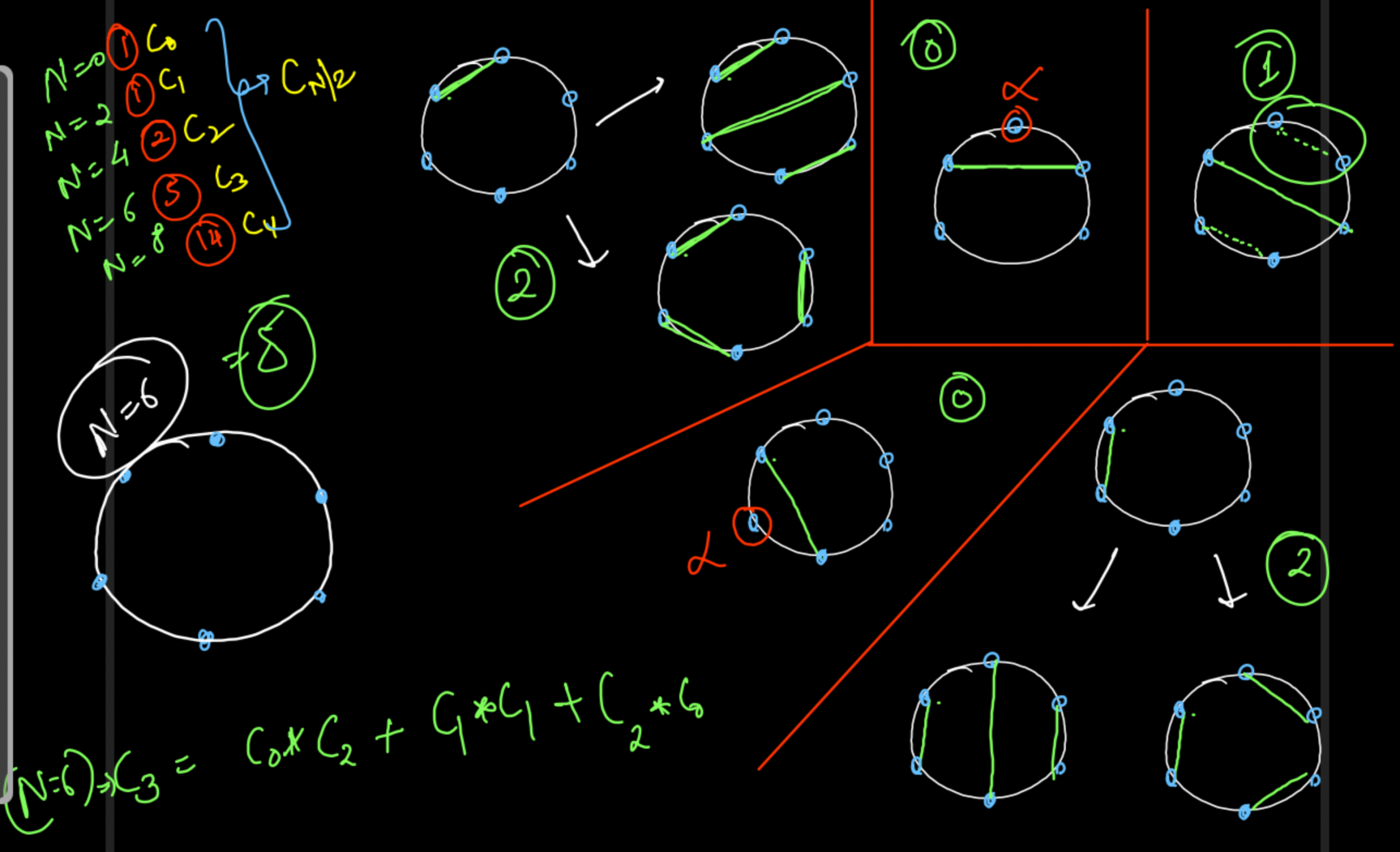
(2)



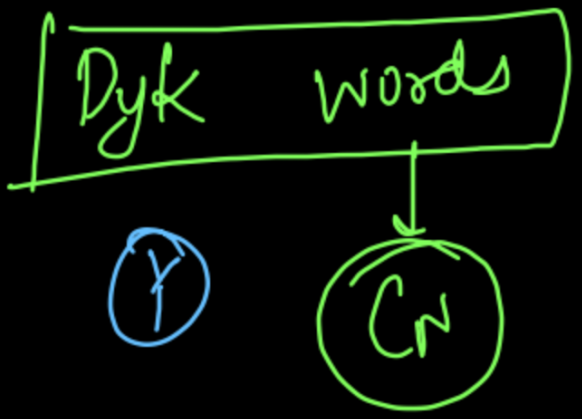


C_0, C_1, C_2, C_3, C_4

1 1 2 5 14


N=0 1 C0
N=2 1 C1
N=4 2 C2
N=6 5 C3
N=8 14 C4
→ CN/2
2
0
α
1
N=6
δ
0
α
2
(N=6) ⇒ C3 = C0 * C2 + C1 * C1 + C2 * C0

Dyck words



substring starting with 0th index
↑
in all prefixes

↳ count of X ≥ count of Y

X = '('

Y = ')'

count of balanced parenthesis strings

N = 2

XYXY

XXYY

2

N = 3

XYXYXY

XXYYXY

XYXXYY

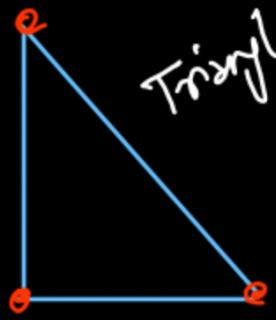
XXXYYY

5

XXYXYY

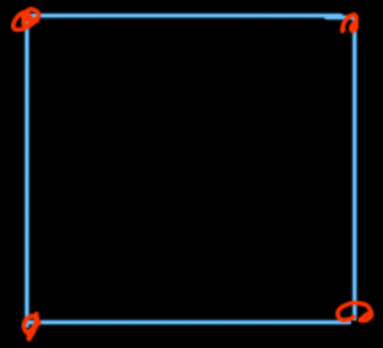
(8) Count ways of Triangulation

N = 2
ans = 1



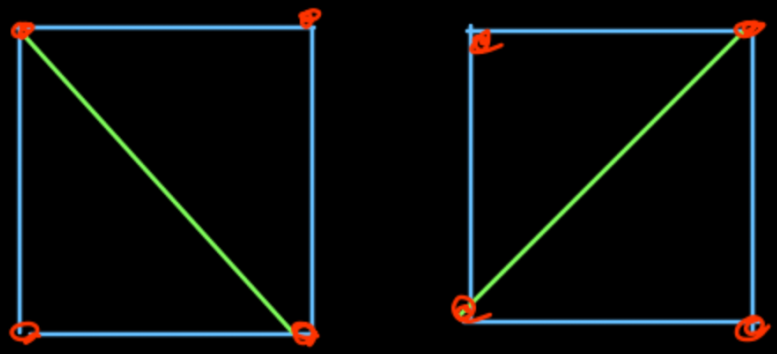
Triangle

N = 3
ans = 1



Rectangle

N = 4
ans = 2



N = 2 ⇒ (1) → C_0

N = 3 ⇒ (1) → C_1

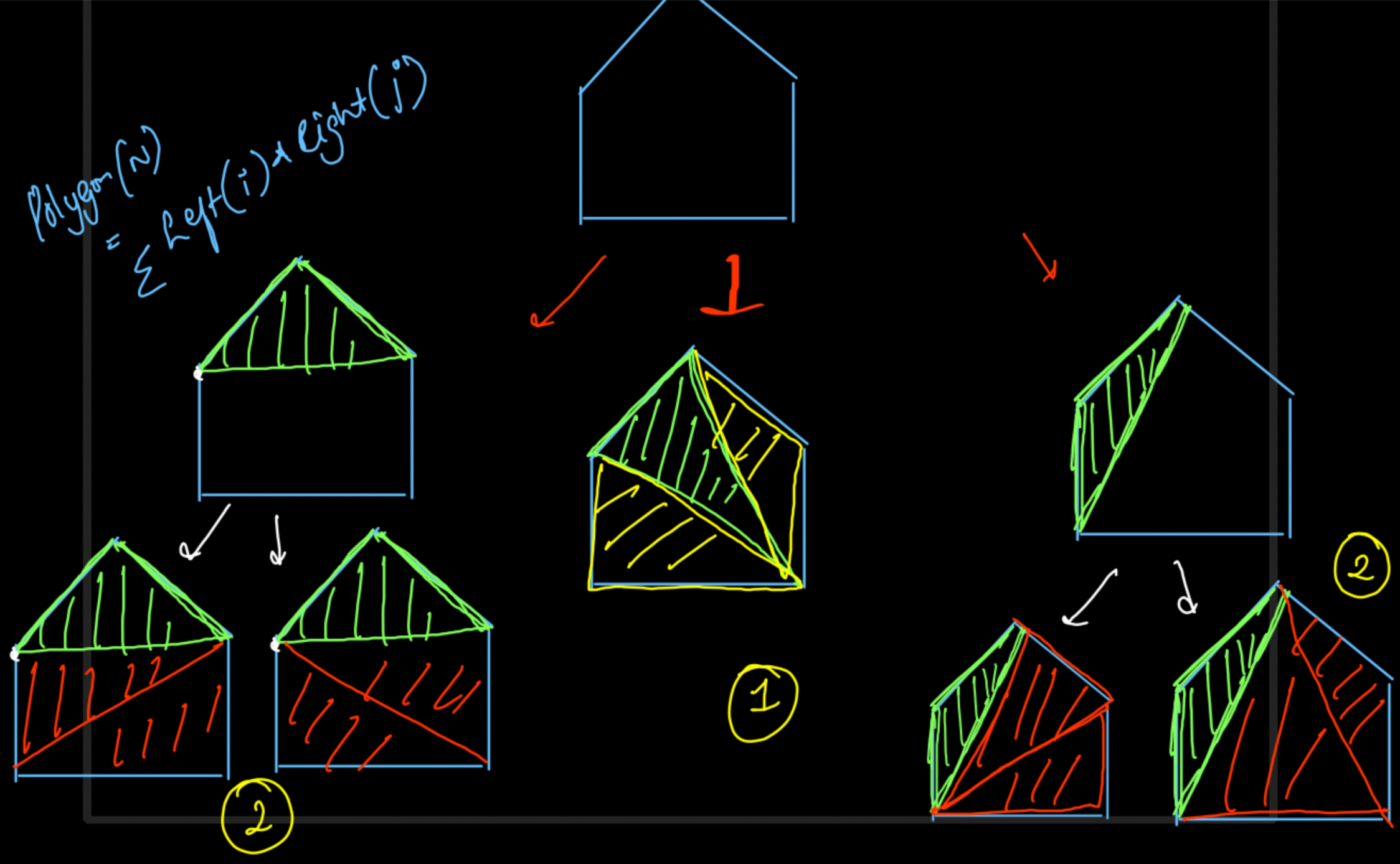
N = 4 ⇒ (2) → C_2

N = 5 ⇒ (5) → C_3

N = 6 ⇒ (14) → C_4

$ans(N) = C_{N-2}$

Polygon(N)
$= \sum$ Left(i) + Right(j)



Kadane's Algorithm

→ Maximum sum Subarray → Kadane (DP)
                        → D&C

→ Maximum sum Circular Subarray → Pref & Suff
                                → Max & Min

→ & Concatenation

→ Maximum Product Subarray → Max & Min

→ Max sum Submatrix (Subrectangle)

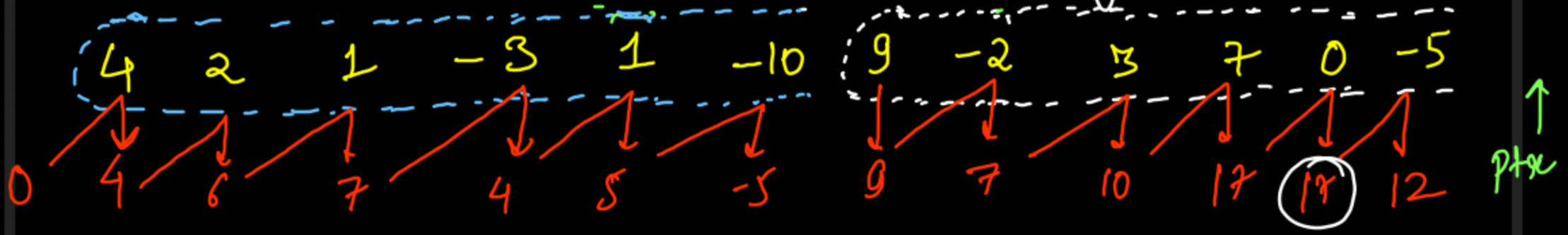
→ Max sum Subarray of size atleast k

→ Max Difference of 0's & 1's

→ max sum Non-Overlapping Subarrays → 2
                                     → 3
                                     → K

Max Sum Subarray

Contiguous



Brute force

↓

All subarrays

↓

$O(N^2)$

Kadane's Algorithm

maxSum = -∞ 4 6 7 7 9 10 17 17

currSum = 0 4+2=6+1=7-3=4+1

=5-10=-5 9-2=

7+3=10+7

=17+0=17

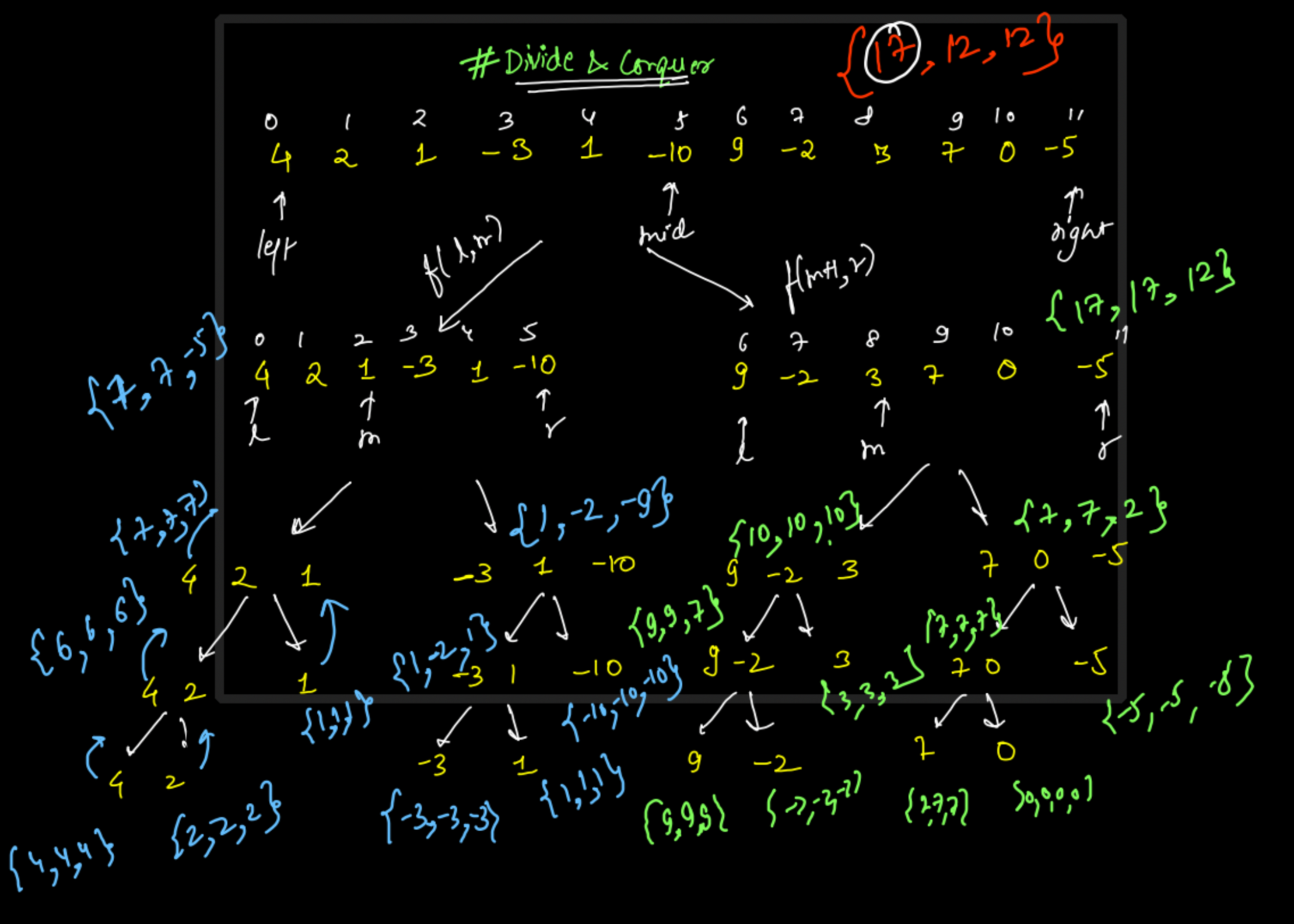
12-5=12

Kadane's Algorithm

```java
public int maxSubArray(int[] nums) {
    int currSum = 0, maxSum = Integer.MIN_VALUE;

    for(int i=0; i<nums.length; i++){

        if(nums[i] + currSum >= nums[i])
            currSum += nums[i];
        else
            currSum = nums[i];

        maxSum = Math.max(maxSum, currSum);
    }

    return maxSum;
}
```

Time → O(N)

Space → O(1)


Divide & Conquer
{17, 12, 12}
0 1 2 3 4 5 6 7 8 9 10 11
4 2 1 -3 1 -10 9 -2 3 7 0 -5
left
mid
right
f(l,m)
f(m+1,r)
{17, 17, 12}
{7, 7, -5}
{7, 7, 7}
{1, -2, -9}
{10, 10, 10}
{7, 7, 2}
{6, 6, 6}
{1, -2, 1}
{9, 9, 7}
{7, 7, 7}
{3, 3, 2}
{-5, -5, -8}
{4, 4, 4}
{2, 2, 2}
{-3, -3, -3}
{1, 1, 1}
{9, 9, 9}
{-7, -7, -7}
{7, 7, 7}
{0, 0, 0}

Return Type

→ Max Sum Subarray

→ Max Sum Prefix

→ Max Sum Suffix

→ Total Sum

maxSum

= { leftMaxSum OR rightMaxSum OR leftSuffix + RightPrefix }

PrefixSum

= leftPrefix OR leftTotal + rightPrefix

SuffixSum

= RightSuffix OR rightTotal + leftSuffix

```
public static class Pair{
    int maxSum = 0;
    int prefix = 0;
    int suffix = 0;
    int total = 0;

    Pair(int val){
        maxSum = prefix = suffix = total = val;
    }
}
```

Time → O(N)

Space → O($\log_2 N$)

↳ R.C.S.

```
public Pair helper(int left, int right, int[] nums){
    if(left == right){
        return new Pair(nums[left]);
    }

    int mid = left + (right - left) / 2;
    Pair lans = helper(left, mid, nums);
    Pair rans = helper(mid + 1, right, nums);

    Pair ans = new Pair(0);
    ans.total = lans.total + rans.total;
    ans.prefix = Math.max(lans.prefix, lans.total + rans.prefix);
    ans.suffix = Math.max(rans.suffix, rans.total + lans.suffix);
    ans.maxSum = Math.max(lans.suffix + rans.prefix,
                    Math.max(lans.maxSum, rans.maxSum));
    return ans;
}
```

```
public int maxSubArray(int[] nums) {
    if(nums.length == 0) return 0;
    return helper(0, nums.length - 1, nums).maxSum;
}
```

$$T(n) = 2^1 T(n/2) + O(1)$$

$$2^1 T(n/2) = 2^2 T(n/4) + 2^1 O(1)$$

$$2^2 T(n/4) = 2^3 T(n/8) + 2^2 O(1)$$

$$2^3 T(n/8) = 2^4 T(n/16) + 2^3 O(1)$$

$$\downarrow$$

$$2^{\log_2 n} T(1) = 2^{\log_2 n + 1} T(0) + 2^{\log_2 n} O(1)$$

$$T(n) = n + (2 + 2^2 + 2^3 + \cdots 2^{\log_2 n})$$

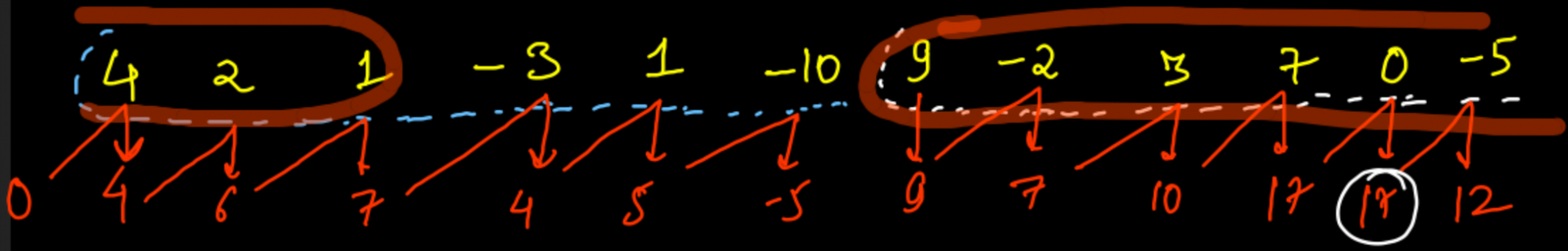
$$= n + \frac{2 * (2^{\log_2 n} - 1)}{2 - 1}$$

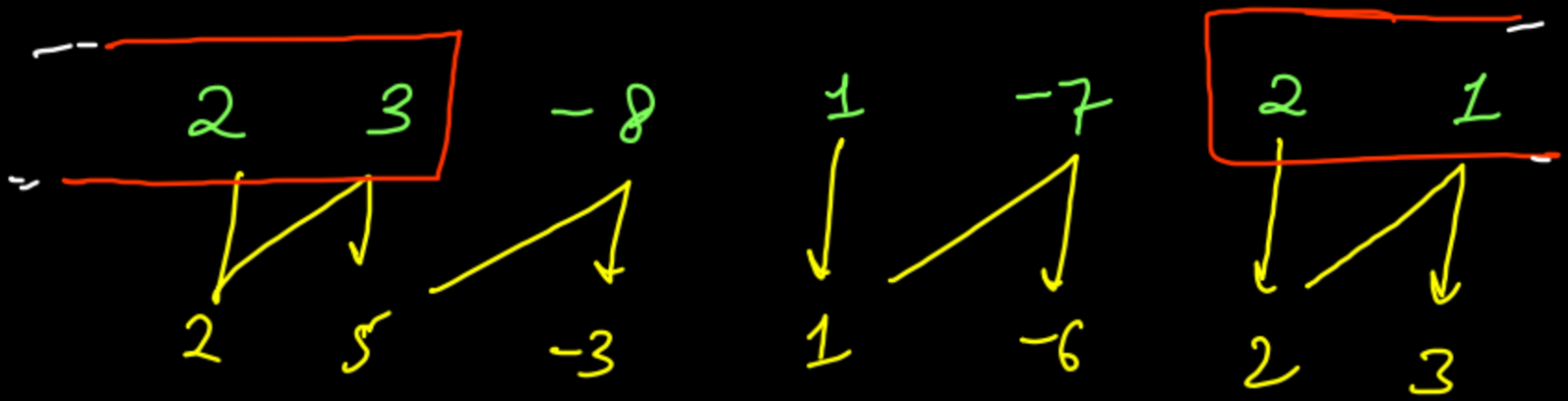
$$= n + \frac{2(n-1)}{2-1}$$

$$= 3n - 1$$

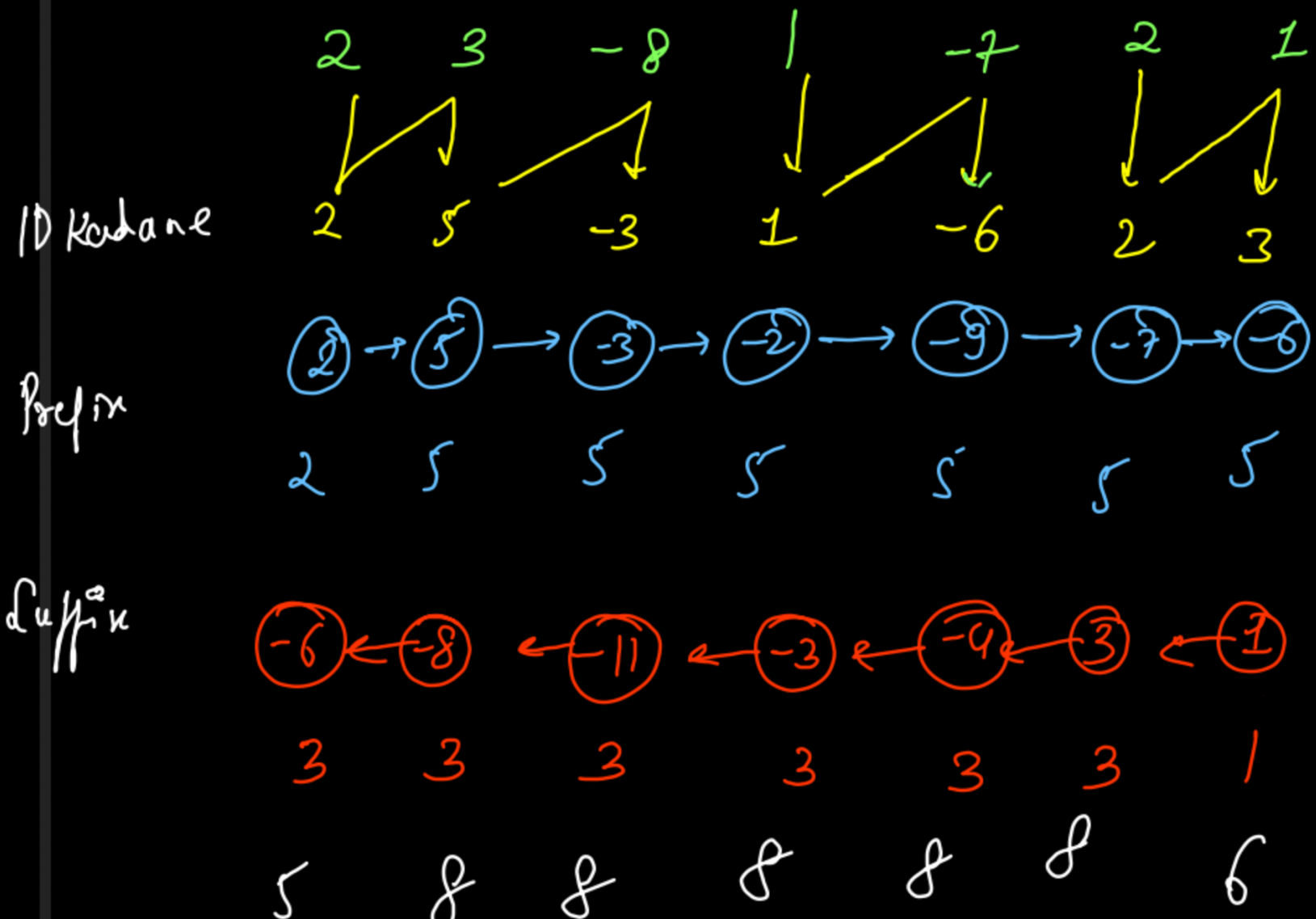
$$\Rightarrow \boxed{O(n)}$$

Max Sum Circular Subarray





$2+3+2+1 = 8$ max Sum



prefix(i)
max sum prefix ending at index $\leq i$

Suffix(i)
max sum suffix starting at index $\geq i$

```java
int currSum = 0, maxSum = Integer.MIN_VALUE;
for(int i=0; i<nums.length; i++){
    currSum = Math.max(currSum + nums[i], nums[i]);
    maxSum = Math.max(maxSum, currSum);
}

int prefSum = nums[0];
int[] prefix = new int[nums.length];
prefix[0] = prefSum;

for(int i=1; i<nums.length; i++){
    prefSum += nums[i];
    prefix[i] = Math.max(prefix[i - 1], prefSum);
}

int suffixSum = nums[nums.length - 1];
int[] suffix = new int[nums.length];
suffix[nums.length - 1] = suffixSum;

for(int i=nums.length-2; i>=0; i--){
    suffixSum += nums[i];
    suffix[i] = Math.max(suffixSum, suffix[i + 1]);
}

for(int i=0; i<nums.length; i++){
    suffixSum = (i + 1 < nums.length) ? suffix[i + 1] : 0;
    maxSum = Math.max(maxSum, prefix[i] + suffixSum);
}

return maxSum;
```

Approach 1

Time → O(N)

Space → O(N)

Ans ; Max Sum Subarray OR Total − Min Sum Subarray

Approach 2

```java
public int maxSubarraySumCircular(int[] nums) {
    int minSum = Integer.MAX_VALUE;
    int maxSum = Integer.MIN_VALUE;
    int currMin = 0, currMax = 0;
    int total = 0;

    for(int i=0; i<nums.length; i++){
        currMin = Math.min(currMin + nums[i], nums[i]);
        currMax = Math.max(currMax + nums[i], nums[i]);

        minSum = Math.min(minSum, currMin);
        maxSum = Math.max(maxSum, currMax);
        total = total + nums[i];
    }

    if(maxSum < 0) return maxSum;
    return Math.max(maxSum, total - minSum);
}
```

Time → O(N)

Space → O(1)

→ Corner Case : All negatives

{ −5, −3, −4, −8 }

totalSum = −20 minSum = −20

maxSum = −3

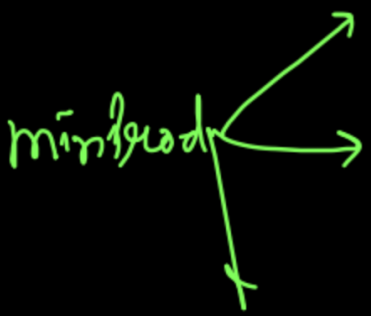
Max^m Product Subarray

	3	5	-1	2	-7	1	-2	5	-9	4
maxProd	3	15	-1	2	+210	210	+28	140	420*5*9	420*5*9 *1
minProd	3	5	-15	-30	-14	-14	-420	-420*5	140*-9	140*-9 *4

nums[i] {new subarray}

maxProd * nums[i]
+ve +ve

minProd * nums[i]
-ve -ve



nums[i] {new subarray}

maxProd * nums[i]
+ve -ve

minProd * nums[i]
-ve +ve

```
class Solution {
public int maxProduct(int[] nums) {
    int maxProd = 1;
    int minProd = 1;
    int ans = Integer.MIN_VALUE;

    for(int i=0; i<nums.length; i++){
        int newMax = Math.max(nums[i], Math.max(maxProd * nums[i], minProd * nums[i]));
        int newMin = Math.min(nums[i], Math.min(maxProd * nums[i], minProd * nums[i]));

        maxProd = newMax;
        minProd = newMin;
        ans = Math.max(ans, maxProd);
    }

    return ans;
}
```

Time → O(N)

Space → O(1)