

Catalan Numbers

Nth Catalan Number

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

Nth Catalan Number

$$C_0 = 1$$

$$C_1 = 1$$

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

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

$$\begin{aligned} C_4 &= C_0 * C_3 + C_1 * C_2 + C_2 * C_1 + C_3 * C_0 \\ &= 1 * 5 + 1 * 2 + 2 * 1 + 5 * 1 = 14 \end{aligned}$$

$$\begin{aligned} C_5 &= C_0 * C_4 + C_1 * C_3 + C_2 * C_2 + C_3 * C_1 + C_4 * C_0 \\ &= 1 * 14 + 1 * 5 + 2 * 2 + 5 * 1 + 14 * 1 \\ &= 42 \end{aligned}$$

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 $N \geq 20$

N^{th} Catalan No \rightarrow Using Dynamic Programming

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

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

WDP

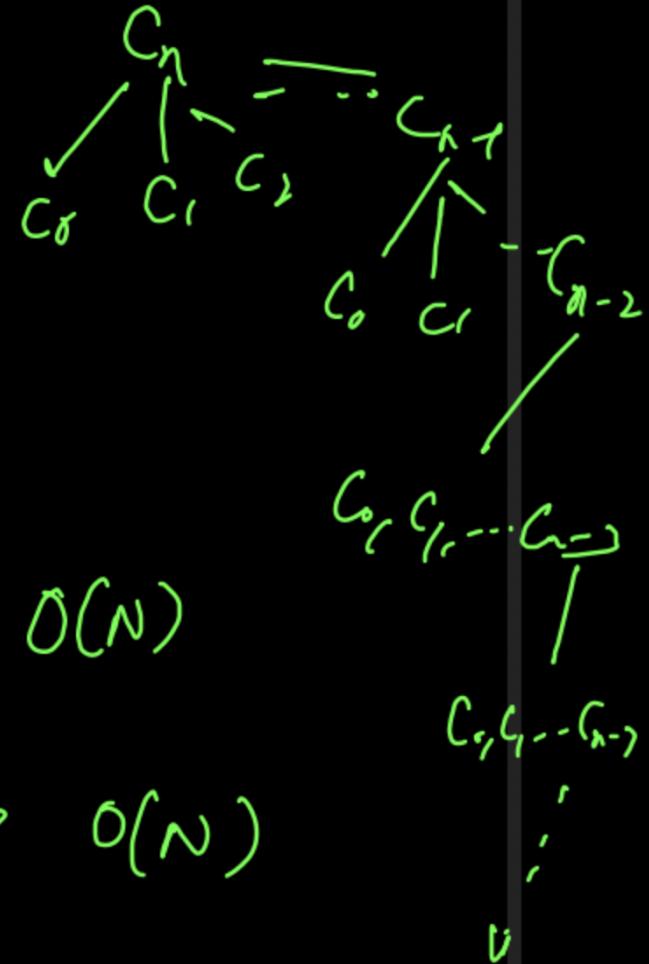
1	1	$1+1 = 2$	$1*2 + 1*1 + 2*1 = 5$	$1*5 + 1*2 + 2*1 + 5*1 = 14$	$1*4 + 1*5 + 2*2 + 5*1 + 14*1 = 42$	132
---	---	-----------	-----------------------	------------------------------	-------------------------------------	-----

$$C_0 \quad C_1 \quad C_2 \quad C_3 \quad C_4 \quad C_5 \quad C_6$$

$$\begin{array}{llll}
 C_0 * C_1 & C_0 * C_2 & C_0 * C_3 & C_0 * C_4 \\
 C_1 * C_0 & C_1 * C_1 & C_1 * C_2 & C_1 * C_3 \\
 C_2 * C_0 & C_2 * C_1 & C_2 * C_1 & C_2 * C_2 \\
 C_3 * C_0 & C_3 * C_1 & C_3 * C_2 & C_3 * C_1 \\
 C_4 * C_0 & C_4 * C_1 & C_4 * C_2 & C_4 * C_3 \\
 C_5 * C_0 & C_5 * C_1 & C_5 * C_3 & C_5 * C_2 \\
 C_6 * C_0 & C_6 * C_1 & C_6 * C_4 & C_6 * C_3
 \end{array}$$

~~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 \rightarrow Height $\rightarrow O(N)$

\rightarrow Breadth
(calls) $\rightarrow O(N)$

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

~~Memoization~~
Recursive

```
Solution 1  
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);  
}
```

Time $\rightarrow O(N \cdot N)$
Space $\rightarrow O(N)$ 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 $\rightarrow O(N \cdot N)$
Space $\rightarrow O(N)$ 1D DP

NM catalan No Using Binomial coeff

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

Combination

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

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

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

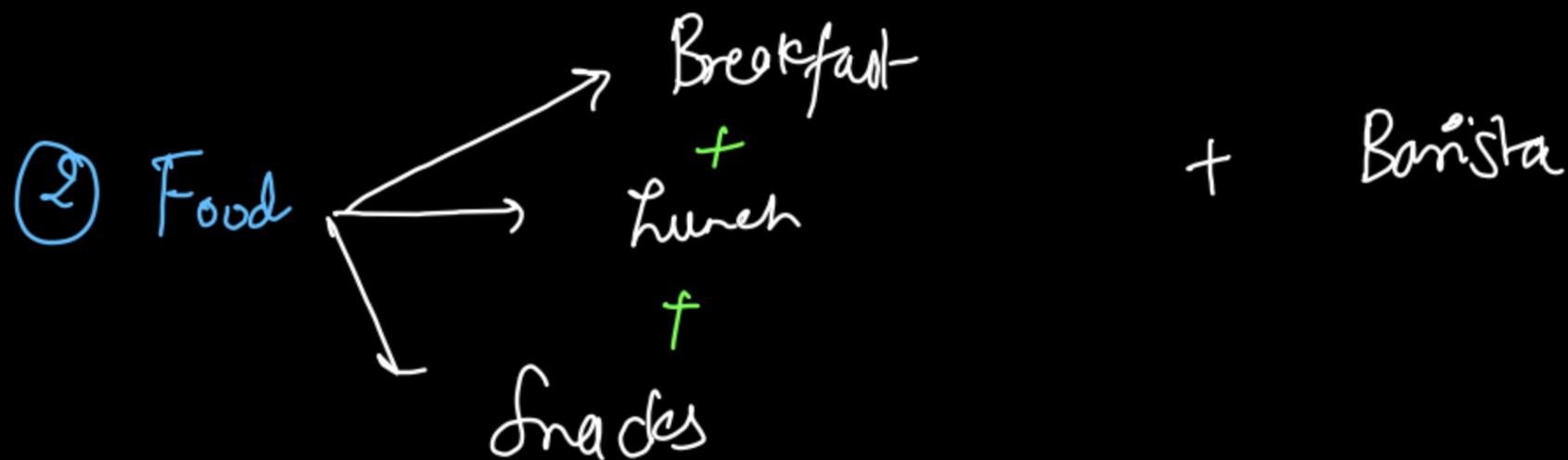
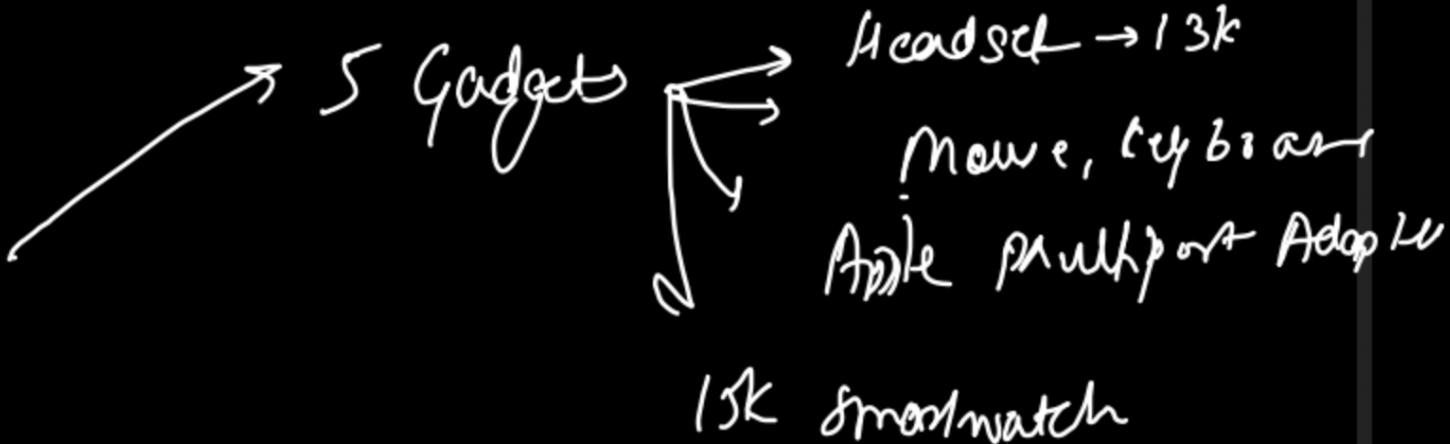
Recurrence Relation

$$n_{G_r} = n_{G_{r-1}} + n_{G_{r-1}}$$

→ 2D DP

Perks

① Gadgets



③ Uber → 20 dollars → \$100 R

①

Unique BST

$N=0$

empty
BST

$\text{root} = \text{null}$

①

$N=1$

A

①

$N=2$

B → A

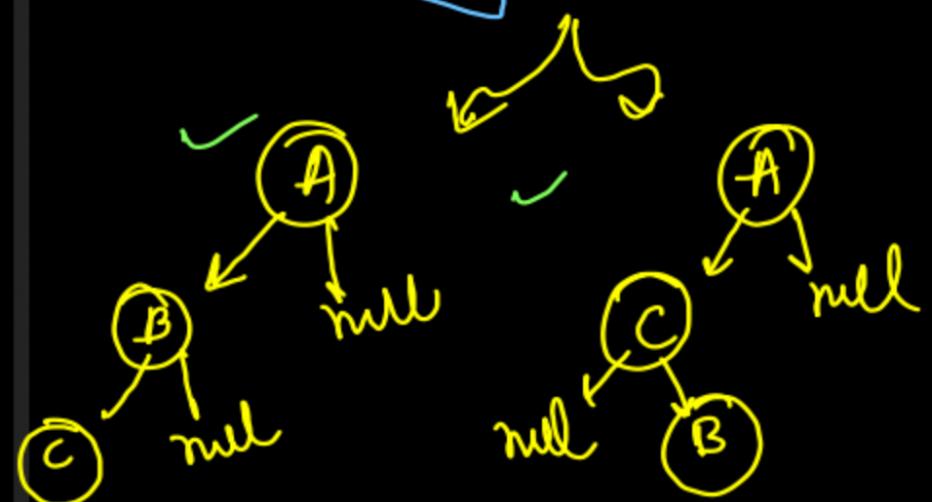
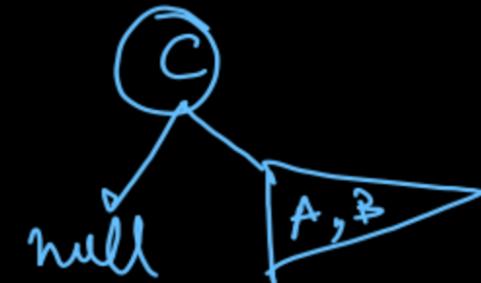
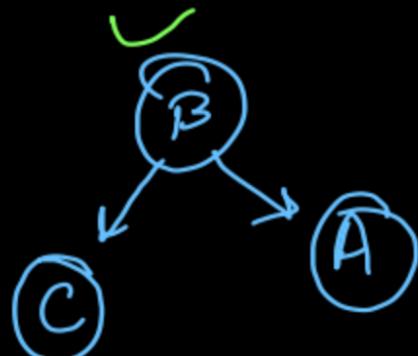
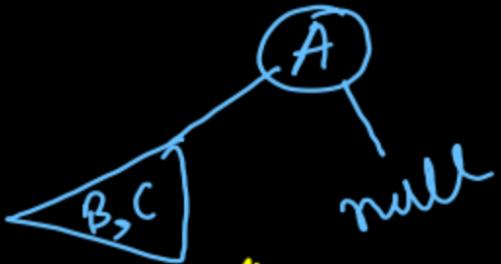
②

A > B

B → A

$N=3$

$20 \ 15 \ 10$
 $A > B > C$



5

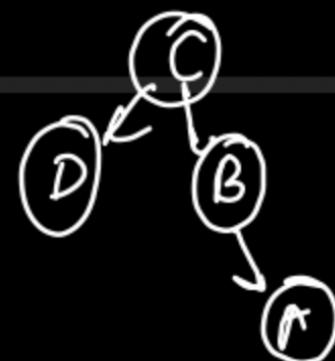
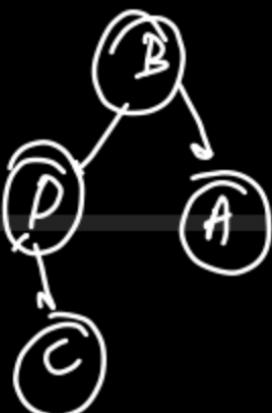
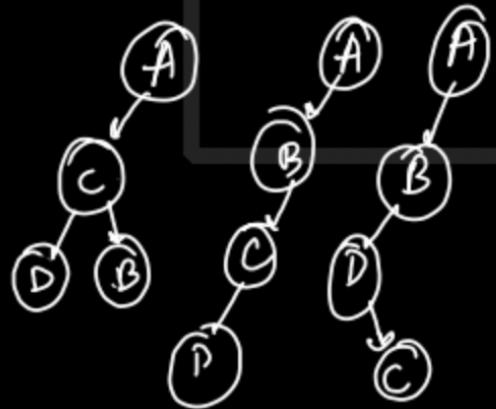
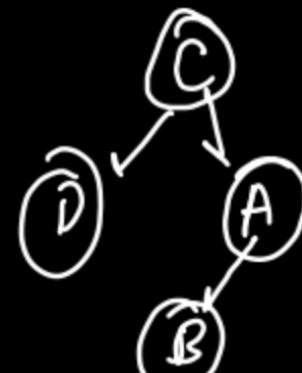
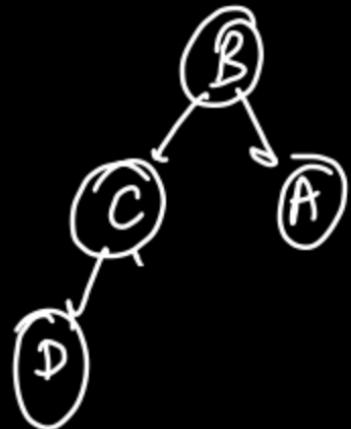
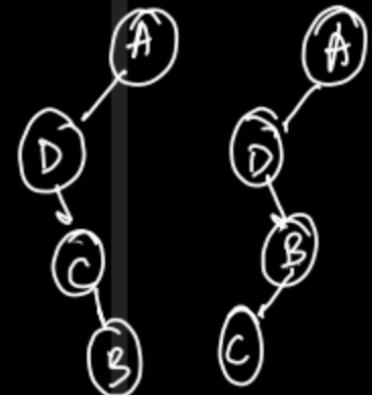
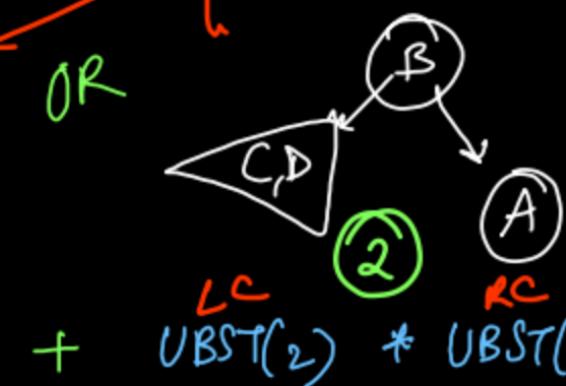
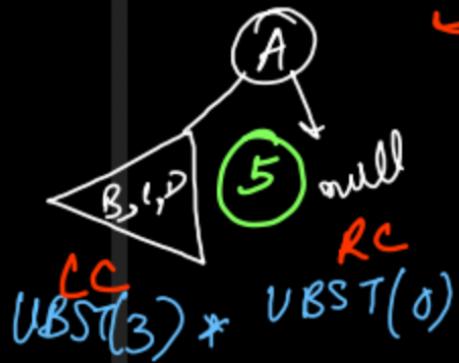


Unique BST(n) = UBST(n) with k_1 as root + k_2 as root
 + k_3 as root + ... + k_n as root

$N=4$

$20 \ 15 \ 10 \ 5$
 $A > B > C > D$

$5 + 2 + 2 + 5 = 14$



② Count Balanced Parentheses

$N \leftarrow$ opening brackets
closing brackets } pairs

$N=0$

" $N=2$ "

$N=3$

①

" $C)C)$ "

②

$N=1$

" $C)$ "

①

$\begin{matrix} (\\) \\ ①\hat{①} \end{matrix}$

$(())$

$()()$

$(())C)$

$()(())$

$()C()$

$(((())$

$(()C())$

⑤

$()$

$(\hat{②})\hat{①}$

$\swarrow \downarrow$

$(\hat{①})\hat{①}$

$(\hat{②})\hat{②}$

$\rightarrow ()C()$

$()C()$

$()\hat{②}$

$(\hat{②})\hat{①}$

$(\hat{①})\hat{①}$

N=4

()

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



$$\begin{array}{ccccccccc} (3)0 & + & (2)1 & + & (1)2 & + & (0)3 & = & 14 \\ \textcircled{5} & & \textcircled{2} & & \textcircled{2} & & \textcircled{5} & & \textcircled{14} \\ (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2})\textcircled{4} & & (\textcircled{1})\textcircled{2}\textcircled{3} & & (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2}\textcircled{3}\textcircled{4}) \\ (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2})\textcircled{4} & & (\textcircled{1})\textcircled{2}\textcircled{3} & & (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2}\textcircled{3}\textcircled{4}) \\ (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2})\textcircled{4} & & (\textcircled{1})\textcircled{2}\textcircled{3} & & (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2}\textcircled{3}\textcircled{4}) \\ (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2})\textcircled{4} & & (\textcircled{1})\textcircled{2}\textcircled{3} & & (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2}\textcircled{3}\textcircled{4}) \\ (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2})\textcircled{4} & & (\textcircled{1})\textcircled{2}\textcircled{3} & & (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2}\textcircled{3}\textcircled{4}) \\ (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2})\textcircled{4} & & (\textcircled{1})\textcircled{2}\textcircled{3} & & (\textcircled{1}\textcircled{2}\textcircled{3}) & & (\textcircled{1}\textcircled{2}\textcircled{3}\textcircled{4}) \end{array}$$

③ Count mountain ranges

n uphills, n downhills \Rightarrow above or on the sea level

uphill \rightarrow (

downhill \rightarrow)



Count

balanced

strings

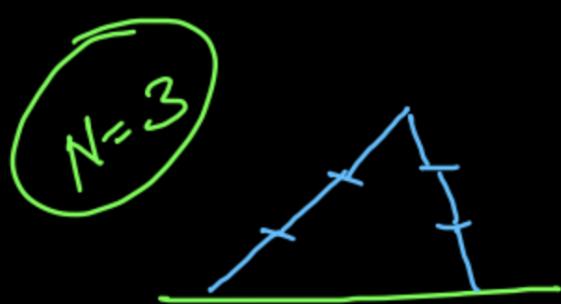
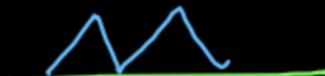


①

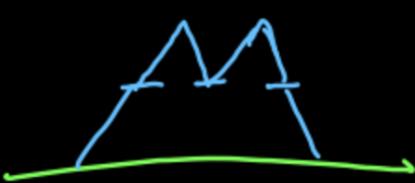
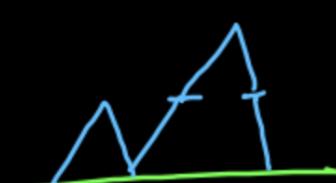


$N=2$

②



$N=3$

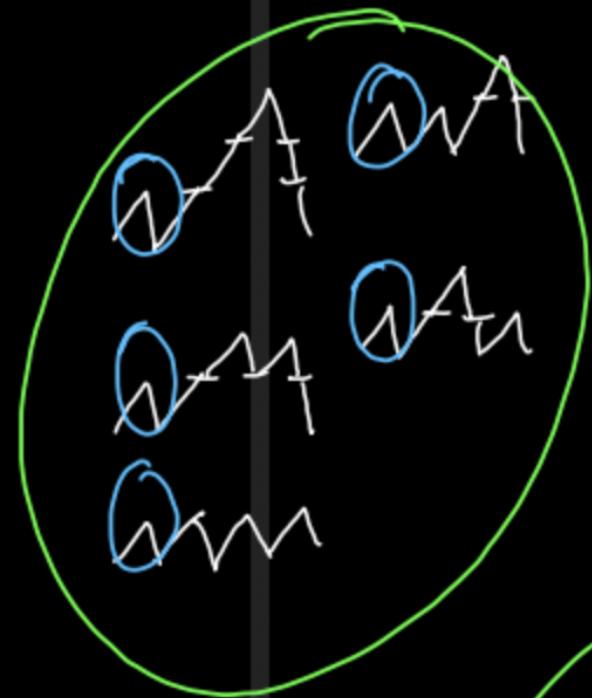


⑤

4 uphills, 4 downhills

$N = 4$

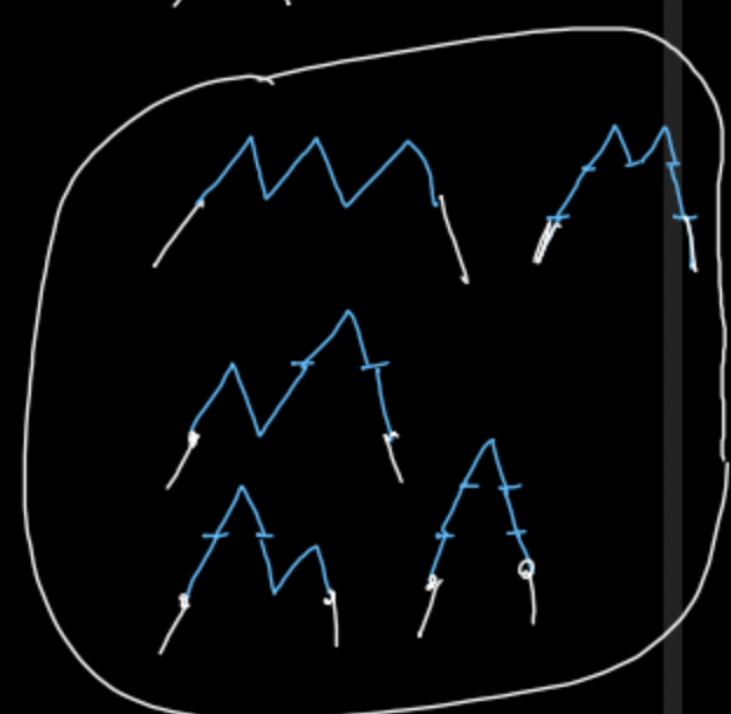
⑥ ③



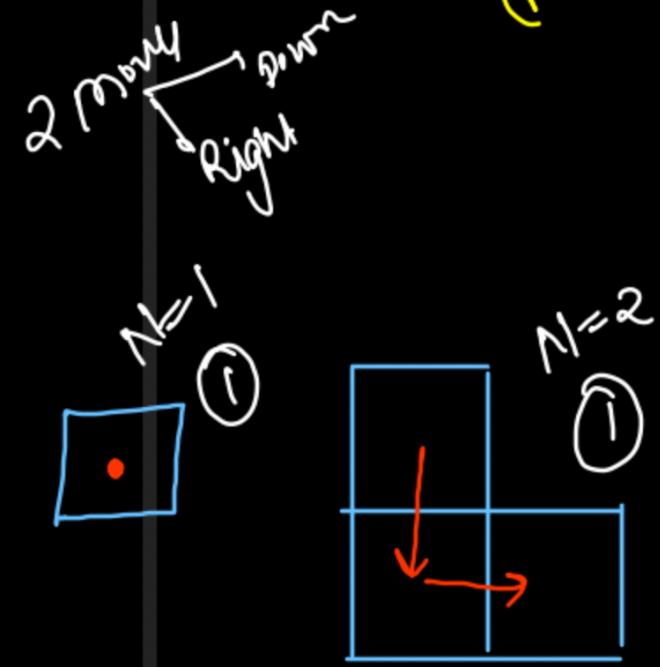
① ②

⑤ ⑦

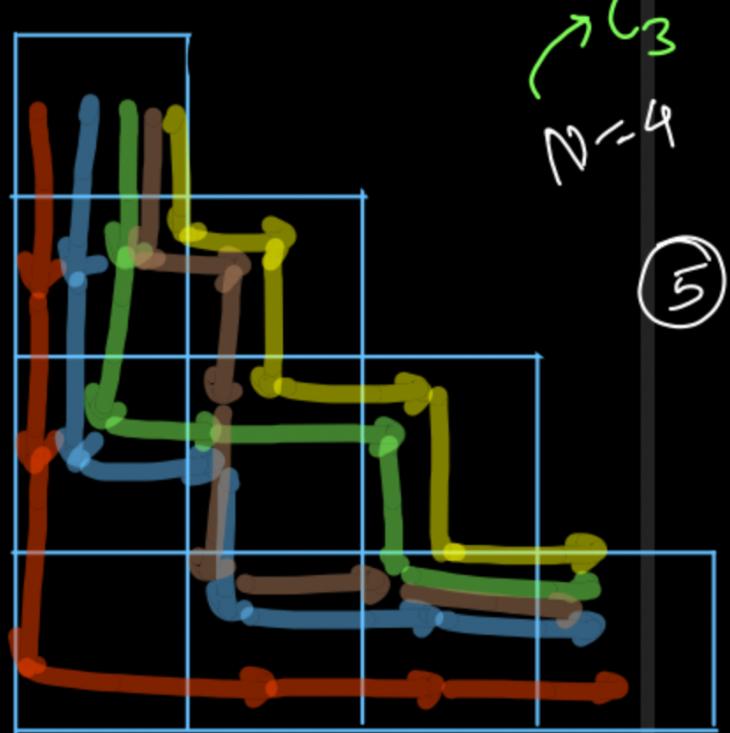
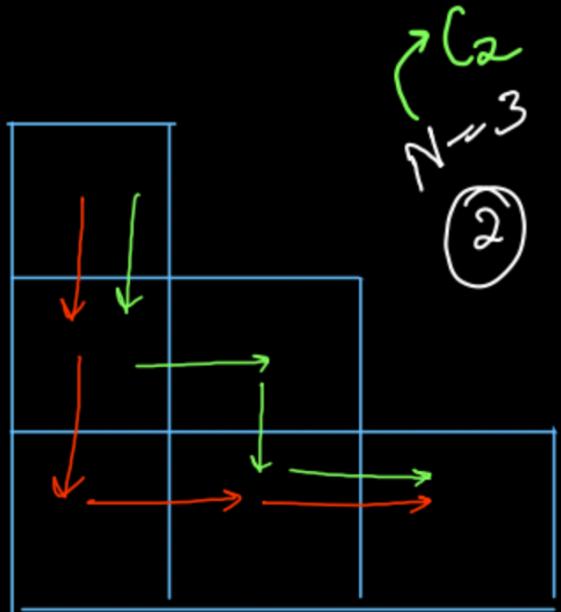
③ ④



If entire matrix
↳ Unique paths
 $\binom{n+m}{n} = \binom{n+m}{m}$



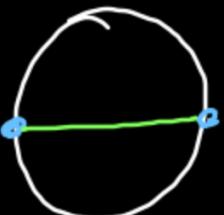
④ Count Paths in Upper-Right Triangle
or Lower-Left Triangle



$(N \text{ rows}, N \text{ cols}) \Rightarrow C_{N-1} \Rightarrow \binom{N-1}{\frac{N-1}{2}} \text{ up hills} + \binom{N-1}{\frac{N-1}{2}} \text{ down hills}$

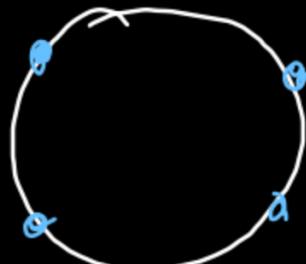
⑤ Handshakes / Non-intersecting chords

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



$$N = 2$$

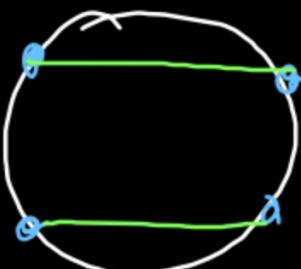
⊕



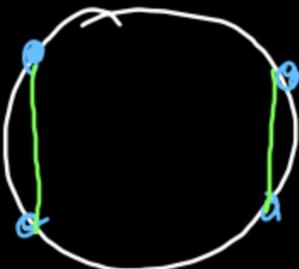
$$N = 4$$

C_0, C_1, C_2, C_3, C_4
 ↓ ↓ ↓ ↓
 1 1 2 5 14

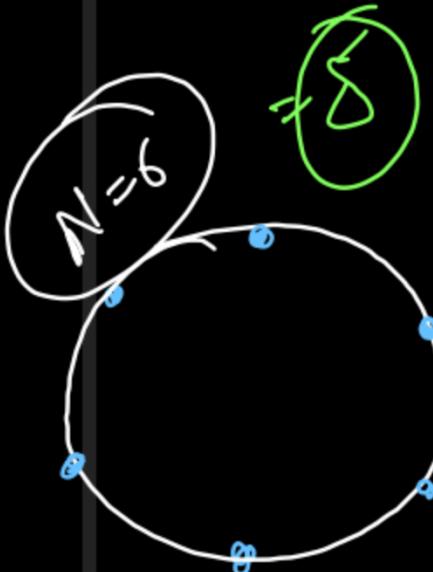
①



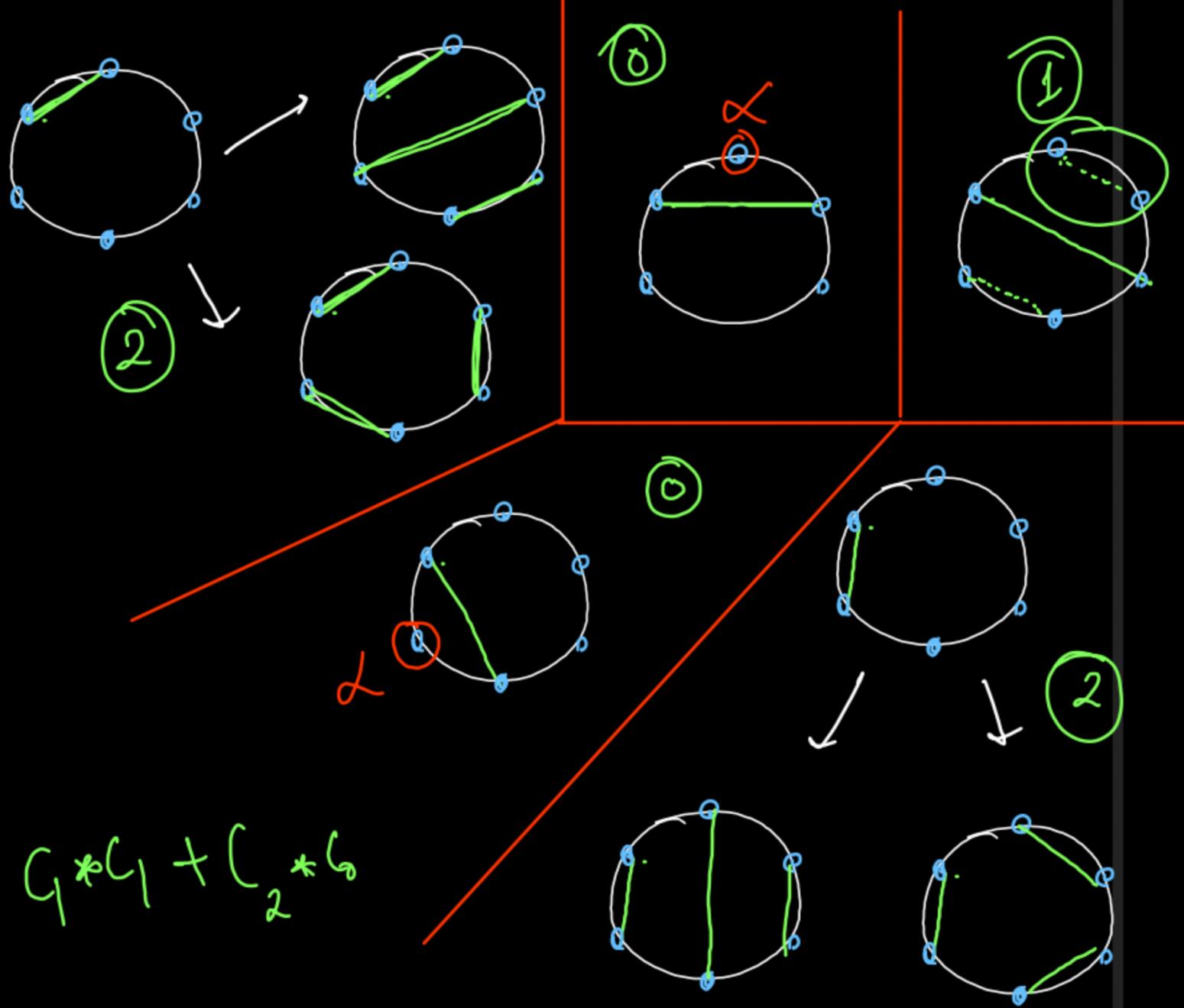
②



$N=0$ (1) C_0
 $N=2$ (1) C_1
 $N=4$ (2) C_2
 $N=6$ (3) C_3
 $N=8$ (4) C_4



$$(N=6) \rightarrow C_3 = C_0 * C_2 + C_1 * C_1 + C_2 * C_0$$





substring starting with 0th index
in all prefix
↳ count of $x \geq$ count of y

$N=2$

XYXY

XXYY

②

$N=3$

xyx y xy

x xyy y xy

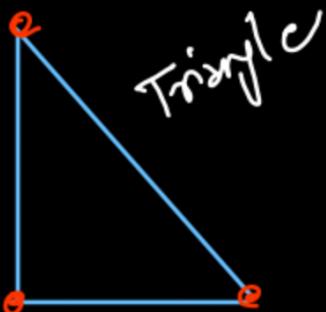
xy xyy y

x x xyy y y
x xy x y y

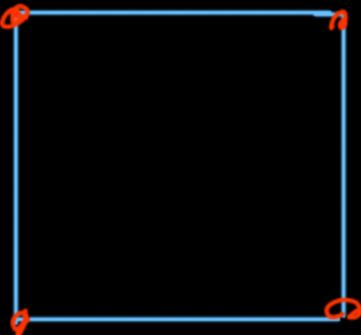
$x = 'C'$ $y = ')'$

↓
count of balanced
parenthesis strings

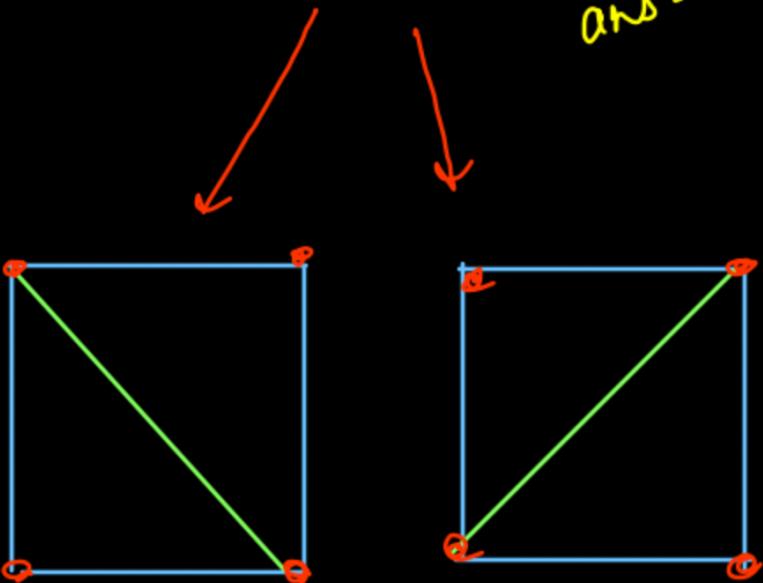
⑧ Count ways of Triangulation



$$N=3 \\ ans = 1$$



$$N=4 \\ ans = 2$$



$$N=2 \\ ans = 1$$

$$N=2 \Rightarrow 1 \rightarrow C_0$$

$$N=3 \Rightarrow 1 \leftarrow C_1$$

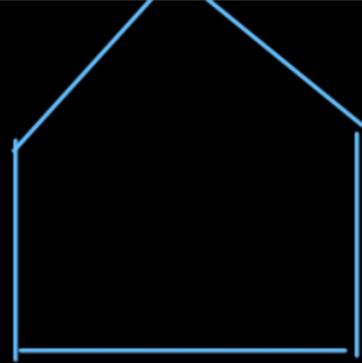
$$N=4 \Rightarrow 2 \leftarrow C_2$$

$$N=5 \Rightarrow 5 \leftarrow C_3$$

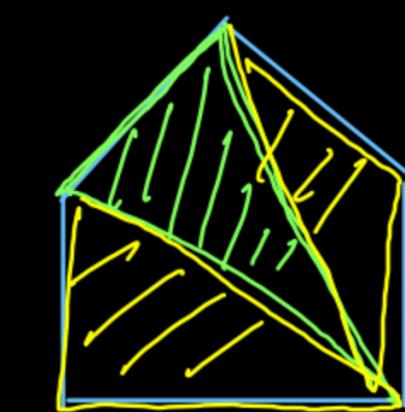
$$N=6 \Rightarrow 14 \leftarrow C_4$$

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

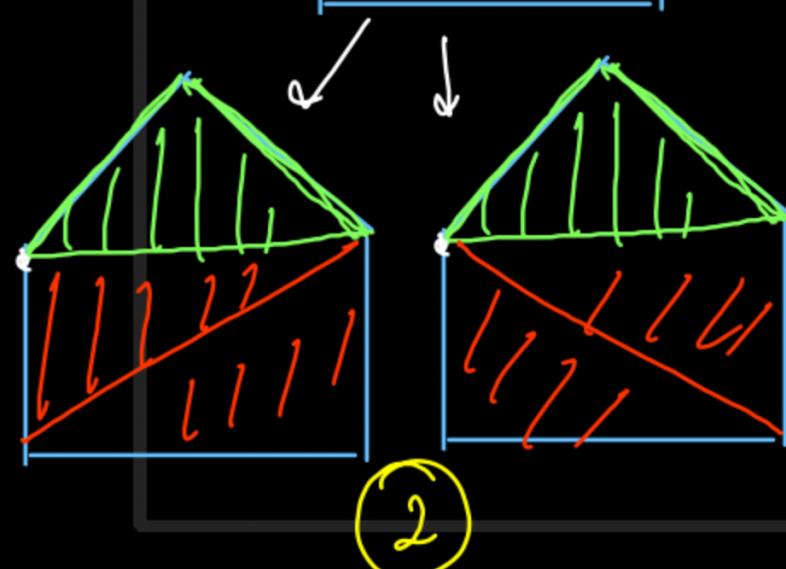
$\text{polygon}(n) = \sum \text{left}(i) * \text{light}(j)$



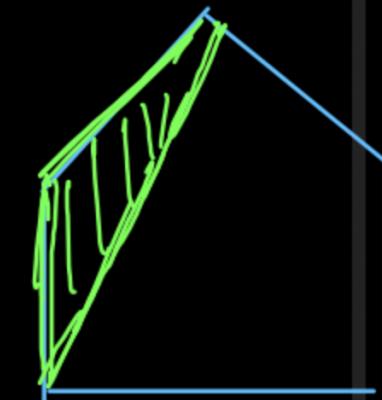
1



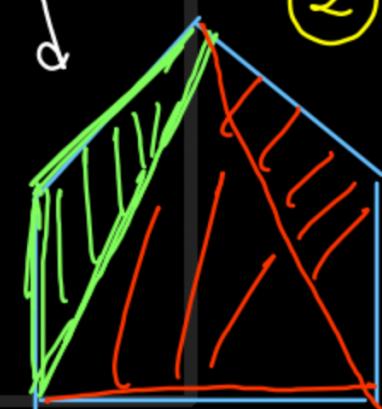
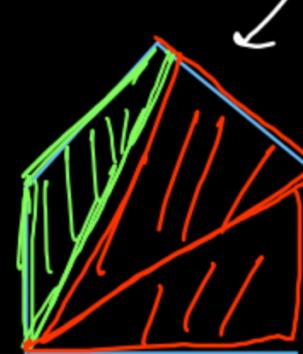
①



②



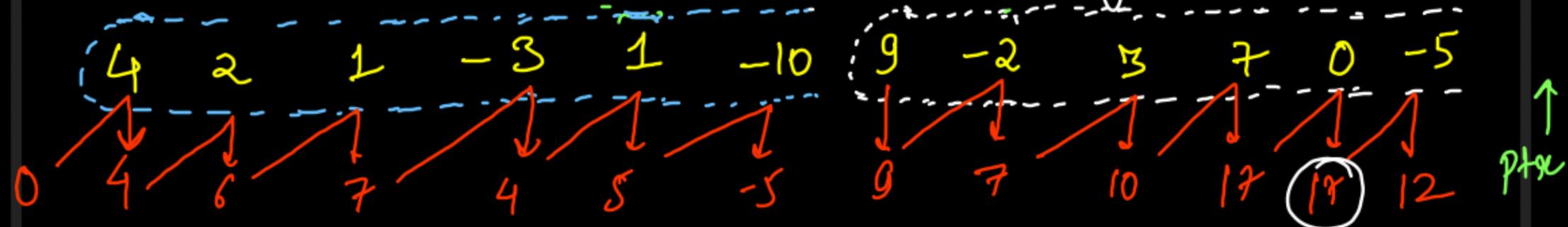
②



Kadane's Algorithm

- Maximum sum Subarray → Kadane (DP)
- Maximum sum Circular Subarray → Preff & Suffix Max & Min
- k 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



Brute force



All subarrays



$O(N^2)$

Kadane's Algorithm

$$\text{maxSum} = -\infty$$

$$\text{currSum} = \emptyset$$

$$4+2=6+1=7-3=4+1$$

$$=5-10=-8$$

$$9-2=$$

$$7+3=10+7$$

$$=18+0=18$$

$$12-5=12$$

Kadane's Algorithm

```
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 $\rightarrow O(N)$

Space $\rightarrow O(1)$

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

f(l, m)

mid

f(m+1, r)

↑
right

{7, 7, 5}

0 1 2 3 4 5
4 2 1 -3 1 -10
↑ m ↑ r

6 7 8 9 10 11
9 -2 3 7 0 -5
↑ m ↑ r

{17, 17, 12}

{7, 3, 7}

↖

↓ {1, -2, -9}

↓ {10, 10, 10}

↓ {2, 7, 2}

{6, 6, 6}

2 1

↓ {1, 2, 3}

↓ {3, 3, 2}

↓ {5, 5, 8}

4 2 1 9

↓ {2, 2, 2}

↓ {-3, -3, -3}

↓ {1, 1, 1}

↓ {2, 2, 2}

↓ {0, 0, 0}

{4, 4, 4}

Return Type

→ manSum Subarray

→ manSum begin

→ manSum suffin

→ totalSum

manSum

$$= \left\{ \begin{array}{l} \text{leftmanSum} \\ \text{OR} \\ \text{rightmanSum} \\ \text{OR} \\ (\text{leftSuffin} + \text{RightPostfix}) \end{array} \right\}$$

PrefixSum

$$= \left\{ \begin{array}{l} \text{leftPrefix} \\ \text{OR} \end{array} \right.$$

leftTotal + rightPrefix

SuffixSum

$$= \left\{ \begin{array}{l} \text{RightSuffix} \\ \text{OR} \end{array} \right.$$

rightTotal + leftSuffin

```
public static class Pair{  
    int maxSum = 0;  
    int prefix = 0;  
    int suffix = 0;  
    int total = 0;  
  
    Pair(int val){  
        maxSum = prefix = suffix = total = val;  
    }  
}
```

```
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;  
}
```

Time $\rightarrow O(n)$

Space $\rightarrow O(\log n)$

$\hookrightarrow R.C.S^*$

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

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

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

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

$$\downarrow$$

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

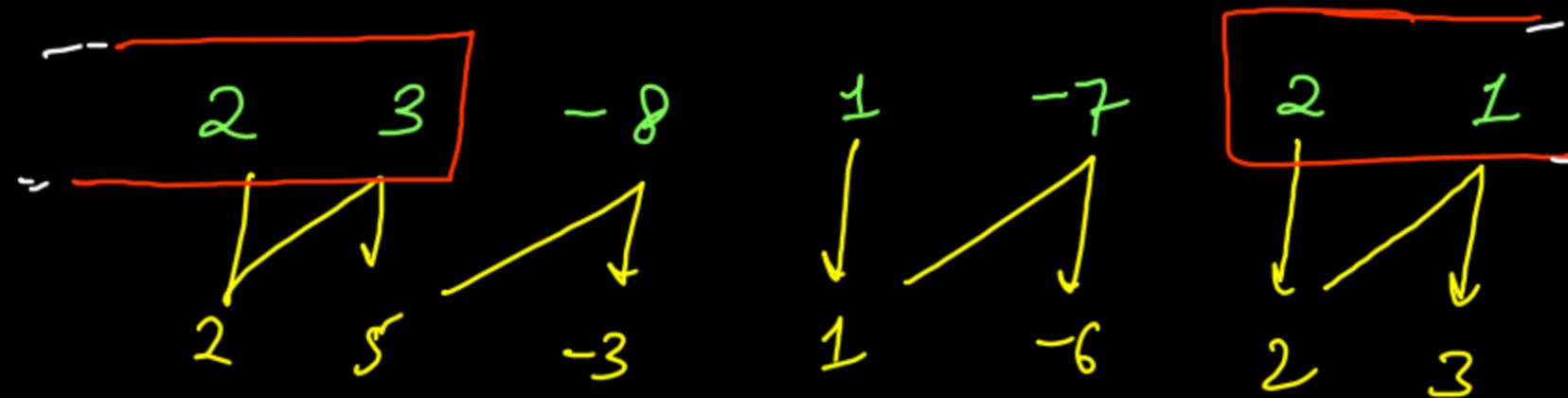
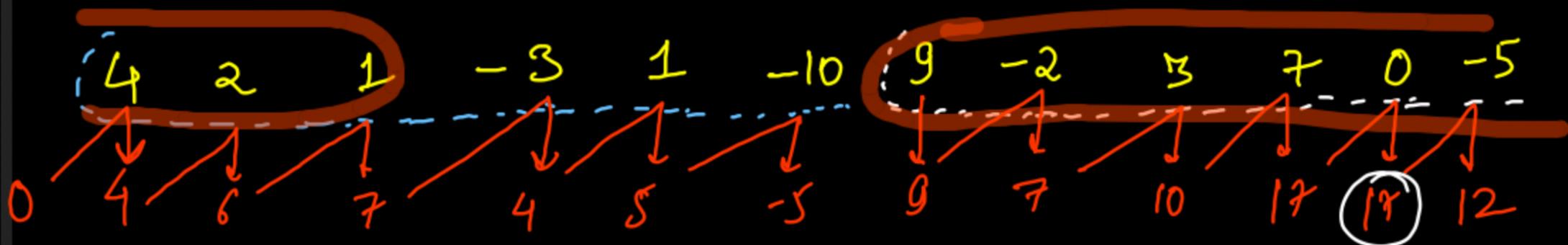
$$T(n) = n + (2 + 2^2 + 2^3 + \dots + 2^{\log_2 n})$$

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

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

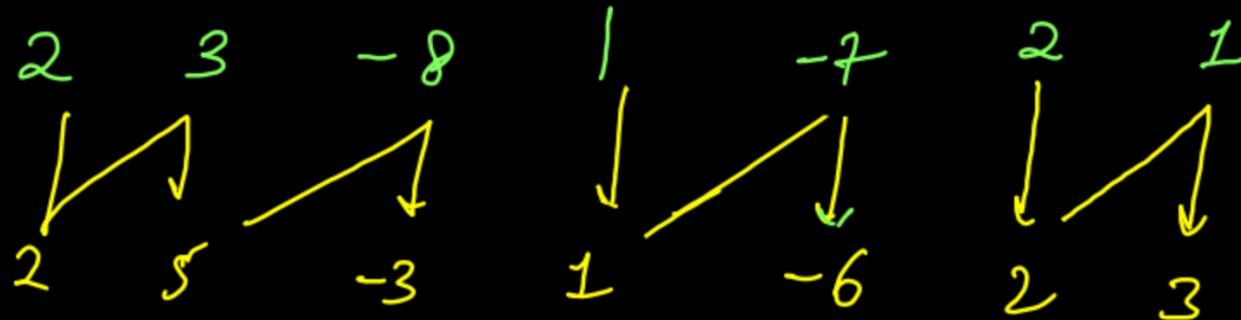
$$= \frac{3n-1}{2} \Rightarrow O(n)$$

Max sum Circular subarray

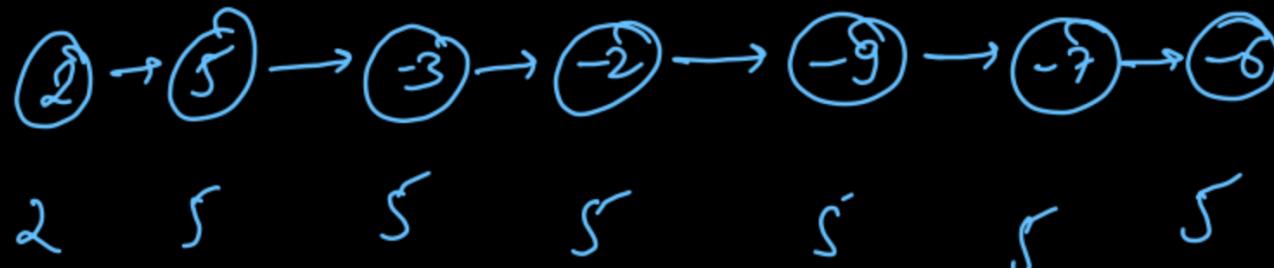


$$2+3+2+1 = 8 \text{ max sum}$$

1D Kadane



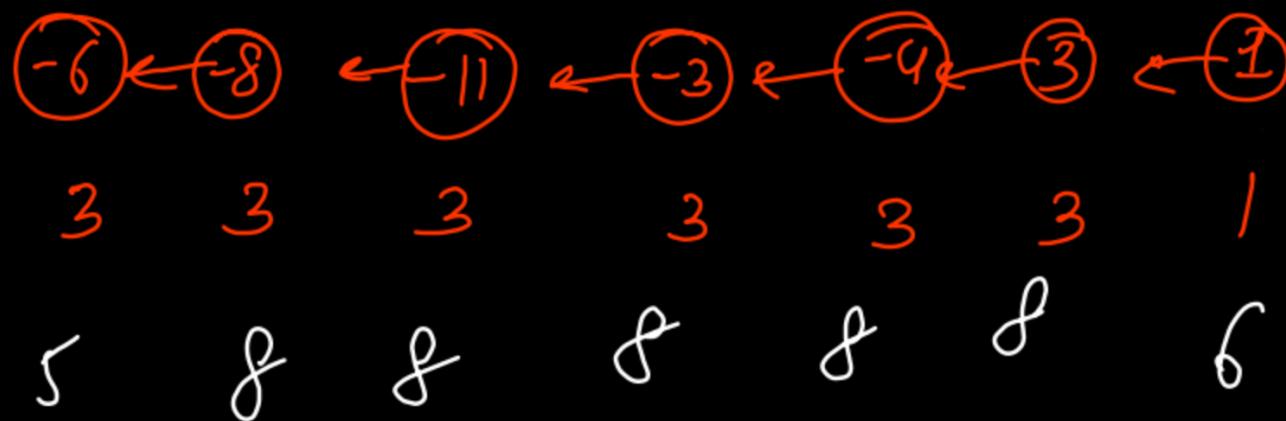
Prefix



prefix⁽ⁱ⁾

max sum prefix
ending at index
 $\leq i$

Suffix



Suffix⁽ⁱ⁾

max sum suffix
starting at index
 $\geq i$

```

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;

```



 Time $\rightarrow O(N)$

 Space $\rightarrow O(N)$

Ans ; Max Sum Subarray OR Total - Min Sum Subarray

Approach 2

Time $\rightarrow O(n)$

Space $\rightarrow O(1)$

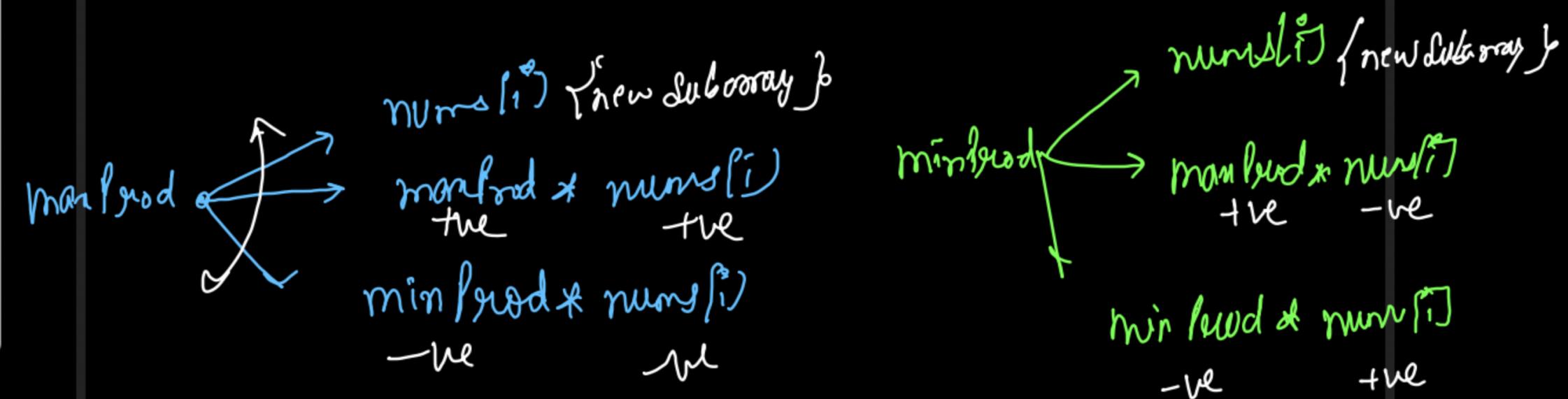
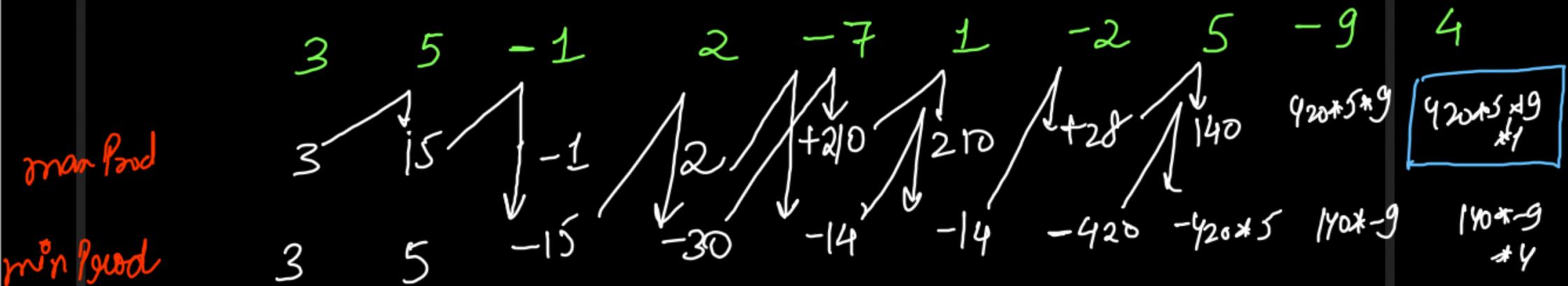
```
public int maxSubarraySumCircular(int[] nums) {  
    int minSum = Integer.MAX_VALUE;  
    int maxSum = Integer.MIN_VALUE;  
    int currMin = 0, currMax = 0;  
    int total = 0;  
  
    for(int i=0; i<nums.length; i++){  
        currMin = Math.min(currMin + nums[i], nums[i]);  
        currMax = 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);  
}
```

Corner Case : All negatives

{ -5, -3, -4, -8 }

totalNum = -20 mindum = -20
maxsum = -3

Max^m Product Subarray



```
ss SOLUTION 1
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 $\rightarrow O(N)$

Space $\rightarrow O(1)$

~~k~~ Concatenation (LC 1191) maximum subarray
of k copies

~~eg 1~~

arr: $\rightarrow 4 \ -2 \ 3 \ 5 \ 1$

$k = 4$

$\text{sum} > 0$

~~eg 3~~

arr: $\rightarrow 5 \ 0 \ 1 \ 2 \ -4$

$k = 4$

~~eg 2~~

arr: $\rightarrow -2 \ 3 \ 0 \ 1 \ 4$

$k = 4$

~~eg 4~~

arr: $\rightarrow -2 \ -3 \ 4 \ 5 \ 3 \ -1 \ -1$

$k = 4$

Bentley force $\rightarrow O(k+n)$ Radane on
 k concatenated array

Entire array is the Answer

4 -2 3 5 1 4 -2 3 5 1 4 -2 3 5 1 4 -2 3 5 1
[] [] [] []

$$\text{dum} = 4 - 2 + 3 + 5 + 1 \\ = (11 * 4) = \textcircled{44}$$

Rejected leading elements

-2 [3 0 1 4 -2 3 0 1 4 -2 3 0 1 4 -2 3 0 1 4]
[] [] [] []

$$\text{dum} = -2 + 3 + 0 + 1 + 4 \\ (6 * 4) + 2 = 26$$

Rejecting some elements from last copy

5 0 1 2 -4

5 0 1 2 -4 5 0 1 2 -4 5 0 1 2 -4

$$\text{dsum} = 5 + 0 + 1 + 2 - 4$$

$$= +4 * 4 - (-4) = 16 + 4 = 20$$

Rejecting some ele from 1st copy & some elements from last copy

-2 -3

4 5 3 -1 -1 -2 -3 4 5 3 -1 -1 -2 -3 4 5 3 -1 -1

Suff.

$$\begin{aligned}\text{dsum} &= -2 - 3 + 4 + 5 + 3 - 1 - 1 \\ &= +5\end{aligned}$$

$k=1 \Rightarrow$ Apply Kadane's

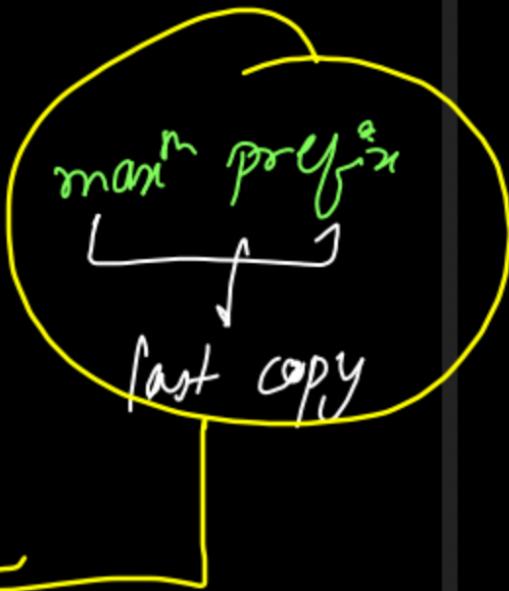
~~Kadane~~

If $\text{sum} > 0$

\Rightarrow



$$\text{total} + (K-2) * \text{sum} + \underbrace{\text{all middle copies}}$$



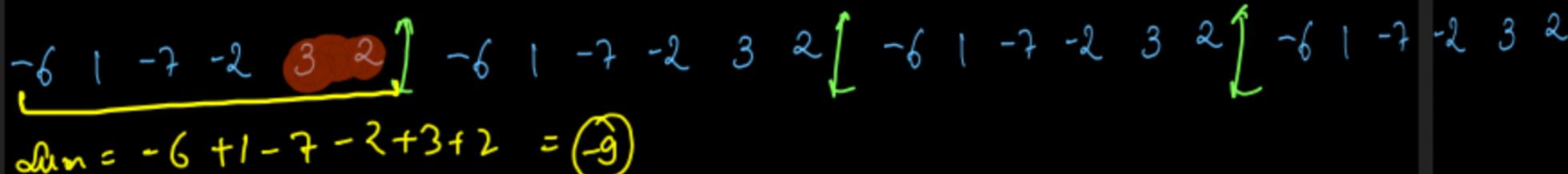
\Rightarrow Kadane on 2 copies

eg5



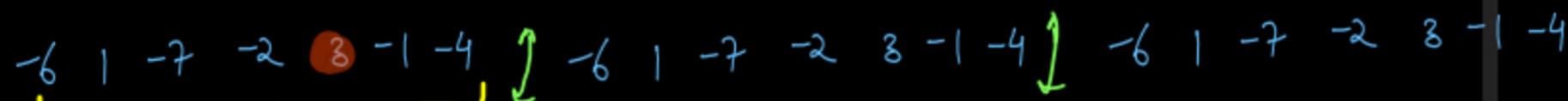
$$\text{sum} = 2 + 3 - 6 + 1 - 7 - 2 = (-9)$$

eg6



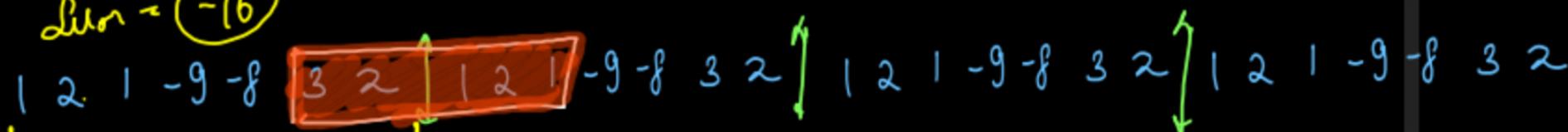
$$\text{sum} = -6 + 1 - 7 - 2 + 3 + 2 = (-9)$$

eg7



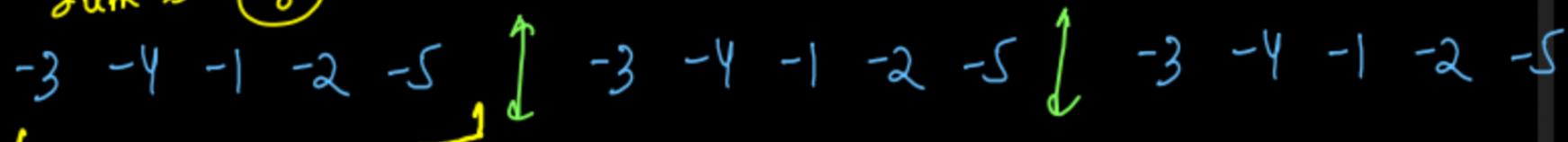
$$\text{sum} = (-16)$$

eg8



$$\text{sum} \sim (-8)$$

eg9



$$\text{sum} = (-15)$$

$\hookrightarrow \text{maxSum} = 0 \{ \text{emptySubarray} \}$

~~eg10
sum < 0~~

Kadane of only 2 copies \Rightarrow Kadane of K copies

```
public int kadane(int[] arr){  
    int currSum = 0, maxSum = 0;  
    for(int i=0; i< arr.length; i++){  
        currSum = Math.max((currSum + arr[i]) % m, arr[i]);  
        maxSum = Math.max(maxSum, currSum);  
    }  
    return maxSum;  
}
```

```
public int kConcatenationMaxSum(int[] arr, int k) {  
    if(k == 1){  
        return kadane(arr);  
    }  
  
    int[] twice = new int[arr.length * 2];  
    int total = 0;  
  
    for(int i=0; i<arr.length; i++){  
        total = add(total, arr[i]);  
        twice[i] = twice[i + arr.length] = arr[i];  
    }  
  
    int ans = kadane(twice);  
    if(total < 0) return ans;  
  
    long res = (ans + (total * (k - 2l)) % m) % m;  
    return (int)res;  
}
```



```

public int kadane(int[] arr){
    int currSum = 0, maxSum = Integer.MIN_VALUE;
    for(int i=0; i< arr.length; i++){
        currSum = Math.max(currSum + arr[i], arr[i]);
        maxSum = Math.max(maxSum, currSum);
    }
    return maxSum;
}

```

```

int maximumSumRectangle(int R, int C, int mat[][]){
    int maxSum = Integer.MIN_VALUE;
    for(int i=0; i<R; i++){
        int[] pref = new int[C];
        for(int j=i; j<R; j++){
            for(int c=0; c<C; c++){
                pref[c] += mat[j][c];
            }
            maxSum = Math.max(maxSum, kadane(pref));
        }
    }
    return maxSum;
}

```

2	3	1
-4	1	0
3	1	-4

1	2	3
-4	1	0
3	1	-4

$\text{Row}(i) = 0$

$\text{Row}(j) = 0$
 $\downarrow 2, 3, 1$

$\text{Row}(j) = 1$
 $\downarrow -4, 1, 0$

$\text{Row}(j) = 2$
 $\downarrow 1, 1, -4$

$\text{Row}(i) = 1$

$\text{Row}(j) = 1$
 $\downarrow -4, 1, 0$

$\text{Row}(j) = 2$
 $\downarrow -1, 2, -4$

$\text{Row}(i) = 2$

$\text{Row}(j) = 2$
 $\downarrow 3, 1, -4$

$\downarrow -1, 2, -4$

DP with Game Theory / ~~Minimax Algo~~

→ Optimal Game Strategy

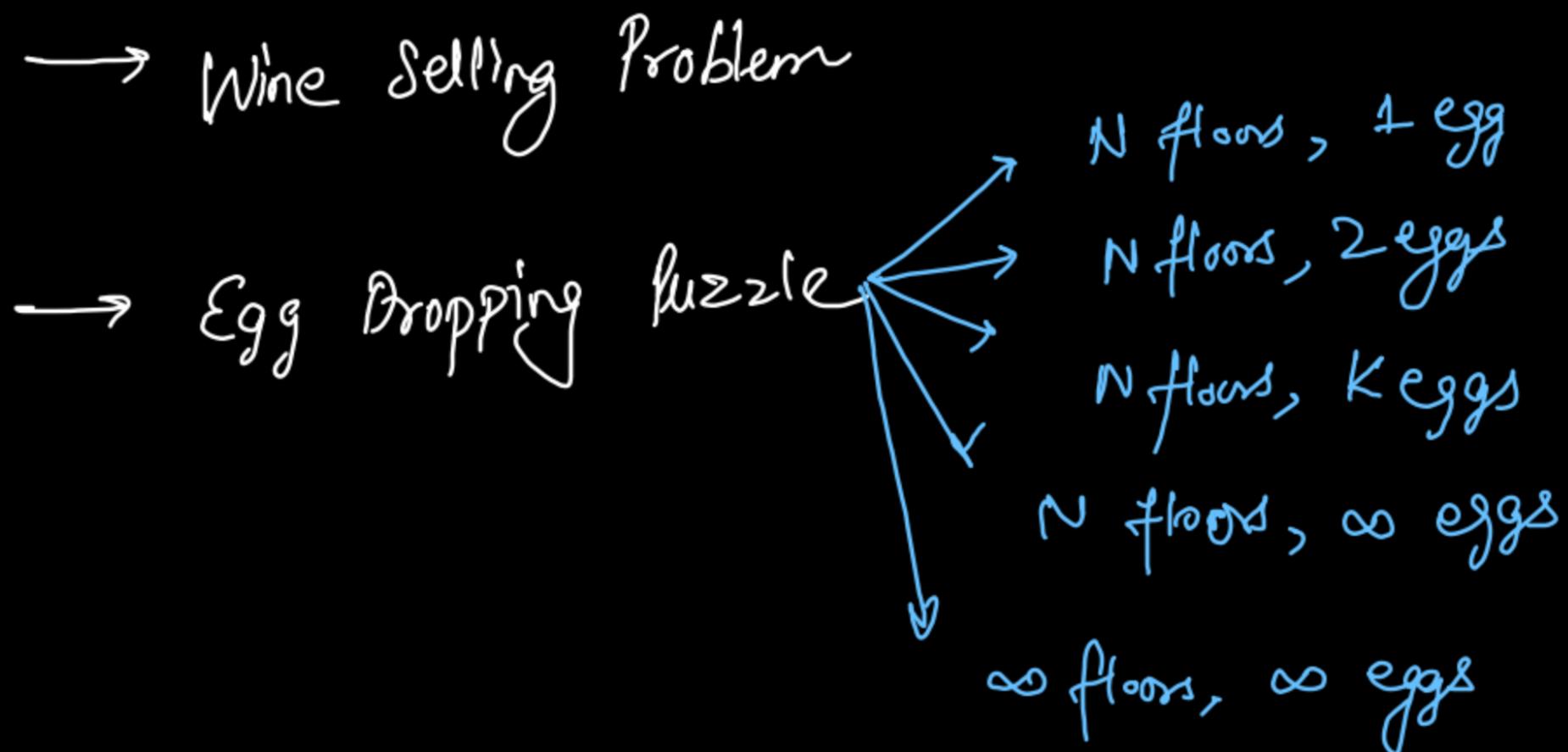
→ Wine Selling Problem

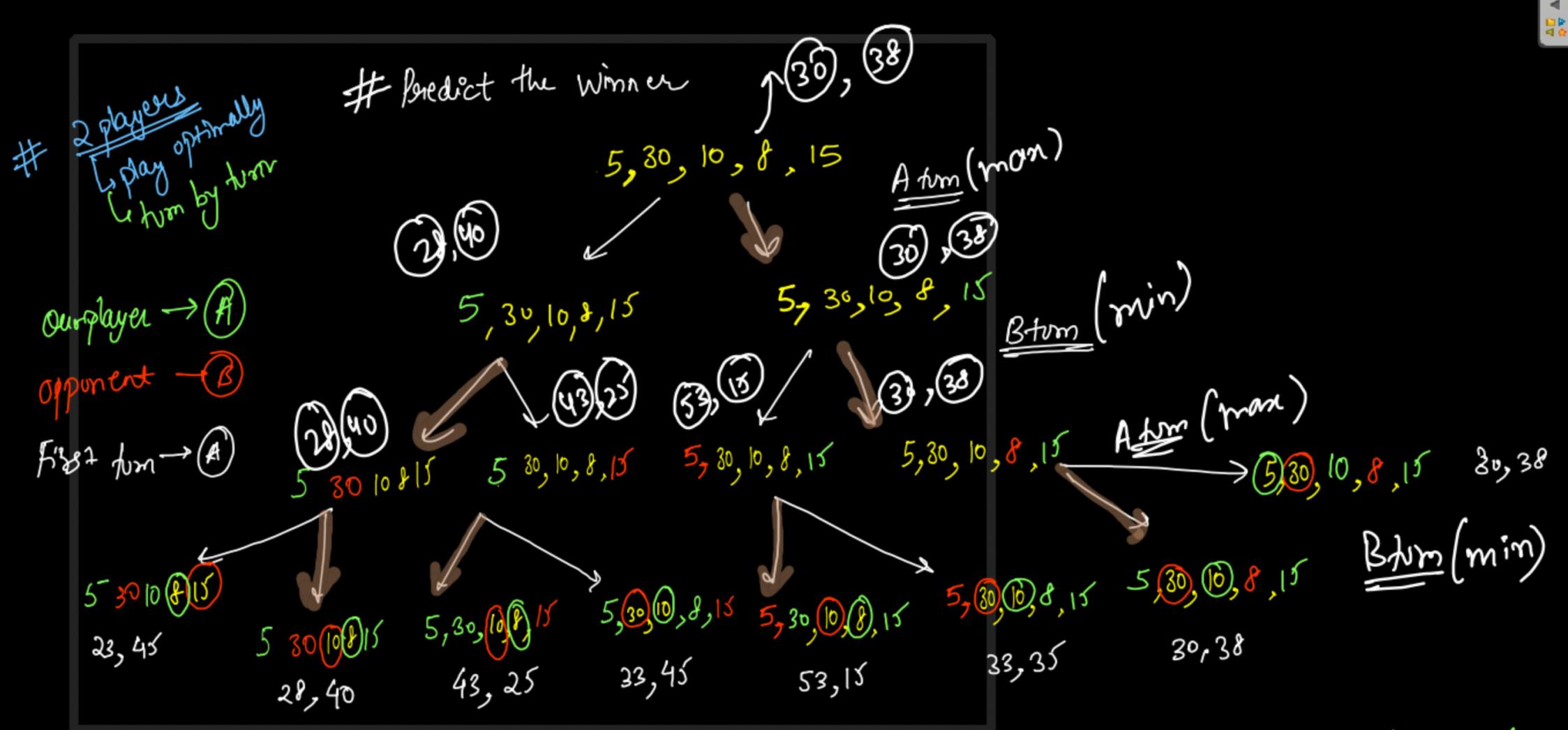
→ Egg Dropping puzzle

Predict Winner (485)

Stone Game (877)

GFG → manasum





your choice → best answer for yourself , Opponent choice → worst answer for you



$\text{left} = 0, \text{right} = n-1$

game(left, right)

You

$\text{arr}[left] + \text{game}(left+1, right)$

OPP

$\text{arr}[right] + \text{game}(left, right-1)$

OPP

$\min \left(\text{arr}[left+1] + \text{game}(left+2, right),$

$\text{arr}[right]$

$+ \text{game}(left+1, right-1)$

$\min \left(\text{arr}[left+1] +$

$\text{game}(left+1, right-1)$

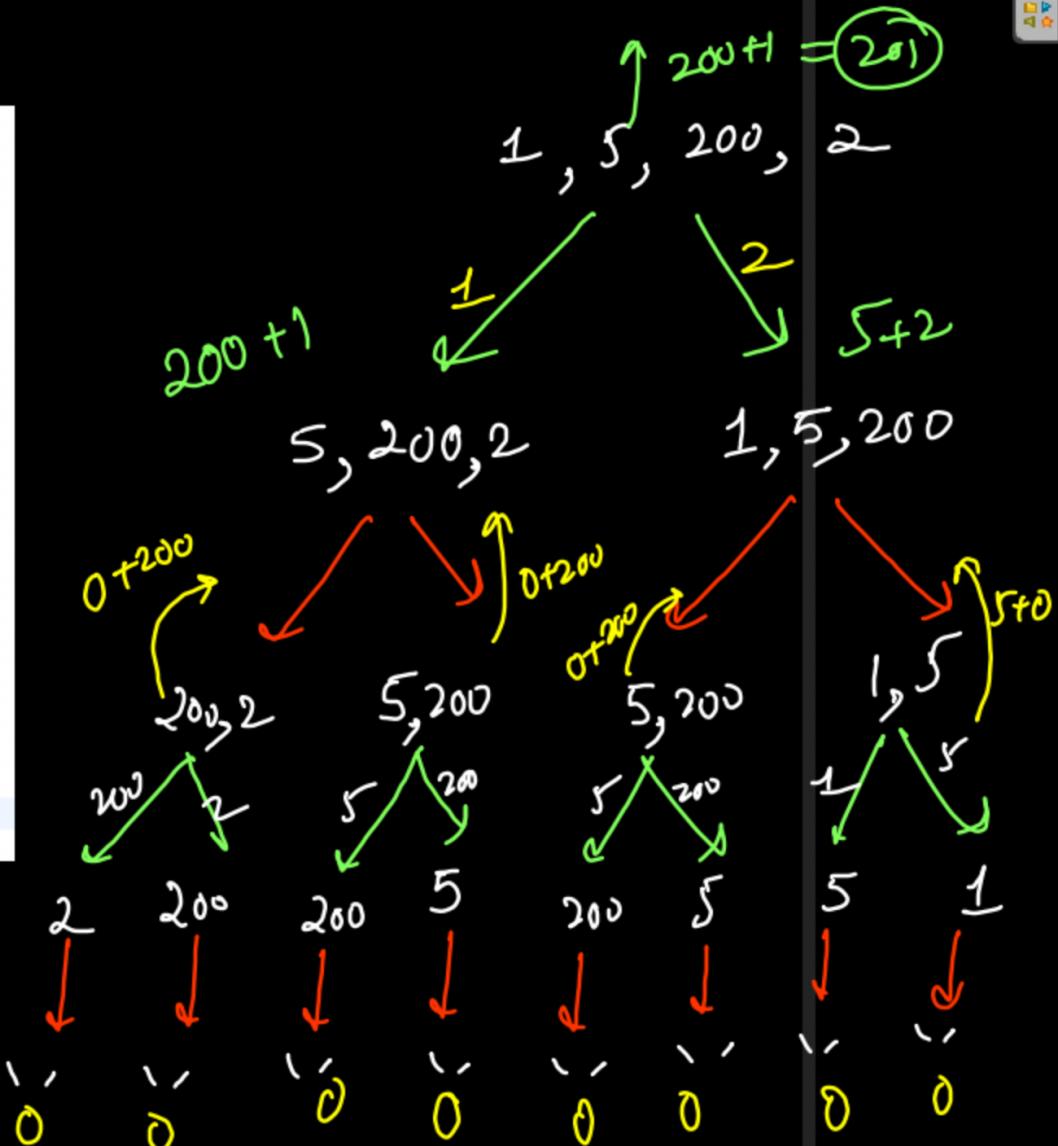
$\text{arr}[right-1]$

+

$\text{game}(left, right-2)$

overlapping

```
public int helper(int left, int right, int[] nums){  
    if(left > right) return 0;  
    if(left == right) return nums[left];  
  
    int c1 = helper(left + 2, right, nums);  
    int c2 = helper(left + 1, right - 1, nums);  
    int c3 = helper(left, right - 2, nums);  
  
    return Math.max(nums[left] + Math.min(c1, c2), nums[right] + Math.min(c2, c3));  
}  
  
public boolean PredictTheWinner(int[] nums) {  
    int ascore = helper(0, nums.length - 1, nums);  
  
    int total = 0;  
    for(int i=0; i<nums.length; i++){  
        total = total + nums[i];  
    }  
  
    int bscore = total - ascore;  
  
    if(ascore >= bscore) return true;  
    return false;  
}
```



~~Approach~~

```

public int helper(int left, int right, int[] nums, int[][] dp){
    if(left > right) return 0;
    if(left == right) return nums[left];
    if(dp[left][right] != -1) return dp[left][right];

    int c1 = helper(left + 2, right, nums, dp);
    int c2 = helper(left + 1, right - 1, nums, dp);
    int c3 = helper(left, right - 2, nums, dp);

    return dp[left][right] = Math.max(nums[left] + Math.min(c1, c2),
                                      nums[right] + Math.min(c2, c3));
}

```

```

public boolean PredictTheWinner(int[] nums) {
    int[][] dp = new int[nums.length + 1][nums.length + 1];
    for(int i=0; i<dp.length; i++)
        for(int j=0; j<dp[0].length; j++)
            dp[i][j] = -1;

    int ascore = helper(0, nums.length - 1, nums, dp);

    int total = 0;
    for(int i=0; i<nums.length; i++){
        total = total + nums[i];
    }

    int bscore = total - ascore;

    if(ascore >= bscore) return true;
    return false;
}

```

Polynomial (memoizat)

Time $\rightarrow O(n^2)$

Space $\rightarrow O(n^2)$

2D DP

~~Approach 2~~

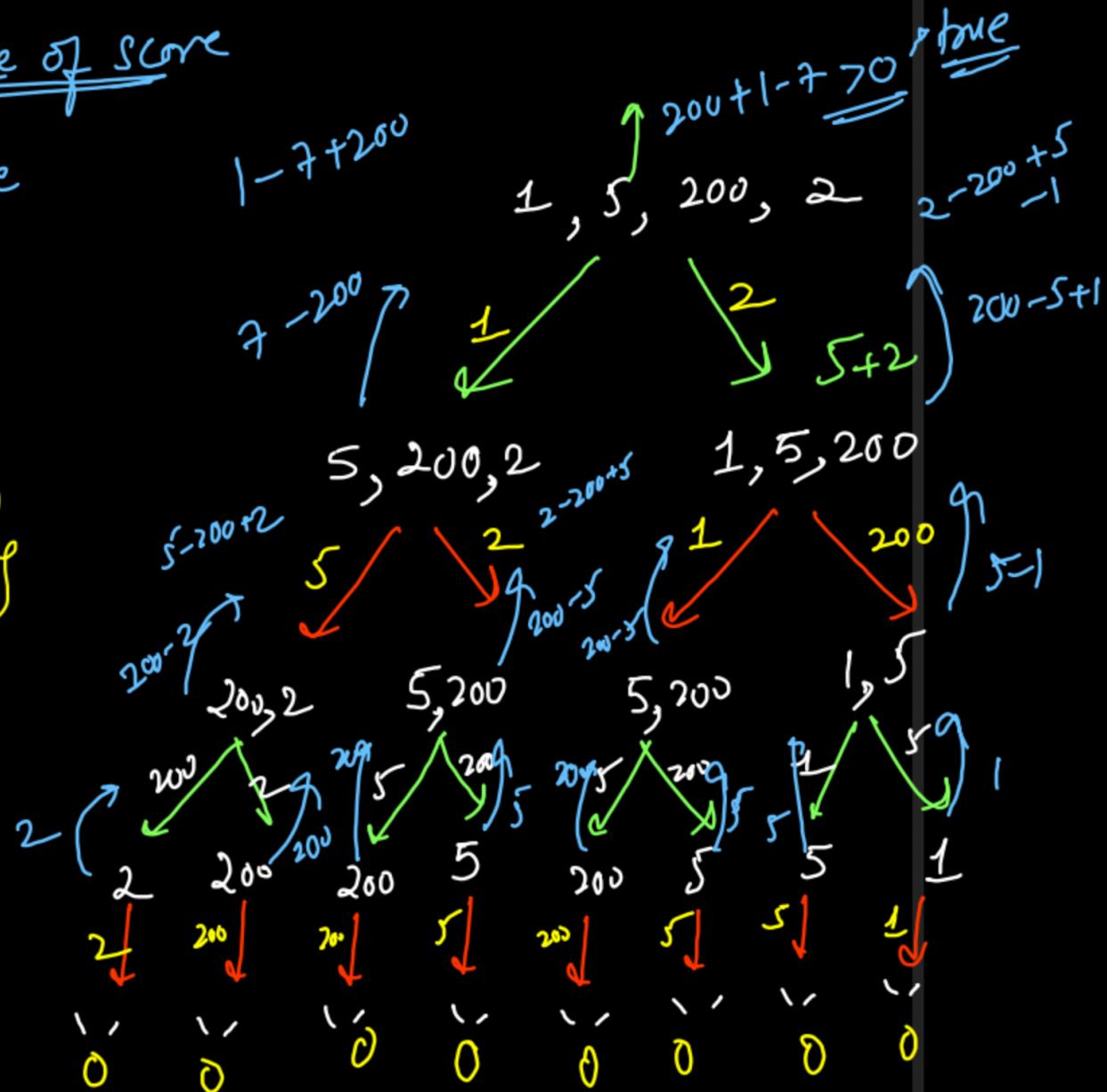
$\textcircled{A} \rightarrow \textcircled{B}$ return difference of score

$\textcircled{A} \geq \textcircled{B} \rightarrow \textcircled{A} - \textcircled{B} \geq 0 \Rightarrow \text{true}$

$\textcircled{B} > \textcircled{A} \rightarrow \textcircled{A} - \textcircled{B} < 0 \Rightarrow \text{false}$

$$\max \left\{ \begin{array}{l} \text{num}_{-}dp(l+1, r), \quad \text{num}_{\text{right}} \\ -dp(l, r-1) \end{array} \right\}$$

$$dp(l)[r] \hookrightarrow \max \left\{ n[l] - (\textcircled{B} - \textcircled{A}), \quad n[r] - (\textcircled{B} - \textcircled{A}) \right\}$$



~~Approach 2~~

```
public int helper(int left, int right, int[] nums, int[][] dp){  
    if(left > right) return 0;  
    if(left == right) return nums[left];  
    if(dp[left][right] != -1) return dp[left][right];  
  
    return dp[left][right] = Math.max(nums[left] - helper(left + 1, right, nums, dp),  
                                      nums[right] - helper(left, right - 1, nums, dp));  
}  
  
public boolean PredictTheWinner(int[] nums) {  
    int[][] dp = new int[nums.length + 1][nums.length + 1];  
    for(int i=0; i<dp.length; i++)  
        for(int j=0; j<dp[0].length; j++)  
            dp[i][j] = -1;  
  
    int diff = helper(0, nums.length - 1, nums, dp);  
    if(diff >= 0) return true;  
    return false;  
}
```

Time $\rightarrow O(N^2)$, Space $\rightarrow O(N^2)$

Optimal Game Theory - II {Stone Game}

Extra Constraints

- ① Even size Array
- ② There will be a definite winner

Answer : → Always  will be answer
Stone