

Recursion - level-1 Revision

- Print Increasing { 1, 2, 3, 4 } → Faith → Postorder
- Print Decreasing { 4, 3, 2, 1 } → Postorder → Faith
- Print Increasing Decreasing { 5, 4, 3, 2, 1, 1, 2, 3, 4 } → Faith → HN

① Expectation : $\rightarrow \text{PI}(N) : \rightarrow 1, 2, \dots, N$

Faith $\rightarrow \text{PI}(N-1) : \rightarrow 1, 2, 3, \dots, N-1$

Meeting expectation : $\rightarrow \text{Sys0}(N)$

Preorder

$\text{Sys0}(N)$

Postorder

$\text{PI}(N-1)$

$$U^N + \{ k^k \} \uparrow N \\ = O(N)$$

Preorder

Sys₀(n)

PD(n-1)

Postorder

PD(n-1)

Sys₀(n)

```
public class Main {
    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        printIncreasing(n);
    }

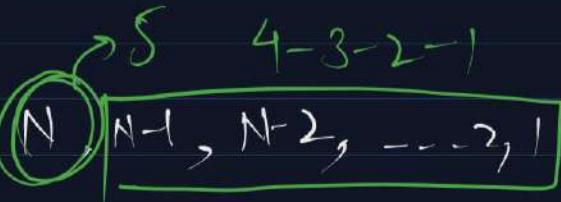
    public static void printIncreasing(int n){
        if(n == 0) return; // Base Case

        printIncreasing(n - 1); // Faith
        System.out.println(n); // Meeting Expectation with Faith
    }
}
```

Call → 1
height → N

$$(1)^N + \{0+k\}^N \Rightarrow \underline{\underline{O(N)}}$$

② Expectation: → PD(n) :→



Faith: → PD(n-1) :→ N-1, N-2, ... , 1

Meeting Expectation: → Preorder: Sys₀(n)

$$\left\{ \begin{array}{l} \text{calls} = 1, \text{height} = N \\ (1)^N + \{k+0\}^N \\ = \underline{\underline{O(N)}} \end{array} \right.$$

```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    printDecreasing(n);
}

public static void printDecreasing(int n){
    if(n == 0) return;
    System.out.println(n);
    printDecreasing(n - 1);
}
```

Power Function

E

Expectation $\mapsto x^n \xleftarrow{\text{pow}(x,n)}$

$$x = 2.0, n = 5 \therefore 2^5 = 32$$

Faith $\rightarrow x^{n-1} * x$ $\xleftarrow{\text{pow}(x,n-1)}$ Meeting expectation

Generic Time Complexity of Recursion

\rightarrow $(\text{Calls})^{\text{height}} + \{\text{Preorder} + \text{Postorder}\} * \text{height}$

Generic Time Complexity of Recursion

$$\rightarrow \boxed{(\text{Calls})^{\text{height}} + \{\text{Preorder} + \text{Postorder}\} * \text{height}}$$

```

public double power(double x, int n){
    if(n == 0) return 1.0;
    double pxn1 = power(x, n - 1); // Faith
    return pxn1 * x; // Meeting Expectation
}

public double myPow(double x, int n) {
    if(x == 0) return 0.0;
    if(x == 1) return 1.0;

    if(n < 0){
        return 1.0 / power(x, -n);
    }

    return power(x, n);
}

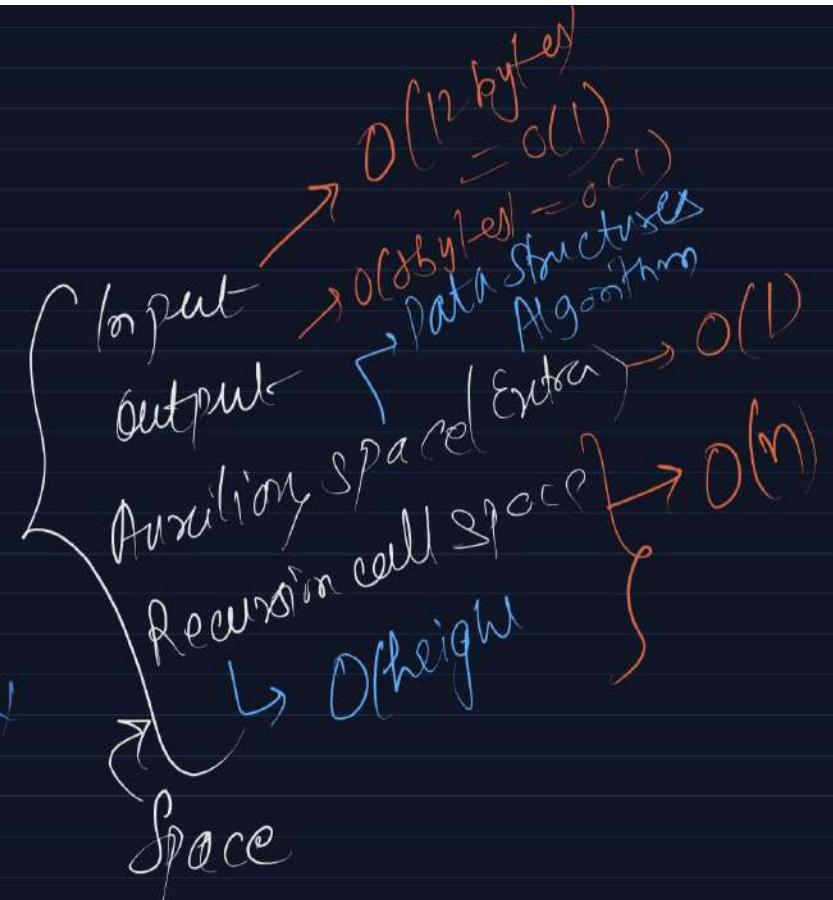
```

Bottom up

$$\begin{aligned} \text{Calls} &= 1 \\ \text{height} &= N \end{aligned}$$

$$1^N + \{k+k\} \times N$$

$$\underline{\underline{O(N)}}$$



Power -> Optimized

$$x^n = x^{n/2} * x^{n/2}$$

~~even~~ $2^6 = 2^3 * 2^3 = 2^{3+3} = 2^6$

~~odd~~ $2^7 = 2^3 * 2^3 * 2 = 2^{6+1} = 2^7$

```
class Solution {
    public double power(double x, int n){
        if(n == 0) return 1.0;

        if(n % 2 == 0)
            return power(x, n/2) * power(x, n/2); // Meeting Expectation
        else
            return power(x, n/2) * power(x, n/2) * x;
    }

    public double myPow(double x, int n) {
        if(x == 0) return 0.0;
        if(x == 1) return 1.0;

        if(n < 0){
            return 1.0 / power(x, -n);
        }

        return power(x, n);
    }
}
```

calls $\Rightarrow 2 \rightarrow$ breadth

height $\Rightarrow \log n$

depth

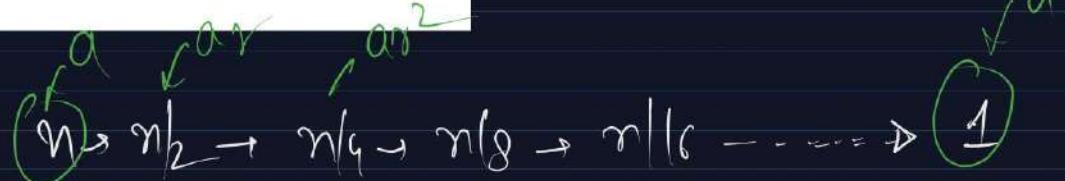
$$\Rightarrow O(2^{\log n}) = O(n)$$

Input $\rightarrow O(1)$

Output $\rightarrow O(1)$

Extra $\rightarrow O(1)$

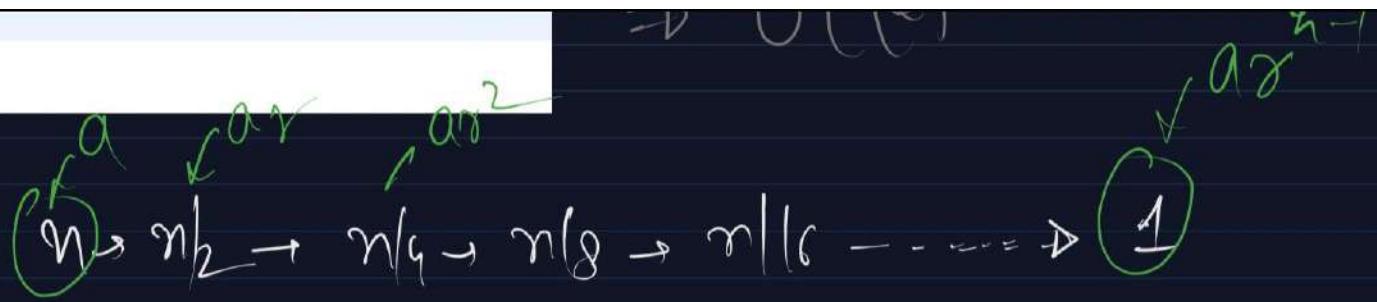
Rec. call stack $\rightarrow O(\log n)$



h terms

```
    return power(x, n);  
}
```

$\rightarrow O(1)$

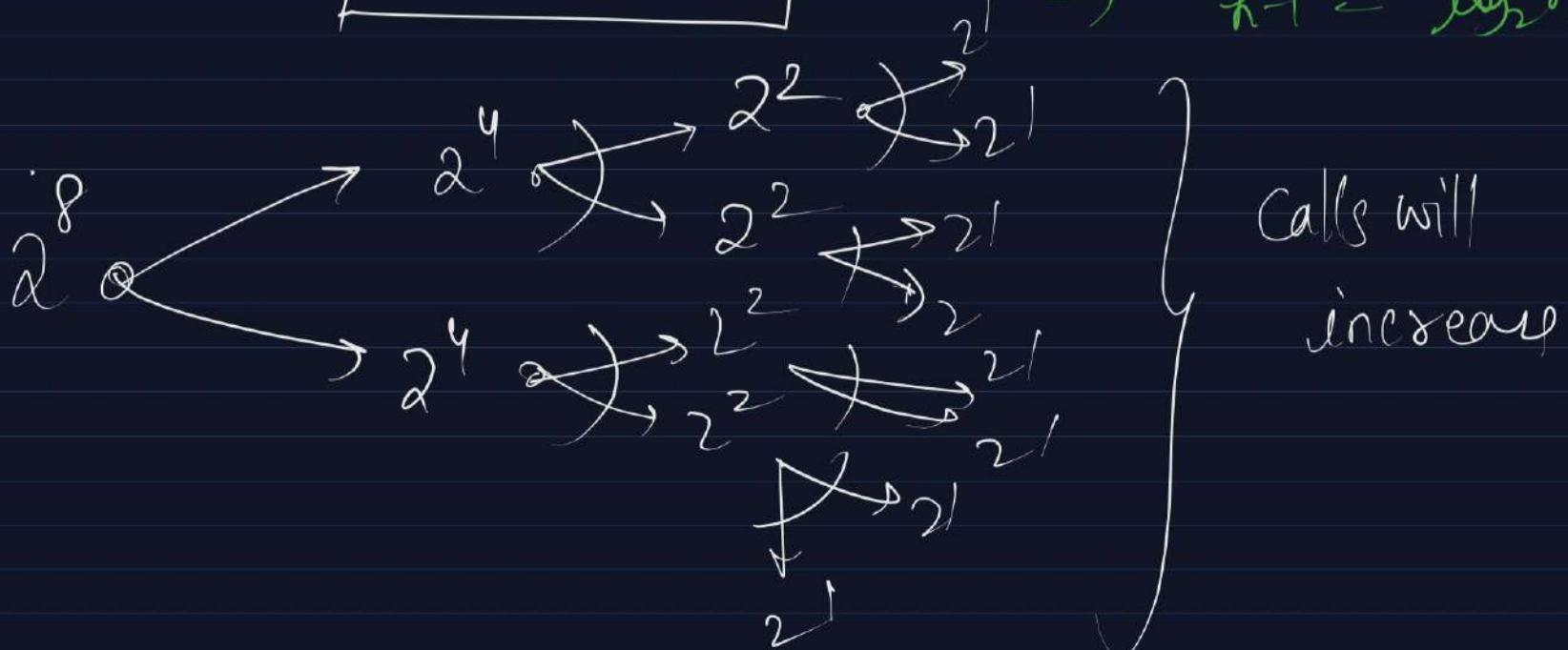


h terms

$$1 = n \times \left(\frac{1}{2}\right)^{h-1} \Rightarrow 2^{h-1} = n$$

$$\boxed{h = O(\log n)} \Rightarrow \log_2(2^{h-1}) = \log_2 n$$

$$\Rightarrow h-1 = \log n$$



```

class Solution {
    public double power(double x, int n){
        if(n == 0) return 1.0;

        double res = power(x, n/2); // Call to S.C
        if(n % 2 == 0)
            return res * res; // Meeting Expectation
        else
            return res * res * x;
    }

    public double myPow(double x, int n) {
        if(x == 0) return 0.0;
        if(x == 1) return 1.0;

        if(n < 0){
            return 1.0 / power(x, -n);
        }

        return power(x, n);
    }
}

```

Same as ~~Power~~ S.C

$x^n \rightarrow x^{n/2} \rightarrow n^{\log_2 n}$

height $\rightarrow h = O(\log_2 n)$

$$O\left(1^{\log_2 n} + k * \log n\right)$$

$$= O(\log_2 n) \approx O(1)$$

$$O(\log_2 n) \ll O(n)$$

$$\begin{aligned} N &= 2^{16} \\ \log_2 n &\approx 16 \end{aligned}$$

Bit Manipulation \rightarrow Modular Exponentiation

\rightarrow TC: $O(\log_2 n)$

\rightarrow SC: $O(1)$

Dynamic Programming

→ "Those who can't remember their past are condemned to repeat it"

① Recursion

- Time Complexity Poor
- Space Complexity Poor

② Memoization

- Time Complexity Good
- Space Complexity Poor

③ Tabulation

- Time Complexity Good
- Space Complexity Good

Topics

Pre-requisites

★ → Dynamic Programming {Level 1 + 2} → Recursion

→ Hashmap & Heap {Level 1 + 2}

→ Graphs {Level 1 + Level 2} → Generic Tree
 {DFS, BFS}

→ Bit Manipulation → No System ↗ Binary
 Decimal

→ Array & String → Remaining Ones

Lecture ① Dynamic Programming {10:30 - 12:00} 19 Apr

→ Fibonacci + Climb Stairs Module

"Those who can't remember their past
are condemned to repeat it"

Exponential

Backtracking

$\sqrt{N} \leq 18$

20

30

TC → L
SC → L

① Recursion { Brute force }

TC → ✓
SC → L

Recursive

② Memoization { Top Down DP }

TC → ✓
SC → ✓

Iterative

③ Tabulation { Bottom Up DP }

④ Space Optimization → limited previous states

Fibonacci Number

0, 1, 1, 2, 3, 5, 8, 13
0th 1st 2nd 3rd 4th 5th 6th 7th

Expectation \rightarrow Nth Fibonacci No. $\{ \text{fib}(n) \}$

Faith \rightarrow $\text{fib}(n-1)$, $\text{fib}(n-2)$

Recurrence

Relation

$$\boxed{\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)}$$

Recurrence Relation

$$f_2(N) = f_2(N-1) + f_2(N-2)$$

worst case

height

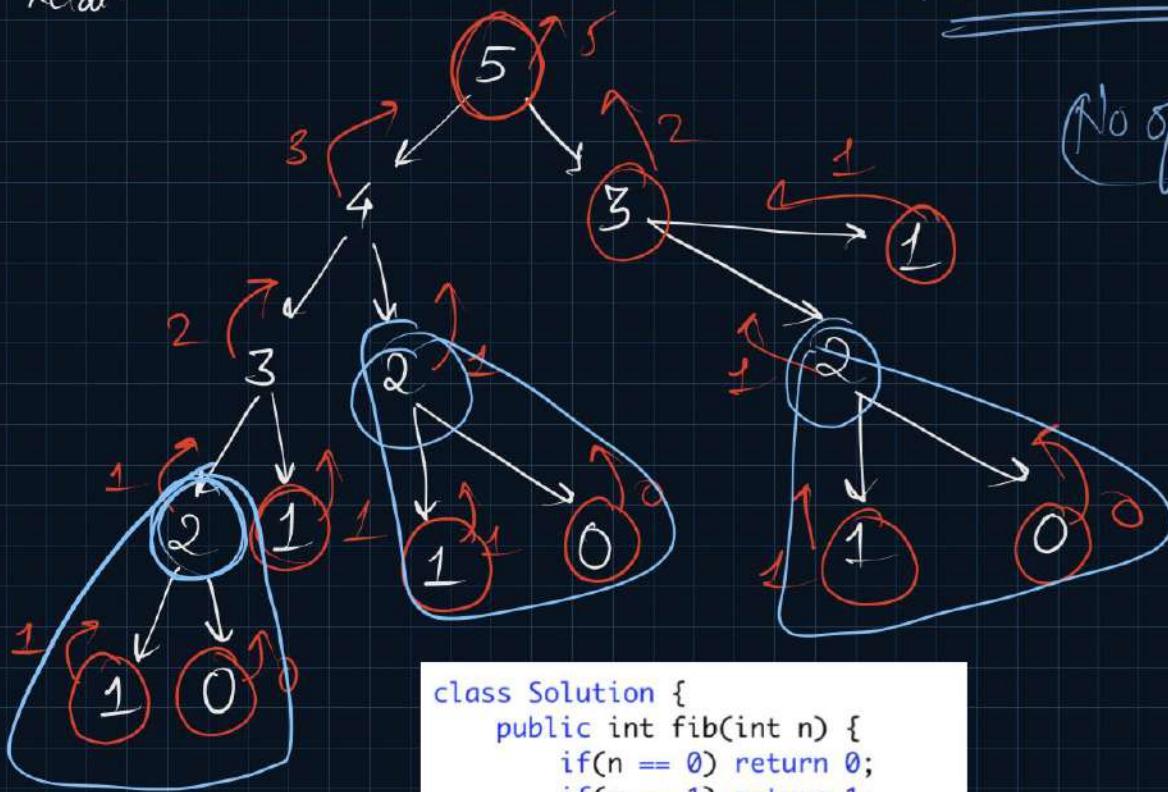
$$(No \ of \ calls) + (pre + post) * height$$

$$(2)^N + (k + k) * N$$

$\Rightarrow O(2^N)$ Time Complexity

Space Complexity $\Rightarrow O(N)$

R.C.S.S

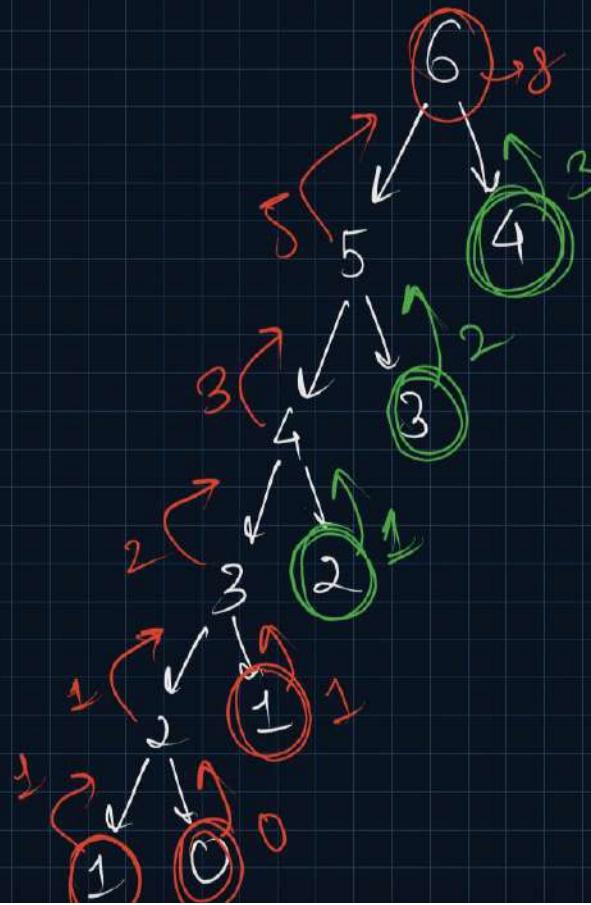
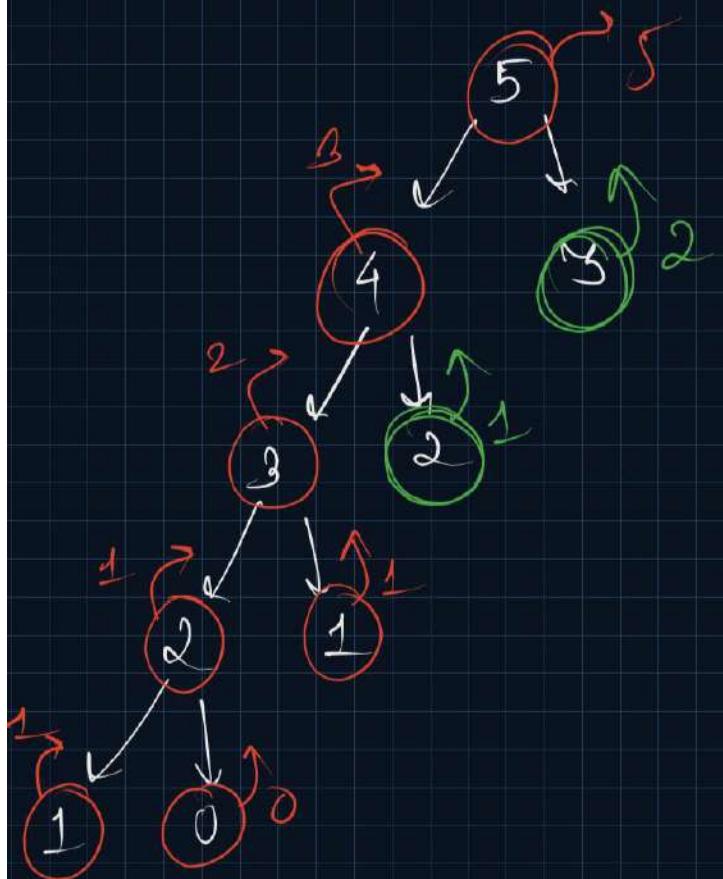
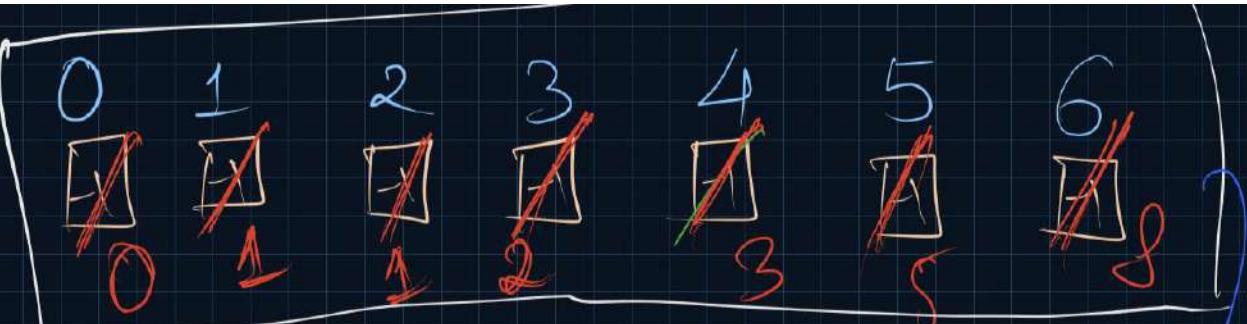


```
class Solution {
    public int fib(int n) {
        if(n == 0) return 0;
        if(n == 1) return 1;

        int prev1 = fib(n - 1);
        int prev2 = fib(n - 2);

        return prev1 + prev2;
    }
}
```

Memoization



$O(N)$ Time Complexity

```

class Solution {
    public int fib(int n, int[] dp){
        if(n == 0) return 0;
        if(n == 1) return 1;
        if(dp[n] != -1) return dp[n];
        // Already Calculated Value should be returned

        int prev1 = fib(n - 1, dp);
        int prev2 = fib(n - 2, dp);

        dp[n] = prev1 + prev2;
        // Before returning the calculated value, store it somewhere
        return prev1 + prev2;
    }

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

```

Recursion Call Stack

$\rightarrow O(N)$

Extra Space: $\rightarrow O(N) \stackrel{DP}{=}$

Time Complexity $\rightarrow O(N)$

Dynamic Programming Identification

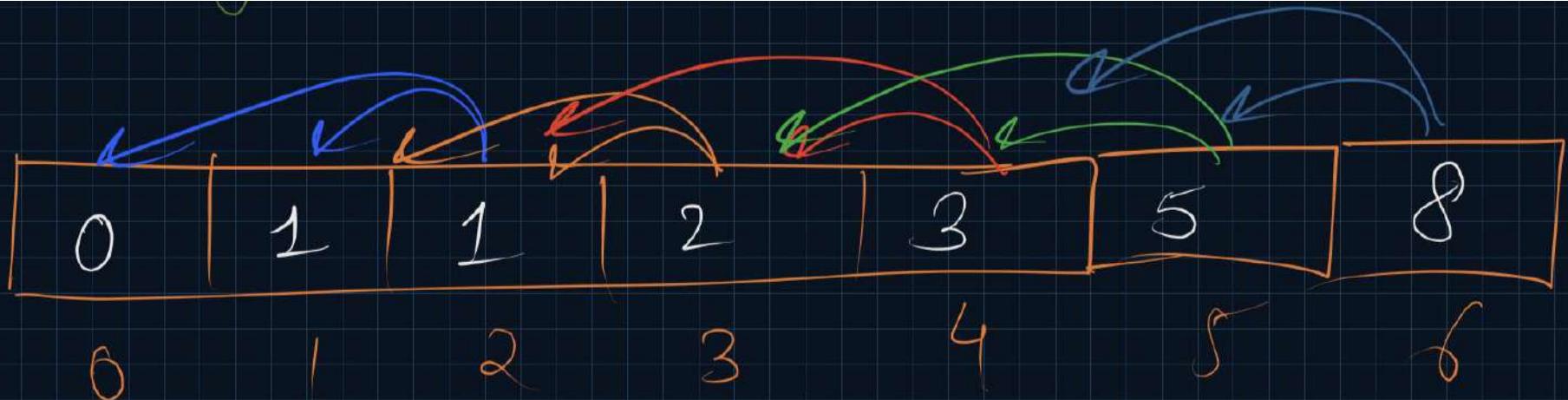
\rightarrow ① Overlapping Subproblems { Repeated Calls }

\rightarrow ② Optimal Substructure of Path Z

3 Tabulation

$$DP[n] = DP[n-1] + DP[n-2]$$

Iterative
soln



```
class Solution {  
    public int fib(int n) {  
        if(n <= 1) return n;  
  
        int[] dp = new int[n + 1];  
        dp[0] = 0; dp[1] = 1;  
  
        for(int i=2; i<=n; i++){  
            dp[i] = dp[i - 1] + dp[i - 2];  
        }  
  
        return dp[n];  
    }  
}
```

TC $\rightarrow O(N)$

SC $\rightarrow O(1)$
(Extra Space)

If R.C.S.S $\rightarrow O(1)$



```
class Solution {  
    public int fib(int n) {  
        if(n <= 1) return n;  
  
        int prev1 = 0, prev2 = 1;  
  
        for(int i=2; i<=n; i++){  
            int curr = prev1 + prev2;  
            prev1 = prev2;  
            prev2 = curr;  
        }  
  
        return prev2;  
    }  
}
```



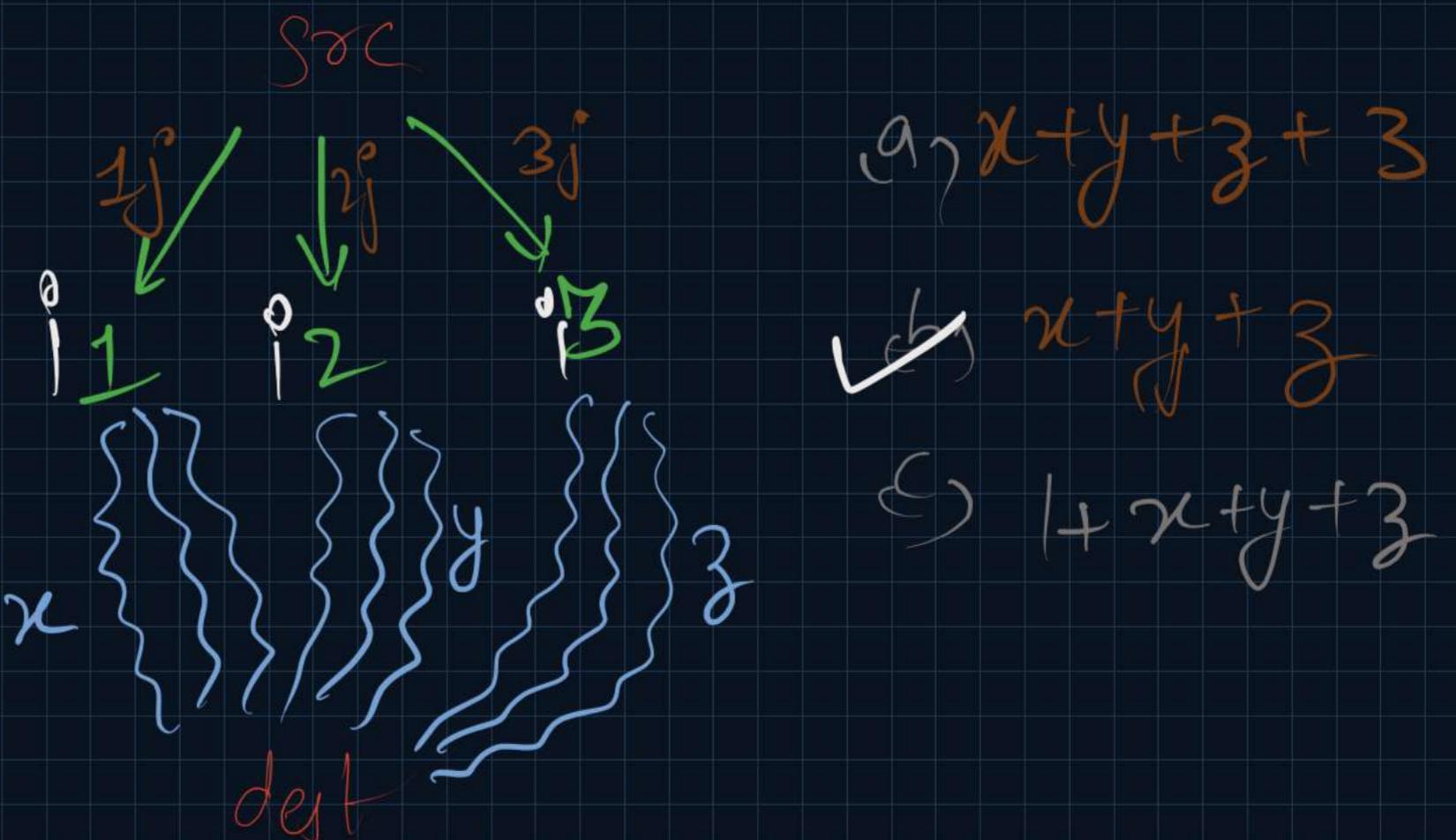
Time Complexity $\rightarrow O(N)$
Space Complexity $\rightarrow O(1)$

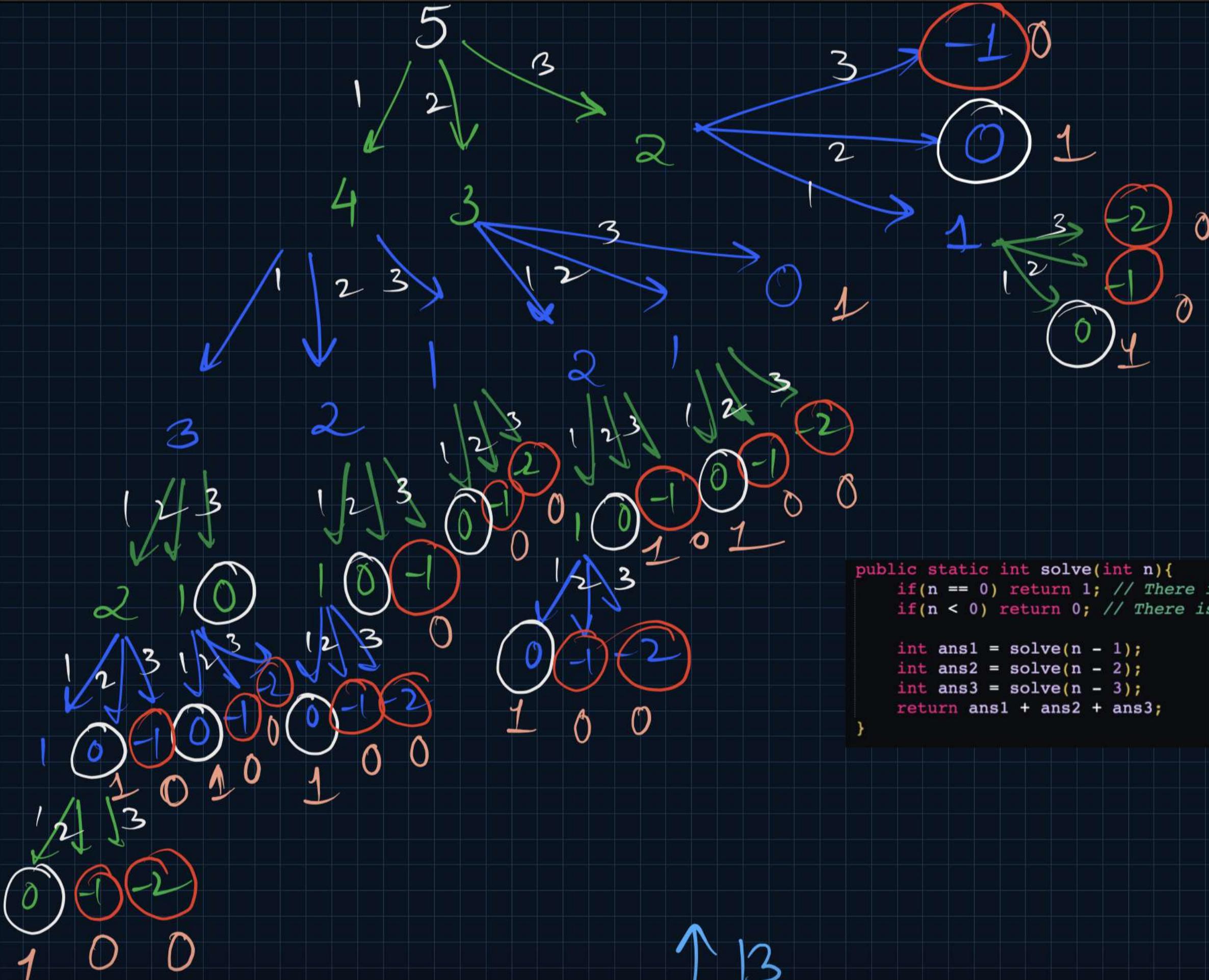
Dynamic Programming - Lecture ②

{ 23 April 2022
9 AM - 12 PM}

→ Climb Stairs & its Variations

① Climb Stairs



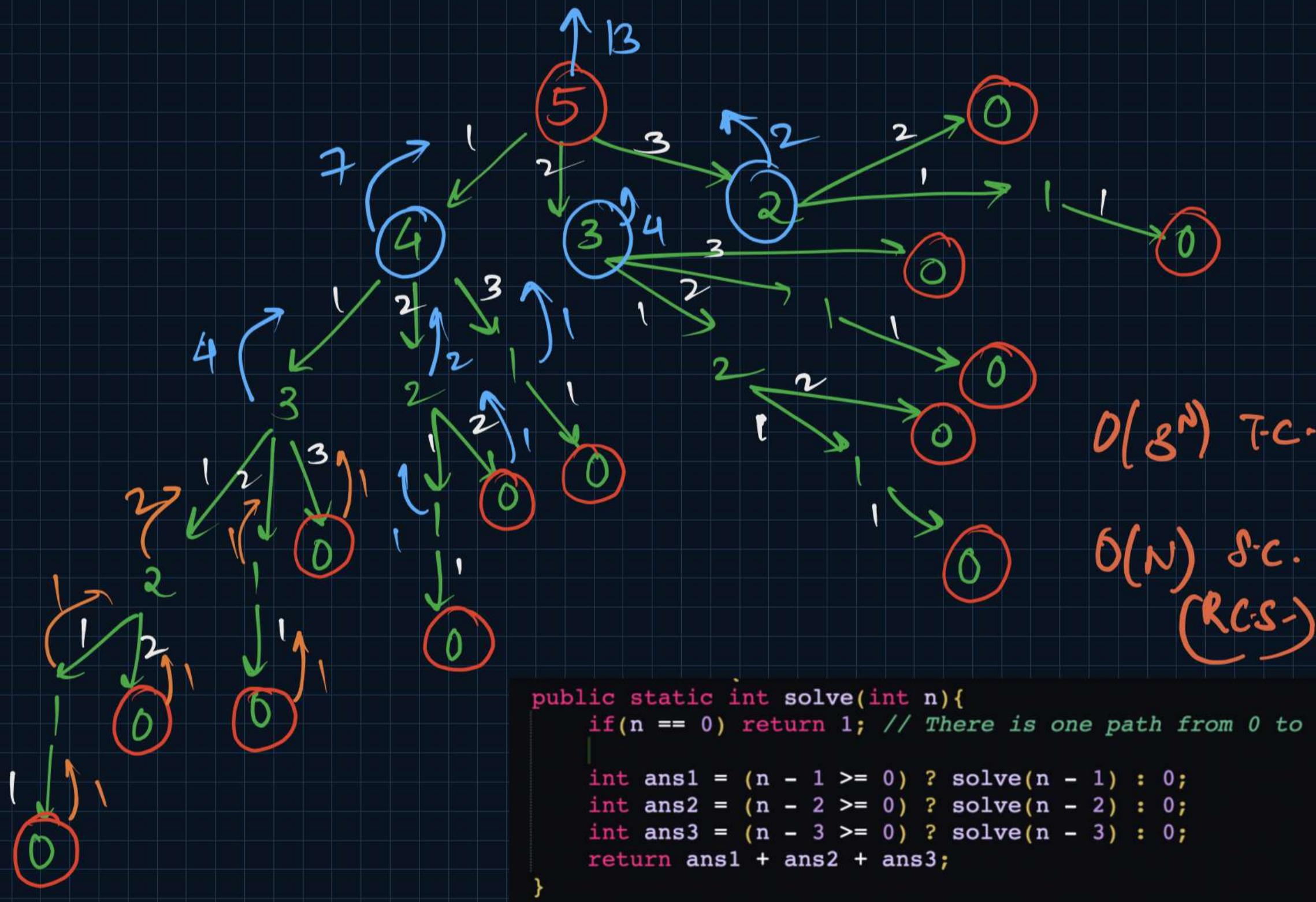


```

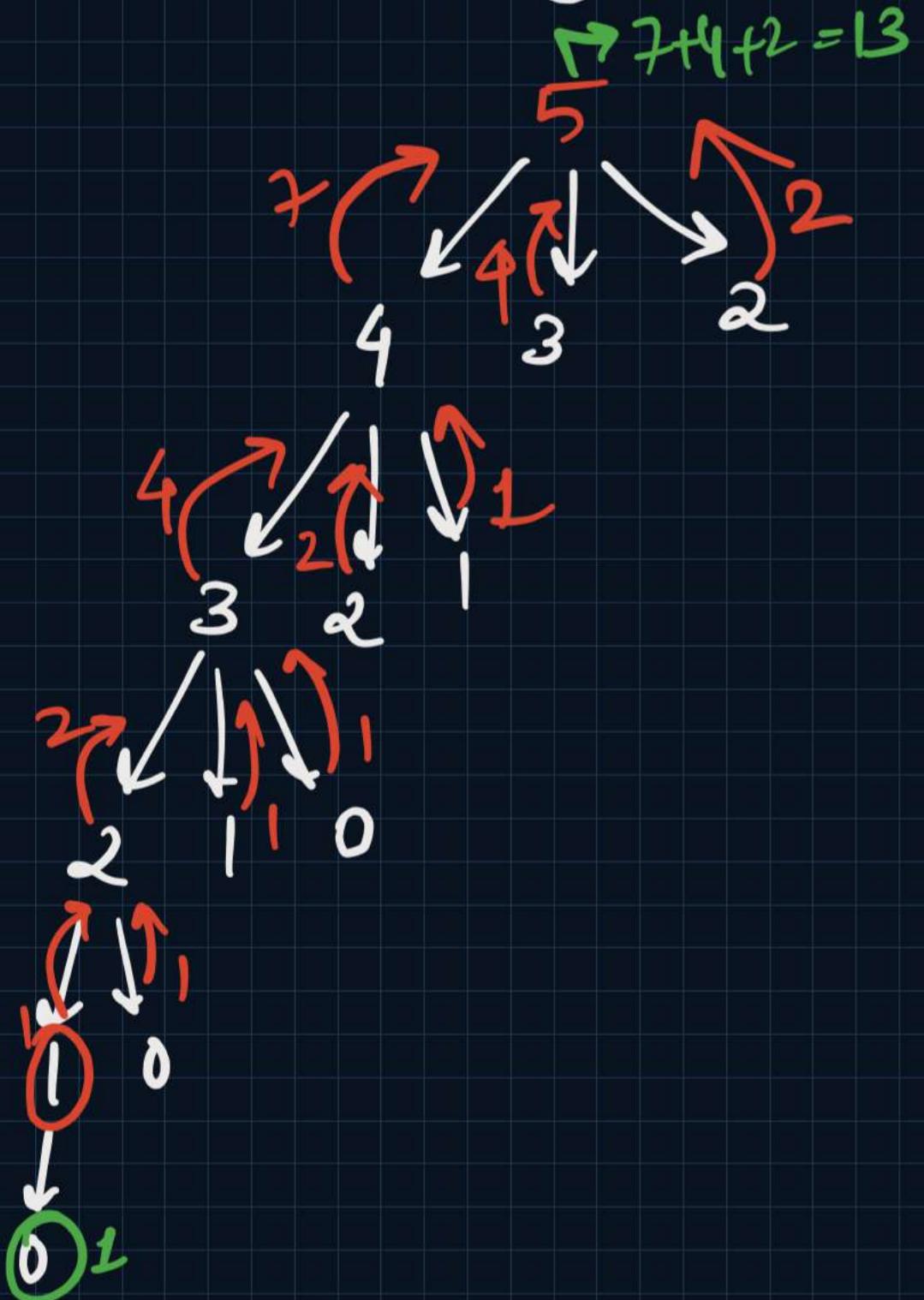
public static int solve(int n){
    if(n == 0) return 1; // There is one path from 0 to 0, i.e. "" empty string
    if(n < 0) return 0; // There is no path from -1 or -2 to 0

    int ans1 = solve(n - 1);
    int ans2 = solve(n - 2);
    int ans3 = solve(n - 3);
    return ans1 + ans2 + ans3;
}

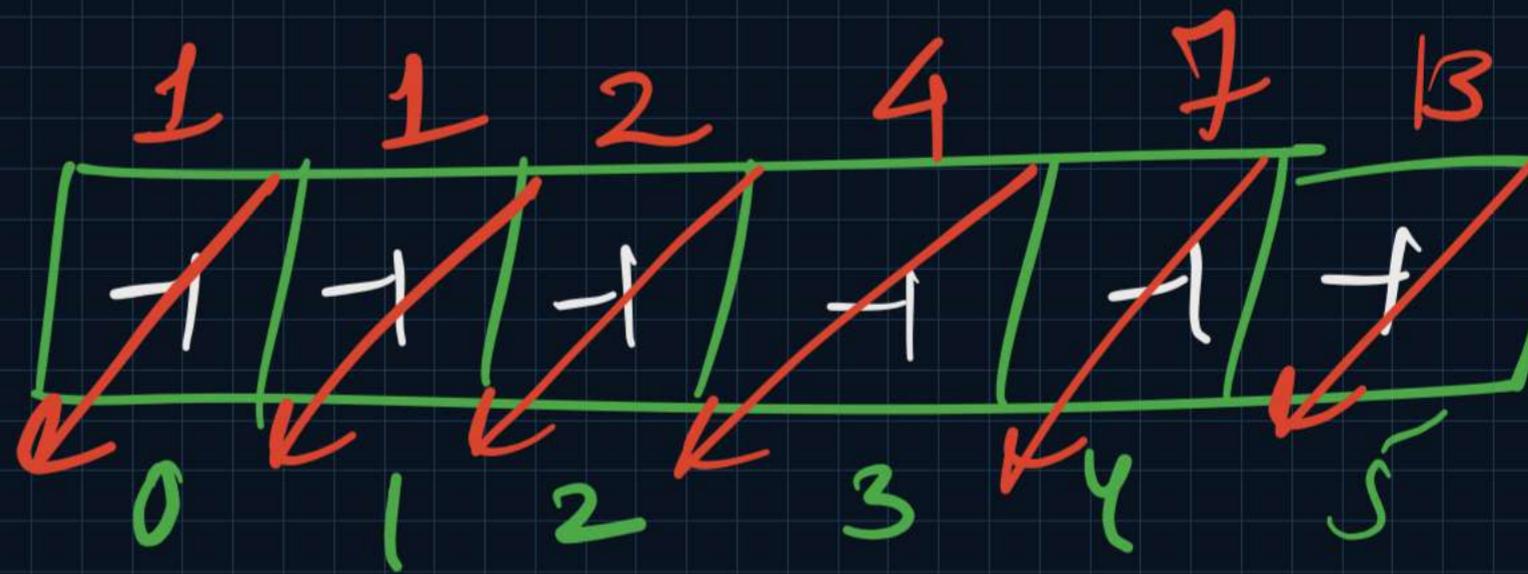
```



Memorization



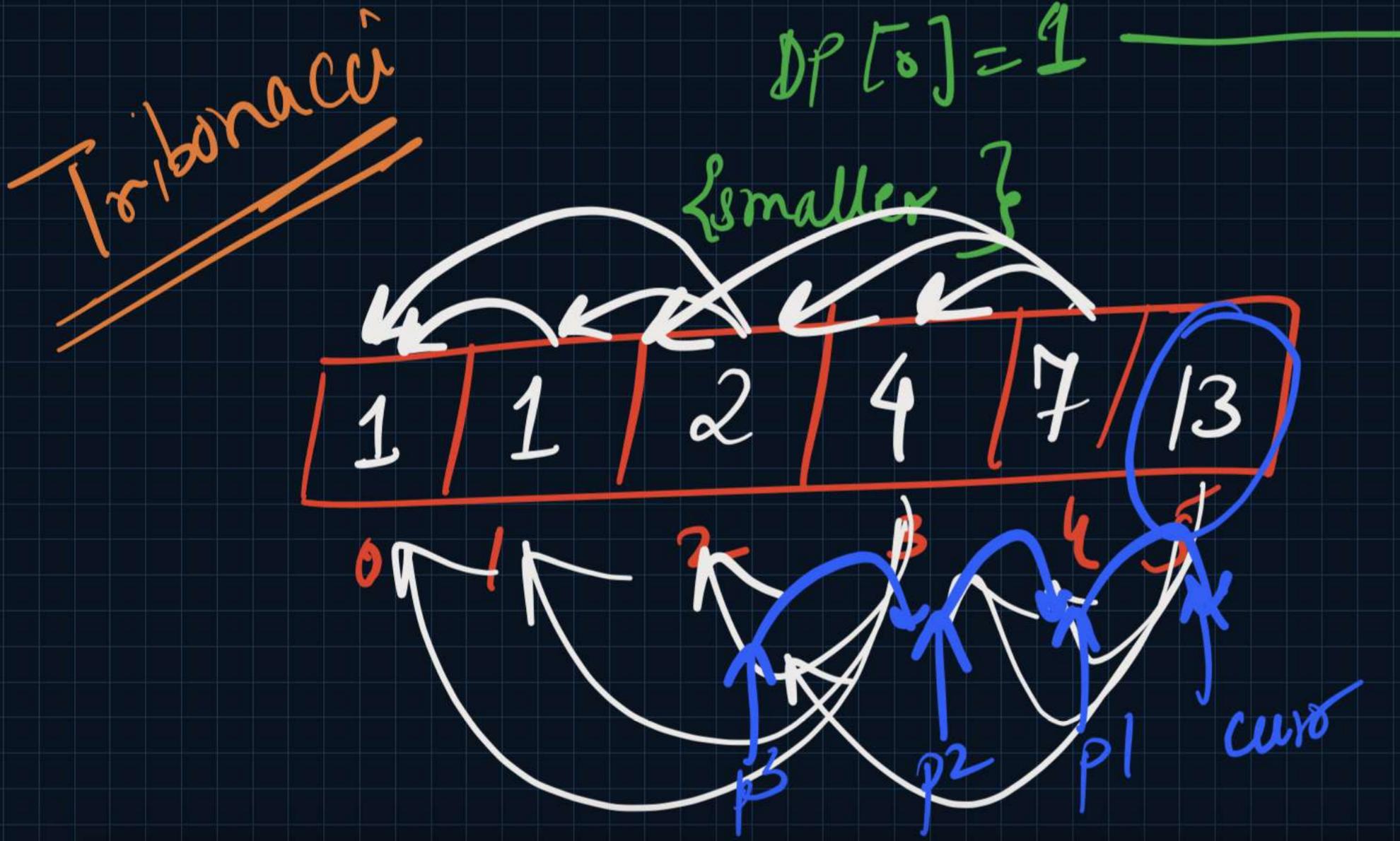
```
public static int solve(int n, int[] dp){  
    if(n == 0) return 1; // There is one path from 0 to 0, i.e. "" empty string  
    if(dp[n] != -1) return dp[n]; // Return the already calculated value (Preorder)  
  
    int ans1 = (n - 1 >= 0) ? solve(n - 1, dp) : 0;  
    int ans2 = (n - 2 >= 0) ? solve(n - 2, dp) : 0;  
    int ans3 = (n - 3 >= 0) ? solve(n - 3, dp) : 0;  
  
    dp[n] = ans1 + ans2 + ans3; // Store the Answer in DP Table (Postorder)  
    return dp[n];
```



Time Complexity $\rightarrow O(n)$

Space Complexity $\rightarrow O(n)$ DTable

Tabulation



TC $\rightarrow O(N)$
SC $\rightarrow O(N)$
extra

Nth state
DP[N] = ?
{larger}

$$DP[N] = DP[N-1] + DP[N-2] + DP[N-3]$$

```
int n = scn.nextInt();
int[] dp = new int[n + 1];
dp[0] = 1;

for(int i=1; i<=n; i++){
    int ans1 = (i - 1 >= 0) ? dp[i - 1] : 0;
    int ans2 = (i - 2 >= 0) ? dp[i - 2] : 0;
    int ans3 = (i - 3 >= 0) ? dp[i - 3] : 0;
    dp[i] = ans1 + ans2 + ans3;
}

System.out.println(dp[n]);
```

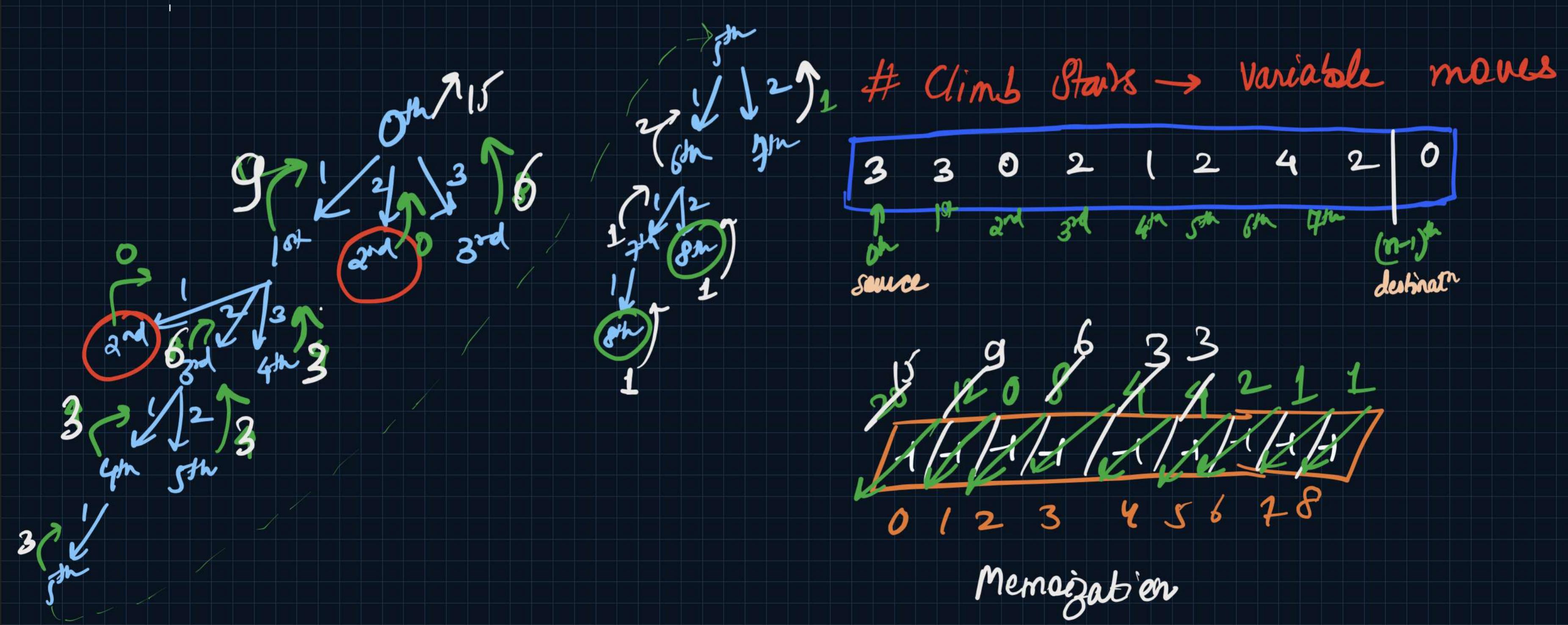
Extra Space
Optimization
O(n) Time
O(n) Space

```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    int prev1 = 1, prev2 = 0, prev3 = 0;
    for(int i=1; i<=n; i++){
        int curr = prev1 + prev2 + prev3;
        prev3 = prev2;
        prev2 = prev1;
        prev1 = curr;
    }

    System.out.println(prev1);
}
```

4 pointer
Technique



```

public class Main {
    public static int memo(int n, int[] jumps, int[] dp){
        if(n == jumps.length) return 1;
        if(dp[n] != -1) return dp[n];

        int ans = 0;
        for(int i=1; i<=jumps[n]; i++){
            if(n + i <= jumps.length)
                ans += memo(n + i, jumps, dp);
        }

        dp[n] = ans;
        return ans;
    }
}

```

```

public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] jumps = new int[n];
    for(int i=0; i<n; i++){
        jumps[i] = scn.nextInt();
    }

    int[] dp = new int[n + 1];
    Arrays.fill(dp, -1);
    System.out.println(memo(0, jumps, dp));
}

```

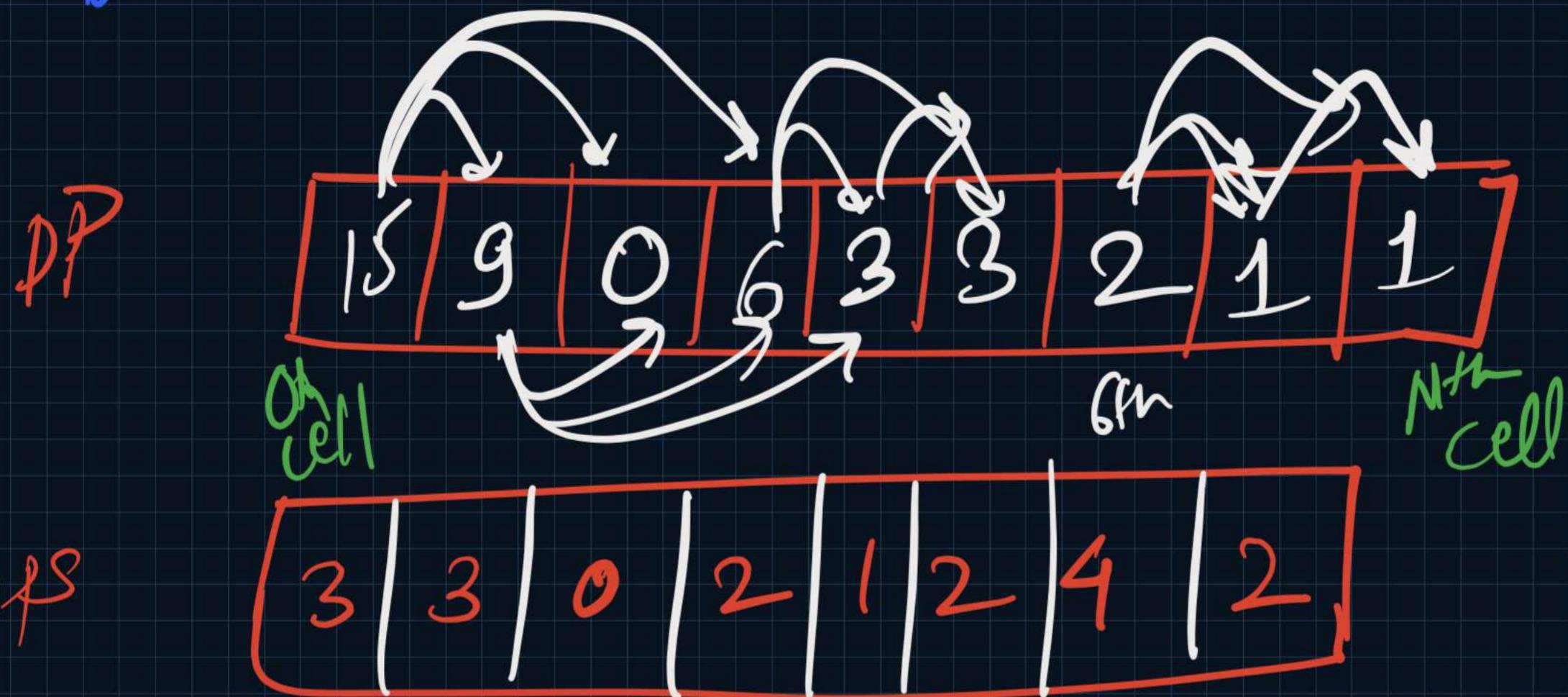
recursion
↓
 $TC \Rightarrow O(jumps^N)$
 $SC \Rightarrow O(N)$
R-C-S

Membr
 $TC \Rightarrow O(N \times jumps)$
 $SC \Rightarrow O(N)$
R-C-S, DF

~~Tabulation~~

hanger
Bth state

Smaller
Nth state



```

public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] jumps = new int[n];
    for(int i=0; i<n; i++){
        jumps[i] = scn.nextInt();
    }

    int[] dp = new int[n + 1];
    dp[n] = 1;

    for(int cell=n-1; cell>=0; cell--){
        int ans = 0;
        for(int jump=1; jump<=jumps[cell]; jump++){
            if(cell + jump <= n){
                ans += dp[cell + jump];
            }
        }
        dp[cell] = ans;
    }
    System.out.println(dp[0]);
}

```

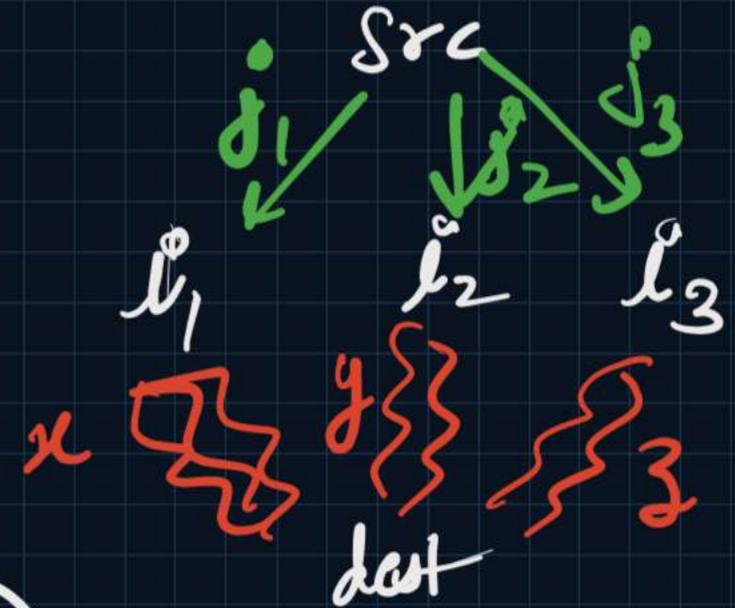
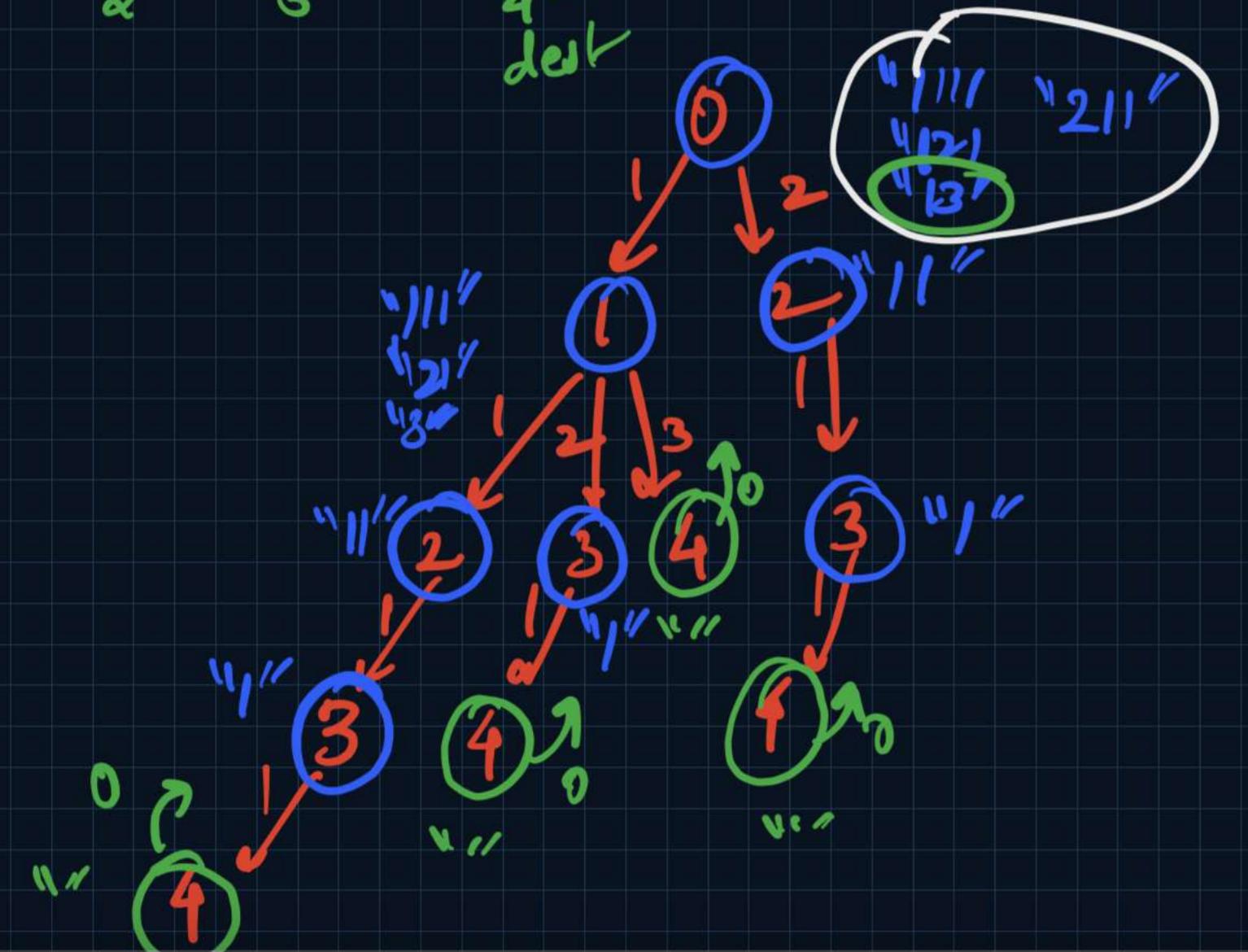
$$O(N * \text{jumps}) \quad \underline{TC}$$

$$O(N) \leq \{ \text{DP table} \}$$

Space optimization

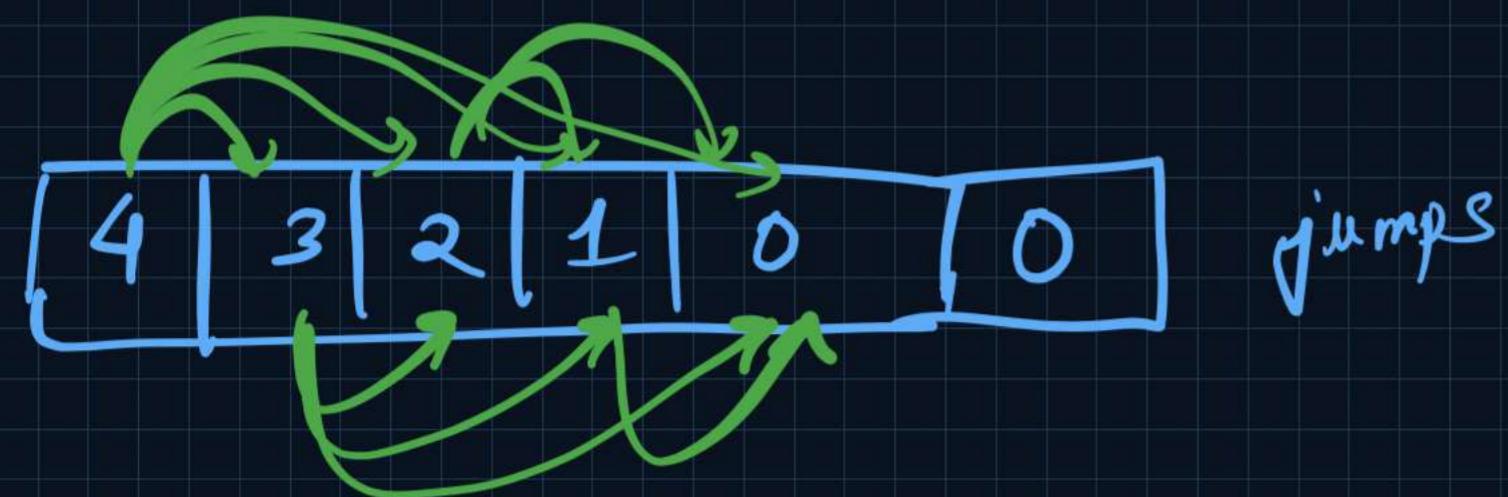
↳ Not possible { jumps value is not fixed }

climb stairs with minimum jumps



ans = $\min(x, y, z)$

eg²



Count of paths = 0

min length = +∞



```
class Solution {  
    public long memo(int src, int[] jumps, long[] dp){  
        if(src == jumps.length - 1)  
            return 0; // min moves to go to dest from dest is 0 (empty string)  
        if(dp[src] != -1) return dp[src];  
  
        long min = Integer.MAX_VALUE;  
        for(int jump=1; jump<=jumps[src]; jump++){  
            if(src + jump < jumps.length){  
                min = Math.min(min, memo(src + jump, jumps, dp) + 1);  
            }  
        }  
  
        dp[src] = min;  
        return min;  
    }  
  
    public int jump(int[] jumps) {  
        long[] dp = new long[jumps.length + 1];  
        Arrays.fill(dp, -1);  
  
        return (int)memo(0, jumps, dp);  
    }  
}
```

TC $\Rightarrow O(N \times \text{jumps})$
SC $\Rightarrow O(N)$

* Tabulation \rightarrow HW \rightarrow TC $\Rightarrow O(n \cdot j^{\text{umps}}) \approx O(N^2)$

* Space Optimization \rightarrow N of possible

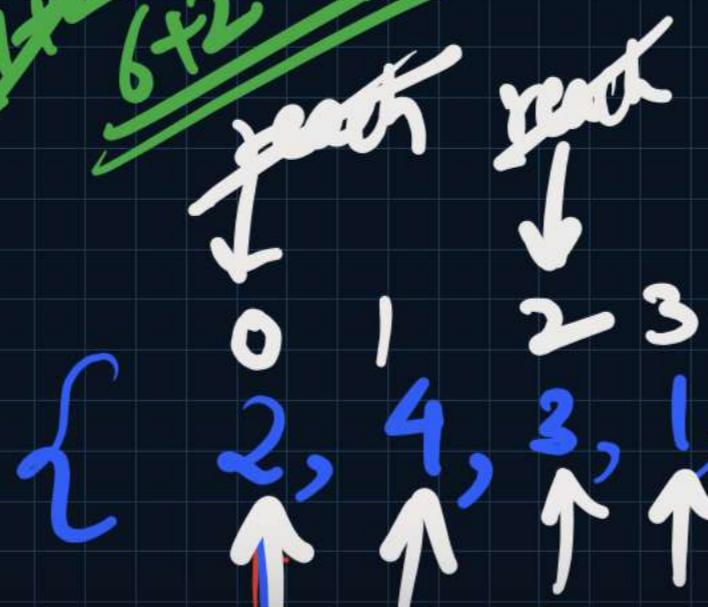
Jump Game - I { check if $+\infty$, \Rightarrow false { path not possible }
 else \Rightarrow true { path possible }

~~array[i] + i~~
~~Greedy soln~~
~~or~~
~~reach = 0~~



~~path reach~~
~~path reach~~
~~path reach~~

~~eq 2~~



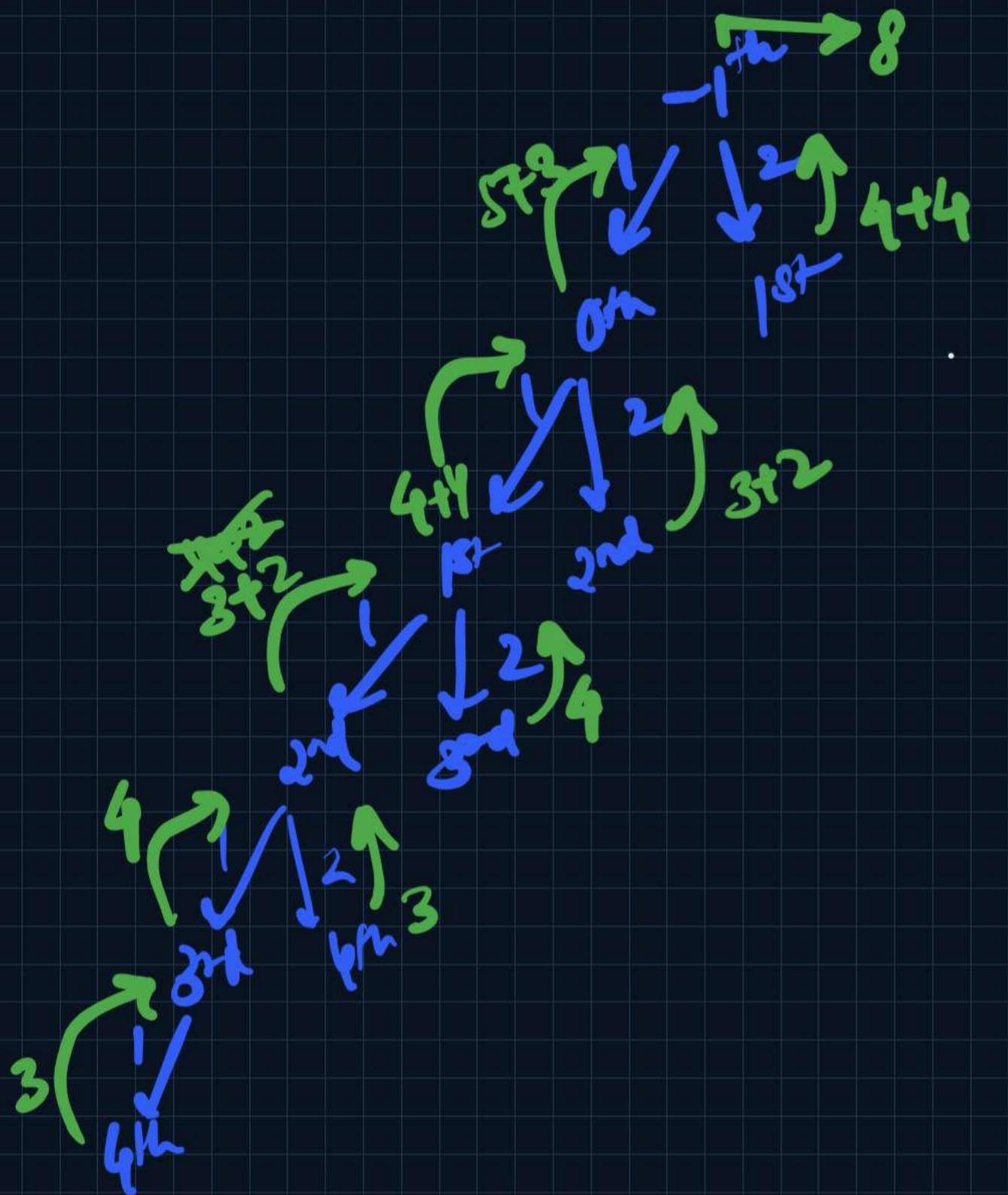
reach in reach
 false
 no path possible

```
class Solution {
    public boolean canJump(int[] jumps) {
        int reach = 0;
        for(int i=0; i<jumps.length; i++){
            if(i > reach) return false;

            if(i + jumps[i] > reach){
                reach = i + jumps[i];
            }
        }

        return true;
    }
}
```

$T \subset \Rightarrow O(N)$
 $S \subset \Rightarrow O(1)$



Climb Stairs with Minimum Cost

$$\{0, 3, 4, 2, 1, 3\}$$

-1 0 1 2 3 4

jumps

1
2

$SRC(x)$

i_1 i_2

$\{ \} p$ $\{ \} q$

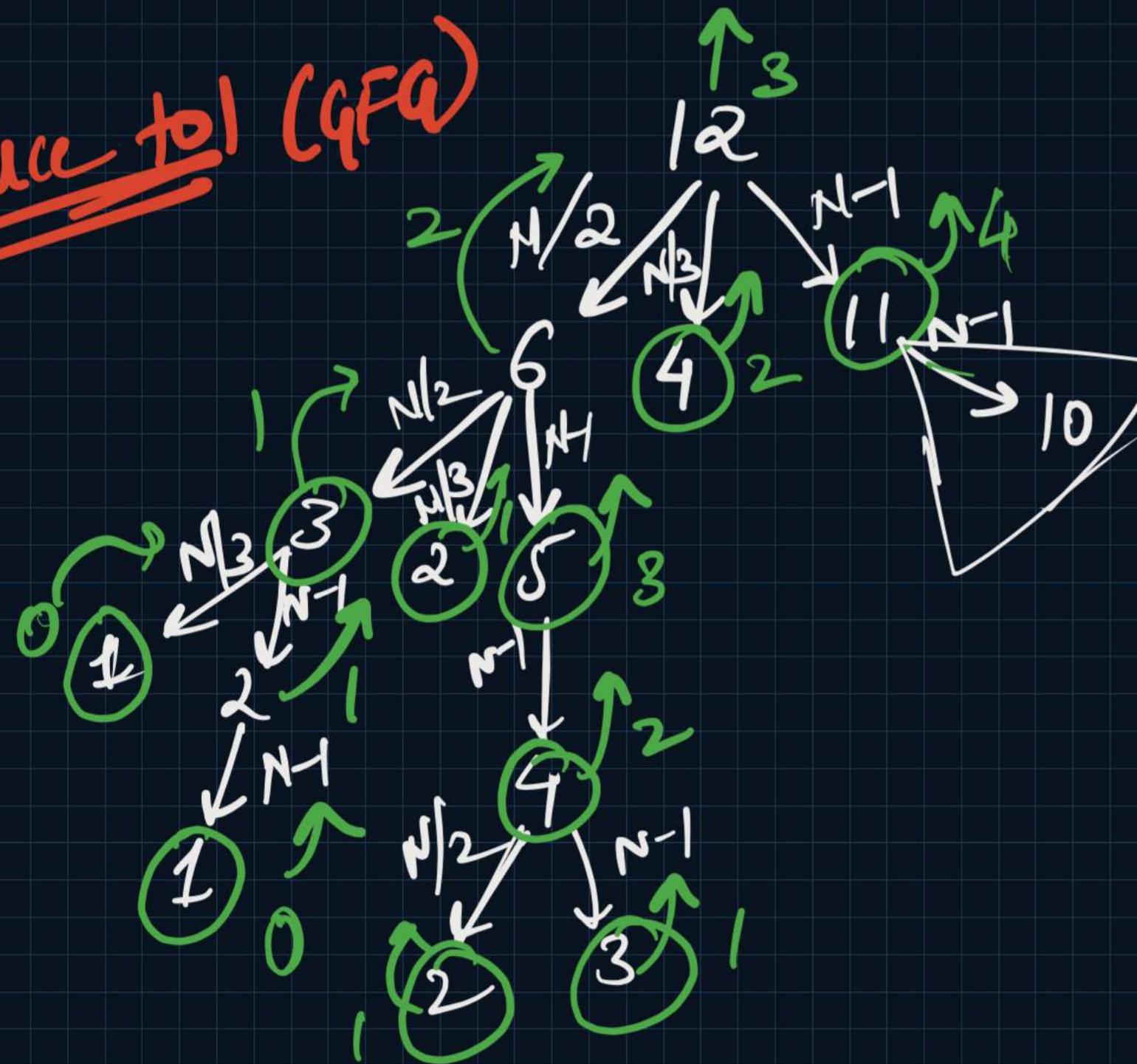
dp

$\min(p, q) + x$

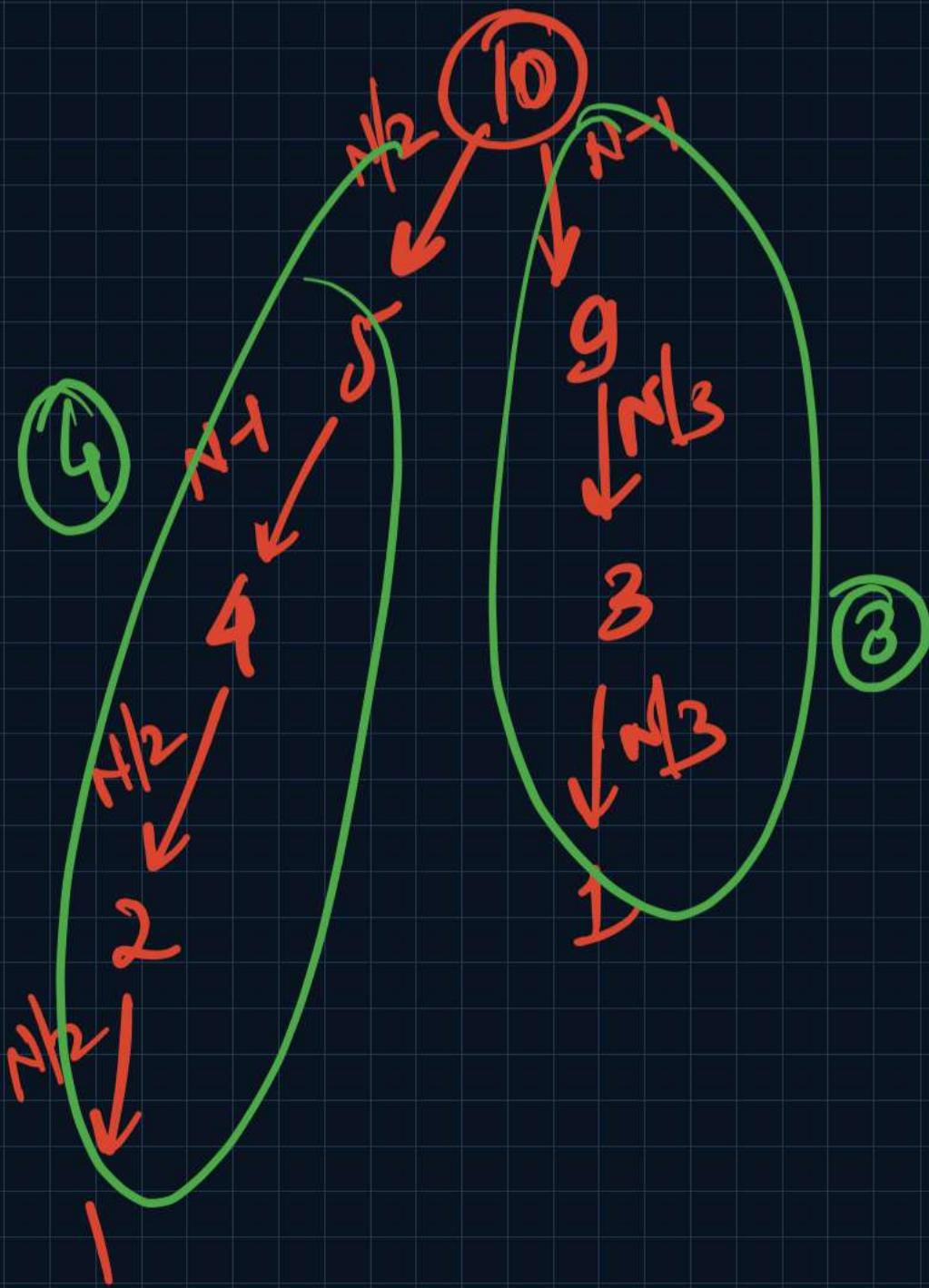
```
class Solution {  
    public int memo(int src, int[] cost, int[] dp){  
        if(src >= cost.length) return 0;  
        if(dp[src] != -1) return dp[src];  
  
        int ans1 = memo(src + 1, cost, dp);  
        int ans2 = memo(src + 2, cost, dp);  
  
        return dp[src] = Math.min(ans1, ans2) + cost[src];  
    }  
  
    public int minCostClimbingStairs(int[] cost) {  
        int[] dp = new int[cost.length + 1];  
        Arrays.fill(dp, -1);  
  
        memo(0, cost, dp);  
        return Math.min(dp[0], dp[1]);  
    }  
}
```

Memoization
 $O(N)$ time
 $O(N)$ space

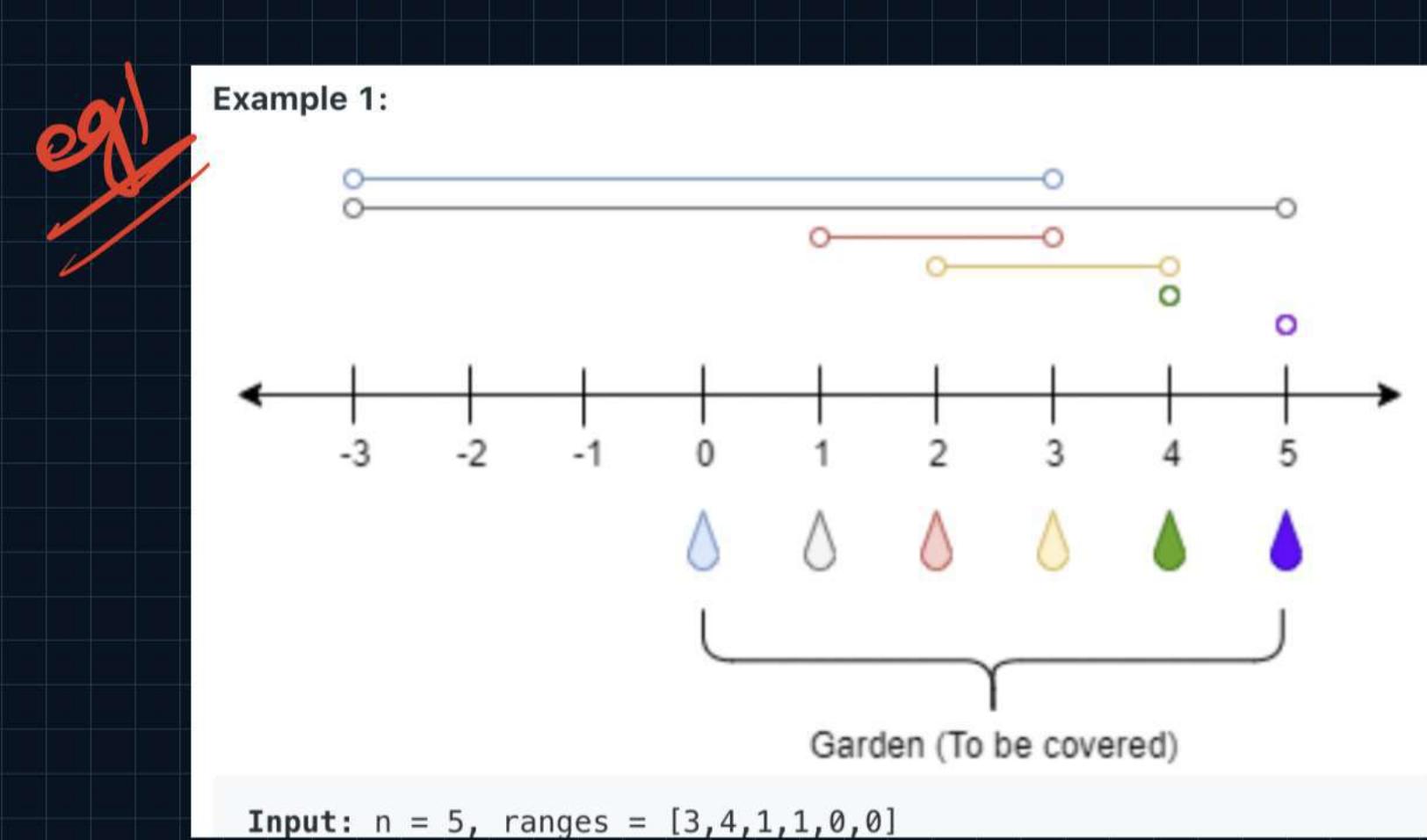
reduce to 1 (GFA)



```
class Solution{  
    public int memo(int N, int[] dp){  
        if(N == 1) return 0;  
        if(dp[N] != -1) return dp[N];  
  
        int ans1 = (N % 2 == 0) ? memo(N/2, dp) : Integer.MAX_VALUE;  
        int ans2 = (N % 3 == 0) ? memo(N/3, dp) : Integer.MAX_VALUE;  
        int ans3 = memo(N - 1, dp);  
  
        return dp[N] = Math.min(ans1, Math.min(ans2, ans3)) + 1;  
    }  
  
    public int minSteps(int N) {  
        int[] dp = new int[N + 1];  
        Arrays.fill(dp, -1);  
  
        return memo(N, dp);  
    }  
}
```



Greedy fail
smaller the no, better the answer

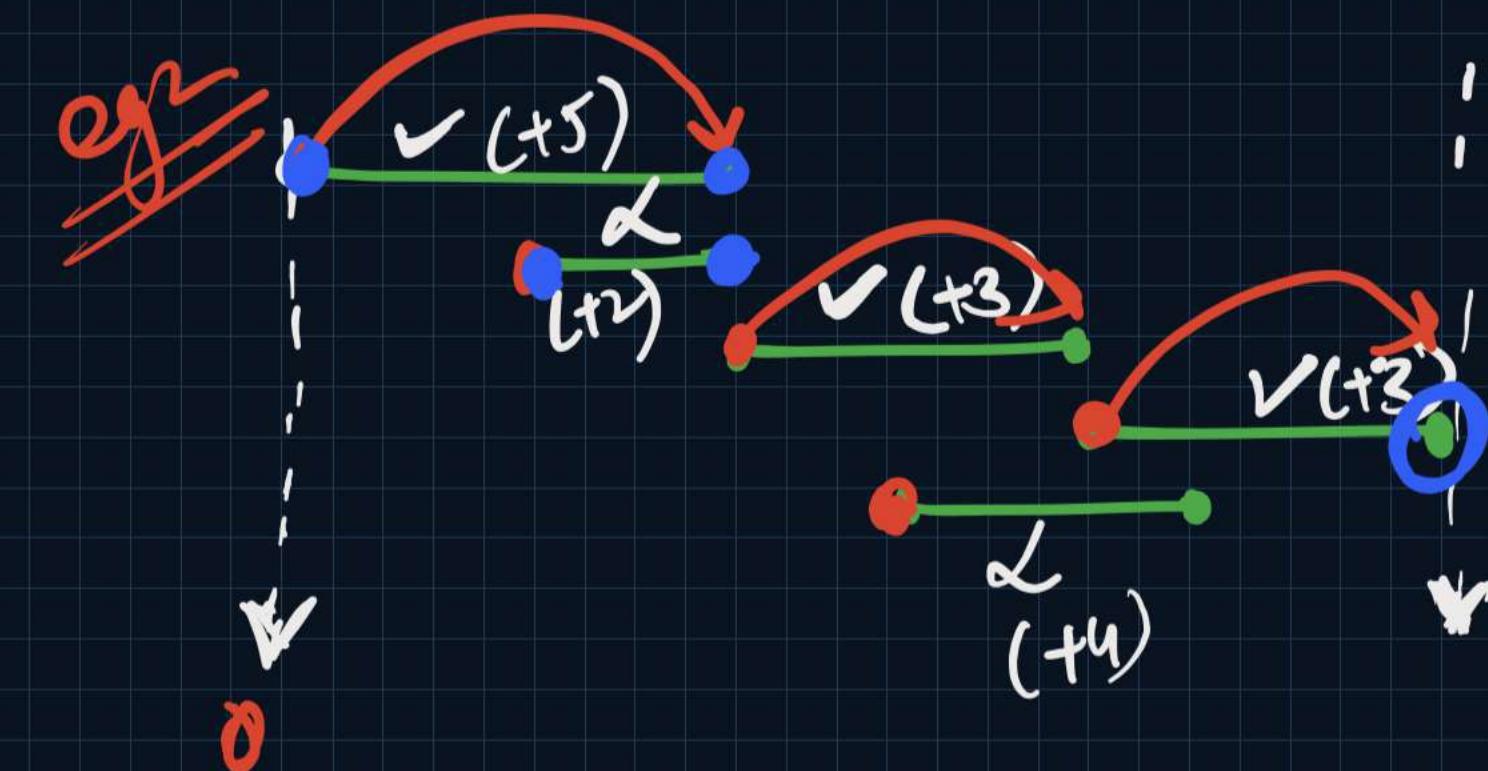


0 → 5

1 → 2

2 → 3

Minimum Taps to Garden



Answer = 3

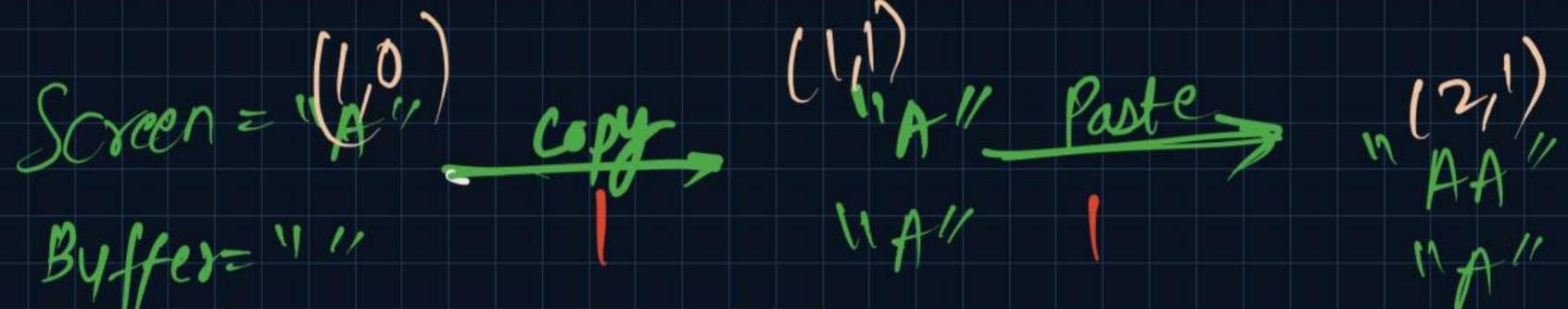
3 → 4
4 → 3



There is only one character '`'A'`' on the screen of a notepad. You can perform one of two operations on this notepad for each step:

- Copy All: You can copy all the characters present on the screen (a partial copy is not allowed).
- Paste: You can paste the characters which are copied last time.

Given an integer `n`, return *the minimum number of operations to get the character '`'A'`' exactly `n` times on the screen.*



```

class Solution {
    public long memo(int screen, int buffer, long dest, long[][] dp){
        if(screen > dest) return Integer.MAX_VALUE;
        if(screen == dest) return 0;
        if(dp[screen][buffer] != -1)
            return dp[screen][buffer];
        long copyPaste = 2l + memo(2 * screen, screen, dest, dp);
        long paste = 1l + memo(screen + buffer, buffer, dest, dp);

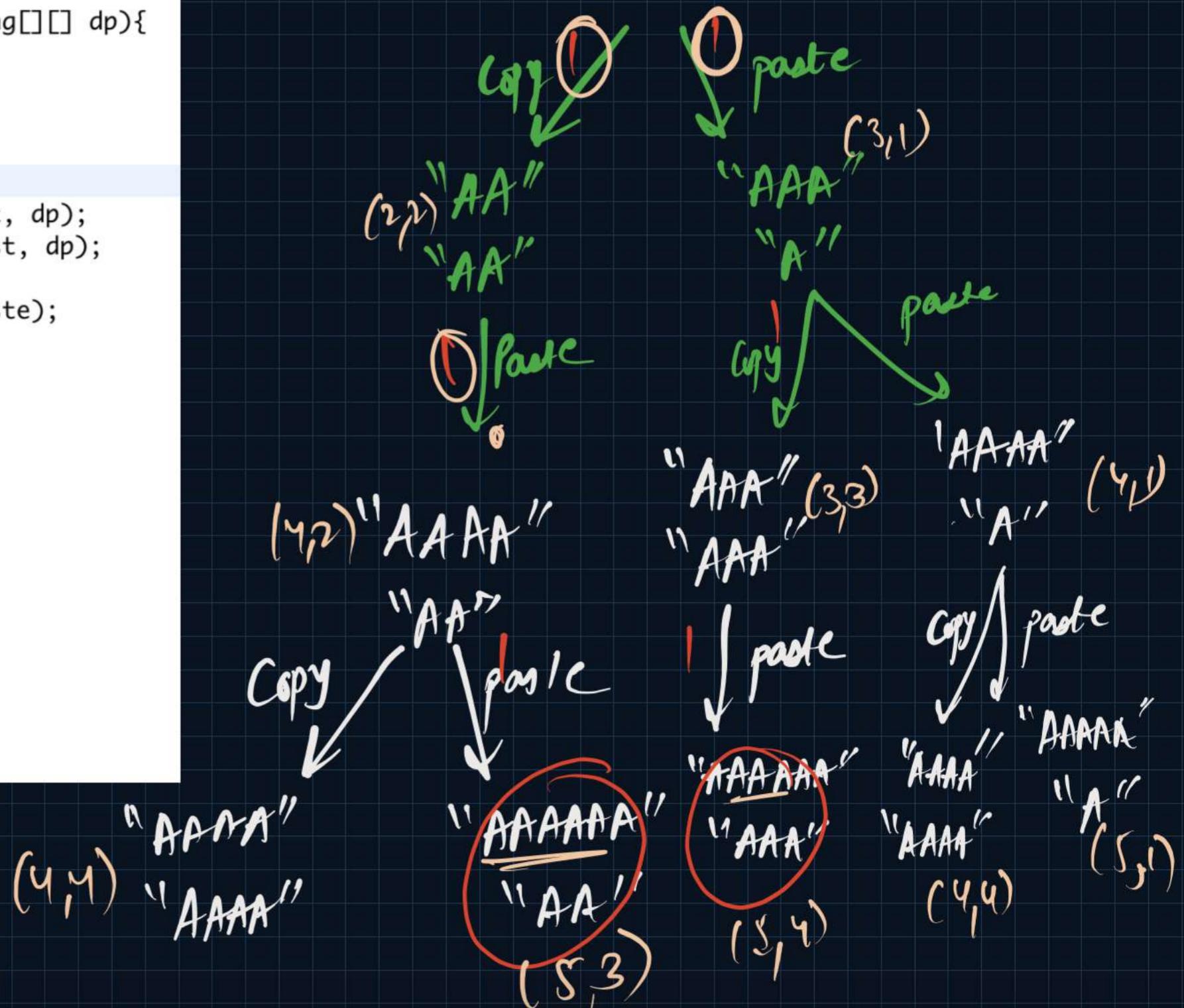
        return dp[screen][buffer] = Math.min(paste, copyPaste);
    }
    public int minSteps(int n) {
        if(n == 1) return 0;

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

        return (int)(1l + memo(1, 0, n, dp));
    }
}

```

TC $\rightarrow O(N^2)$



Dynamic Programming - Lecture 4

{ 24 April, Sunday }
9 AM - 12 PM

- Consecutive 1's not allowed X
- Decode Ways - LeetCode X
- Ways To Tile A Floor | Practice X
- Count the number of ways X
- Friends Pairing Problem X
- Ugly Number II - LeetCode X

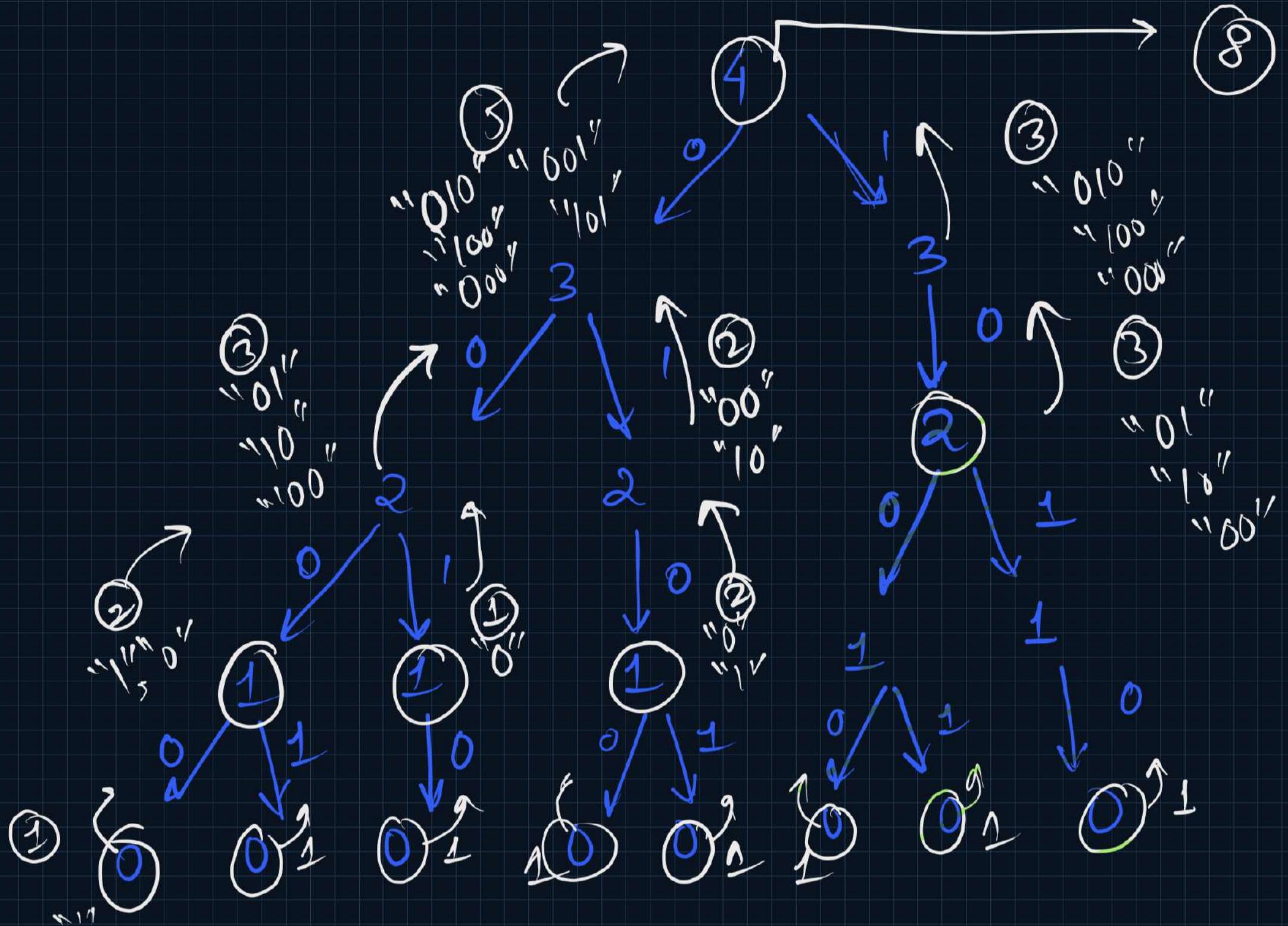
{ Fibonacci Variations }

① Count valid binary strings

(2) ① → "0", "1"

(3) ② → "00", "01", "10", ~~"11"~~

(5) ③ → "000", "001", "010", ~~"011"~~,
"100", "101", ~~"110"~~, ~~"111"~~



```

class Solution {
    int mod = 1000000007;

    long memo(int noOfDigits, int prevDigit, long[][] dp){
        if(noOfDigits == 0) return 1; // Empty String
        if(dp[noOfDigits][prevDigit] != 0)
            return dp[noOfDigits][prevDigit];

        long appending0 = memo(noOfDigits - 1, 0, dp);
        long appending1 = (prevDigit == 0) ? memo(noOfDigits - 1, 1, dp) : 0l;
        return dp[noOfDigits][prevDigit] = (appending0 + appending1) % mod;
    }

    long countStrings(int n) {
        long[][] dp = new long[n + 1][2];
        return memo(n, 0, dp);
    }
}

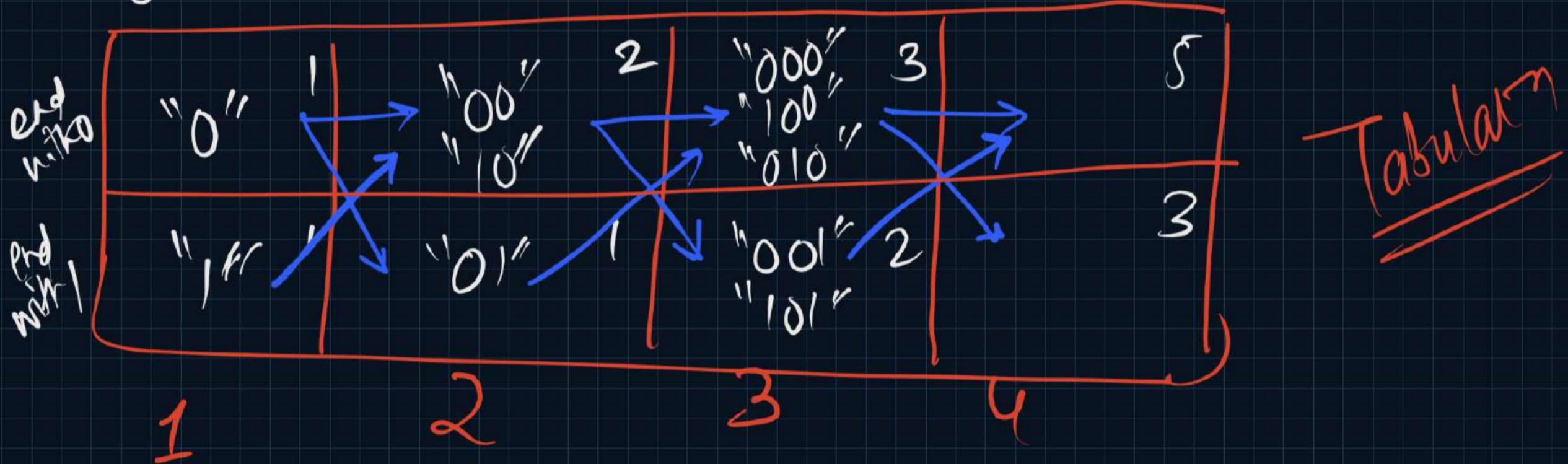
```

TC
 $O(2^N)$
 $\approx O(N)$

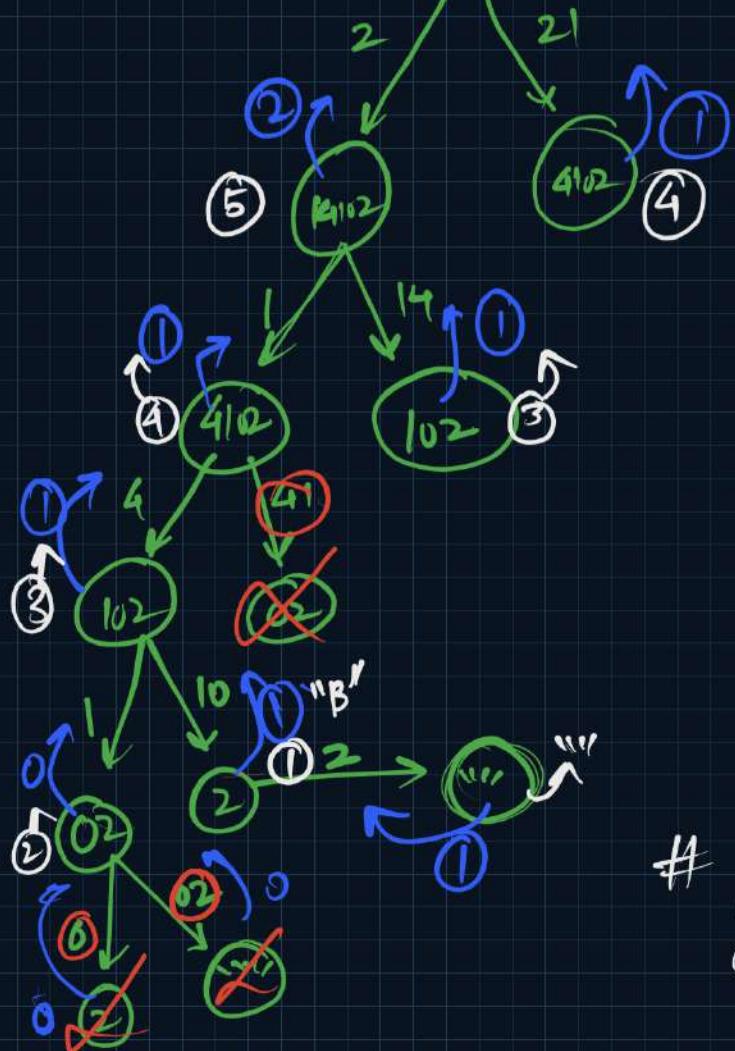
SC
 RCS Extra Space
 $O(N)$ $O(2^N)$
 $= O(N)$

Smaller Problem

Lage et Probleme



$$⑥ \text{ "214102"} \uparrow_{2+1} = ③$$



② Current Encodings { Decode Ways }

1	10	20
2	11	21
3	12	22
4	1	23
5	1	24
6	19	25
7	1	26

count →
(ways)

deci-
dest
n..

des-
dest
0

min →
steps
(length)

DP table can be made
by taking indexing as length
of input string

```

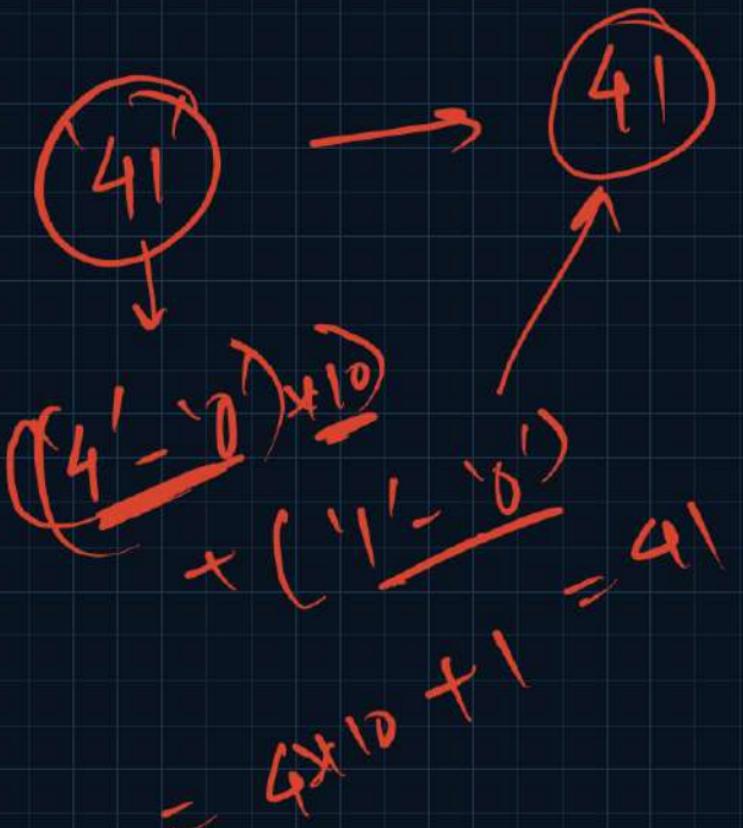
class Solution {
    public int memo(String s, int idx, int[] dp){
        if(idx == s.length()) return 1;
        if(dp[idx] != -1) return dp[idx];

        int ans1 = 0, ans2 = 0;
        if(s.charAt(idx) != '0'){
            ans1 = memo(s, idx + 1, dp);
        }

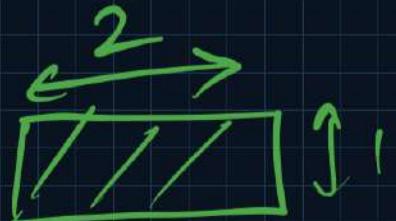
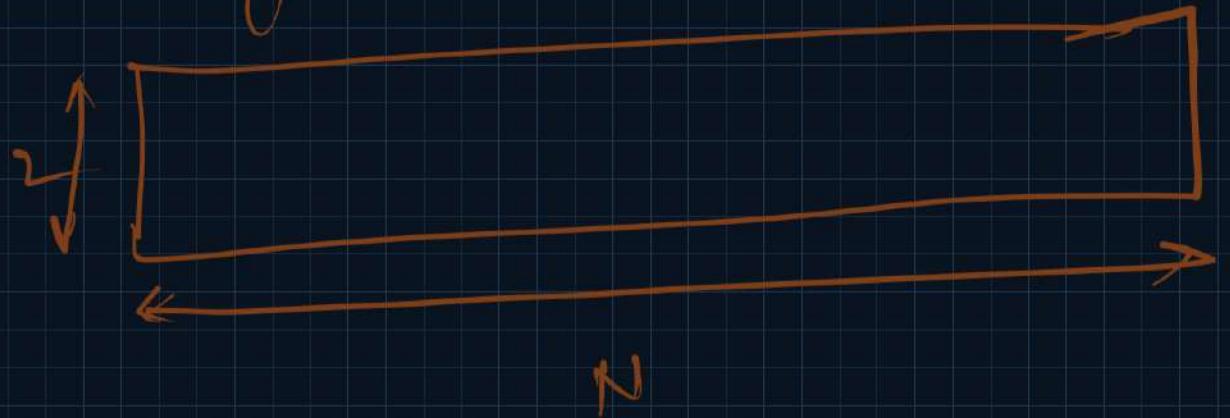
        if(idx + 1 < s.length()){
            int code = (s.charAt(idx) - '0') * 10 + (s.charAt(idx + 1) - '0');
            if(code >= 10 && code <= 26){
                ans2 = memo(s, idx + 2, dp);
            }
        }

        return dp[idx] = ans1 + ans2;
    }
}

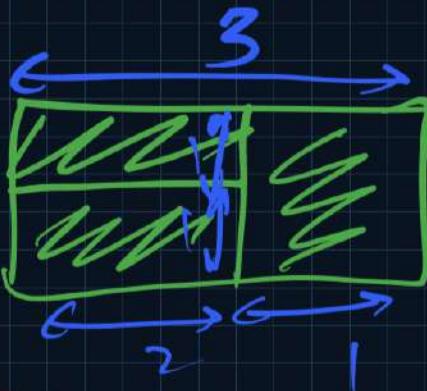
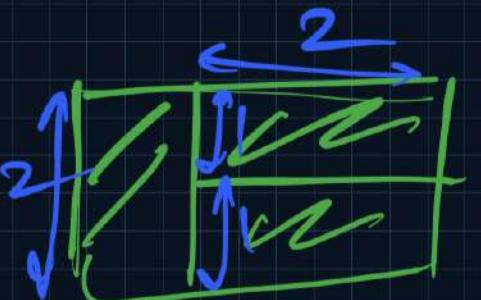
```

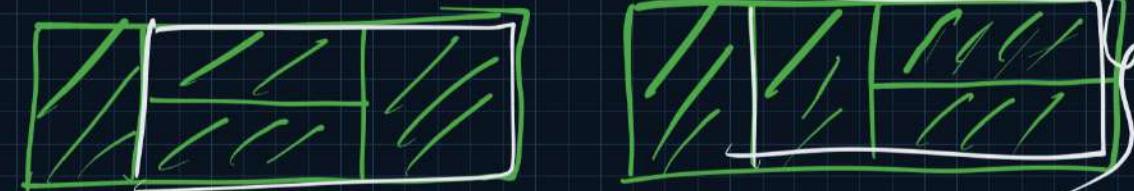
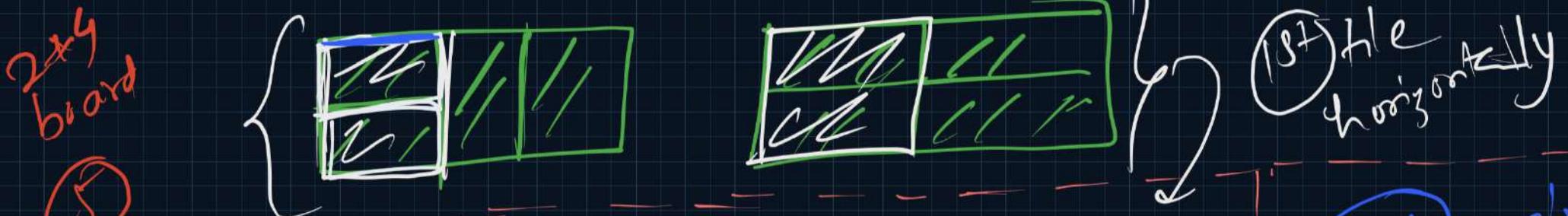
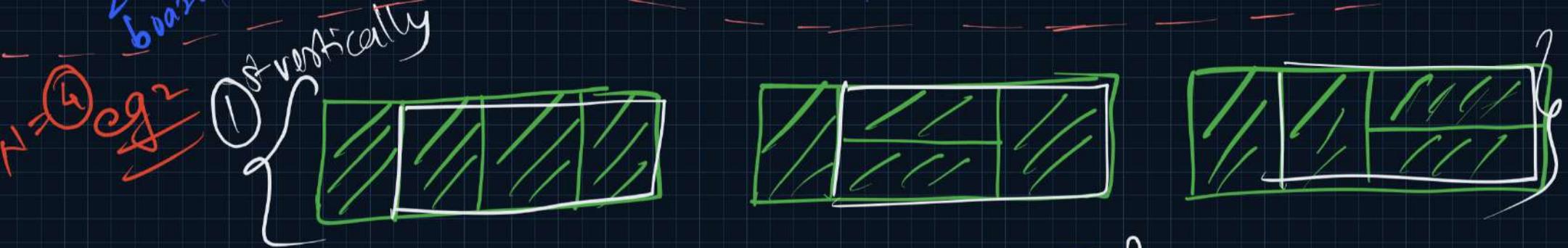


Tiling Problem - ①



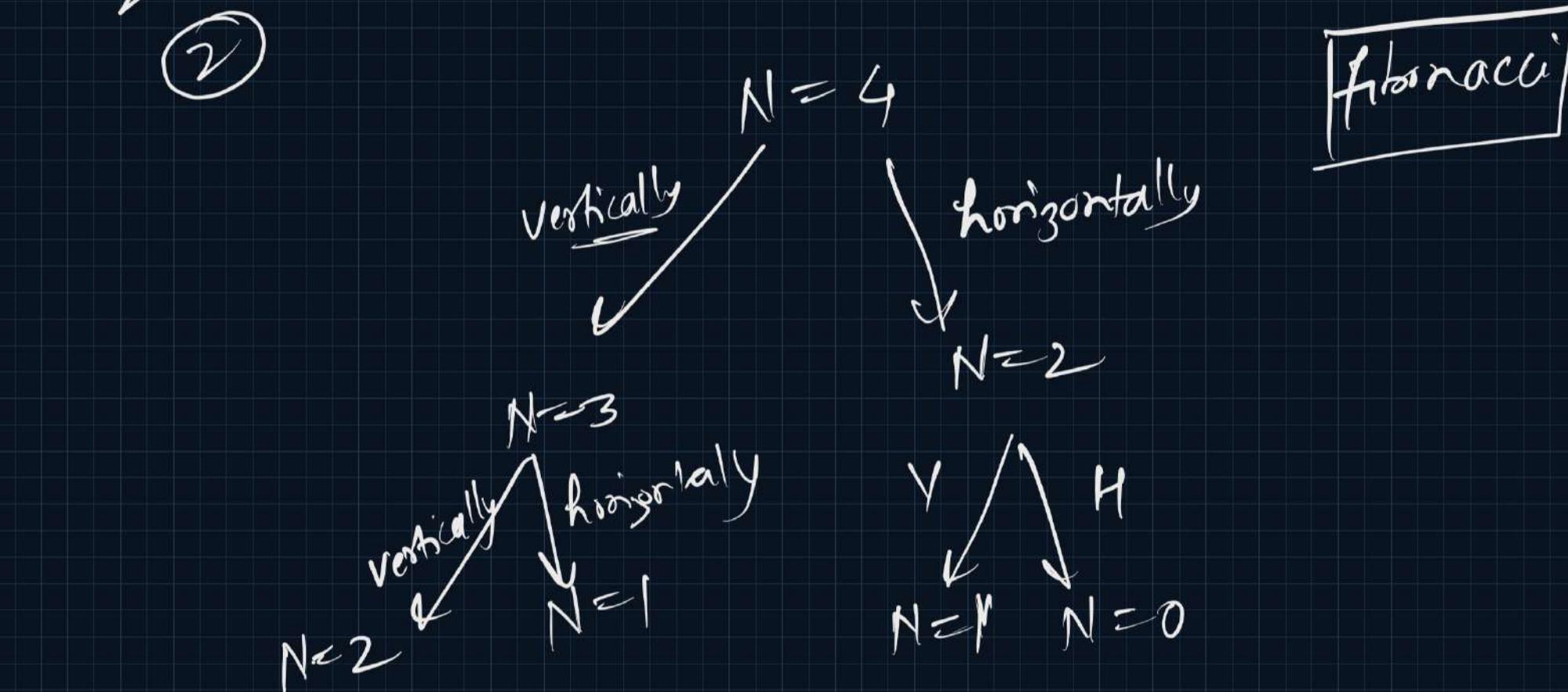
N → ③ cgl
2x3 board



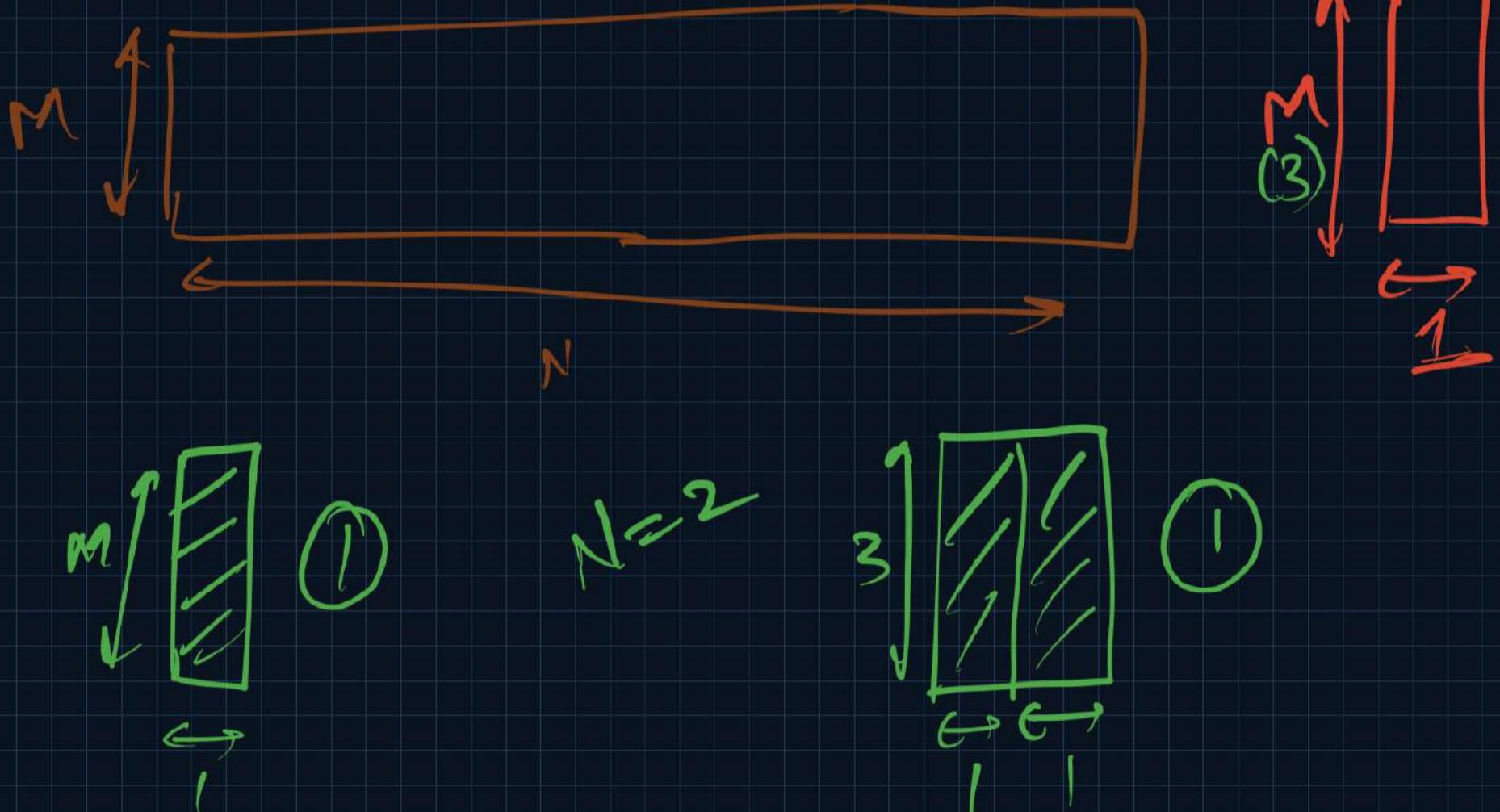


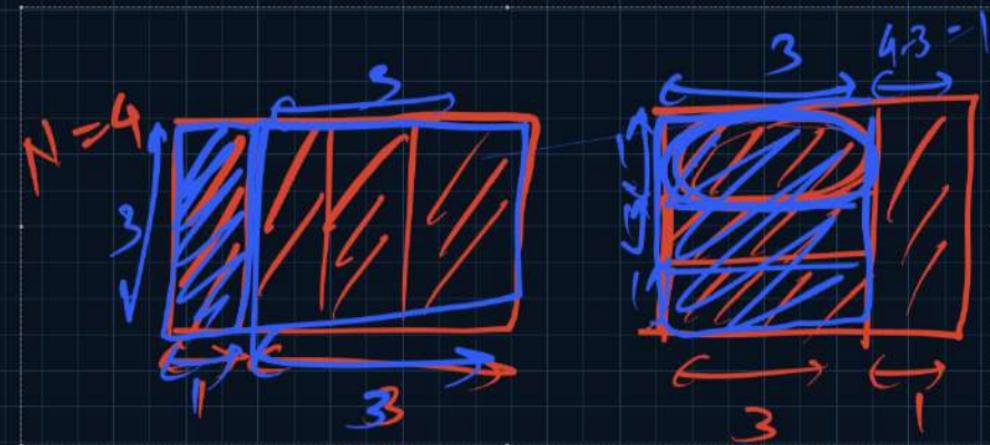
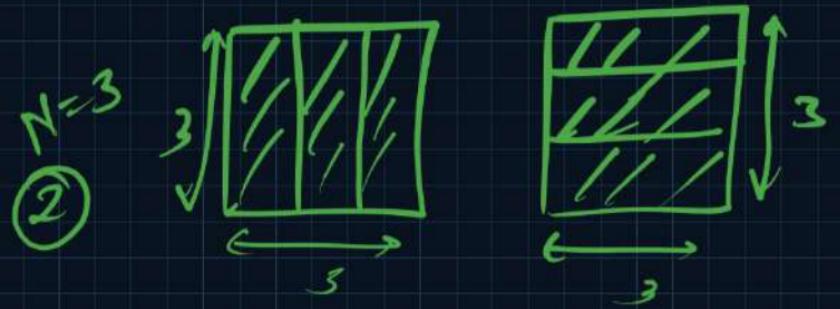
$N=4$

Fibonacci



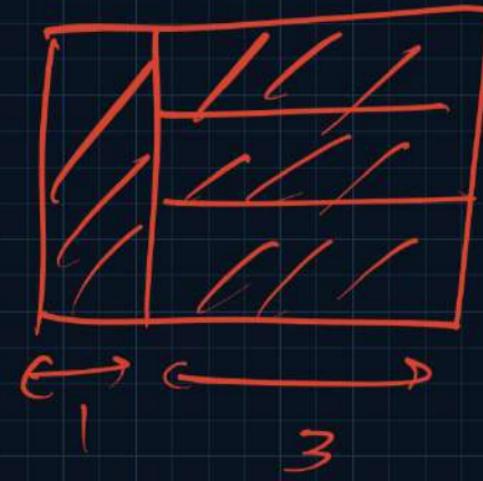
Tiling Problem - II



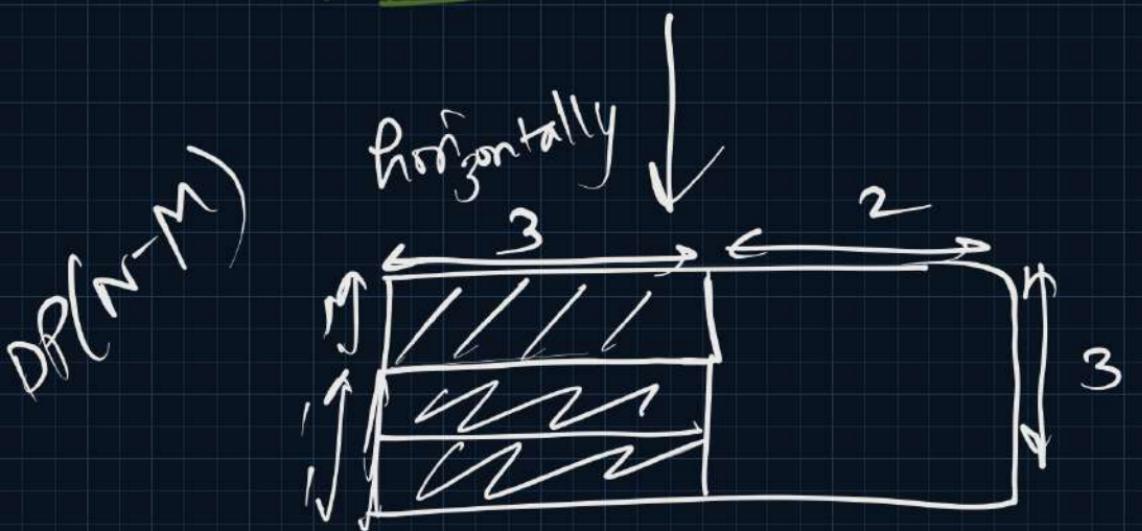
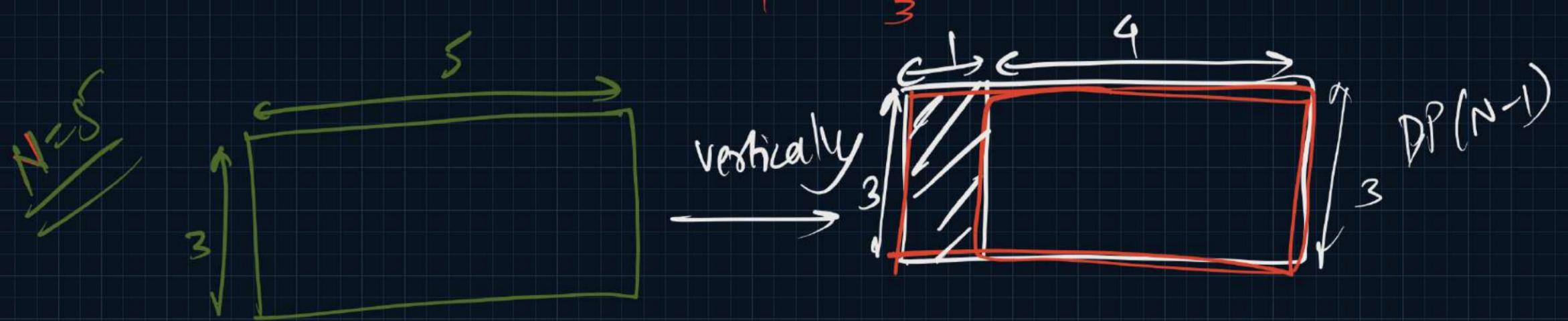


$$DP(N) = DP(N-1) + DP(N-2)$$

↑ vertically ↑ horizontally



$$DP(N=4) = DP(N=3) + DP(N=2)$$

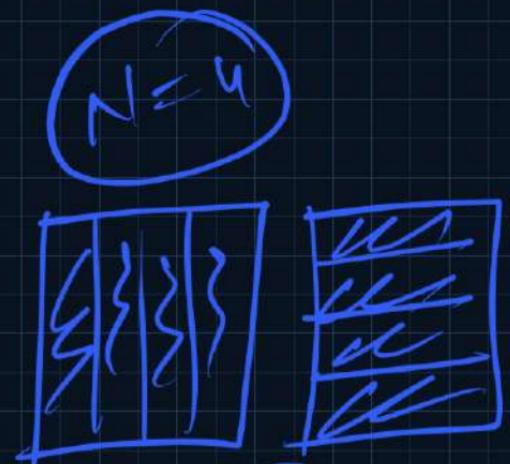


$$DP(N) = \frac{DP(N-1)}{DP(N-M)}$$

$$DP(N-1)$$

$M=4$

Base Cases



If $N < M$ \rightarrow count = ①

If $N = M$ \rightarrow count = ②

```

int mod = 1000000007;
public int memo(int n, int m, int[] dp){
    if(n < m) return 1;
    if(n == m) return 2;
    if(dp[n] != -1) return dp[n];

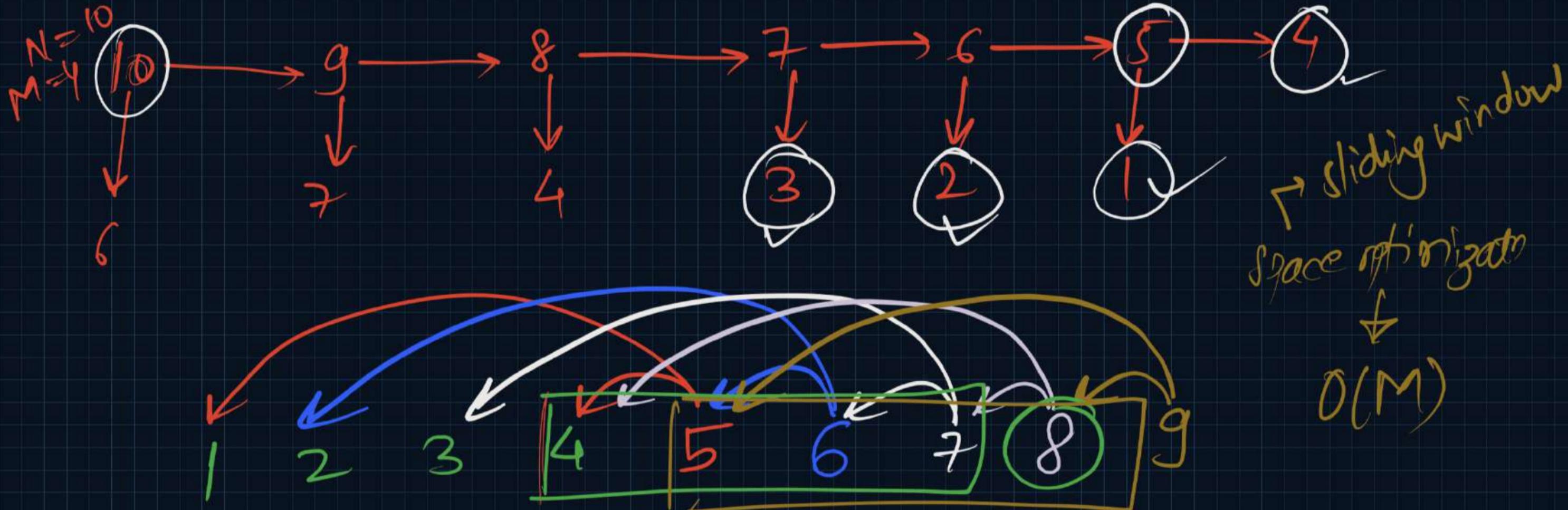
    int ans1 = memo(n - 1, m, dp);
    int ans2 = memo(n - m, m, dp);
    return dp[n] = (ans1 + ans2) % mod;
}

public int countWays(int n, int m)
{
    int[] dp = new int[n + 1];
    Arrays.fill(dp, -1);
    return memo(n, m, dp);
}

```

Time : $\rightarrow O(N)$

Space : $\rightarrow O(N)$
 ↴ R.C.S ↴ DP Table



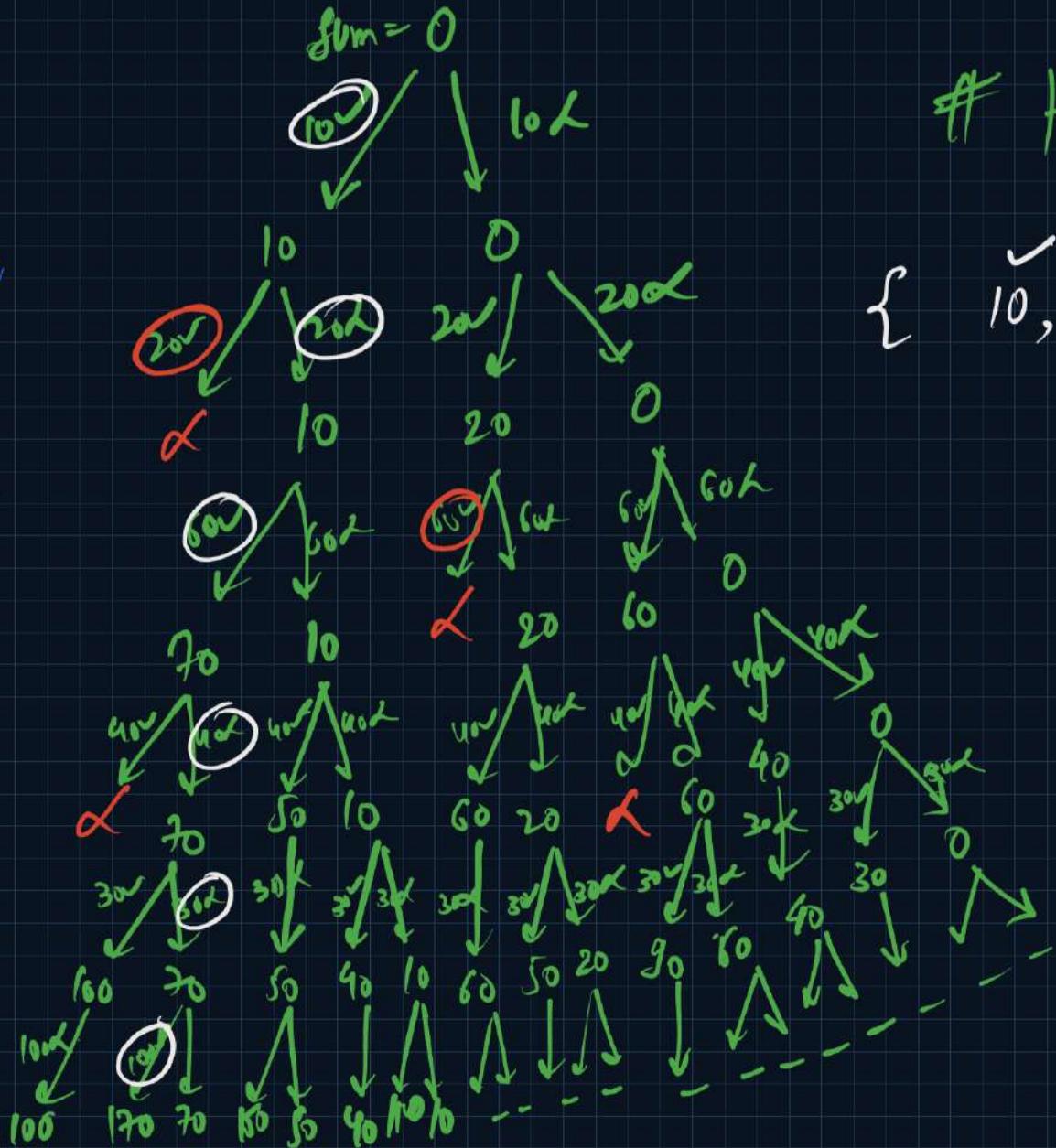
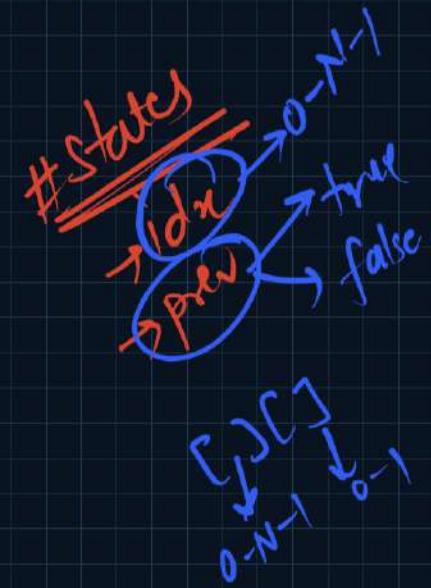
```
public int countWays(int n, int m)
{
    if(n < m) return 1;
    if(n == m) return 2;
    // int[] dp = new int[n + 1];
    // Arrays.fill(dp, -1);
    // return memo(n, m, dp);
```

```
Deque<Integer> dp = new ArrayDeque<>();
for(int i=1; i<m; i++){
    dp.add(1); // DP[N < M]
}
dp.add(2); // DP[N == M]

for(int i=m+1; i<=n; i++){
    int ans = (dp.getFirst() + dp.getLast()) % mod;
    dp.removeFirst();
    dp.addLast(ans);
}

return dp.getLast();
}
```

Time $\rightarrow O(N)$
Space $\rightarrow O(M)$



House Robber

man sum

$$\{ \checkmark 10, \cancel{20}, \checkmark 60, \cancel{40}, \cancel{30}, \checkmark 100 \}$$

```
graph TD; Greedy[Greedy] --> Alternating[Alternating]; Alternating --> fail[fail]
```

```
class Solution {
    public int memo(int[] nums, int idx, int prev, int[][] dp){
        if(idx == nums.length) return 0;
        if(dp[idx][prev] != -1) return dp[idx][prev];

        int yes = (prev == 0) ? (memo(nums, idx + 1, 1, dp) + nums[idx]) : 0;
        int no = memo(nums, idx + 1, 0, dp);

        return dp[idx][prev] = Math.max(yes, no);
    }

    public int rob(int[] nums) {
        int n = nums.length;
        int[][] dp = new int[n + 1][2];
        for(int i=0; i<=n; i++){
            dp[i][0] = -1;
            dp[i][1] = -1;
        }

        return memo(nums, 0, 0, dp);
    }
}
```

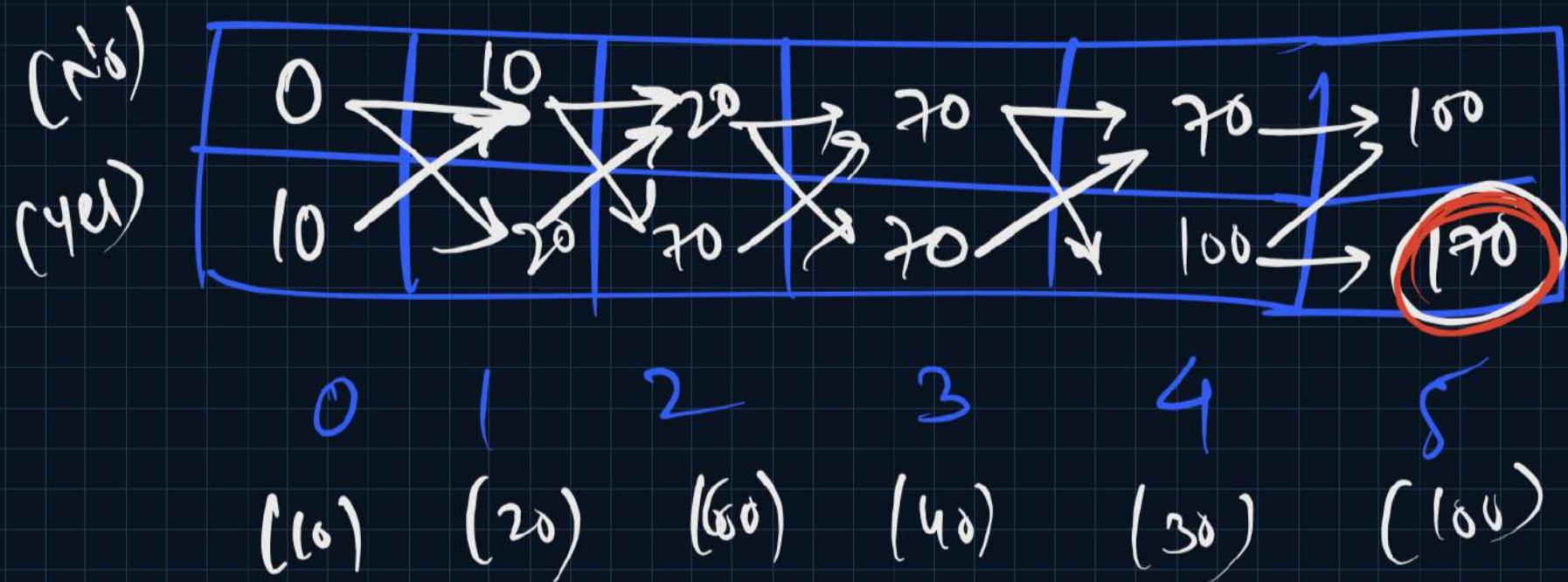
Memo

→ $O(N)$ Time Comp

→ $O(N)$ Space Comp

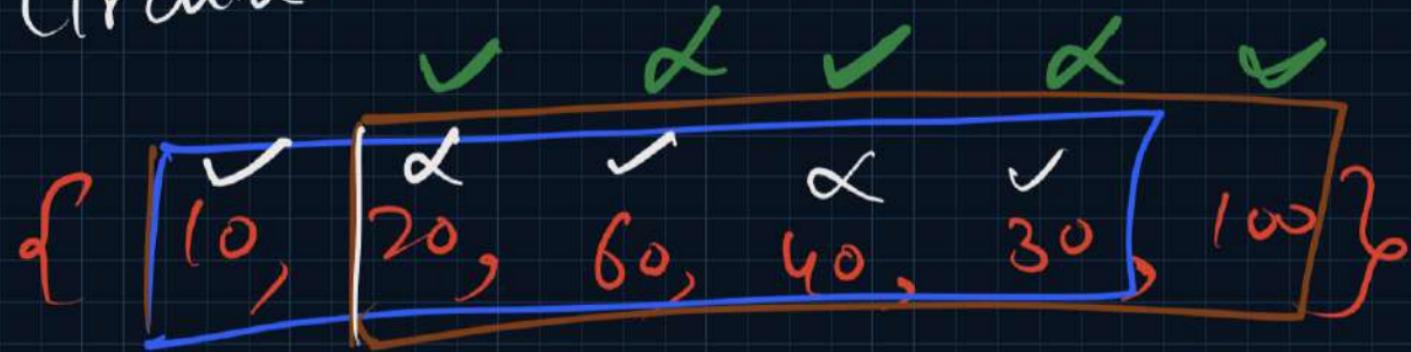
↓
Extra R.C.S

Smaller → Larger



Homework: \Rightarrow Tabulation (Code)

House Robber - Circular



Extra constraint → 10 & 100 are adjacent

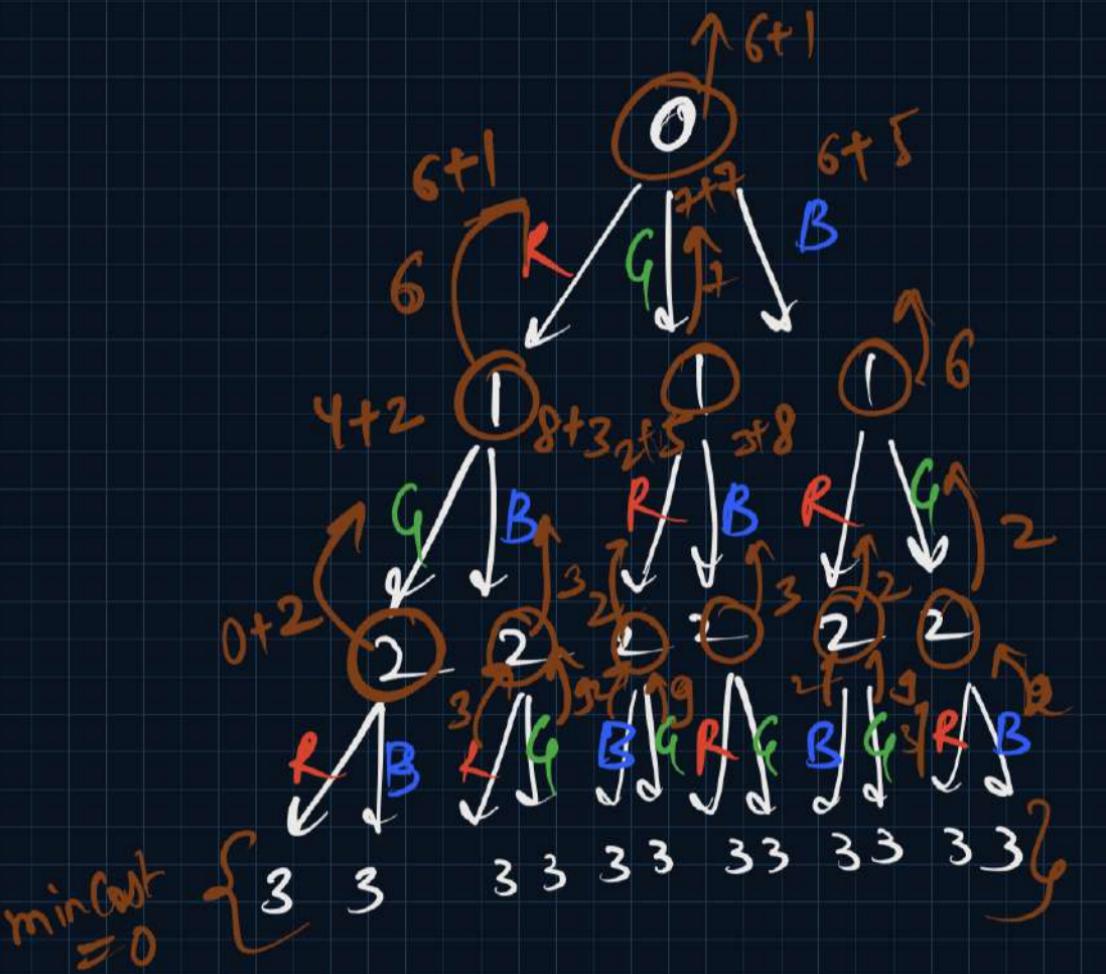
arr[0] & arr[n-1]
are also adjacent!

```
public int memo(int[] nums, int idx, int n, int prev, int[][] dp){  
    if(idx == n + 1) return 0;  
    if(dp[idx][prev] != -1) return dp[idx][prev];  
  
    int yes = (prev == 0) ? (memo(nums, idx + 1, n, 1, dp) + nums[idx]) : 0;  
    int no = memo(nums, idx + 1, n, 0, dp);  
  
    return dp[idx][prev] = Math.max(yes, no);  
}  
  
public int rob(int[] nums, int start, int end){  
    int n = nums.length;  
  
    int[][] dp = new int[n + 1][2];  
    for(int i=0; i<=n; i++){  
        dp[i][0] = -1;  
        dp[i][1] = -1;  
    }  
  
    return memo(nums, start, end, 0, dp);  
}
```

```
public int rob(int[] nums) {  
    if(nums.length == 0) return 0;  
    if(nums.length == 1) return nums[0];  
  
    return Math.max(rob(nums, 0, nums.length - 2)  
                    , rob(nums, 1, nums.length - 1));  
}
```

$$\begin{aligned} \text{TC} &\Rightarrow O(N^2) \\ \text{SC} &\Rightarrow O(N) \end{aligned}$$

(R)	1	5	3	
(B)	5	8	2	
(G)	7	4	9	
0	1	2	3	



```

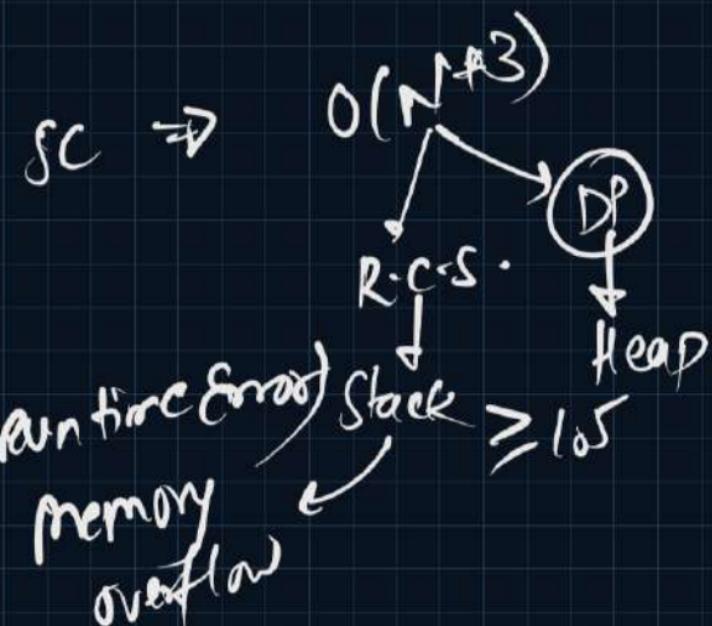
public int helper(int[][] costs, int idx, int prev, int[][] dp){
    if(idx == costs.length) return 0;
    if(prev >= 0 && dp[idx][prev] != -1) return dp[idx][prev];

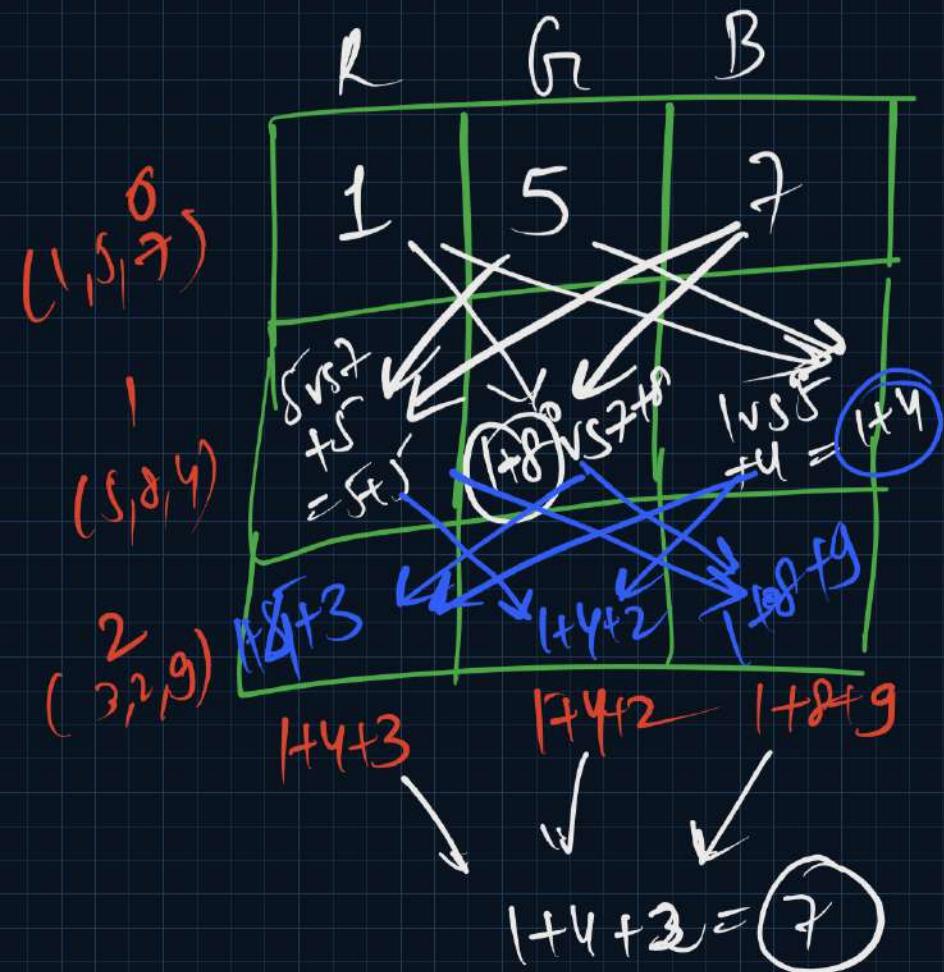
    int ansR = (prev == 0) ? Integer.MAX_VALUE
                          : helper(costs, idx + 1, 0, dp) + costs[idx][0];
    int ansB = (prev == 1) ? Integer.MAX_VALUE
                          : helper(costs, idx + 1, 1, dp) + costs[idx][1];
    int ansG = (prev == 2) ? Integer.MAX_VALUE
                          : helper(costs, idx + 1, 2, dp) + costs[idx][2];

    if(prev == -1)
        return Math.min(ansR, Math.min(ansB, ansG));
    return dp[idx][prev] = Math.min(ansR, Math.min(ansB, ansG));
}

```

Memoization
 $T.C \rightarrow O(N \times 3)$
 $\approx O(N)$





$DP(i, j)$ = $\min_{k=1}^N \{C_{i,k} + DP(i+1, k)\}$

① Storage & meaning
 ② Small or large problem
 ③ Travel & solve
 { Recurrence relation }

$O(N^{i-1})$ = $O(i)$ houses
 paint
 min cost
 such that i^{th} house has
 color j^o

```

int[][] dp = new int[n + 1][3];
dp[0][0] = costs[0][0];
dp[0][1] = costs[0][1];
dp[0][2] = costs[0][2];

for(int i=1; i<n; i++){
    // (0 - i houses) -> ith House Red
    dp[i][0] = costs[i][0] + Math.min(dp[i - 1][1], dp[i - 1][2]);

    // (0 - i houses) -> ith House Blue
    dp[i][1] = costs[i][1] + Math.min(dp[i - 1][0], dp[i - 1][2]);

    // (0 - i houses) -> ith House Green
    dp[i][2] = costs[i][2] + Math.min(dp[i - 1][0], dp[i - 1][1]);
}

return Math.min(dp[n - 1][0], Math.min(dp[n - 1][1], dp[n - 1][2]));

```

$T.C \Rightarrow O(N^3)$
 $= O(N^3)$

$S.C \Rightarrow O(N^3)$
 $= O(N^3)$

\downarrow
 b DP
 "Stack overflow
 is not
 occurring"

```
int prev0 = costs[0][0];
int prev1 = costs[0][1];
int prev2 = costs[0][2];

for(int i=1; i<n; i++){
    // (0 - i houses) -> ith House Red
    int curr0 = costs[i][0] + Math.min(prev1, prev2);

    // (0 - i houses) -> ith House Blue
    int curr1 = costs[i][1] + Math.min(prev0, prev2);

    // (0 - i houses) -> ith House Green
    int curr2 = costs[i][2] + Math.min(prev0, prev1);

    prev0 = curr0; prev1 = curr1; prev2 = curr2;
}

return Math.min(prev0, Math.min(prev1, prev2));
```

"Space Optimization"
Optimization
in Tabulation

→ O(n) Time
O(1) Space

Paint House - II

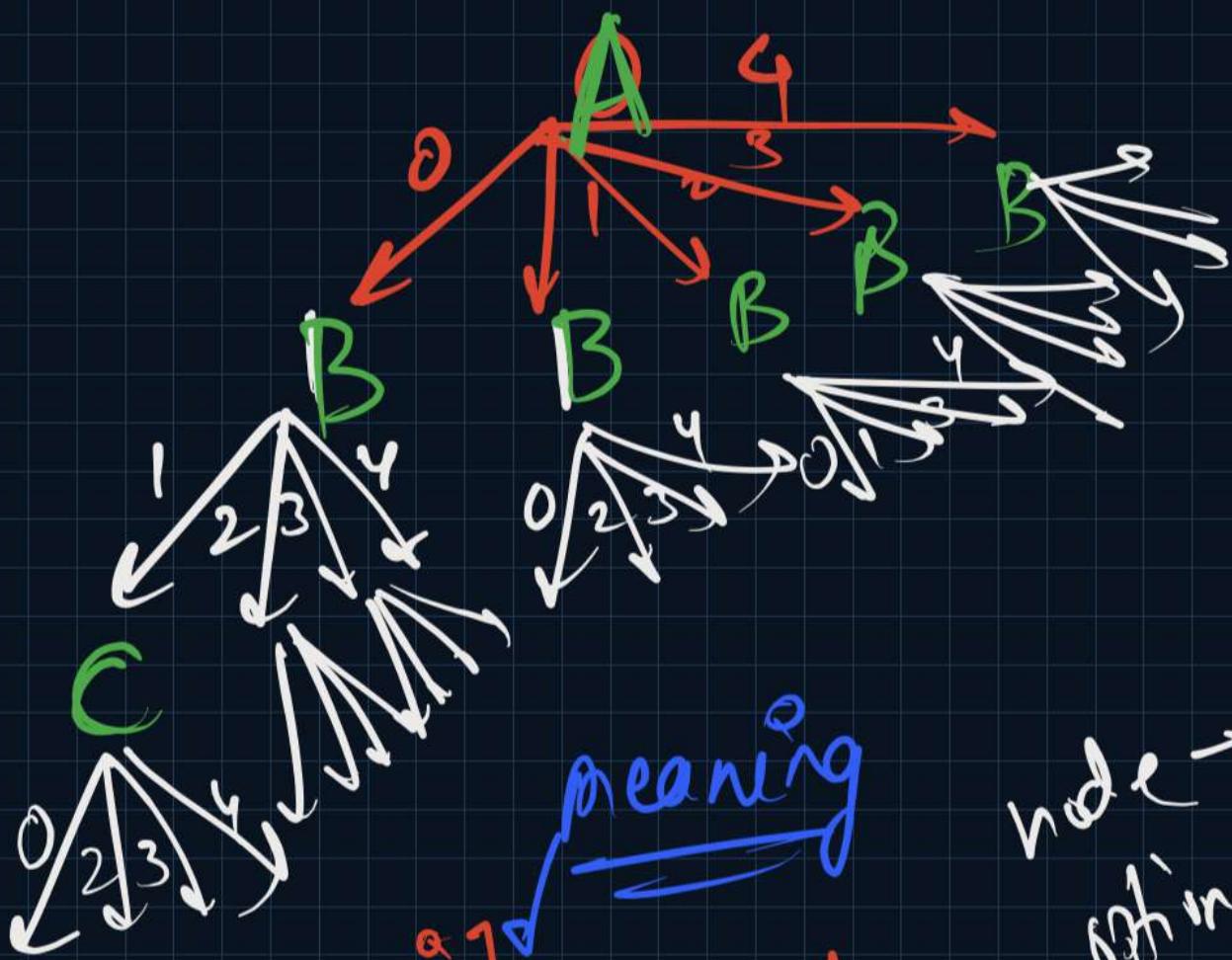
cost
DP

House
(N)

colors = $\{1, 2, \dots, k\}$

	0	1	2	3	4	5	6	7	8
A	2	4	1	3	5				
B	1+6	2+1	2+3	1+4	1+1				
C	1+1+2	1+1+2	1+1+1	1+1+1	1+1+1				
D	1+1+1+3	1+1+1+1	1+1+1+3	1+1+1+4	1+1+1+4				

min cost



$D P[i][j]$
 $= D P[i-1][j]$
 such that
 if house
 has color
 i then
 min cost
 such that
 house
 has color
 j

meaning

node \rightarrow house
 option \rightarrow color

```

int[][] dp = new int[costs.length + 1][k];
for(int c=0; c<k; c++){
    dp[0][c] = costs[0][c];
}

for(int i=1; i<n; i++){
    for(int c=0; c<k; c++){
        int min = Integer.MAX_VALUE;

        // Extracting Min of Previous Row Excluding Our Column
        for(int prev=0; prev<k; prev++){
            if(prev == c) continue;
            min = Math.min(min, dp[i - 1][prev]);
        }

        dp[i][c] = costs[i][c] + min;
    }

    int min = Integer.MAX_VALUE;
    for(int c=0; c<k; c++){
        min = Math.min(dp[n - 1][c], min);
    }
    return min;
}

```

Time Comp : $O(N * K^2)$

space Comp : $O(n * k)$

For calculating every cell value we are looping on the previous row $\rightarrow O(k)$

A 0 1 2 3 4
 $\begin{matrix} 2 \\ 2 \end{matrix}$ $\begin{matrix} 4 \\ 4 \end{matrix}$ $\begin{matrix} 1 \\ 1 \end{matrix}$ $\begin{matrix} 3 \\ 3 \end{matrix}$ $\begin{matrix} 5 \\ 5 \end{matrix}$ 1st min = 1, 2nd min = 2

B 6 2 3 4 1
 $\begin{matrix} 1+6 \\ 1+2 \end{matrix}$ $\begin{matrix} 2+3 \\ 2+3 \end{matrix}$ $\begin{matrix} 4+4 \\ 4+4 \end{matrix}$ $\begin{matrix} 1+1 \\ 1+1 \end{matrix}$ 1st min = 1+1, 2nd min = 1+2

C $\begin{matrix} 2 \\ 1+1+2 \end{matrix}$ $\begin{matrix} 2 \\ 1+1+2 \end{matrix}$ $\begin{matrix} 1 \\ 1+1+1 \end{matrix}$ $\begin{matrix} 1 \\ 1+1+1 \end{matrix}$ $\begin{matrix} 2 \\ 1+2+2 \end{matrix}$ 1st min = 1+1+1
 2nd min = 1+1+1

D $\begin{matrix} 3 \\ 1+1+3 \end{matrix}$ $\begin{matrix} 1 \\ 1+1+1 \end{matrix}$ $\begin{matrix} 3 \\ 1+1+3 \end{matrix}$ $\begin{matrix} 4 \\ 1+1+1+1 \end{matrix}$ $\begin{matrix} 4 \\ 1+1+1+1 \end{matrix}$

```

for(int i=1; i<n; i++){
    int firstMin = Integer.MAX_VALUE;
    int secondMin = Integer.MAX_VALUE;
    for(int prev=0; prev<k; prev++){
        if(dp[i - 1][prev] <= firstMin){
            secondMin = firstMin;
            firstMin = dp[i - 1][prev];
        } else if(dp[i - 1][prev] < secondMin){
            secondMin = dp[i - 1][prev];
        }
    }

    for(int c=0; c<k; c++){
        int min = Integer.MAX_VALUE;
        // Extracting Min of Previous Row Excluding Our Column
        if(dp[i - 1][c] == firstMin)
            dp[i][c] = costs[i][c] + secondMin;
        else dp[i][c] = costs[i][c] + firstMin;
    }
}

```

finding

$O(k)$

1st min & 2nd min
of previous row

calculating current

dp

$O(k)$

$T.C \Rightarrow O(n * (k+k))$
 $= O(n * 2k)$
 $\approx O(n * k)$

$C \Rightarrow O(N * k)$

Paint Fence

n , k colors
 ↓
 fence houses

No more than 2 fences have the same color

$$\underline{n=1 \rightarrow k} \\ k=1 \quad \textcircled{1}$$

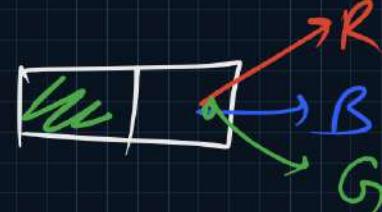
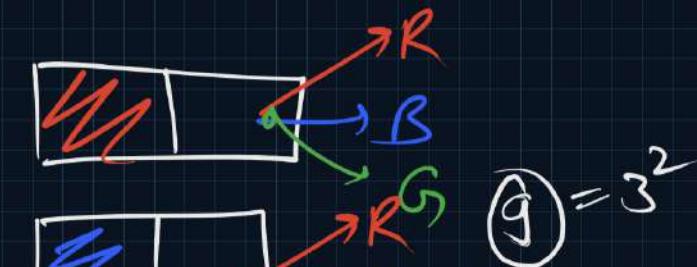
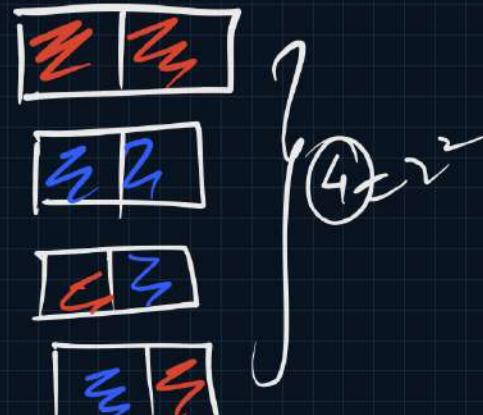
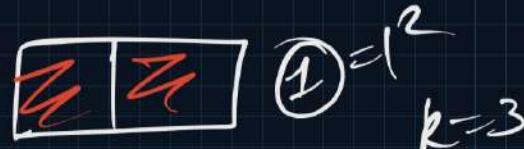
$$\underline{n=2 \rightarrow k^2} \\ k=1$$

$$k=2 \quad \textcircled{2}$$

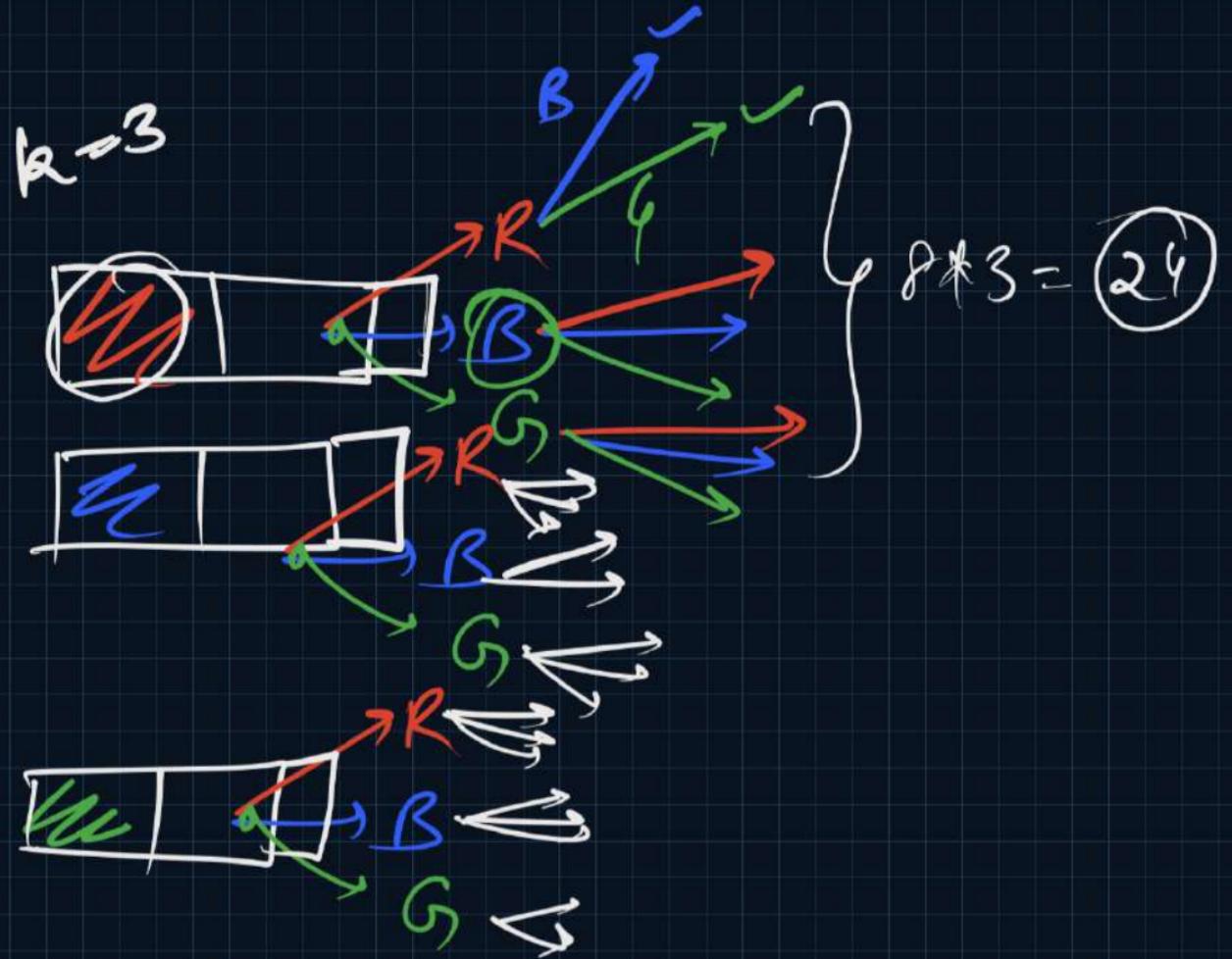
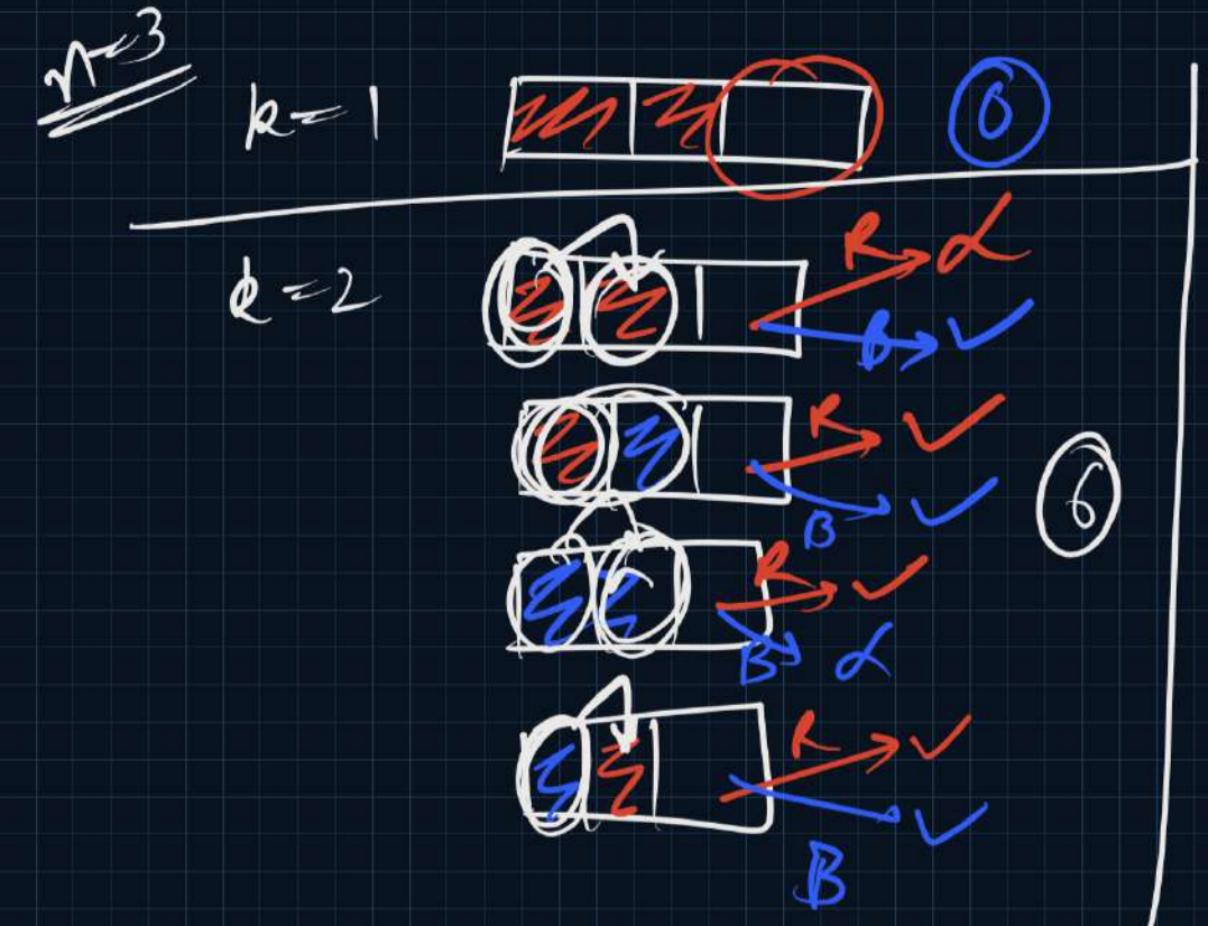
$$k=2$$

$$k=3 \quad \textcircled{3}$$

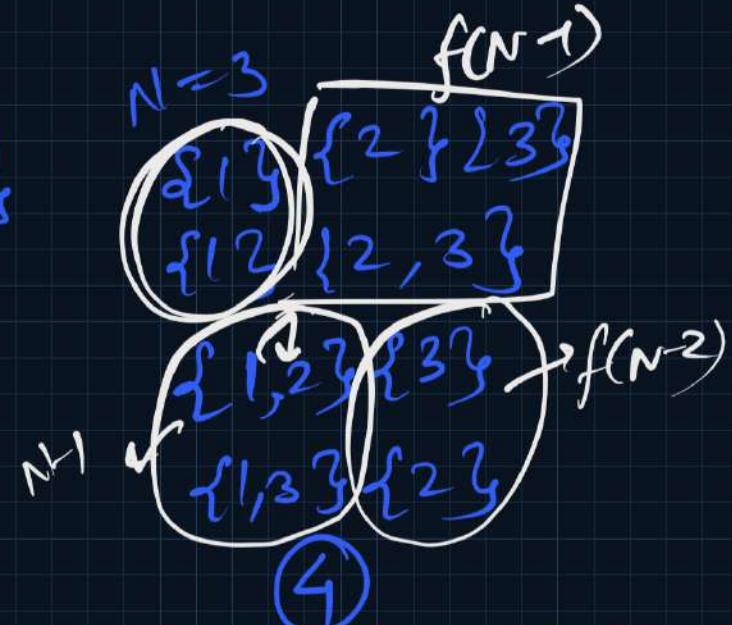
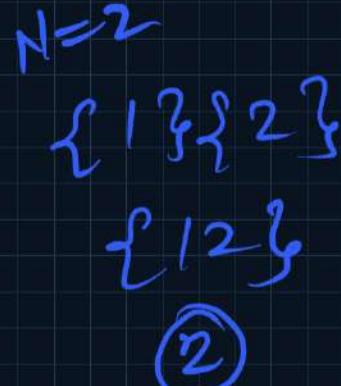
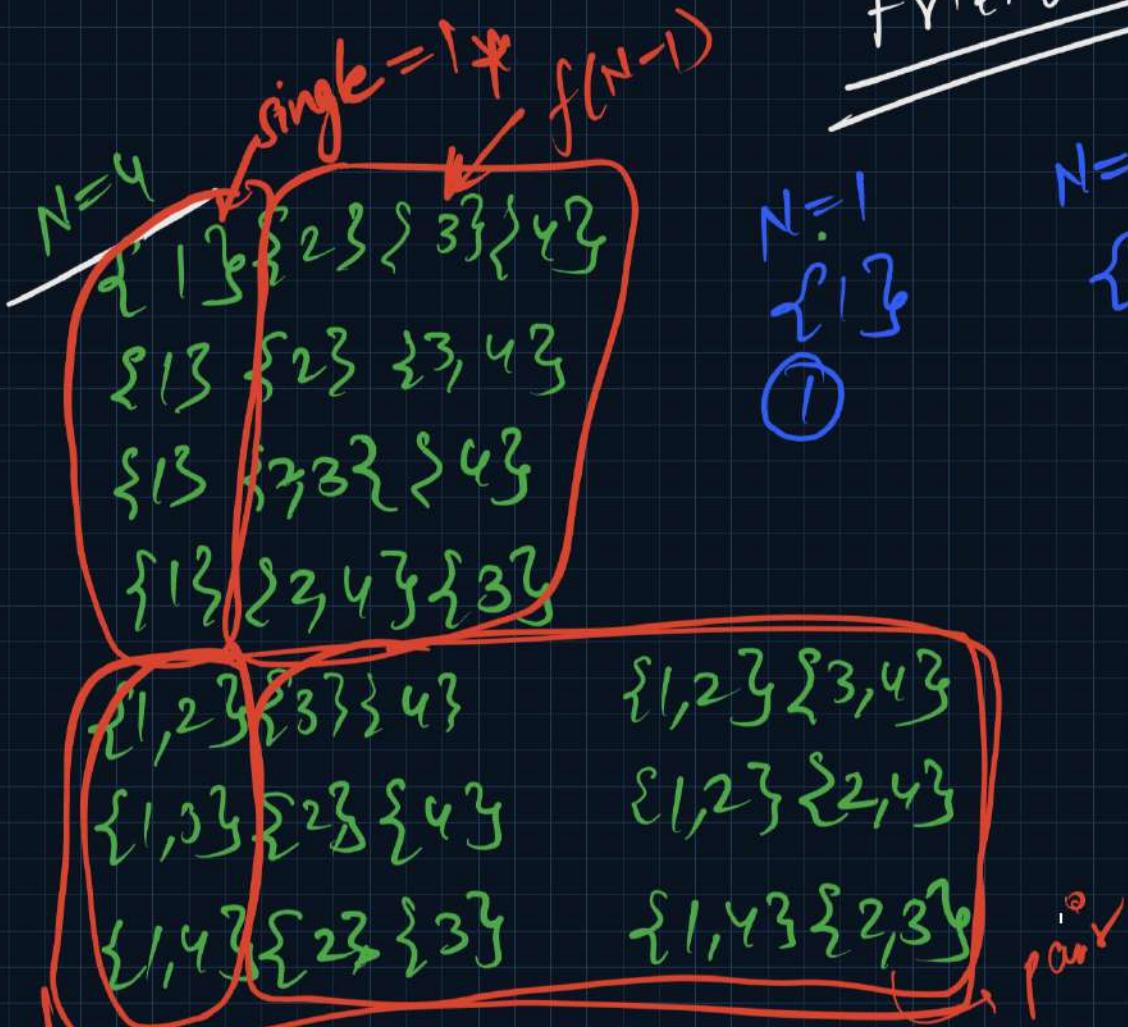
$$\downarrow \\ k=k \quad \textcircled{k}$$



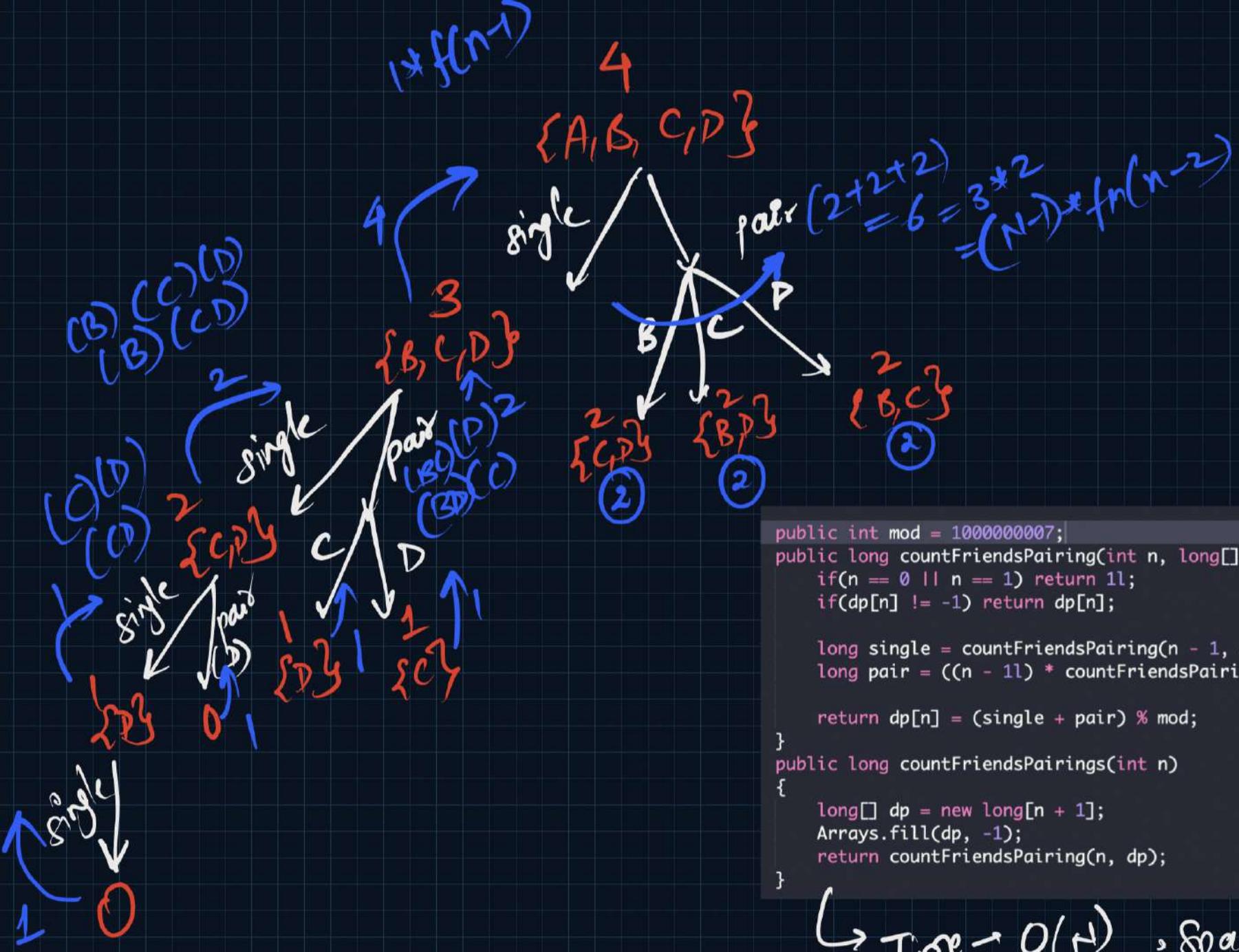
$$k=k \quad \text{count of ways} = k^2 = k*k$$



Friends Pairing



$$f(N) = f(N-1) + (f(N-2) * (N-1))$$



```

public int mod = 1000000007;
public long countFriendsPairing(int n, long[] dp){
  if(n == 0 || n == 1) return 1L;
  if(dp[n] != -1) return dp[n];

  long single = countFriendsPairing(n - 1, dp);
  long pair = ((n - 1L) * countFriendsPairing(n - 2, dp)) % mod;

  return dp[n] = (single + pair) % mod;
}
public long countFriendsPairings(int n)
{
  long[] dp = new long[n + 1];
  Arrays.fill(dp, -1);
  return countFriendsPairing(n, dp);
}
  
```

Space Optimization → Possible → Tabulation
 { Two Pointer }

Ugly No - 1

$$N = p_1^a * b_2^b * b_3^c \dots$$

$$N = 2^a * 3^b * 5^c$$

where $\{a, b, c \in [0, \infty)\}$

rotation

where $N_1 = 2^2 \times 3^2 \times 5^2 = 1$ ✓ true

$N_2 = 2^2 \times 3^2 \times 7^2 \neq 1$ false

Prime Factorization
↓
Worst $O(\sqrt{N})$
Avg $O(\log_2 N + \log_3 N)$

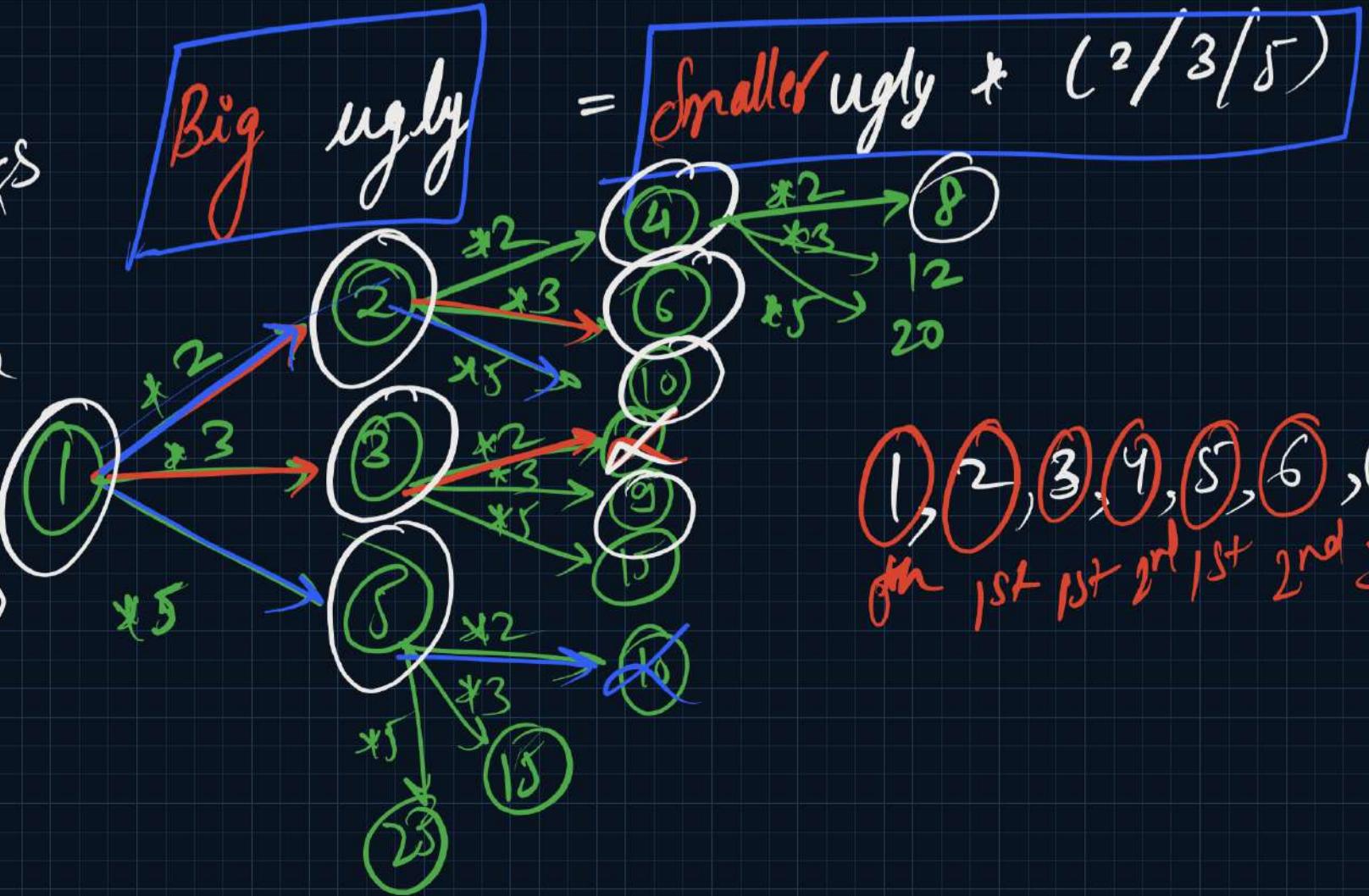
Udy No - 11

① Brute force \rightarrow loop * prime fact $\rightarrow O(N \times \sqrt{N})$ TC
 $N^{3/2}$

② 1, 2, 3, 4, 5, 6, 8, 10, 12, 15
 2×2 , 2×3 , $2 \times 2 \times 2$, 2×5 , $2 \times 2 \times 3$, 3×5

It is neither
BFS nor DFS

There can be
repetitions
 $2^2 \times 3 = 3 \times 2^2$



Dynamic Programming

→ Lecture - 8 9 AM to 12 PM Saturday
30 April

- ① Paint fence
- ② Ugly No - II
- ③ Ugly No - III
- ④ Egg Jump
- ⑤ Jump Game - All Paths

Paint Fence

(n houses, k colors)

no more than 2 houses
have the same color

$$\# \overbrace{N=1}^{k=1} \quad \boxed{\text{RR}} \quad ①$$

$$\# \overbrace{N=2}^{k=1} \quad \boxed{\text{RR}} \quad \boxed{\text{RR}} \quad ①$$

$$k=2 \quad \boxed{\text{R}}, \quad \boxed{\text{B}} \quad ②$$

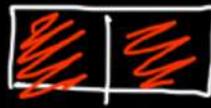
$$k=2 \quad \boxed{\text{R}} \quad \boxed{\text{B}} \quad ③$$

$$k=3 \quad \boxed{\text{R}}, \quad \boxed{\text{B}}, \quad \boxed{\text{G}} \quad ④$$

$$\vdots \downarrow \quad k=D \quad \circled{k}$$

$$\left. \begin{array}{l} \boxed{\text{R}} \quad \boxed{\text{B}} \\ \boxed{\text{B}} \quad \boxed{\text{R}} \\ \boxed{\text{G}} \quad \boxed{\text{G}} \end{array} \right\} ④$$

$N=2$
 $k=3$



⑨



$N=2$
 k

$\rightarrow k^2$

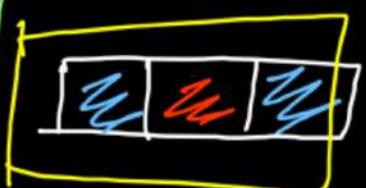
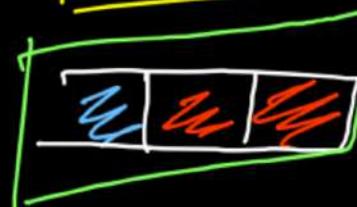
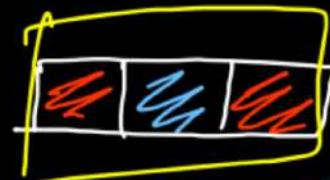
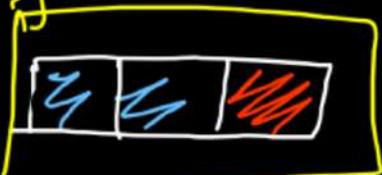
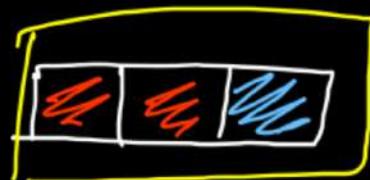
$N=3$

$k=1$



$R=2$ ⑥

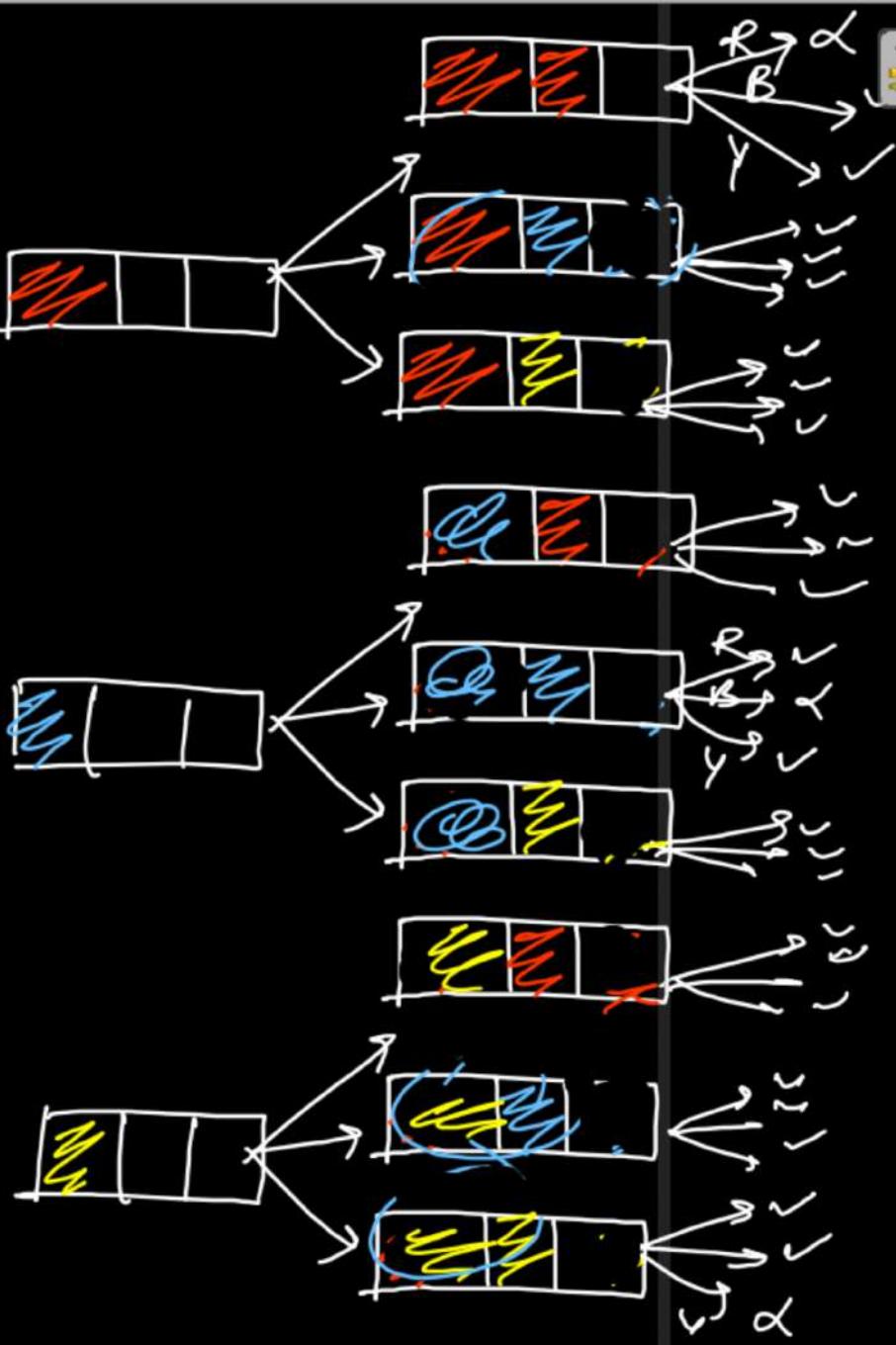
$fd(n,k)$



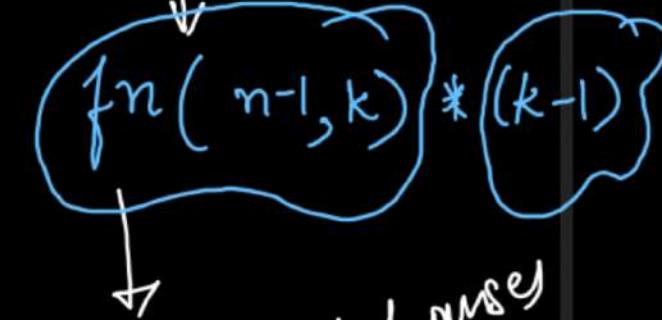
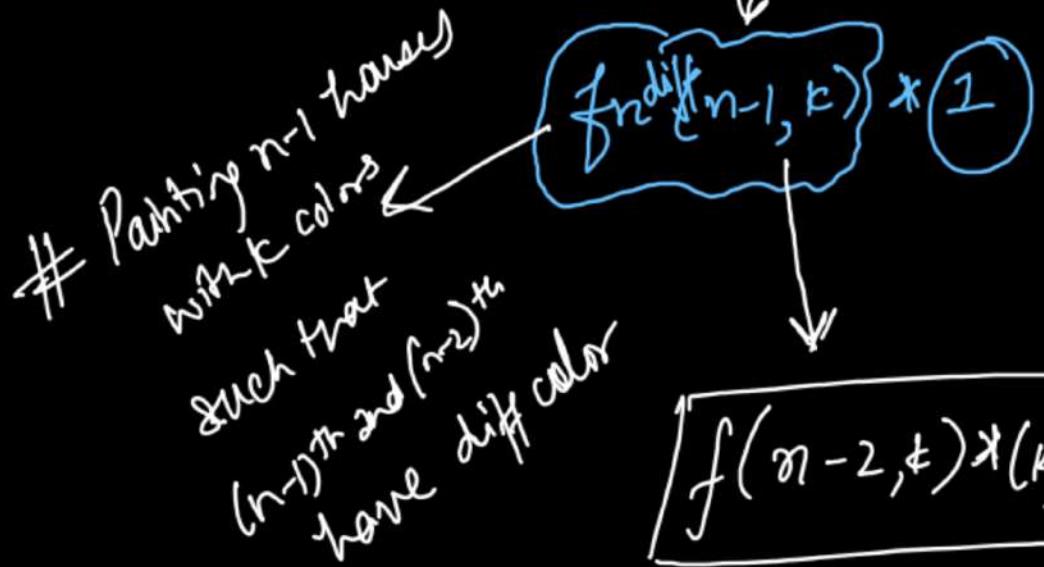
$k=3$

24

$fs(n,k)$



$$f_n(n, k) = \underbrace{f_{n \text{ same}}(n, k)}_{\substack{n \& n+1 \\ \text{last 2 houses}}} + \underbrace{f_{n \text{ diff}}(n, k)}_{\substack{n \& n+1 \\ \text{last 2 houses} \\ \text{have diff color}}}$$



$$f(n, k) = \boxed{f(n-2, k) + f(n-1, k) * (k-1)}$$

\Rightarrow we are painting
 n^{th} house with
any color other
than $(n-1)^{\text{th}}$ house

Tabulation

(last 2 houses)
Same

	$N=1$	$N=2$	$N=3$	$N=4$
diff		3	3×2	18
same	3	3×2	$(3+3 \times 2) \times 2 = 18$	$(18+6) \times 2 = 48$

⋮
⋮
⋮
⋮
⋮

⋮
⋮
⋮
⋮
⋮

$k=3$

~~TC $\Rightarrow O(n)$~~
~~SC $\Rightarrow O(n)$~~ \rightarrow space can be
optimized

```
public class Solution {
    public int memo(int n, int k, int[] dp){
        if(n == 1) return k;
        if(n == 2) return k * k;
        if(dp[n] != -1) return dp[n];

        int ans = (memo(n - 1, k, dp) + memo(n - 2, k, dp)) * (k - 1);
        return dp[n] = ans;
    }

    public int numWays(int n, int k) {
        if(n == 1) return k;
        if(n == 2) return k * k;
        if(k == 1) return 0;

        int[] dp = new int[n + 1];
        Arrays.fill(dp, -1);
        return memo(n, k, dp);
    }
}
```



0.7x

```
public int numWays(int n, int k) {  
    if(n == 1) return k;  
    if(n == 2) return k * k;  
    if(k == 1) return 0;  
  
    int[] same = new int[n + 1];  
    int[] diff = new int[n + 1];  
    same[2] = k; diff[2] = k * (k - 1);  
  
    for(int i=3; i<=n; i++){  
        same[i] = diff[i - 1];  
        diff[i] = (same[i - 1] + diff[i - 1]) * (k - 1);  
    }  
  
    return same[n] + diff[n];  
}
```



0.7 x

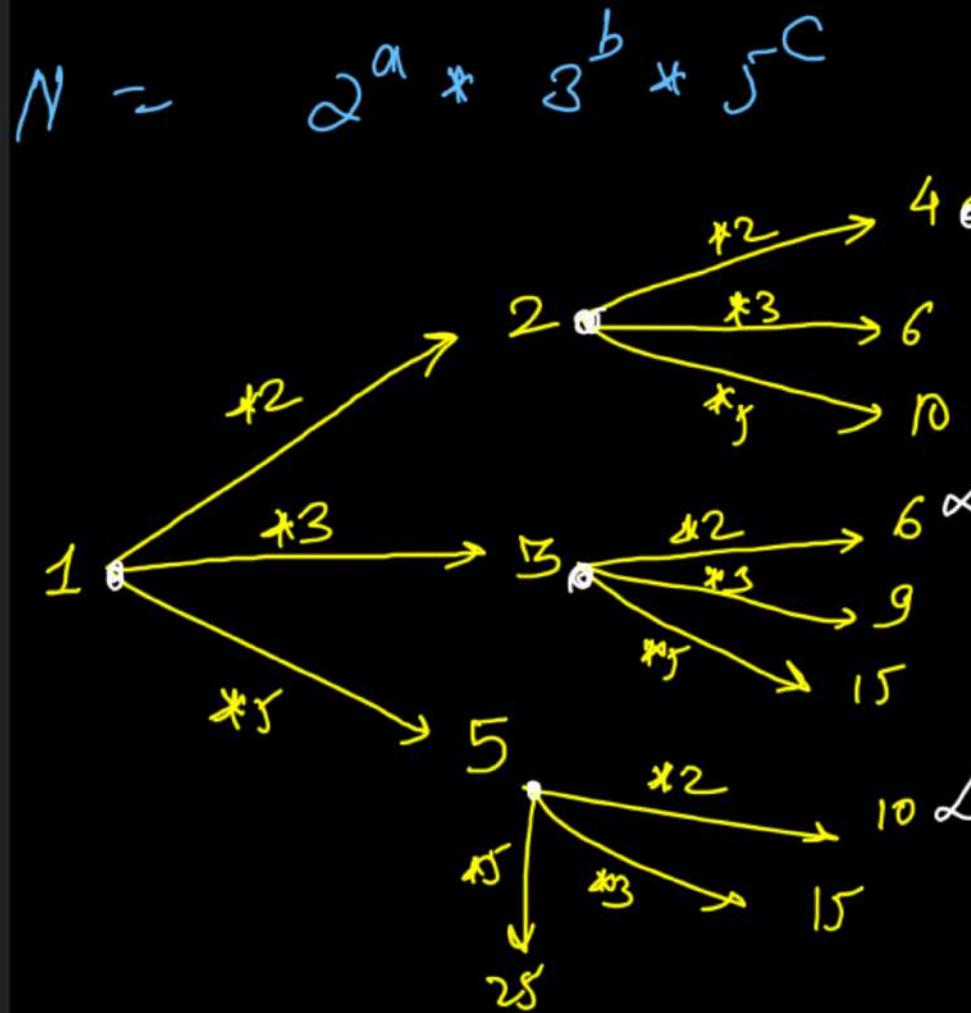
```
public int numWays(int n, int k) {  
    if(n == 1) return k;  
    if(n == 2) return k * k;  
    if(k == 1) return 0;  
  
    int same = k;  
    int diff = k * (k - 1);  
  
    for(int i=3; i<=n; i++){  
        int newSame = diff;  
        int newDiff = (same + diff) * (k - 1);  
  
        same = newSame; diff = newDiff;  
    }  
  
    return same + diff;  
}
```



0.7 x

~~# Ugly No - 11~~

$\frac{1^0, 2, 3, 4, 5, 6, 8}{10}$



Recursion
DFS or BFS

Overlapping subproblems (DP)

Optimal substructure (Faith)

Bigger Ugly = $\min_{i=2} \text{Ugly} * (2/3/5)$

~~ptr 2~~
~~5*2~~

4th
9th 3
4th 3

~~ptr 5~~
~~1*5~~

①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧, ⑨, ⑩
1st 2nd 3rd 4th 5th 6th 7th 8th 9th 10th

```
// O(N) Time, O(N) Space
public int nthUglyNumber(int n) {
    if(n == 1) return 1;

    // Pointers Pointing to Indices not Values
    int ptr2 = 0, ptr3 = 0, ptr5 = 0;

    ArrayList<Integer> ugly = new ArrayList<>(); // visited array
    ugly.add(1); // to add the 1st ugly no at index 0

    for(int i=1; i<n; i++){
        int a = ugly.get(ptr2) * 2;
        int b = ugly.get(ptr3) * 3;
        int c = ugly.get(ptr5) * 5;

        int min = Math.min(a, Math.min(b, c));
        ugly.add(min);

        if(min == a) ptr2++;
        if(min == b) ptr3++;
        if(min == c) ptr5++;
    }

    return ugly.get(n - 1);
}
```



0.7 x

```
// O(N * Log N) Time, O(N) Space
public int nthUglyNumber(int n) {
    if(n == 1) return 1;

    PriorityQueue<Long> q = new PriorityQueue<>();
    q.add(1L);
    HashSet<Long> vis = new HashSet<>();

    int idx = 0;
    while(q.size() > 0){
        long min = q.remove();
        if(vis.contains(min) == true)
            continue;

        idx++;
        if(idx == n) return (int)min;

        vis.add(min);
        q.add(min * 2L);
        q.add(min * 3L);
        q.add(min * 5L);
    }

    return 1;
}
```

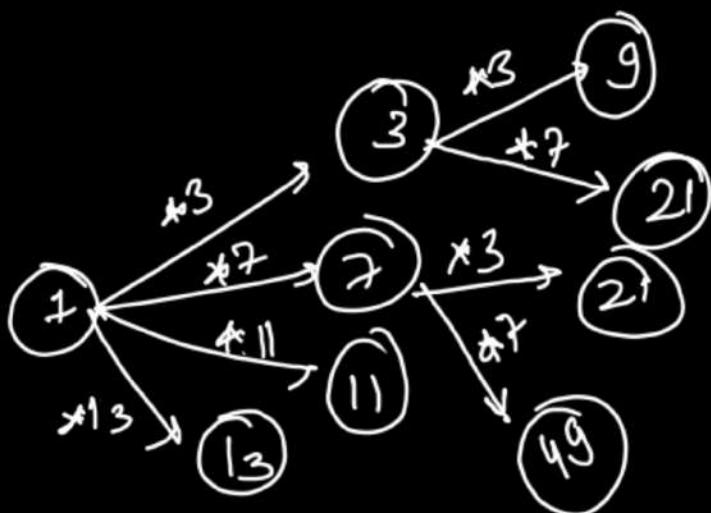


0.5 x

~~Diaper Ugly No~~

Q Input prime = { 3, 7, 11, 13 } Q ugly ?
 $N=7$

$\text{extra} = \alpha(F)$ for = { 4, 3, 2, 2 }



space
comp

Ans (Extra) = $\alpha(N)$
1st, 2nd, 3rd, 4th, 5th, 6th, 7th

$O(N + k)$
for pointers index

```
// O(N * K) Time, O(N + K) Space
public int nthSuperUglyNumber(int n, int[] primes) {
    int[] ptr = new int[primes.length];

    ArrayList<Integer> ugly = new ArrayList<>();
    ugly.add(1); // Add 1st Ugly No at Index 0

    for(int i=1; i<n; i++){
        // Finding the next Smallest Ugly No
        int min = Integer.MAX_VALUE;
        for(int j=0; j<primes.length; j++)
            min = Math.min(min, ugly.get(ptr[j]) * primes[j]);

        ugly.add(min);

        // Updating All Pointers Pointing to Min
        for(int j=0; j<primes.length; j++)
            if(ugly.get(ptr[j]) * primes[j] == min)
                ptr[j]++;
    }

    return ugly.get(n - 1);
}
```



0.5 x

Dynamic Programming - Lecture ⑨

9 AM - 12 PM

Sunday, 1 May

- Count of ways {
 - Coin Change Permutations
 - Coin Change Combinations
- minimum steps {
 - Minimum Coin change
 - Indian Coin change

#Coin change → Minimum coins (Indian)

eg:

{1, 2, 5, 10}

target = 79

$$\underbrace{(10 * 7)}_{\downarrow} + \underbrace{(5 * 1)}_{\downarrow} + \underbrace{(2 * 2)}_{\downarrow}$$

$$7 + 1 + 2 = 10 \text{ coins}$$

Why greedy is working?

uniformity
multiples of

eg:

{1, 2, 5, 10, 20, 50, 100, 500, 2000}

target = 2499

$$2000 * 1 + 100 * 4 + 50 * 1$$

$$+ 20 * 2 + 5 * 1 + 2 * 2$$

$$1 + 4 + 1 + 2 + 1 + 2 = 11 \text{ coins}$$

indian
denomination
↳ greedy will
also work

```
static List<Integer> minPartition(int target)
{
    List<Integer> res = new ArrayList<>();
    int[] coins = {1, 2, 5, 10, 20, 50, 100, 200, 500, 2000};
    int count = 0;

    for(int i=coins.length-1; i>=0; i--){
        while(target - coins[i] >= 0){
            res.add(coins[i]);
            target -= coins[i];
        }

        if(target == 0) break;
    }

    return res;
}
```

Time Comp

$$\hookrightarrow \frac{N}{2000} + \frac{N}{500} + \frac{N}{200} + \dots + \frac{N}{1}$$

$$\Rightarrow O(N + N + N + \dots + N) = O(N \times 10) \\ \approx O(N)$$

Space Complexity \rightarrow $O(1)$ extra space
 $O(n)$ output space

322. Coin Change → Minimum

$$\text{Coins} = \{2, 3, 7\} \quad \text{target} = 8$$

~~Greedy~~ target = $8 - 7 = 1$ } no answer possible
target

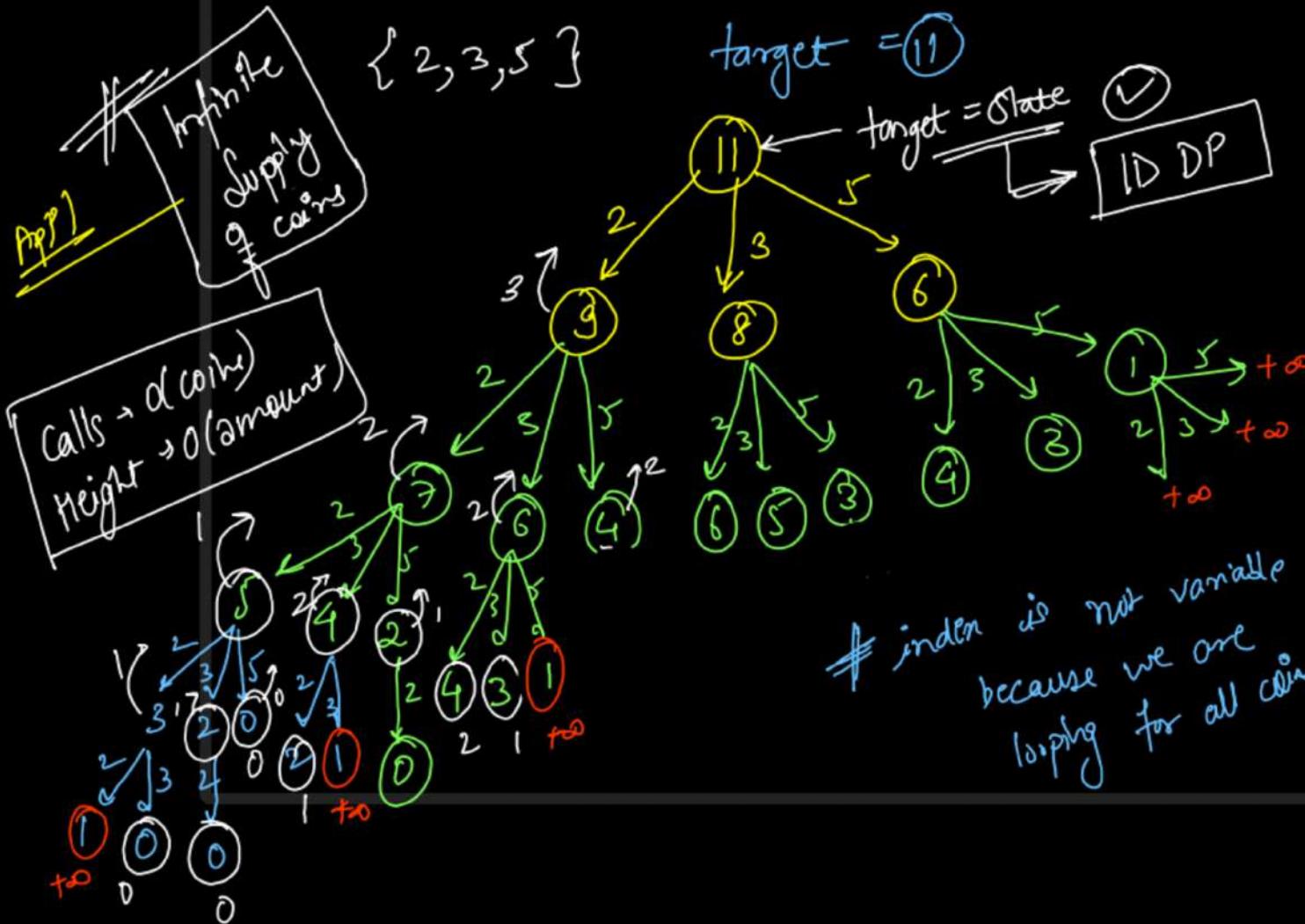
not the biggest coin

Actualized

$$\begin{aligned} \text{target} &= 8 - 3 = 5 \\ 5 - 3 &= 2 \\ 2 - 2 &= 0 \end{aligned}$$

target

ans = 3



index is not variable
 because we are
 looping for all coins

```
// Time Complexity: O(Amount * Coins), Space Complexity: O(Amount)
public int memo(int amount, int[] coins, int[] dp){
    if(amount == 0) return 0;
    if(dp[amount] != -1) return dp[amount];

    int minCoins = Integer.MAX_VALUE;
    for(int i=0; i<coins.length; i++){
        if(amount - coins[i] >= 0){
            minCoins = Math.min(minCoins, memo(amount - coins[i], coins, dp));
        }
    }

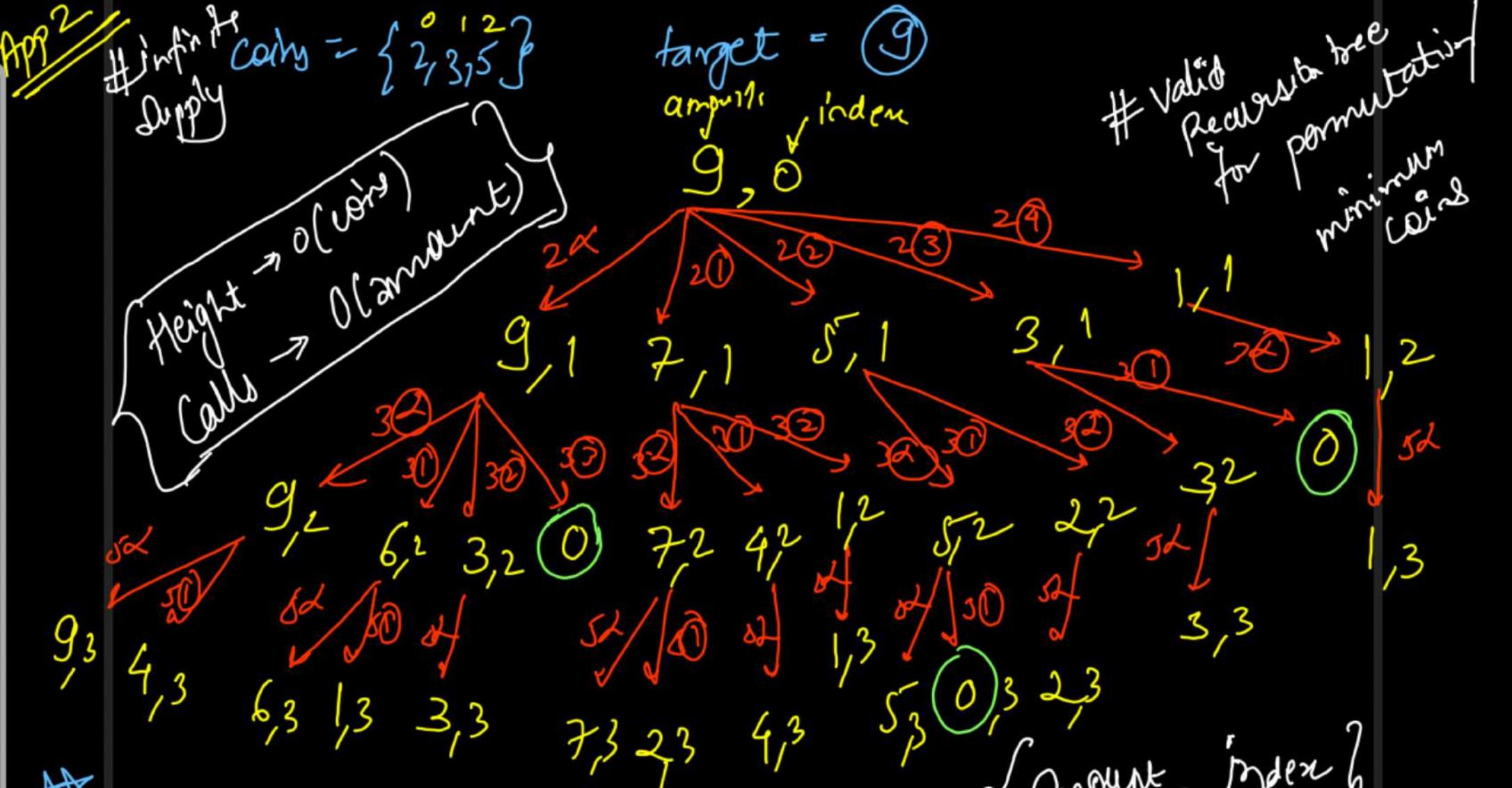
    if(minCoins < Integer.MAX_VALUE) minCoins += 1;
    return dp[amount] = minCoins;
}

public int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, -1);

    int ans = memo(amount, coins, dp);
    return (ans == Integer.MAX_VALUE) ? -1 : ans;
}
```



0.5 x



There is a edge for 0 even also

{
 |
 / states
 \ amount, index

```
// Time Complexity: O(Amount * Coins), Space Complexity: O(Amount * Coins)
public int memo(int amount, int idx, int[] coins, int[][] dp){
    if(amount == 0) return 0;
    if(idx == coins.length) return Integer.MAX_VALUE;
    if(dp[amount][idx] != -1) return dp[amount][idx];

    int minCoins = Integer.MAX_VALUE;
    for(int coin=0; amount >= coins[idx]*coin; coin++){
        int ans = memo(amount - coins[idx] * coin, idx + 1, coins, dp);
        if(ans < Integer.MAX_VALUE) ans += coin;
        minCoins = Math.min(minCoins, ans);
    }

    return dp[amount][idx] = minCoins;
}

public int coinChange(int[] coins, int amount) {
    int[][] dp = new int[amount + 1][coins.length];
    for(int i=0; i<=amount; i++){
        for(int j=0; j<coins.length; j++){
            dp[i][j] = -1;
        }
    }

    int ans = memo(amount, 0, coins, dp);
    if(ans == Integer.MAX_VALUE) return -1;
    return ans;
}
```

App③

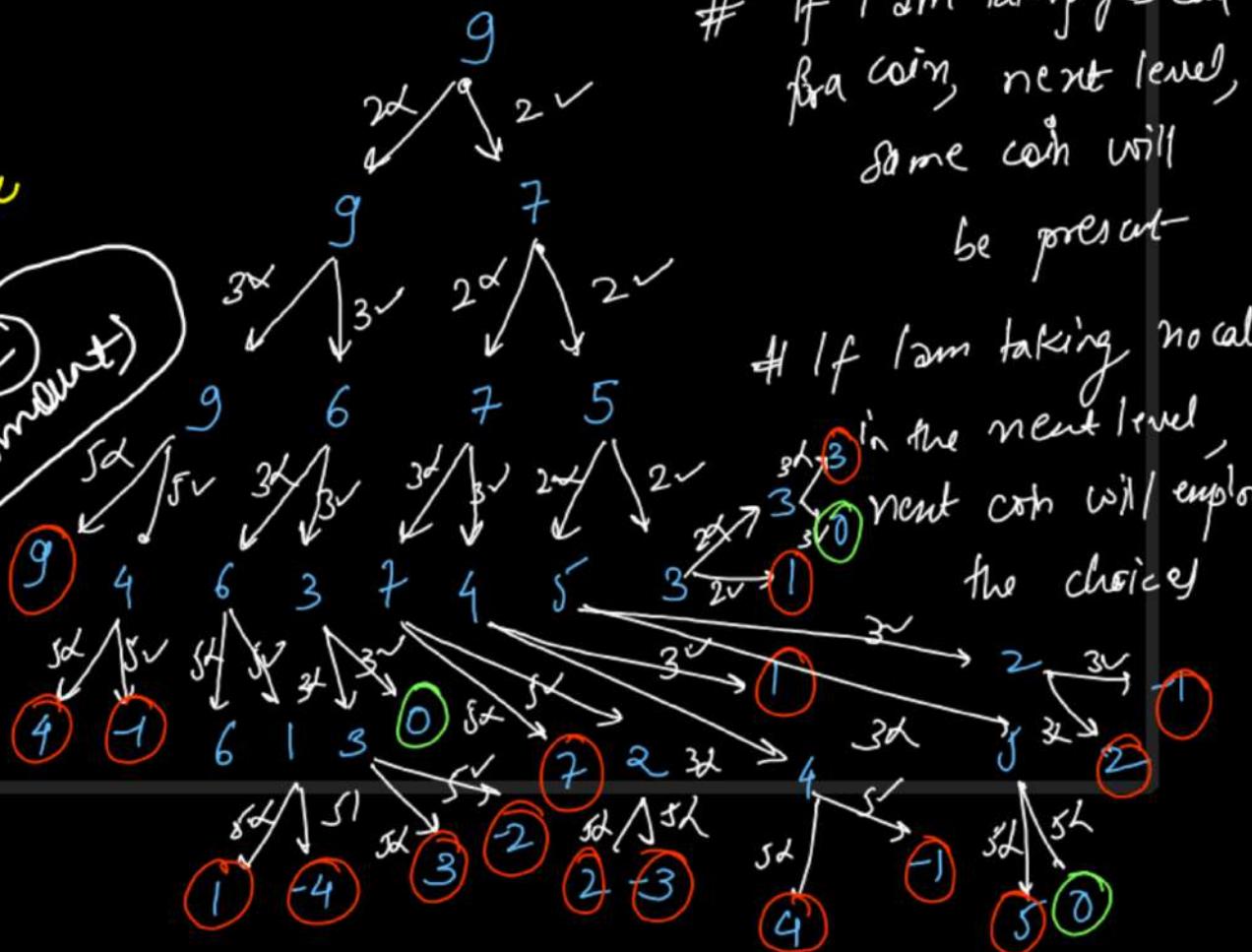
{2, 3, 5}

target = 9

↑ small nt

States → index

Call → ②
Height → demand



```
// Time Complexity: O(Amount * Coins), Space Complexity: O(Amount * Coins)
public int memo(int amount, int idx, int[] coins, int[][] dp){
    if(amount < 0) return Integer.MAX_VALUE;
    if(amount == 0) return 0;
    if(idx == coins.length) return Integer.MAX_VALUE;
    if(dp[amount][idx] != -1) return dp[amount][idx];

    int yes = memo(amount - coins[idx], idx, coins, dp);
    if(yes != Integer.MAX_VALUE) yes += 1;

    int no = memo(amount, idx + 1, coins, dp);
    |
    return dp[amount][idx] = Math.min(yes, no);
}

public int coinChange(int[] coins, int amount) {
    int[][] dp = new int[amount + 1][coins.length];
    for(int i=0; i<=amount; i++){
        for(int j=0; j<coins.length; j++){
            dp[i][j] = -1;
        }
    }

    int ans = memo(amount, 0, coins, dp);
    if(ans == Integer.MAX_VALUE) return -1;
    return ans;
}
```



0.5 x

Tabulation

{ 2, 3, 5 }

target = 7

0	+∞	i	1	2	1	2	2
0	1	2	3	4	5	6	7



smaller
problem

larger
problem

$dp[i] = \min$ using to form
target ?

1D array
(array)
(is a must)
space opt
↓
Not possible

```
// Time -> O(Amount * Coin), Space -> O(Amount)
public int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, Integer.MAX_VALUE);
    dp[0] = 0;

    for(int i=1; i<=amount; i++){
        for(int coin: coins){
            if(i - coin >= 0 && dp[i - coin] != Integer.MAX_VALUE)
                dp[i] = Math.min(dp[i], dp[i - coin] + 1);
        }
    }

    return (dp[amount] == Integer.MAX_VALUE) ? -1 : dp[amount];
}
```



0.7 x

Coin change Permutation vs Combinations

coins = {2, 3, 5}

target = ⑩

Permutations

④

{2, 2, 2, 2, 2} {5, 5}

{2, 2, 3, 3} {2, 3, 3} {2, 3, 3, 2}

{3, 2, 2, 3} {3, 2, 3, 2} {3, 3, 2, 2}

{2, 2, 3, 5} {2, 2, 5, 3} {3, 2, 5}
{3, 5, 2} {5, 2, 3} {5, 3, 2}

⑤

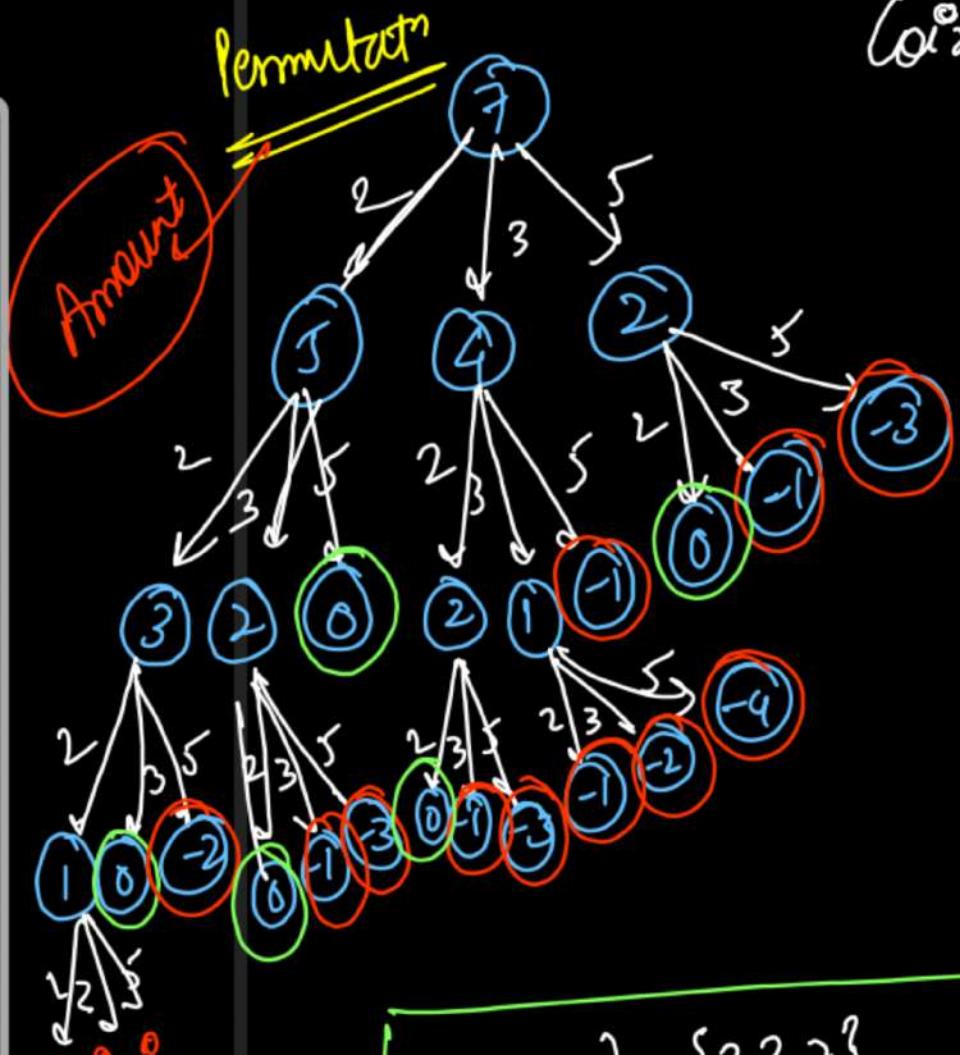
Combinations

{2, 2, 2, 2, 2} {2, 2, 3, 3}

{5, 5} {2, 3, 5}

$$3! = 6$$

$$\frac{4!}{2! 2!} = 6$$



$$\text{Color} \rightarrow \{2, 3, 5\}$$



$$\boxed{\begin{array}{cc} \{1, 2, 3\} & \{2, 3, 2\} \\ \{3, 2\} & \{1, 3\} \\ \{5, 2\} \end{array}}$$

Dynamic Programming lecture ⑩

3 May 2022,
8 PM - 11 PM

Coin change

→ Permutations (377)

→ Combinations (518)

+

Knapsack Variations

→ 0/1 Knapsack

→ Unbounded Knapsack

{ QFG }

Permutation

```
class Solution {
    public int memo(int amount, int[] coins, int[] dp) {
        if (amount < 0) return 0;
        if (amount == 0) return 1;
        if (dp[amount] != -1) return dp[amount];

        int perm = 0;
        for (int i = 0; i < coins.length; i++) {
            perm += memo(amount - coins[i], coins, dp);
        }

        return dp[amount] = perm;
    }

    public int combinationSum4(int[] coins, int amount) {
        int[] dp = new int[amount + 1];
        Arrays.fill(dp, -1);

        return memo(amount, coins, dp);
    }
}
```

Combination

```
// Time - O(Amount * Coins), Space - O(Amount * Coins): 2D DP + RCS
public int memo(int amount, int lastCoin, int[] coins, int[][] dp) {
    if (amount < 0) return 0;
    if (amount == 0) return 1;
    if (dp[amount][lastCoin] != -1) return dp[amount][lastCoin];

    int perm = 0;
    for (int coin = lastCoin; coin < coins.length; coin++) {
        perm += memo(amount - coins[coin], coin, coins, dp);
    }

    return dp[amount][lastCoin] = perm;
}

public int change(int amount, int[] coins) {
    int[][] dp = new int[amount + 1][coins.length + 1];
    for (int i=0; i<=amount; i++){
        for (int j=0; j<=coins.length; j++){
            dp[i][j] = -1;
        }
    }

    return memo(amount, 0, coins, dp);
}
```

Tabulation

Permutation ways

Digitized by srujanika@gmail.com

Ways →

Arnold
(ndem)

11



2

3

1

2

1

2

5

{ "2" }

13"

22"}

2" } { "22"
"1" } "2

7

Smaller -

7

Biggs

xtyle

def

i) $\{x\} \{y\} \{z\}$

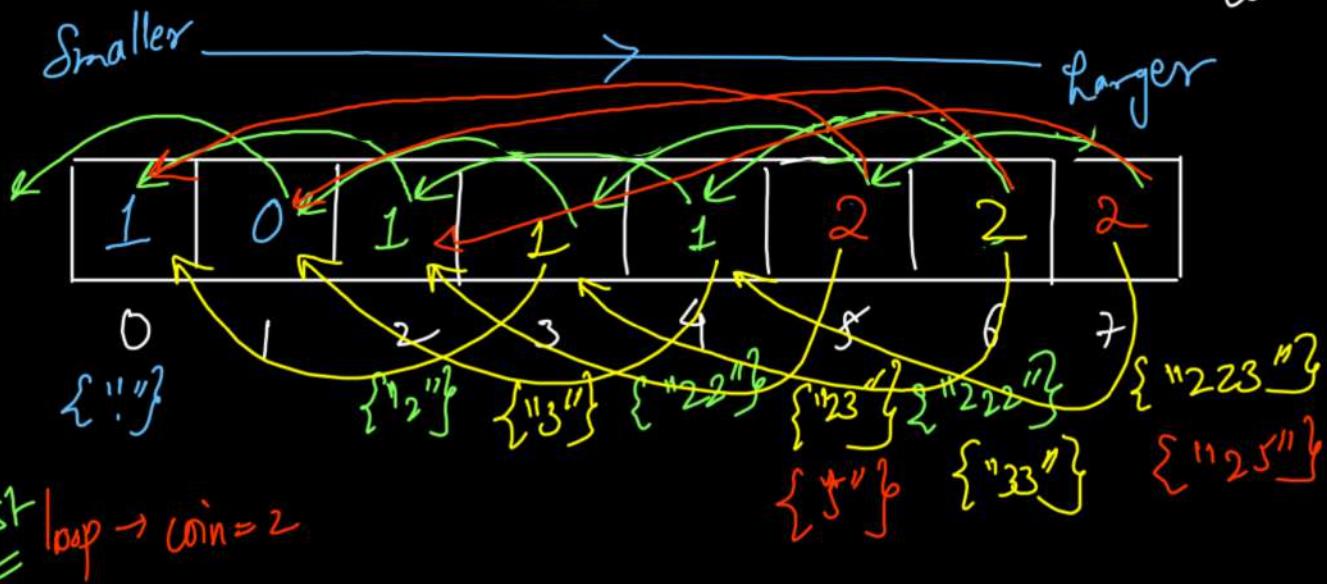
88

Tabulation

```
// Time - O(Amount * Coins), Space - O(Amount)
for(int i=1; i<= amount; i++){
    for(int coin: coins){
        if(i >= coin){
            dp[i] += dp[i - coin];
        }
    }
}
return dp[amount];
```

Combination
(Tabulation)

target = 7
coins = {2, 3, 5}



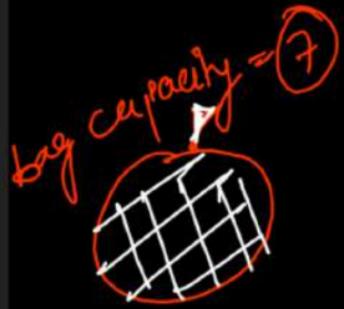
1st loop \rightarrow coin = 2

2nd loop \rightarrow coin = 3

3rd loop: coin = 5

Tabulation

```
public int change(int amount, int[] coins) {  
    int[] dp = new int[amount + 1];  
    dp[0] = 1; // Ways to reach dest when src = dest is 1.  
  
    // Time - O(Amount * Coins), Space - O(Amount)  
    for(int coin: coins)  
        for(int i=coin; i<= amount; i++)  
            dp[i] += dp[i - coin];  
  
    return dp[amount];  
}
```

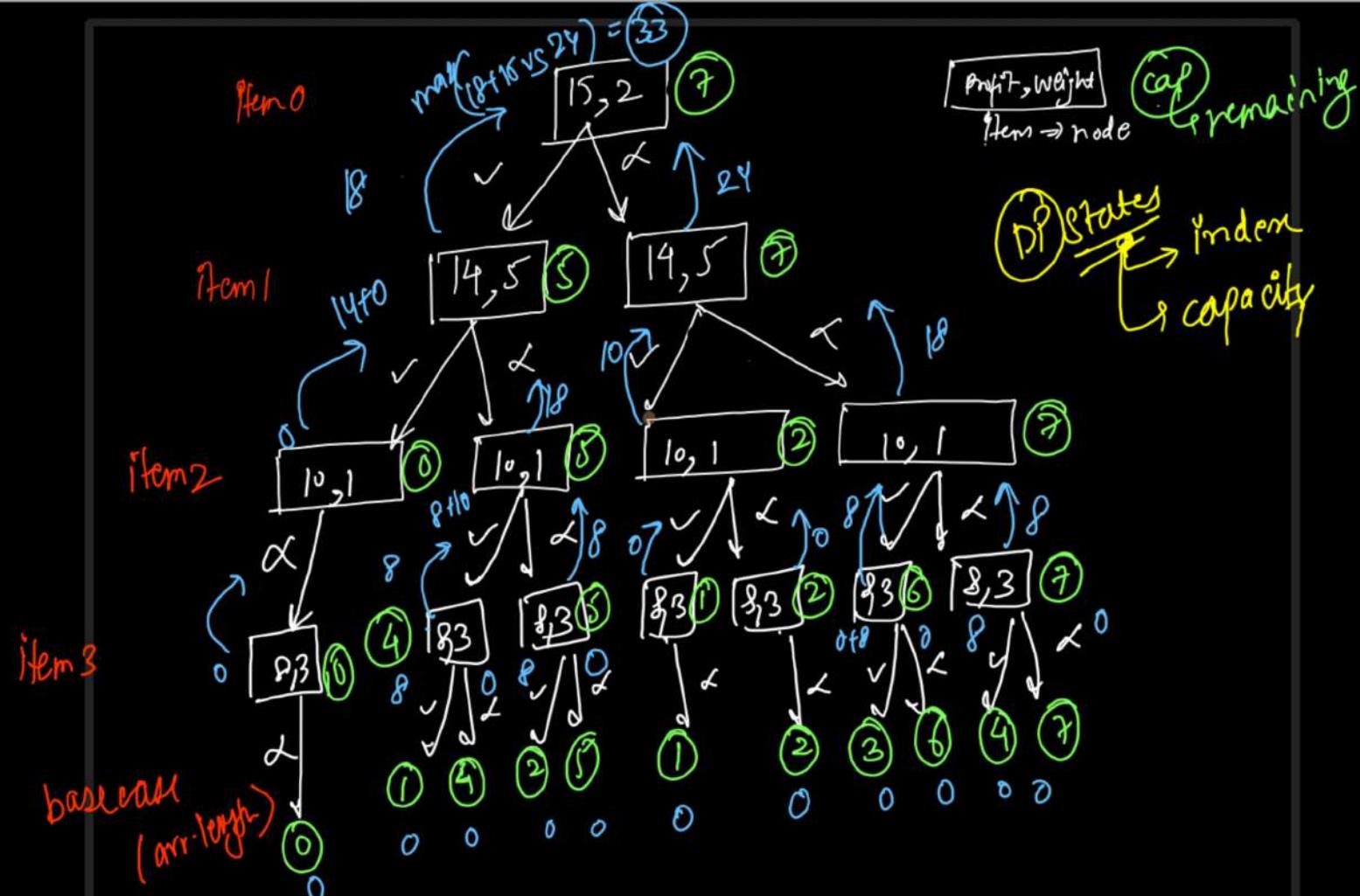


0-1 Knapsack

cost	15	14	10	8	20
weight	2	5	1	3	4
	i ₀	i ₁	i ₂	i ₃	i ₄

constraint

- Maximum loot(profit) → Sort in decreasing order of cost array (greedy^① fails)
- items selected
Weight sum \leq capacity → Sort in increasing order of weight array (greedy^② fails)
- You can either take the item in entirety or do not take the item $\leftarrow \left(\text{Sort in } \frac{\text{cost}}{\text{weight}} \right)$ (greedy^③ fails)
because item is not divisible



```

static int memo(int cap, int item, int[] wt, int[] cost, int[][] dp){
    if(item == cost.length) return 0; // No Item No Profit
    if(dp[cap][item] != -1) return dp[cap][item];

    int yes = (cap >= wt[item]) ?
        memo(cap - wt[item], item+1, wt, cost, dp) + cost[item]: -1;
    int no = memo(cap, item+1, wt, cost, dp);

    return dp[cap][item] = Math.max(yes, no);
}

static int knapSack(int cap, int wt[], int cost[], int n)
{
    int[][] dp = new int[cap + 1][cost.length];
    for(int i=0; i<=cap; i++){
        for(int j=0; j<cost.length; j++){
            dp[i][j] = -1;
        }
    }

    return memo(cap, 0, wt, cost, dp);
}

```

Time
 $O(Cap * item)$

Space
 $O(Cap * item)$
 DP
 2D

R.C.S $\Rightarrow O(items)$

Space optimization ? Yes
 ↳ Tabulation
 ↳ 1D DP
 $O(Cap)$

Overlapping Subproblems

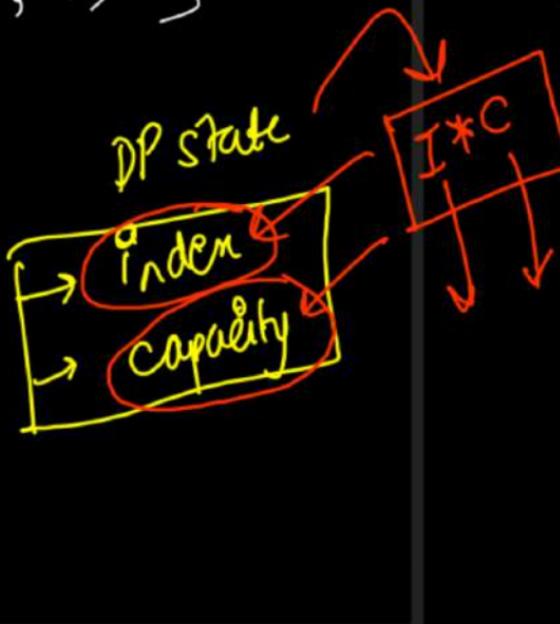
0th item $(15, 7)$ 16
✓ ↴ ↴

$$\{ \begin{matrix} 15, & 14, & 8 \\ 0 & 1 & 2 \end{matrix} \}$$

Recursion
4 items

1st Item
 $(8,2) 2$ $(8,2) 3$
 $(8,2) 7$ $(8,2) 9$

$(14,2) 8$ $(14,2) 9$
 $(14,2) 14$ $(14,2) 16$



Tabulation DP Table

Follow up: Space Optimization (?)

items (el)

Items = ④, capacity = ⑦

		No jform	(15, 2)	(14, 5)	(10, 1)	(8, 3)	
Shaller		0	0	0	0	0	
2000 (at)		1	0 vs -1	0	0 vs -1	0	0 vs 10 + 0 10
2		0	0 vs 15 + 0 15	15 vs -1	15	15 vs 10 + 0 15	15 vs -1 15
3		0	0 vs 15 + 0 15	15 vs -1	15	15 vs 10 + 15 25	25 vs 8 + 0 25
4		0	0 vs 15 + 0 15	15 vs -1	15	15 vs 10 + 15 25	25 vs 8 + 10 25
5		0	0 vs 15 + 0 15	15 vs 14 + 0 15	15	15 vs 10 + 15 25	25 vs 8 + 15 25
6		0	0 vs 15 + 0 15	15 vs 14 + 0 15	15	15 vs 10 + 15 25	25 vs 8 + 25 33
larger 7		0	0 vs 15 + 0 15	15 vs 14 + 15 29	29	29 vs 10 + 15 29	29 vs 8 + 25 33

```

static int knapSack(int caps, int wt[], int cost[], int n)
{
    int[] dp = new int[caps + 1];

    for(int item=1; item<=cost.length; item++){
        int[] newDp = new int[caps + 1];

        for(int cap=1; cap<=caps; cap++){

            int no = dp[cap];
            int yes = (cap >= wt[item - 1])
                ? cost[item - 1] + dp[cap - wt[item - 1]]
                : -1;

            newDp[cap] = Math.max(yes, no);
        }

        dp = newDp;
    }

    return dp[caps];
}

```

Space Opt

TC $\rightarrow O(1)$

SC $\rightarrow O(\text{Cap})$

(IP)

```

int[][] dp = new int[caps + 1][cost.length + 1];

for(int item=1; item<=cost.length; item++){
    for(int cap=1; cap<=caps; cap++){

        int no = dp[cap][item - 1];
        int yes = (cap >= wt[item - 1])
                  ? cost[item - 1] + dp[cap - wt[item - 1]][item - 1]
                  : -1;

        dp[cap][item] = Math.max(yes, no);
    }
}

return dp[caps][cost.length];

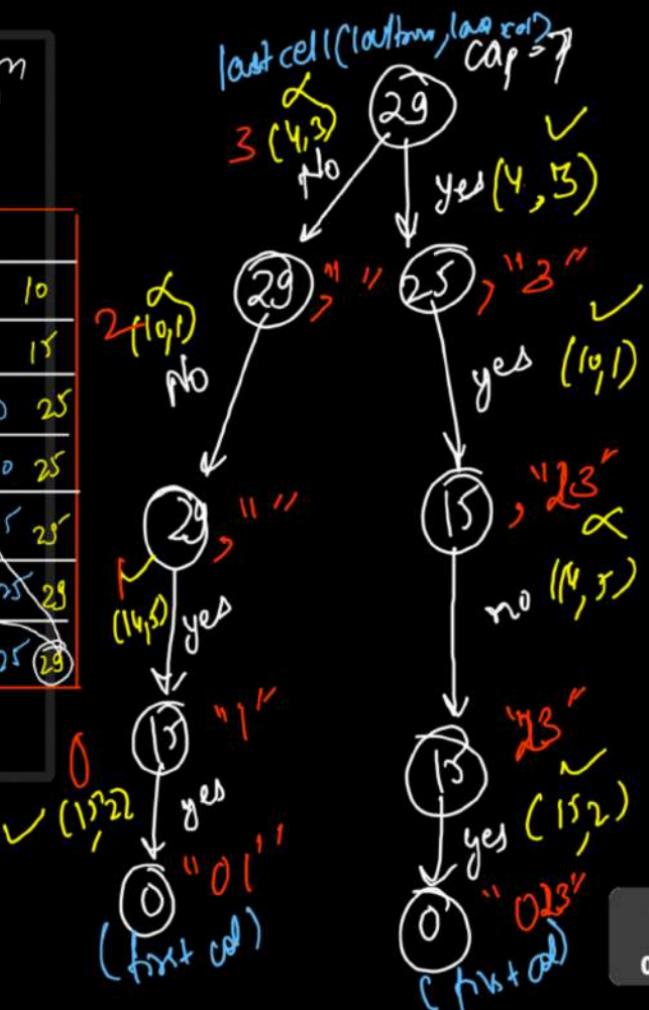
```

$T_C \Rightarrow O(\text{Items} * \text{Cap})$
 $S_C \Rightarrow O(\text{Items} * \text{Cap}) \left\{ 2P \right\}$

Point Always to select items in knapsack such that maximum profit is achieved

	No item	(15, 2)	(14, 5)	(10, 1)	(9, 3)
strat ^r	0	0	0	0	0
var ^s	1	0 vs -1	0	0 vs 10 + 0	10 vs -1
var ^t	2	0 vs 15 + 0	15	15 vs -1	15
var ^t	3	0 vs 15 + 0	15	15 vs -1	15
var ^t	4	0 vs 15 + 0	15	15 vs -1	15
var ^t	5	0 vs 15 + 0	15	15 vs 10 + 15	25
var ^t	6	0 vs 15 + 0	15	15 vs 14 + 0	15
var ^t	7	0 vs 15 + 0	15	15 vs 14 + 15	25

#For Backtracking
 src → last cell
 dest → first col



BFS

```
Queue<Pair> q = new ArrayDeque<>();
q.add(new Pair(caps, cost.length, ""));

while(q.size() > 0){
    Pair top = q.remove();
    int row = top.row;
    int col = top.col;
    String psf = top.psf;

    if(top.col == 0){ /
        System.out.println(top.psf);
        continue;
    }

    // If Item Can be Included and It gives Maximum Profit, then explore this
    if([row >= wt[col - 1] &&
       dp[row][col] == cost[col - 1] + dp[row - wt[col - 1]][col - 1]){

        q.add(new Pair(row - wt[col - 1], col - 1, (col - 1) + " " + psf));
    }

    // If no call gives maximum profit, then only explore this edge
    if(dp[row][col] == dp[row][col - 1]){
        q.add(new Pair(row, col - 1, psf));
    }
}
```

PRO

$T.C \Rightarrow O(\text{exponential})$
in worst case
 $O(V \times E)$ BFS
in avg case

$S.C \Rightarrow O(Col * N)$ DP
 $O(I^{avg} * Col)$ Queue

Dynamic Programming

Lecture 12

Saturday, 9 AM - 12 PM, 7 May

- Knapsack
 - Unbounded Knapsack
 - Rod Cutting
 - Buy & Sell Stocks

Unbounded Knapsack

Henry

bag cap = 7

constraint

	cost	15	14	10	8	20
	weight	2	5	1	3	4
	i ₀	i ₁	i ₂	i ₃	i ₄	

→ maximum loot (profit)

→ items selected

Weight sum \leq capacity

→ items cannot be divided

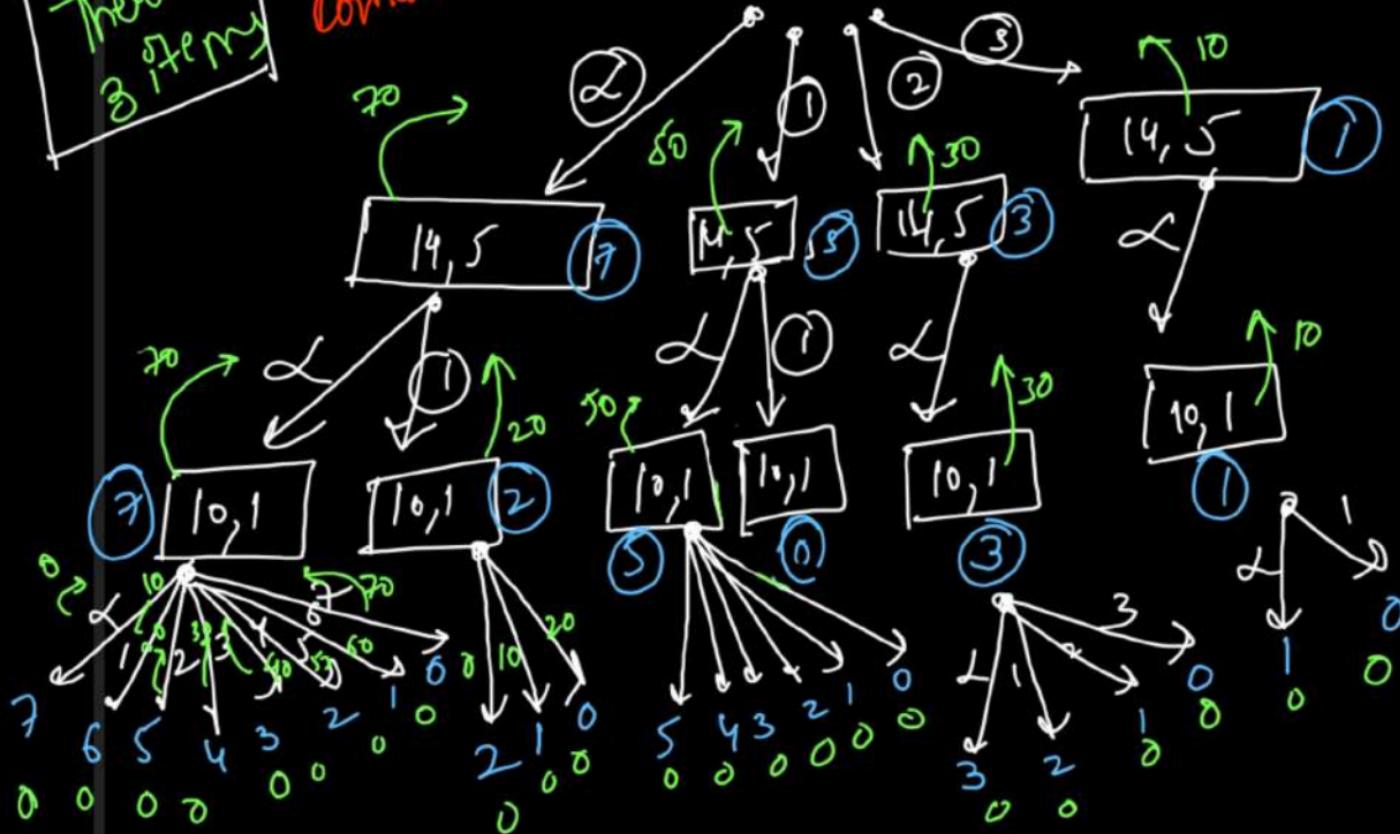
→ You have infinite supply
of each items

There are
3 items

Coin change
combination

15, 2 → 70

70 vs $5_0 + 15 * 1$ vs $3_0 + 15 * 2$
vs $1_0 + 15 * 3$
= 70



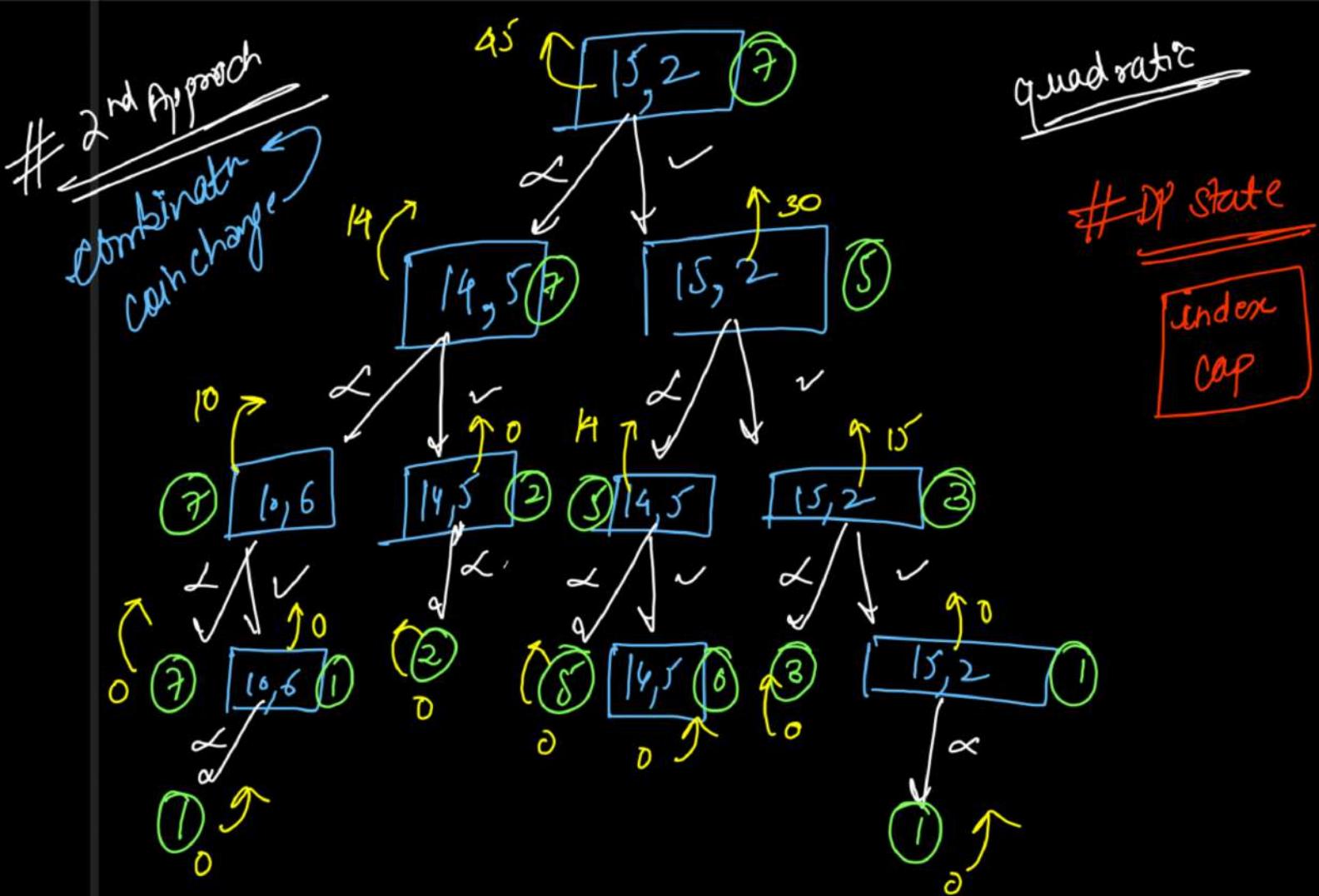
```
static int memo(int index, int cap, int cost[], int wt[], int N, int[][] dp){
    if(index == N || cap == 0) return 0;
    if(dp[index][cap] != -1) return dp[index][cap];

    int ans = -1;
    for(int freq=0; cap >= freq * wt[index]; freq++){
        int temp = memo(index + 1, cap - freq * wt[index], cost, wt, N, dp)
                  + freq * cost[index];
        ans = Math.max(ans, temp);
    }

    return dp[index][cap] = ans;
}
```

```
static int knapSack(int N, int cap, int cost[], int wt[])
{
    int[][] dp = new int[N + 1][cap + 1];
    for(int i=0; i<=N; i++){
        for(int j=0; j<=cap; j++){
            dp[i][j] = -1;
        }
    }

    return memo(0, cap, cost, wt, N, dp);
}
```



```

static int memo(int index, int cap, int cost[], int wt[], int N, int[][] dp){
    if(index == N || cap == 0) return 0;
    if(dp[index][cap] != -1) return dp[index][cap];

    int no = memo(index + 1, cap, cost, wt, N, dp);
    int yes = (cap >= wt[index])
        ? cost[index] + memo(index, cap - wt[index], cost, wt, N, dp) : -1;

    return dp[index][cap] = Math.max(yes, no);
}

static int knapSack(int N, int cap, int cost[], int wt[])
{
    int[][] dp = new int[N + 1][cap + 1];
    for(int i=0; i<=N; i++){
        for(int j=0; j<=cap; j++){
            dp[i][j] = -1;
        }
    }

    return memo(0, cap, cost, wt, N, dp);
}

```

$$TC \rightarrow O(N \times cap)$$

$$SC \rightarrow O(N \times cap)$$

$2^N \times DP$

(Cap)

0 1 2 3 4 5 6 7

0

0 ↑

6

0 ↑

0

0 ↑

0

0

$\boxed{(\sqrt{2}, 1)}$

1

0

0

$0 \text{ vs } 15 + 0$

$0 \text{ vs } 15 + 0$

$0 \text{ vs } 15 + 0$

$15 + 2 + 0$

$0 \text{ vs } 15 + 5$

$0 \text{ vs } 15 + 5$

$0 + 15 + 2$

$30 + 5 + 45$

$30 + 5 + 45$

45

Fence

$\boxed{\frac{4}{5}}$

2

0

0 ↑

15

15

30

34

34

45

45

49

$\boxed{\frac{8}{3}}$

3

0

0

15

30

30

34

45

60

60

60

0	0	6	0	0	0	0	0
0	0	15	15	$15 + 2 + 0$	$0 + 15 + 5$	$0 + 15 + 2$	$30 + 5 + 45$
0	0	15	15	30	$30 + 0 + 34$	$45 + 5 + 0 + 34$	$45 + 5 + 15 + 34$
0	0	15	15	30	$15 + 5 + 30 + 0$	$34 + 5 + 30 + 15$	$45 + 5 + 45 + 60$

(Cap)

0 1 2 3 4 5 6 7

0

0

6

0

0

0

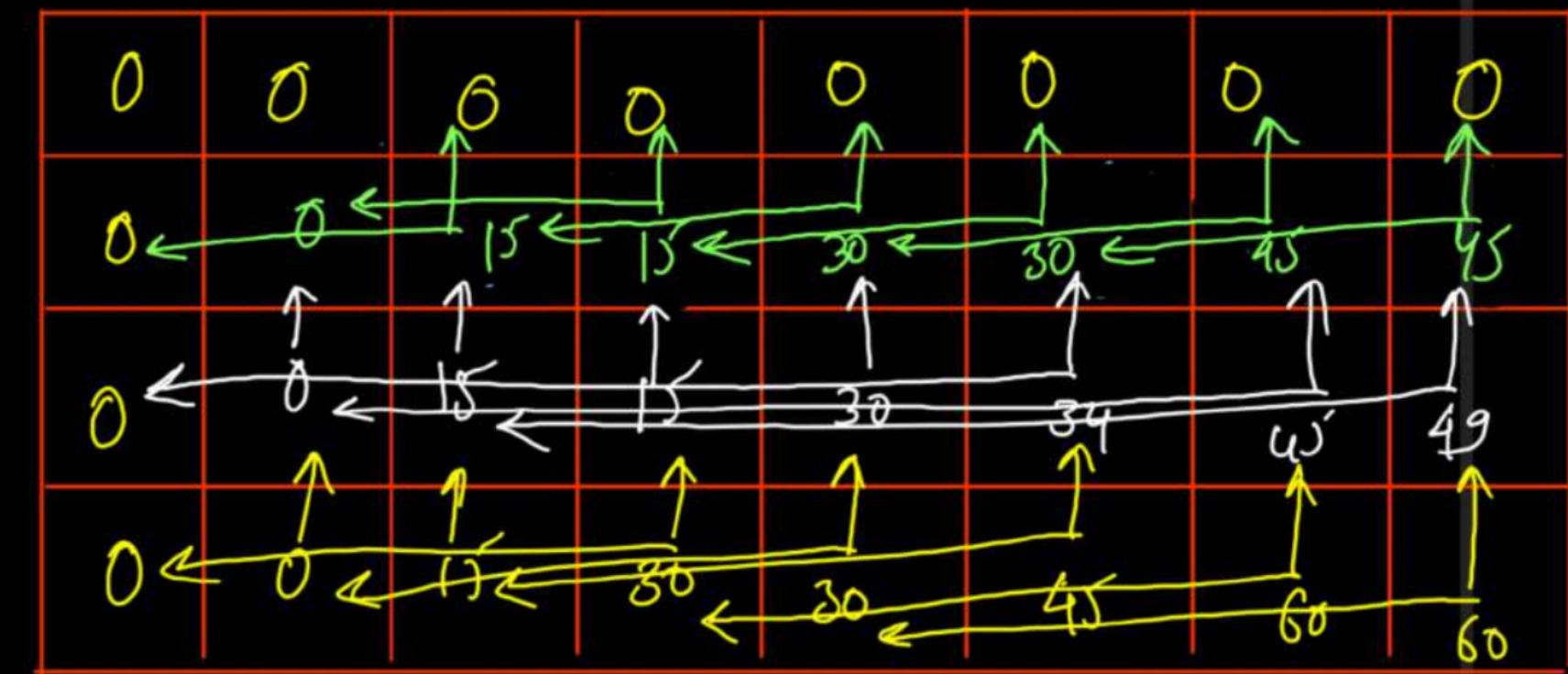
0

0

$\boxed{W_1, 2}$

$\boxed{34, 5}$

$\boxed{30, 3}$



```

static int knapSack(int N, int caps, int cost[], int wt[])
{
    int[][] dp = new int[N + 1][caps + 1];

    for(int item=1; item<=N; item++){
        for(int cap=1; cap<=caps; cap++){
            int no = dp[item - 1][cap];
            int yes = (cap >= wt[item - 1])
                ? dp[item][cap - wt[item - 1]] + cost[item - 1]
                : -1;

            dp[item][cap] = Math.max(no, yes);
        }
    }

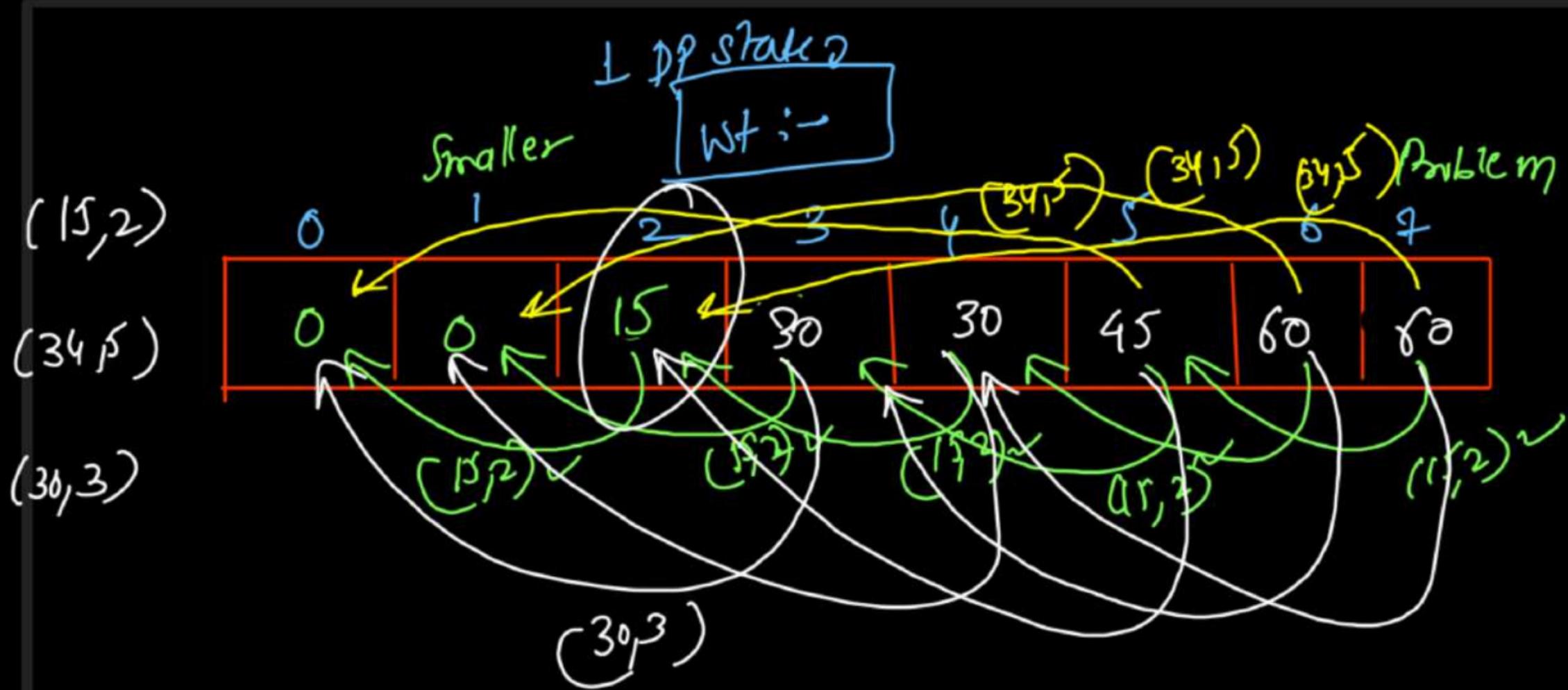
    return dp[N][caps];
}

```

2D DP

 TC → $O(N * caps)$

 SC → $O(N * caps)$



```

static int knapSack(int N, int caps, int cost[], int wt[])
{
    int[] dp = new int[caps + 1];

    for(int item=1; item<=N; item++){
        for(int cap=1; cap<=caps; cap++){
            int no = dp[cap];
            int yes = (cap >= wt[item - 1])
                ? dp[cap - wt[item - 1]] + cost[item - 1]
                : -1;

            dp[cap] = Math.max(no, yes);
        }
    }

    return dp[caps];
}

```

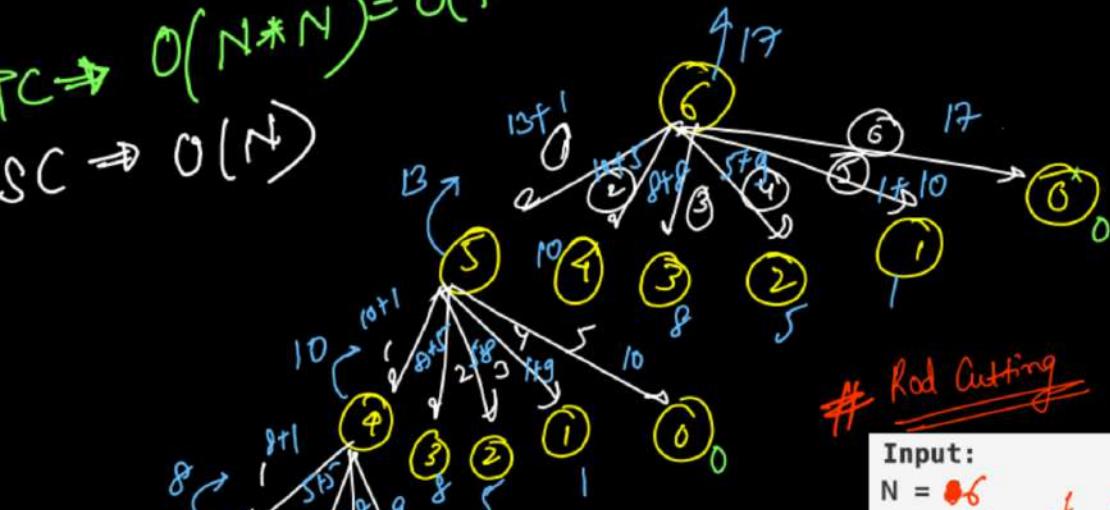
ID DP

TC $\rightarrow O(n^2)$
 SC $\rightarrow O(caps)$

$\left\{ \begin{array}{l} \text{Infinite Supply} \rightarrow \text{yes} \rightarrow \text{same row} \\ \text{Limited Supply} \rightarrow \text{yes} \rightarrow \text{previous row} \end{array} \right\} \left\{ \begin{array}{l} \text{No call} \\ \text{inf} \wedge \text{frnle} \\ \text{prv row} \end{array} \right\}$

TC $\Rightarrow O(N \cdot N) = O(N^2)$

SC $\Rightarrow O(N)$

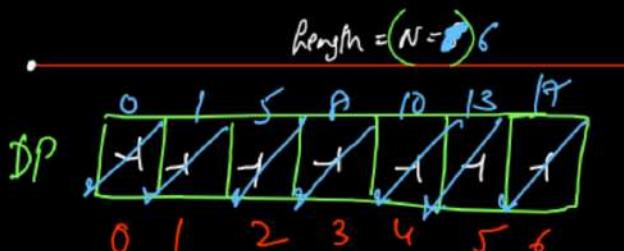
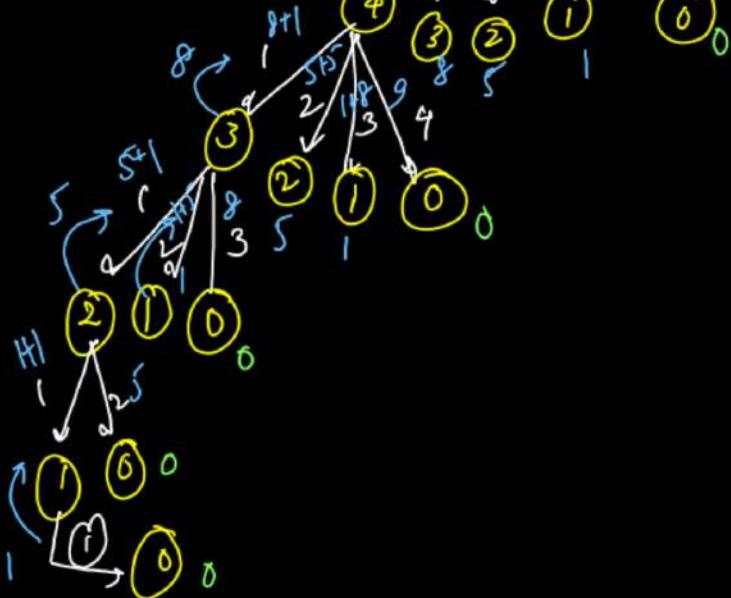


Rod Cutting

Input:

N = 6

Price[] = {1, 5, 8, 9, 10, 17, 13, 17}



$DP[i] = \max^n \text{ profit of that length}$

Variations of
Knapsack
problems
Club stores
minimum cost

```
class Solution{
    public int memo(int n, int price[], int[] dp){
        if(n == 0) return 0;
        if(dp[n] != -1) return dp[n];

        int ans = 0;
        for(int cut=1; cut<=n; cut++){
            ans = Math.max(ans, price[cut - 1] + memo(n - cut, price, dp));
        }

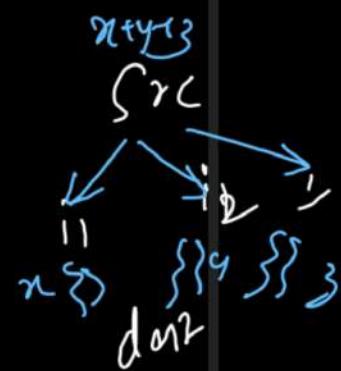
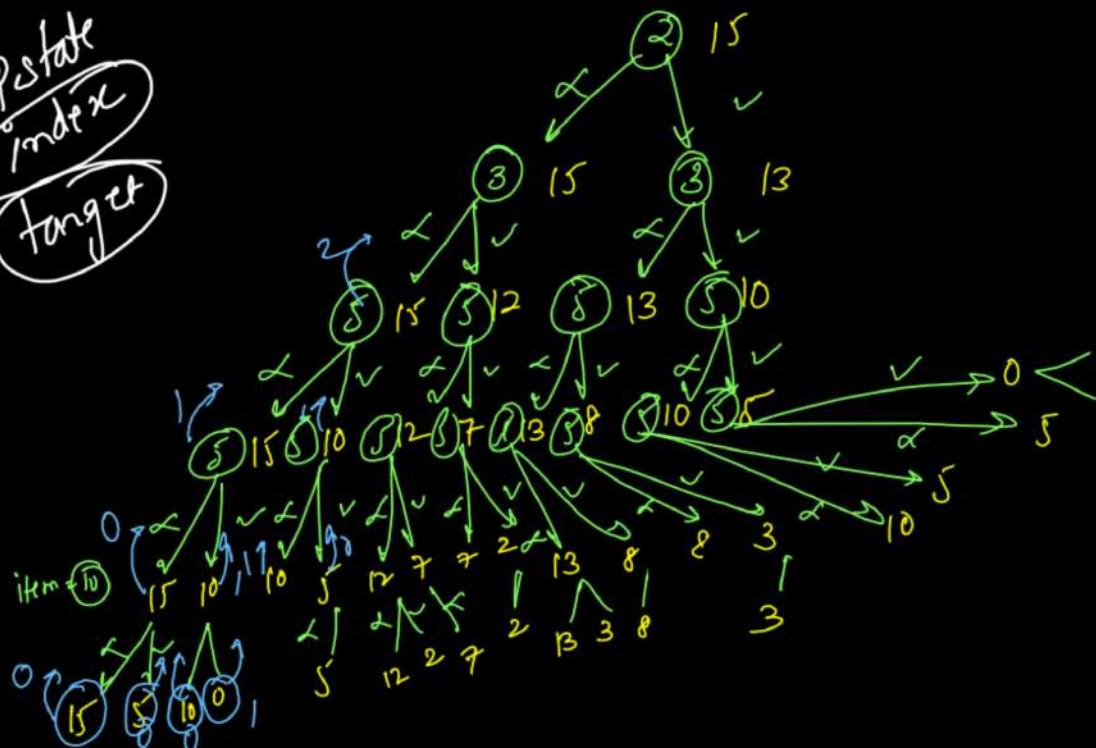
        return dp[n] = ans;
    }

    public int cutRod(int price[], int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, -1);
        return memo(n, price, dp);
    }
}
```

Target sum subset (^{Variation of} 0-1 knapsack)

{2, 3, 5, 5, 10} target = 15

D_i state
index
target



Dynamic Programming - lecture 13

Sunday, 8 May, 9 AM - 12 PM

Buy & Sell stocks

- Variations
- Fibonacci
- House Robber
- Knapsack

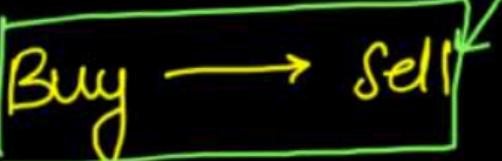
121 Buy & Sell Stocks - 1 Transaction

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

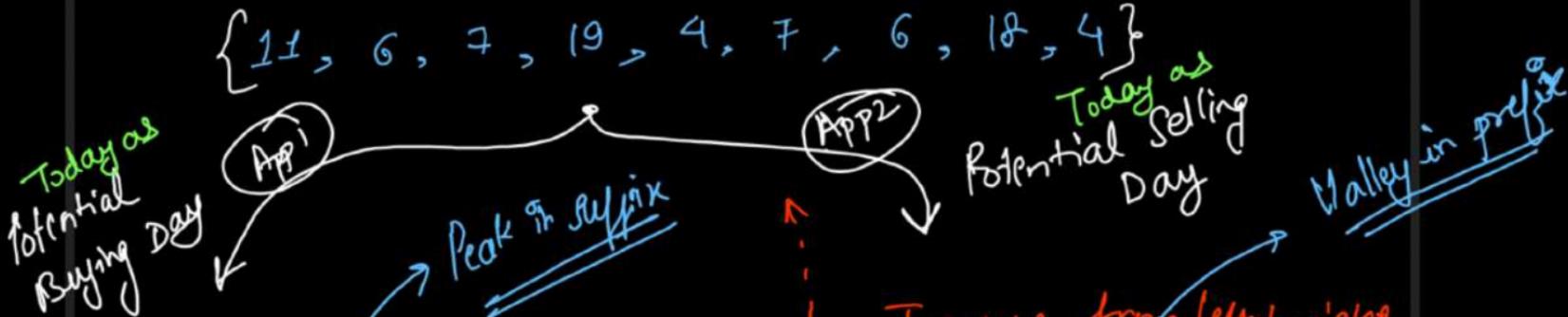
Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Constraint

 | transaction

BBSS α

{11, 6, 7, 19, 4, 1, 6, 18, 4}



Traverse from right to left
 Take \max^m of suffix

$$\text{Profit} = (\max^m - \text{cost[today]})$$

$$\begin{array}{ccccccccc}
 & 11 & , & 6 & , & 7 & , & 19 & , \\
 \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 11-11 & = 0 & 19-6 & = 13 & 19-7 & = 12 & 19-19 & = 0 & 19-4 = 15 \\
 & & & & & & & & \\
 & & & & 18-4 & = 14 & 18-7 = 11 & 18-6 = 12 & 18-19 = 0 \\
 & & & & & & & & \\
 & & & & 18-6 & = 12 & 18-19 = 0 & 18-4 = 14 & 18-4 = 14
 \end{array}$$

$$\max^m = \cancel{11} \cancel{13} \cancel{12} \cancel{0} \cancel{15} \cancel{11} \cancel{12} \cancel{0} \cancel{14} \cancel{14} \cancel{14}$$

$$\text{profit} = \cancel{0} \cancel{13} \cancel{12} \cancel{0} \cancel{14}$$

Today as
 Potential Selling
 Day

Valley in prefix

Traverse from left to right
 Take \min^m of prefix

$$\text{Profit} = (\text{cost[today]} - \min^m)$$

$$\begin{array}{ccccccccc}
 & 11 & , & 6 & , & 7 & , & 19 & , \\
 \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 11-11 & = 0 & 6-6 & = 0 & 7-6 = 1 & 19-6 & = 13 & 19-19 & = 0 \\
 & & & & & & & & \\
 & & & & 19-4 & = 15 & 19-7 = 12 & 19-6 = 13 & 19-4 = 15 \\
 & & & & & & & & \\
 & & & & 19-4 & = 15 & 19-7 = 12 & 19-6 = 13 & 19-4 = 15
 \end{array}$$

$$\min^m = \cancel{11} \cancel{13} \cancel{12} \cancel{0} \cancel{15} \cancel{11} \cancel{12} \cancel{0} \cancel{14} \cancel{14} \cancel{14}$$

Buying day

$$\text{Profit} = \cancel{0} \cancel{13} \cancel{12} \cancel{0} \cancel{14}$$

```
class Solution {
    public int maxProfit(int[] prices) {
        int n = prices.length;

        int selling = prices[n - 1], profit = 0;

        for(int i=n-1; i>=0; i--){
            selling = Math.max(selling, prices[i]);

            int buying = prices[i]; // Today - Potential Buying Day

            profit = Math.max(profit, selling - buying);
        }

        return profit;
    }
}
```

$O(N)$ Time
 $O(1)$ Space

App 1

```
public int maxProfit(int[] prices) {
    int n = prices.length;

    int buying = prices[0], profit = 0;

    for(int i=0; i<n; i++){
        buying = Math.min(buying, prices[i]);

        int selling = prices[i]; // Today - Potential Selling Day

        profit = Math.max(profit, selling - buying);
    }

    return profit;
}
```

App 2

122. Best Time to Buy and Sell Stock II

Infinite Transactions

#BBS is not possible
#BSBS is possible

$$\left\{ 11, \textcircled{6}, \textcircled{7}, \textcircled{19}, 4, 1, \textcircled{6}, \textcircled{18}, \textcircled{4}, \textcircled{12}, 2 \right\}$$

$19 - 7 = 12 \checkmark$

$7 - 6 = 1$

$18 - 6 = 12 \checkmark$

$6 - 1 = 5$

$12 - 4 = 8 \checkmark$

(top) buying day = $\cancel{11} \cancel{6} \cancel{7} \cancel{19} \cancel{4} \cancel{1} \cancel{6} \cancel{18} \cancel{4} \cancel{12} \cancel{2}$
 $\text{profit} = 0 + 1 + 12 + 5 + 12 + 8$

11	$\cancel{6}$	$\cancel{7}$	19	4	$\cancel{1}$	$\cancel{6}$	18	$\cancel{4}$	12	2
----	--------------	--------------	----	---	--------------	--------------	----	--------------	----	---

```

class Solution {
    public int maxProfit(int[] prices) {
        Stack<Integer> stk = new Stack<>();
        int profit = 0;

        for(int i=0; i<prices.length; i++){
            if(stk.size() > 0 && stk.peek() < prices[i]){
                profit += prices[i] - stk.pop();
            }

            stk.push(prices[i]);
        }

        return profit;
    }
}

```

$O(N)$ time
 $O(N)$ space

Greedy Approach

```

class Solution {
    public int maxProfit(int[] prices) {
        int buying = prices[0];
        int profit = 0;

        for(int i=0; i<prices.length; i++){
            if(buying < prices[i]){
                profit += prices[i] - buying;
            }

            buying = prices[i];
        }

        return profit;
    }
}

```

$O(N)$ time
 $O(1)$ space

DP

Buy & Sell Stock - Infinite Transaction
 - Transaction fee
 - Cooldown

$\{11, 6, 7, 9, 4, 1, 6, 18, 4\}$

(1 Extra Stock)
 Buy
 (0 extra Stock)
 Sell

-11	-11 vs -6	-7 vs -6	-6 vs -18	6 vs -4 + 13	9 vs 13 - 1	13 - 6 vs 12	18 - 18 vs 12	12 vs 30 - 4
0	0 vs -11 + 6	1 vs 0	13 vs 1	-2 vs 13	13 vs 9 + 1	13 vs 12 + 6	18 vs 12 + 8	30 vs 12 + 4

Today's
Potential Buying Day

Do not buy today

with 0 Stock

Prev profit - Today's Buy by cost

$$dp[\text{Buy}] = \max(dp[\text{Buy-1}], dp[\text{sell-1} - \text{cost}[i]])$$

$$dp[\text{sell}] = \max(dp[\text{sell-1}], dp[\text{buy-1}] + \text{cost}[i])$$

Today's
Potential
Selling Day

Do not sell
today

Prev profit + Today's selling
cost

with 1 stock
in hand

$$\{6, 7, 19 \rightarrow 1, 6\}$$

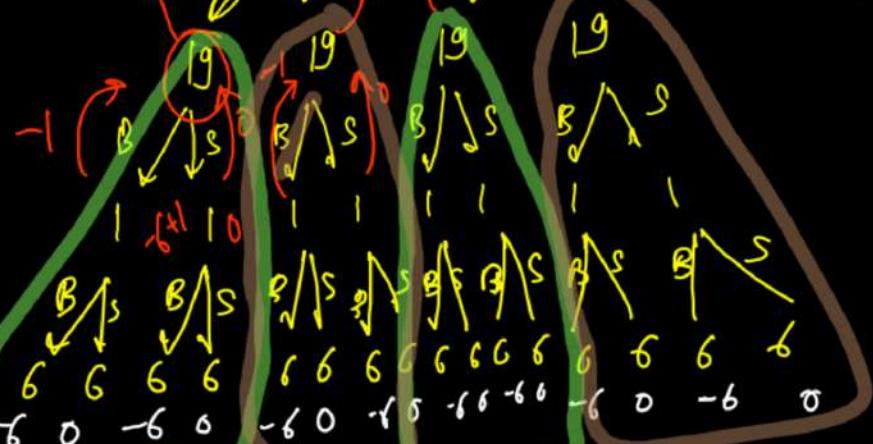
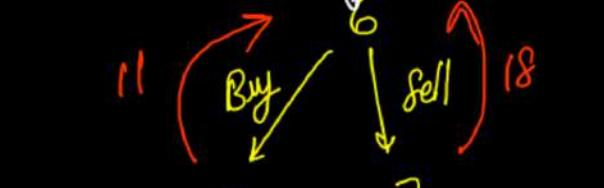
Constraints

BXSS BXSS

sell

Buy

sell



DP State

→ Buy / sell (Row)

→ Days (col)

```

public int maxProfit(int[] prices) {
    int[] buy = new int[prices.length];
    int[] sell = new int[prices.length];

    buy[0] = -prices[0];
    sell[0] = 0;

    for(int i=1; i<prices.length; i++){
        // Treat today as Buying Day
        buy[i] = Math.max(buy[i - 1], sell[i - 1] - prices[i]);

        // Treat today as Selling Day
        sell[i] = Math.max(sell[i - 1], buy[i - 1] + prices[i]);
    }

    return sell[prices.length - 1];
}

```

```

public int maxProfit(int[] prices) {
    int buy = -prices[0];
    int sell = 0;

    for(int i=1; i<prices.length; i++){
        // Treat today as Buying Day
        int newBuy = Math.max(buy, sell - prices[i]);

        // Treat today as Selling Day
        int newSell = Math.max(sell, buy + prices[i]);

        buy = newBuy; sell = newSell;
    }

    return sell;
}

```

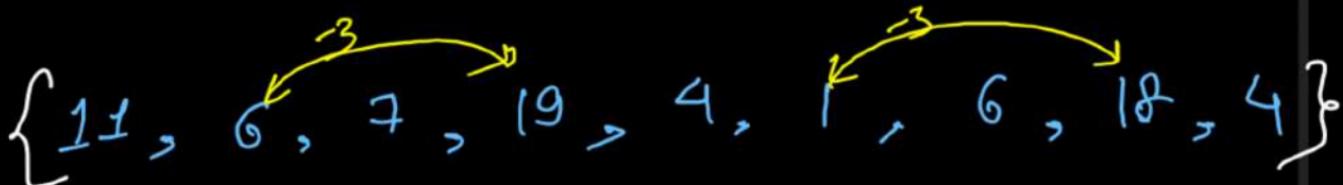
$O(N)$ Time
 $O(1)$ Space

DP

$O(N)$ Time
 $O(1)$ Constant Space

DP

transaction fee \Rightarrow +3 → on completion of every transaction ($B \rightarrow S$)



(Initial State)
Buy
Sell
(Extra State)

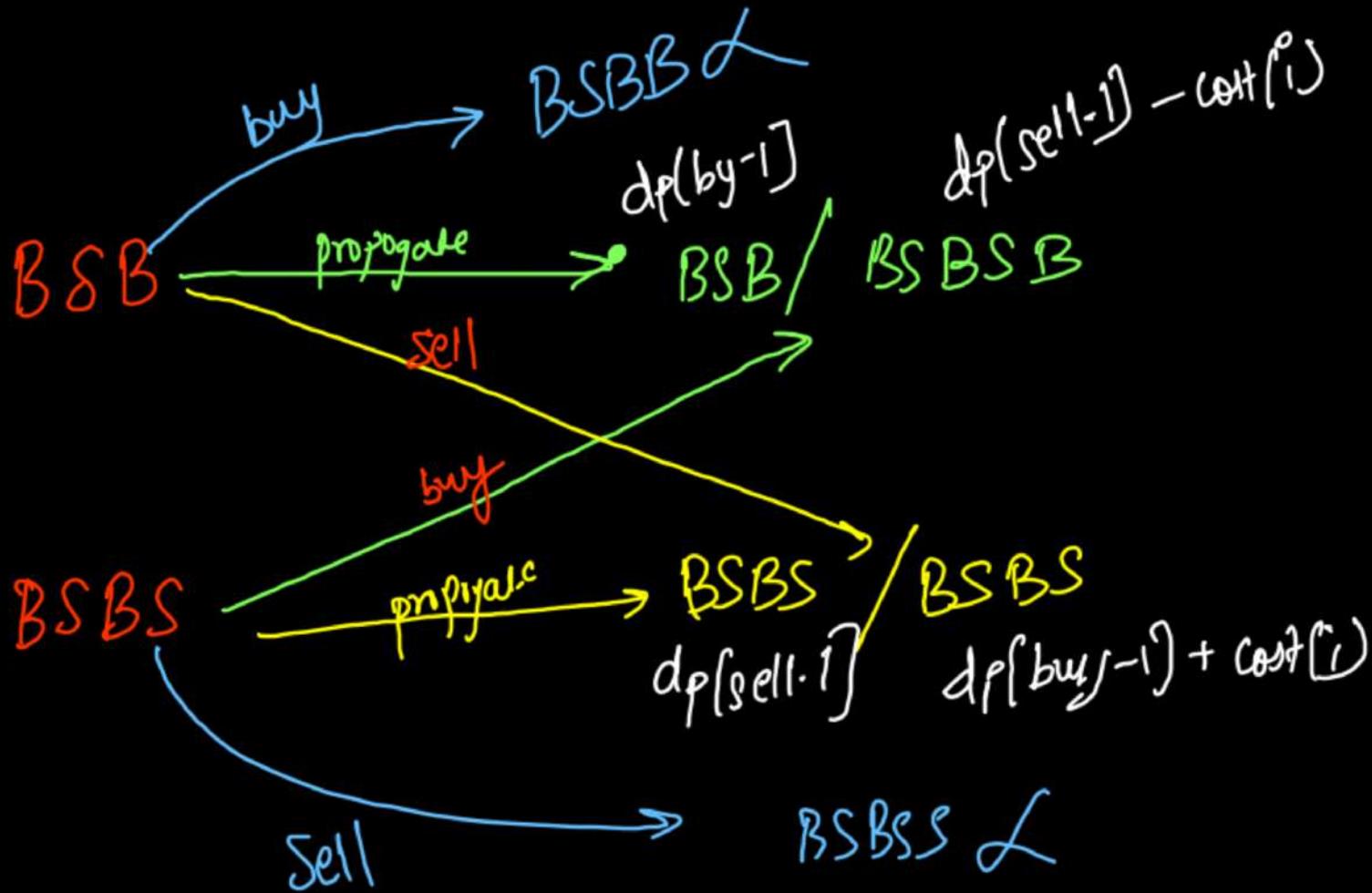
-11	-11 vs -6	-6 vs 0-7	-6 vs 0-9	-6 vs 10-4	6 vs 10-1	9 vs 10-1	9 vs 12-18	9 vs 24-4
0	$-11+6=3 \text{ vs } 0$	$0 \text{ vs } 7-6=1$	$0 \text{ vs } -6+9=3$	$10 \text{ vs } -6+10-4=6$	$10 \text{ vs } 6-1=3$	$10 \text{ vs } 9+6-3=12$	$12 \text{ vs } 9+18-3=24$	$9+4-3 \text{ vs } 24=20$

$$dp[\text{buy}] = dp[\text{buy-1}], (dp[\text{sell-1}] - \text{cost}[i])$$

$$dp[\text{sell}] = dp[\text{sell-1}], (dp[\text{buy-1}] + \text{cost}[i] - \text{fee})$$

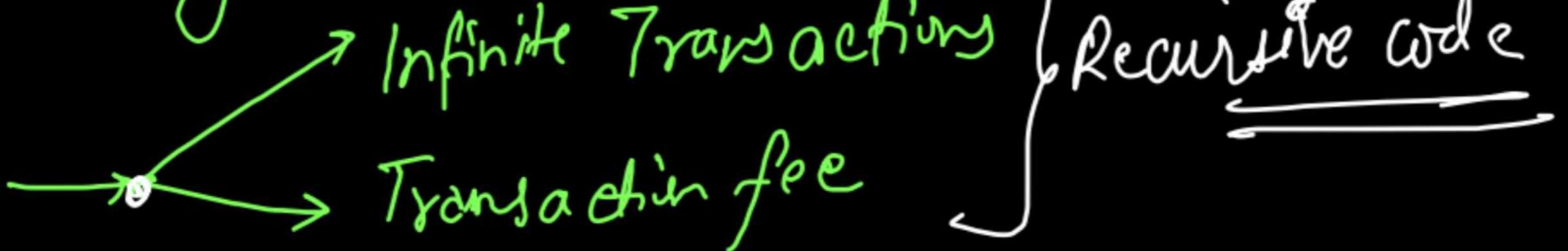
```
public int maxProfit(int[] prices, int fee) {  
    int buy = -prices[0];  
    int sell = 0;  
  
    for(int i=1; i<prices.length; i++){  
  
        // Treat today as Buying Day  
        int newBuy = Math.max(buy, sell - prices[i]);  
  
        // Treat today as Selling Day  
        int newSell = Math.max(sell, buy + prices[i] - fee);  
  
        buy = newBuy; sell = newSell;  
    }  
  
    return sell;  
}
```

$O(N)$ time, $O(1)$ constant space



DP Lecture ⑭

Buy & sell stocks



→ Cooldown

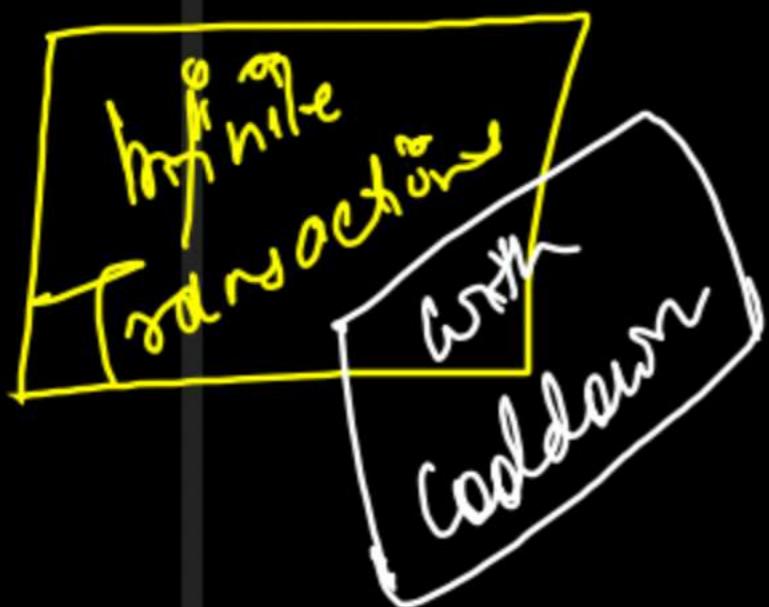
→ 2 Transaction Fees

→ K Transaction fees

Recursive code

memoization

#Buy & sell stock $\{11, 6, 7, 19, 4, 1, 6, 18, 4\}$



$$\left\{ \begin{array}{l} \partial p(\text{buy}) = \partial p(\text{buy-1}), \partial p(\text{sell-2}) + p(1) \\ \partial p(\text{sell}) = \partial p(\text{sell}), \partial p(\text{buy-1}) + p(1) \end{array} \right.$$

1st transaction 2nd transaction 2nd transaction

(BS) (BS)

$\boxed{S \setminus B}$ $\boxed{B \setminus S}$

Current state: B_{buy}

Buy	11	-11 vs -6	-6 vs 0 -7	0 -19 vs -6	-6 vs 1 -4	-3 vs 3 -1	3 -6 vs 12	13 -18 vs 12	18 -4 vs 12
Sell	0	0 vs -11 + 6	0 vs -1 + 7	1 vs -6 + 19	13 vs -6 + 4	13 vs -3 + 1	13 vs 12 + 6	18 vs 12 + 18	30 vs 12 + 4
(0, exp, t)				1	13	13	13	18	30

Final value: 30

```

class Solution {
    public int maxProfit(int[] prices) {
        if(prices.length <= 1) return 0;

        int[] buy = new int[prices.length];
        int[] sell = new int[prices.length];

        buy[0] = -prices[0];
        sell[0] = 0;
        buy[1] = Math.max(-prices[0], -prices[1]); // Either buy 0th stock or 1st stock
        sell[1] = Math.max(0, prices[1] - prices[0]); // Either do nothing or Buy 0 Sell 1

        for(int i=2; i<prices.length; i++){
            buy[i] = Math.max(buy[i - 1], sell[i - 1] - prices[i]);
            sell[i] = Math.max(sell[i - 1], buy[i - 1] + prices[i]);
        }

        return sell[prices.length - 1];
    }
}

```

$O(n)$ time
 $O(n)$ space

```

class Solution {
    public int maxProfit(int[] prices) {
        if(prices.length <= 1) return 0;

        int buy0 = -prices[0];
        int sell0 = 0;
        int buy1 = Math.max(-prices[0], -prices[1]); // Either buy 0th stock or 1st stock
        int sell1 = Math.max(0, prices[1] - prices[0]); // Either do nothing or Buy 0 Sell 1

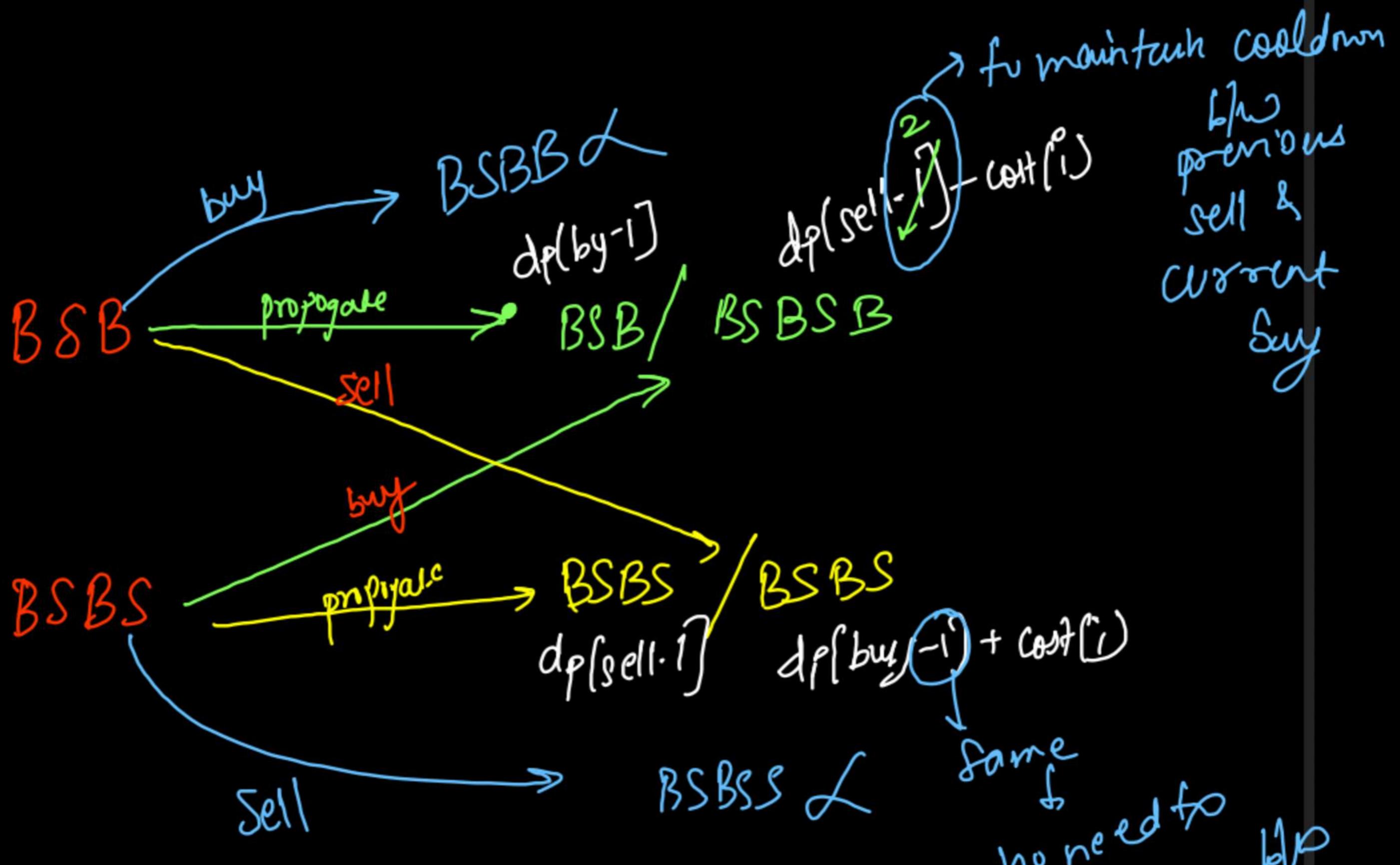
        for(int i=2; i<prices.length; i++){
            int newbuy = Math.max(buy1, sell0 - prices[i]);
            int newsell = Math.max(sell1, buy1 + prices[i]);

            buy0 = buy1; sell0 = sell1;
            buy1 = newbuy; sell1 = newsell;
        }

        return sell1;
    }
}

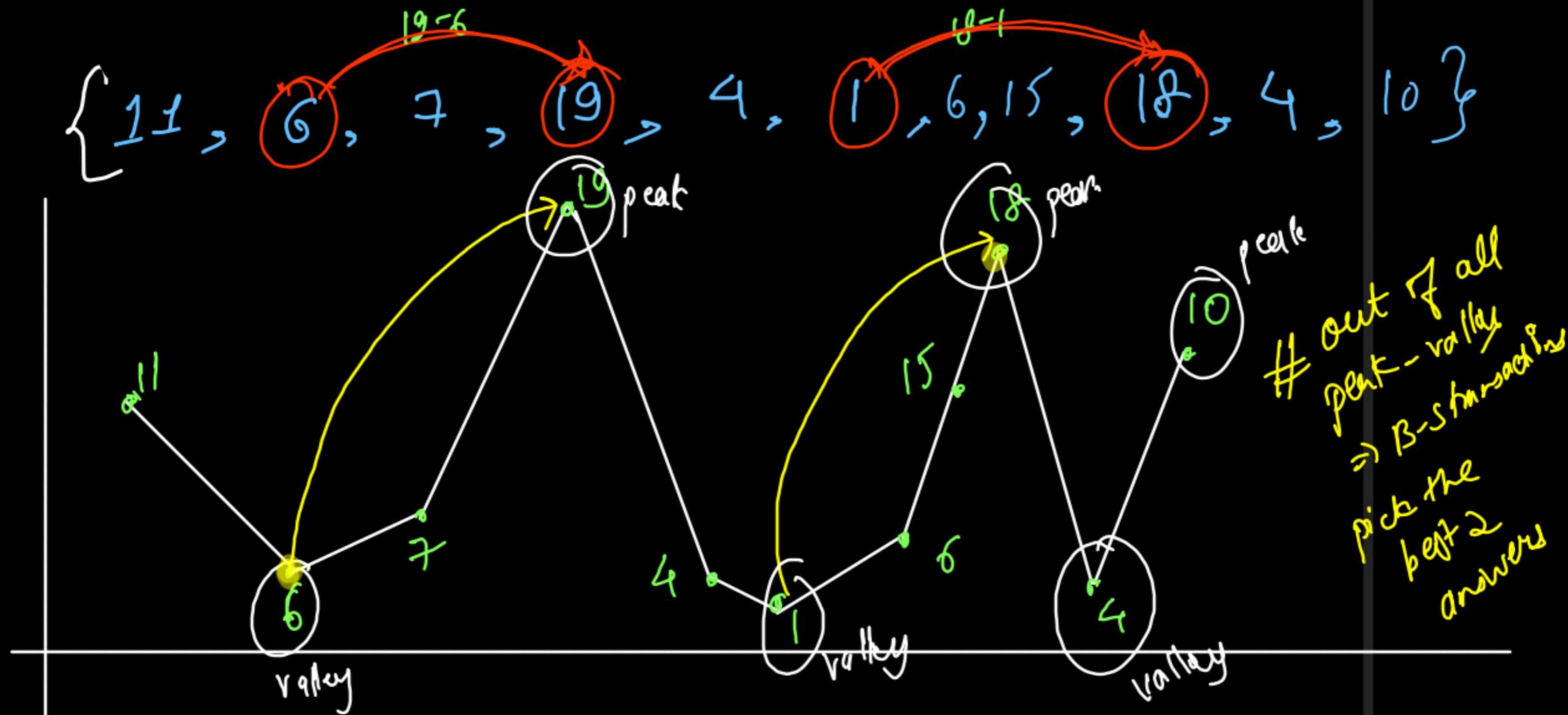
```

$O(n)$ time
 $O(1)$ space



same
 no need to
 cooldown b/w
 a transaction

Infinite Greedy
Buy 2 & sell 1 stocks → 2 Transactions
DP
At most

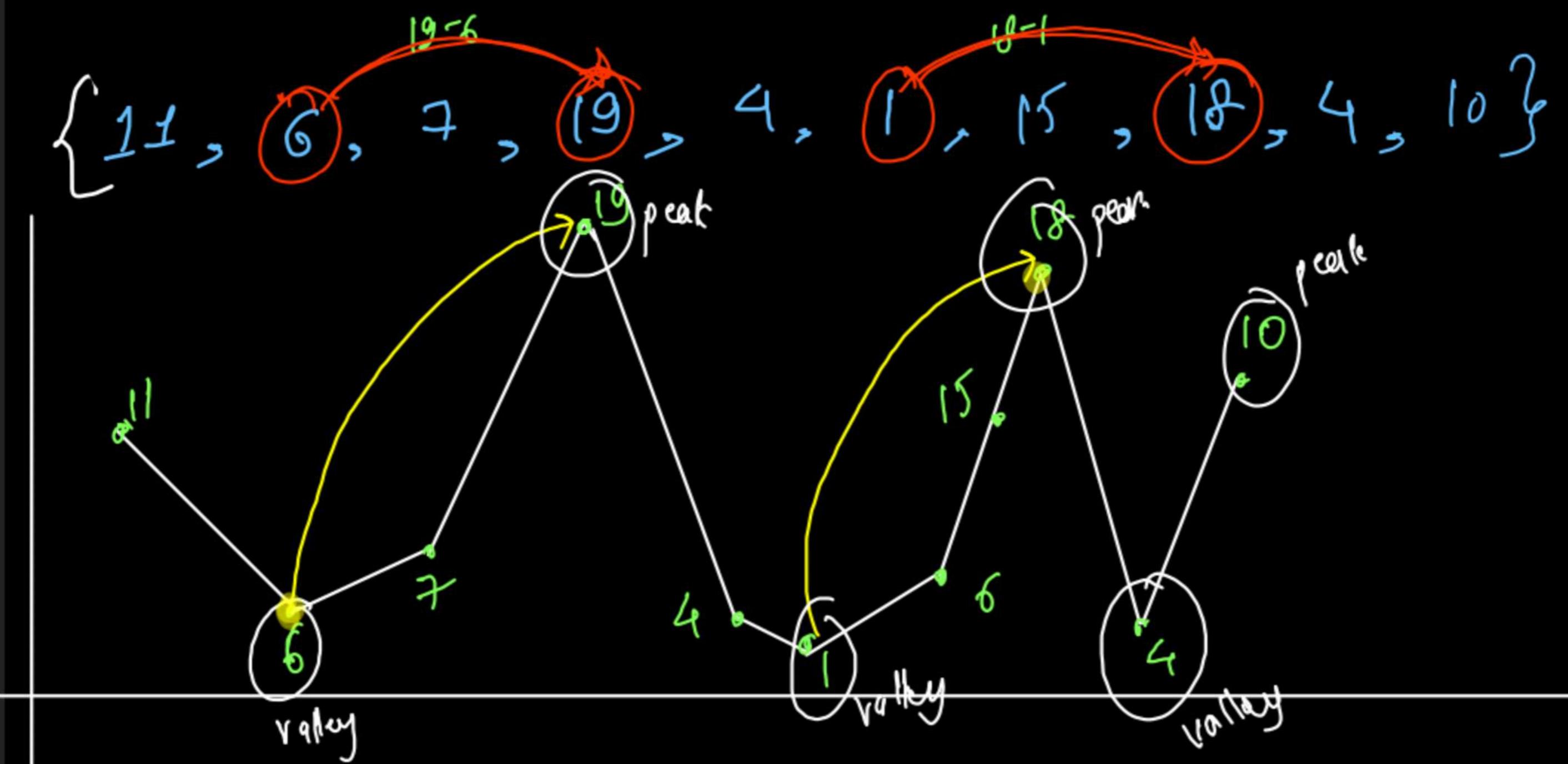


122

#Infinite Trade actions (Greedy)

```
class Solution {  
    public int maxProfit(int[] prices) {  
        int valley = 0, profit = 0;  
        while(valley < prices.length){  
            int peak = valley;  
            while(peak + 1 < prices.length && prices[peak + 1] >= prices[peak]){  
                peak++;  
            }  
  
            profit = profit + (prices[peak] - prices[valley]);  
            valley = peak + 1;  
        }  
        return profit;  
    }  
}
```

buying day	17	17	17	17	17	7	6	6	6	0	Rtol
selling day	0	0	1	13	13	13	14	17	17	17	
lowR											



```
int[] selling = new int[prices.length];

int min = prices[0];
for(int i=0; i<prices.length; i++){
    min = Math.min(min, prices[i]);

    if(i - 1 >= 0)
        selling[i] = Math.max(selling[i - 1], prices[i] - min);
    else selling[i] = prices[i] - min;
}
```

~~Greedy~~ time
 $O(N)$ space
$O(N)$ space

```
int[] buying = new int[prices.length];

int max = prices[prices.length - 1];
for(int i=prices.length-1; i>=0; i--){
    max = Math.max(prices[i], max);

    if(i != prices.length - 1)
        buying[i] = Math.max(buying[i+ 1], max - prices[i]);
    else buying[i] = max - prices[i];
}
```

```
int ans = 0;
for(int i=0; i<prices.length; i++){
    ans = Math.max(ans, buying[i] + selling[i]);
}
return ans;
```

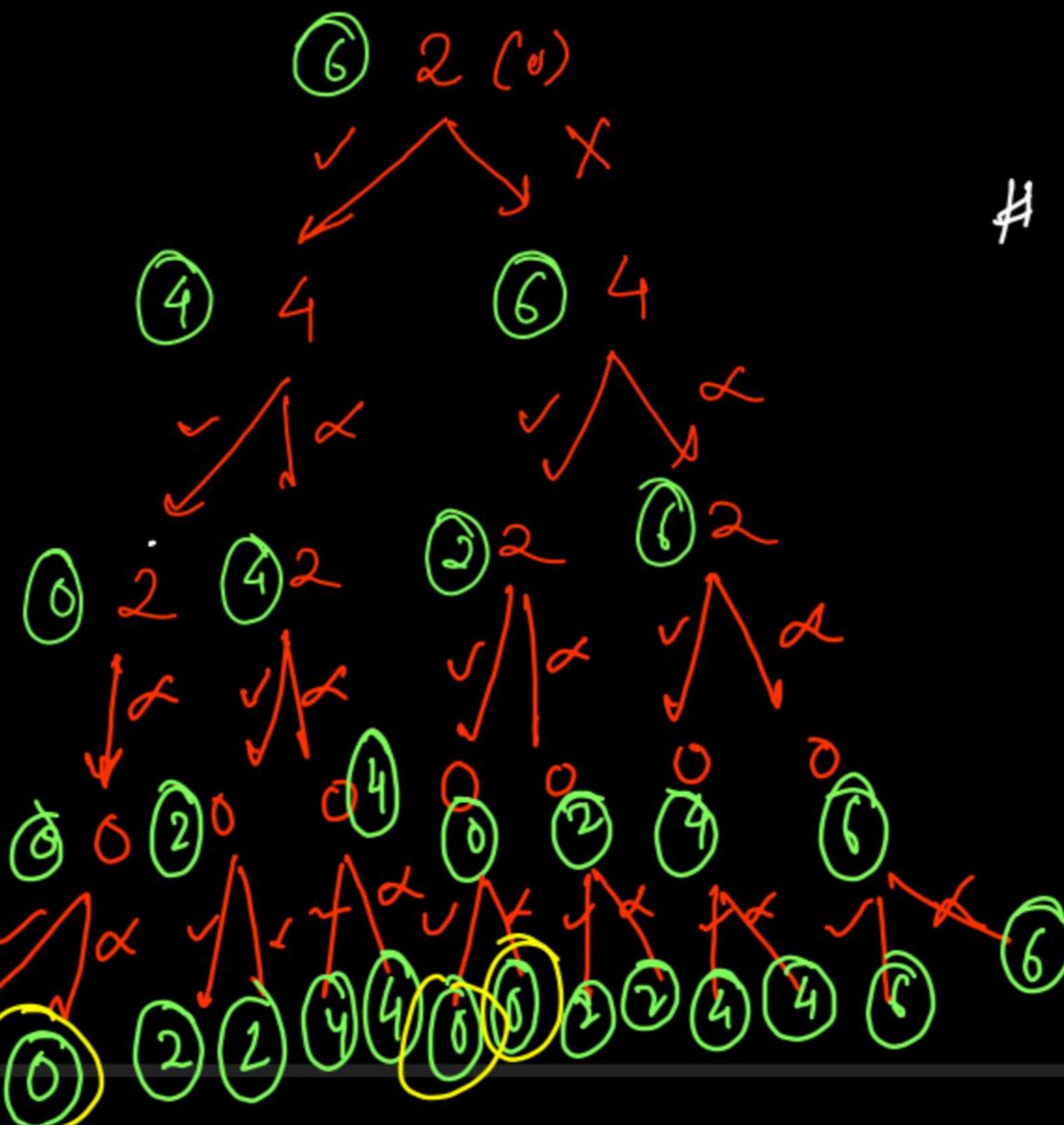
Q&A

Count Target Sum Subset

{ Perfect Sum }

12P DP
DP State
index
rem target

{ 0, 1, 2, 3 }
{ 2, 4, 2, 0 } target = 6



if 0s are present
& count the subset,
you will have to
go to the
last level

if (target == 0)
return 1;

```

public int memo(int index, int target, int[] arr, int[][] dp){
    if(index == arr.length){
        if(target == 0) return 1;
        return 0;
    }

    if(dp[index][target] != -1) return dp[index][target];

    int no = memo(index + 1, target, arr, dp);
    int yes = (target >= arr[index])
        ? memo(index + 1, target - arr[index], arr, dp) : 0;
    return dp[index][target] = (no + yes) % 1000000007;
}

```

```

public int perfectSum(int arr[], int n, int sum)
{
    int[][] dp = new int[n + 1][sum + 1];
    for(int i=0; i<=n; i++){
        for(int j=0; j<=sum; j++){
            dp[i][j] = -1;
        }
    }
    return memo(0, sum, arr, dp);
}

```

Memo

Time $\rightarrow O(N * \text{Target})$

Space $\rightarrow O(N * \text{Target})$
 (DP)

R.C.S $\rightarrow O(N)$

$\{0, 1, 2, 3, 4\}$

$\{2, 4, 0, 0, 2\}$

Tabulation $\left\{ \begin{array}{l} \text{0-1 knapsack} \\ \text{(variant)} \end{array} \right\}$ $\left\{ \begin{array}{l} \{0, 1\} \{0, 1\} \\ \{0, 1\}, \{0, 0\} \end{array} \right\}$

10:45

	0	1	2	3	4	5	6 (target)
0 $\{\}$	1 $\{\}$	0	0	0	0	0	0
1 $[2]$	1 $\{\}$	0	1 $\{2\}$	0	0	0	0
2 $[4]$	1 $\{\}$	0	1 $\{2\}$	0	1 $\{4\}$	0	1 $\{2, 4\}$
3 $\{0\}$	2 $\{\}$	0	2 $\{2\}$	0	2 $\{4\}$	0	2 $\{2, 4\}$
4 $[0]$	4 $\{\}$	0	4 $\{2\}$	0	4 $\{4\}$	0	4 $\{2, 4\}$
5 $[2]$	4	0	8	0	8	0	8

Row by Row
Top to down

```

public int perfectSum(int arr[], int n, int target)
{
    int[][] dp = new int[n + 1][target + 1];
    dp[0][0] = 1; // Empty Subset to form 0 Target

    for(int i=1; i<=n; i++){
        for(int j=0; j<=target; j++){
            int no = dp[i - 1][j]; // No Call
            int yes = (j >= arr[i - 1]) ? dp[i - 1][j - arr[i - 1]] : 0;

            dp[i][j] = (no + yes) % 1000000007;
        }
    }

    return dp[n][target];
}

```

Time $\rightarrow O(N * \text{Target})$
 Space $\rightarrow O(N * \text{Target})$

2D DP

```

public int perfectSum(int arr[], int n, int target) {
    int[] dp = new int[target + 1];
    dp[0] = 1; // Empty Subset to form 0 Target

    for(int i=1; i<=n; i++){
        int[] newdp = new int[target + 1];

        for(int j=0; j<=target; j++){
            int no = dp[j]; // No Call
            int yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : 0;

            newdp[j] = (no + yes) % 1000000007;
        }

        dp = newdp;
    }

    return dp[target];
}

```

Time $\rightarrow O(N * \text{Target})$
 Space $\rightarrow O(\text{Target})$



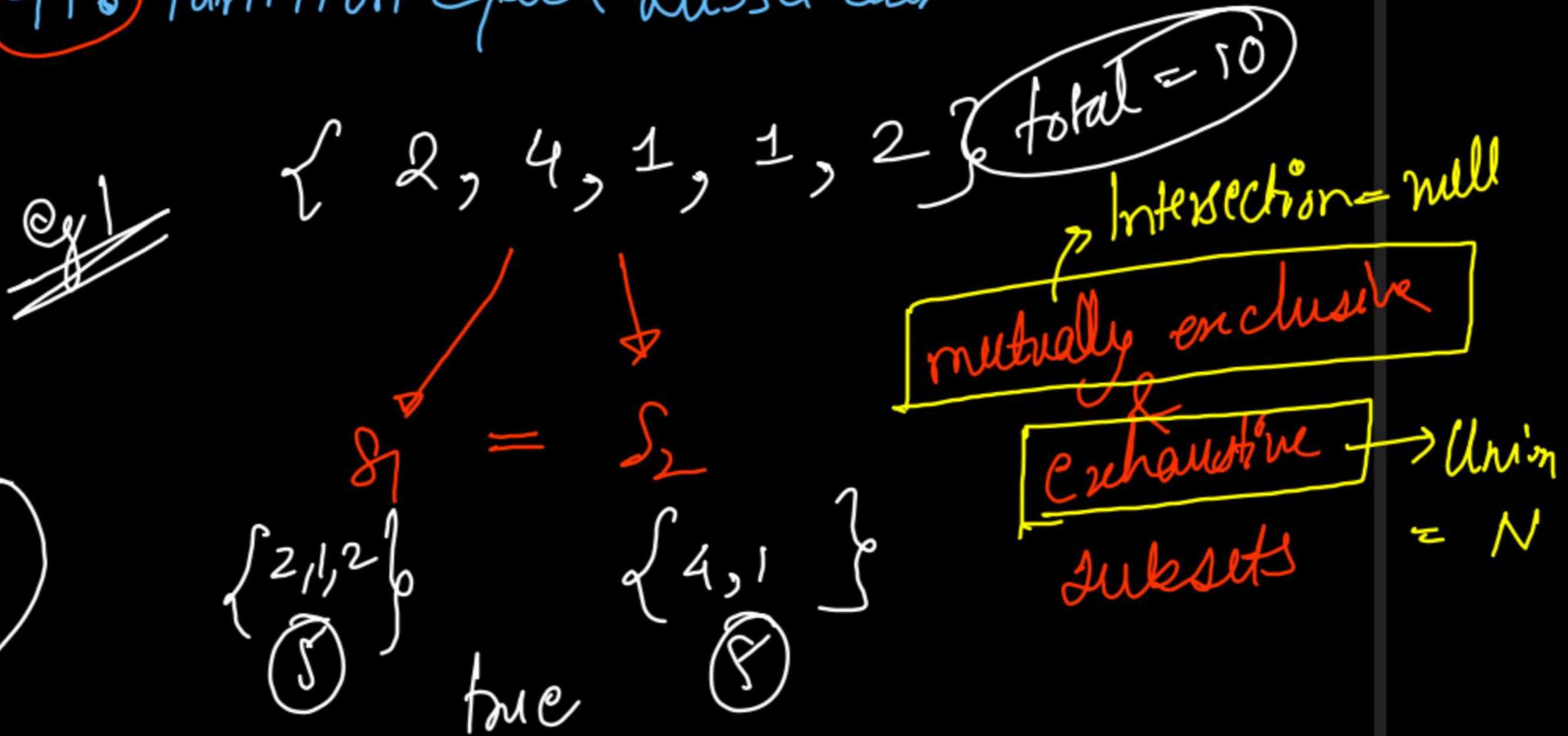
Check if Target Sum subset is Present or Not

```
public class Solution {  
    public int solve(int[] arr, int target) {  
        int n = arr.length;  
        boolean[] dp = new boolean[target + 1];  
        dp[0] = true; // Empty Subset to form 0 Target  
  
        for(int i=1; i<=n; i++){  
            boolean[] newdp = new boolean[target + 1];  
  
            for(int j=0; j<=target; j++){  
                boolean no = dp[j]; // No Call  
                boolean yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : false;  
  
                newdp[j] = no || yes;  
            }  
  
            dp = newdp;  
        }  
  
        return (dp[target] == true) ? 1 : 0;  
    }  
}
```

416 Partition Equal Subset Sum

$$\begin{aligned} S_1 + S_2 &= \text{total} \\ S_1 &= S_2 \end{aligned}$$

$$\begin{aligned} 2S_1 &= \text{total} \\ S_1 &= \frac{\text{total}}{2} \end{aligned}$$



~~Q2~~

$\{3, 4, 2, 5, 2, 1\}$ total = 17
 $\{3, 4, 1\} \neq \{2, 5, 2\}$ odd \downarrow false

~~eg2~~

$$\{1, 2, 3, 8\} \quad \text{total} = 14$$

⑦ ⑧

$$\{1, 2, 3\} \neq \{8\} \quad \text{false}$$

s_1 s_2

Algorithm

$$s_1 + s_2 = \text{total}; \quad s_1 = s_2$$

$$\Rightarrow \text{total \% 2 == 1 : false}$$

\Rightarrow check target sumsubset(arr, total/2)

```

public boolean checkTargetSumSubset(int[] arr, int target) {
    int n = arr.length;
    boolean[] dp = new boolean[target + 1];
    dp[0] = true; // Empty Subset to form 0 Target

    for(int i=1; i<=n; i++){
        boolean[] newdp = new boolean[target + 1];

        for(int j=0; j<=target; j++){
            boolean no = dp[j]; // No Call
            boolean yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : false;

            newdp[j] = no || yes;
        }

        dp = newdp;
    }

    return dp[target];
}

```

$O(N^k \text{target})$

time
space

```

public boolean canPartition(int[] nums) {
    int total = 0;
    for(int val: nums) total += val;

    if(total % 2 == 1) return false; // No Division Possible
    return checkTargetSumSubset(nums, total / 2);
}

```

main logic

leetcode [494] Target Sum

You are given an integer array `nums` and an integer `target`.

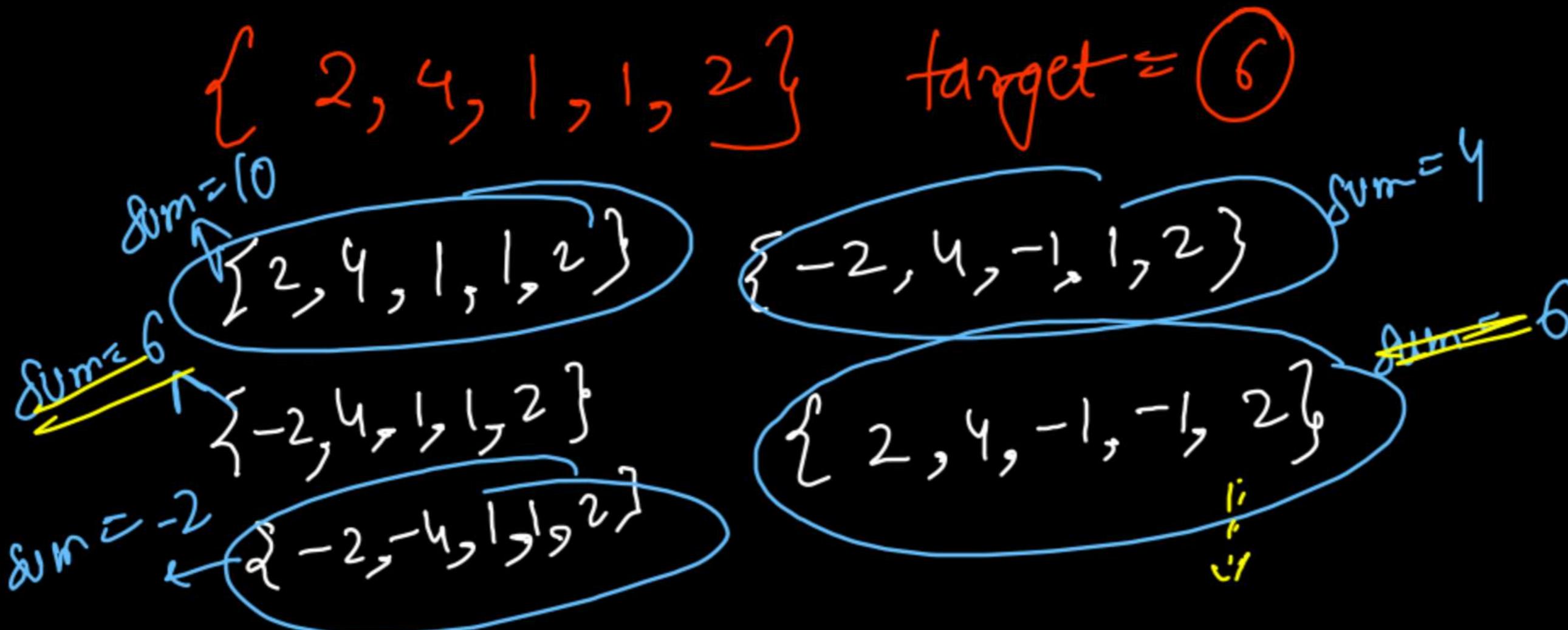
You want to build an **expression** out of `nums` by adding one of the symbols `'+'` and `'-'` before each integer in `nums` and then concatenate all the integers.

- For example, if `nums = [2, 1]`, you can add a `'+'` before `2` and a `'-` before `1` and concatenate them to build the expression `"+2-1"`.

Return the number of different **expressions** that you can build, which evaluates to `target`.

s_1

s_2



integer
7,0

{2, 4, 1, 1, 2} target = 6

way 1 {4, 1, 1, 2}

{2}

δ_1

δ_2

represents
subset of
elements
having

$$\delta_1 - \delta_2$$

$$= (4 + 1 + 1 + 2) - (2)$$
$$= 6$$

X sign

represents
subset of
elements
having -ve
sign

way 2

{2, 4, 2}

δ_1

{1, 1}

δ_2

$$\delta_1 - \delta_2$$

$$2 + 4 + 2 - 1 - 1$$

$$= 6$$

We can also find $S_2 \approx \frac{\text{total} + \text{target}}{2}$
But have larger target, hence more time
Conditionally

Divide array into two mutually-exclusive
and exhaustive S_1 & S_2 if $S_1 - S_2 = \text{target}$

$$S_1 + S_2 = \text{total} \quad \text{--- (1)}$$

$$2S_2 = \text{total} - \text{target}$$

$$\Rightarrow S_2 = \frac{\text{total} - \text{target}}{2}$$

Check if there exists a subset with target as S_2 .

constraints
1
2
total-target > target
total-target < target
hence lesser TC

```
public int findTargetSumWays(int[] nums, int target) {  
    int total = 0;  
    for(int val: nums) total += val;  
  
    if(target > total)  
        return 0; // Even if every element is +ve, S1 - S2 = total  
  
    if((total - target) % 2 == 1)  
        return 0;  
  
    return countTargetSumSubset(nums, (total - target) / 2);  
}
```

$O(N * \text{target})$ time

$O(\text{target})$ space

28 // { 10, 4, 3 } target = 2

{ 10, 4, 3 }
sum = 17



{ 10, 4, 3 } { -10, -4, 3 }

{ -10, 4, 3 } { 10, 4, -3 } ve

{ 10, -4, 3 } { -10, -4, -3 }

{ 10, 4, -3 } { 10, -4, -3 }

{ -10, -4, -3 }
sum = -17

Dynamic Programming → lecture 16
Target sum subset variations

Minimum Difference Subsets!

{Tug of War - Different Size}

Print All Paths With Target
Sum Subset



K-partitions

Partition Into Subsets

Tug Of War - Same Size

Minimize Difference Subsets.

Given an integer array A containing N integers.

You need to divide the array A into two subsets S1 and S2 such that the absolute difference between their sums is minimum.

Find and return this minimum possible absolute difference.

NOTE:

- Subsets can contain elements from A in any order (not necessary to be contiguous).
- Each element of A should belong to any one subset S1 or S2, not both.
- It may be possible that one subset remains empty.

{Subarray} {subset}
(mutually exclusive & exhaustive)

$$S_1 = \{4, 7, 10, 13\}$$

$$S_2 = \{16, 20\}$$

34

$$|S_1 - S_2| = |34 - 36| = 2$$

Eg

$$\{4, 7, 10, 13, 16, 20\}$$

$$|S_1 - S_2| = \text{minimum}$$

$$S_1 = \{4, 20, 10\}$$

34

$$|S_1 - S_2| = |34 - 36| = 2$$

$$S_2 = \{2, 13, 16\}$$

36

$\{4, 7, 10, 13, 16, 20\}$ total = $\textcircled{12}$
 we have the closest
 Target sum subset to
 $\textcircled{1} \quad S_1 - S_2 = \min \text{diff}$ S_1 is big, S_2 is smaller
 $\textcircled{2} \quad S_1 + S_2 = \text{total}$ $(\text{total}/2)$

$\{0, 1, 2, 3\}$
 $\{2, 4, 0, 0\}$
 $4-2 = \textcircled{2}$
 $S_1 - S_2 = \textcircled{2}$
 $S_1 \quad S_2$

Sum \leftarrow total sum
 Row by Row
 Top to Bottom

	0	1	2	3	4	5	6
0	1 ^{}	0	0	0	0	0	0
1 [2]	1 ^{}	0	1 ^{2}	0	0	0	0
2 [4]	1 ^{}	0	1 ^{2}	0	1 ^{4}	0	1 ^{2,4}
3 [0]	2 ^{0}	0	2 ^{2}	0	2 ^{4}	0	2 ^{2,4}
4 [0]	4 ^{0}	0	4 ^{2,0}	0	4 ^{4,0}	0	4 ^{2,4,0}

```

int total = 0;
for(int val: arr) total += val;

boolean[] dp = new boolean[total + 1];
dp[0] = true; // Empty Subset to form 0 total

for(int i=1; i<=n; i++){
    boolean[] newdp = new boolean[total + 1];

    for(int j=0; j<=total; j++){
        boolean no = dp[j]; // No Call
        boolean yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : false;

        newdp[j] = no || yes;
    }

    dp = newdp;
}

```

```

int half = (total + 1) / 2;
for(int s1=half; s1<=total; s1++){
    if(dp[s1] == true){
        return (s1 - (total - s1));
    }
}

return total;

```

$$\frac{(6+1)}{2} = 3$$

total = 6

$$\frac{(7+1)}{2}$$

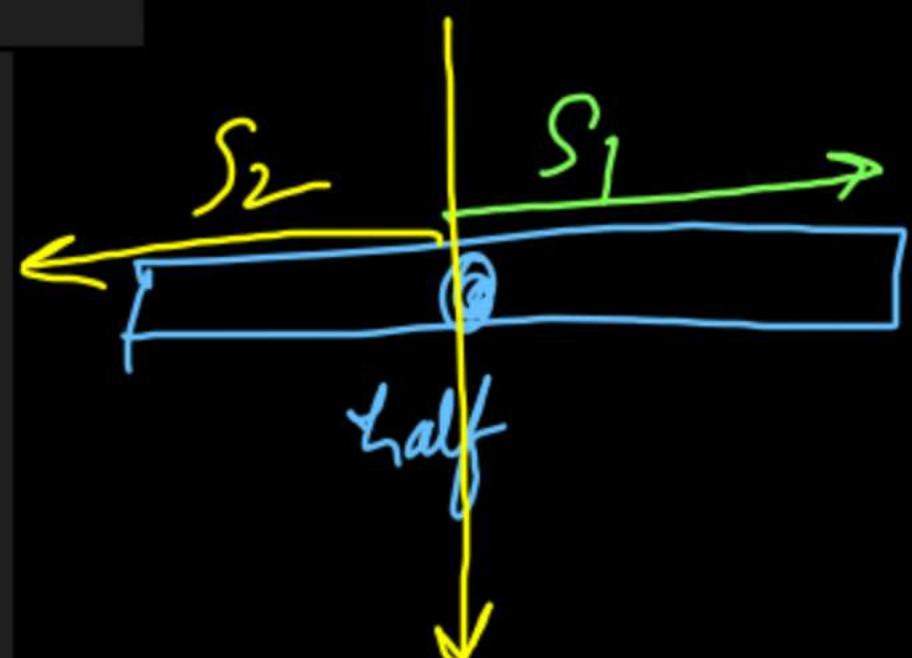
total = 7

$$S_1 - S_2$$

$$3 - 3 = 0$$

$$S_1 - S_2$$

$$4 - 3 =$$



Print All Subsets with given target

#Recursion $\rightarrow O(2^N)$

using DP
(Tabulation)
BFS | DFS

worst case $\rightarrow O(2^N)$
avg case \rightarrow (either exponential or polynomial)

	0	1	2	3	4	5	6 (target)
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	1	0	0	0	0
3	0	1	2	1	0	0	0
4	0	1	2	3	1	0	0
5	0	1	2	3	2	0	0
6	0	1	2	3	4	0	0
7	0	1	2	3	4	1	0
8	0	1	2	3	4	2	0
9	0	1	2	3	4	3	0
10	0	1	2	3	4	4	0
11	0	1	2	3	4	5	0
12	0	1	2	3	4	6	0
13	0	1	2	3	4	7	0
14	0	1	2	3	4	8	0
15	0	1	2	3	4	8	1
16	0	1	2	3	4	8	2
17	0	1	2	3	4	8	3
18	0	1	2	3	4	8	4
19	0	1	2	3	4	8	5
20	0	1	2	3	4	8	6
21	0	1	2	3	4	8	7
22	0	1	2	3	4	8	8
23	0	1	2	3	4	8	9
24	0	1	2	3	4	8	10
25	0	1	2	3	4	8	11
26	0	1	2	3	4	8	12
27	0	1	2	3	4	8	13
28	0	1	2	3	4	8	14
29	0	1	2	3	4	8	15
30	0	1	2	3	4	8	16
31	0	1	2	3	4	8	17
32	0	1	2	3	4	8	18
33	0	1	2	3	4	8	19
34	0	1	2	3	4	8	20
35	0	1	2	3	4	8	21
36	0	1	2	3	4	8	22
37	0	1	2	3	4	8	23
38	0	1	2	3	4	8	24
39	0	1	2	3	4	8	25
40	0	1	2	3	4	8	26
41	0	1	2	3	4	8	27
42	0	1	2	3	4	8	28
43	0	1	2	3	4	8	29
44	0	1	2	3	4	8	30
45	0	1	2	3	4	8	31
46	0	1	2	3	4	8	32
47	0	1	2	3	4	8	33
48	0	1	2	3	4	8	34
49	0	1	2	3	4	8	35
50	0	1	2	3	4	8	36
51	0	1	2	3	4	8	37
52	0	1	2	3	4	8	38
53	0	1	2	3	4	8	39
54	0	1	2	3	4	8	40
55	0	1	2	3	4	8	41
56	0	1	2	3	4	8	42
57	0	1	2	3	4	8	43
58	0	1	2	3	4	8	44
59	0	1	2	3	4	8	45
60	0	1	2	3	4	8	46
61	0	1	2	3	4	8	47
62	0	1	2	3	4	8	48
63	0	1	2	3	4	8	49
64	0	1	2	3	4	8	50
65	0	1	2	3	4	8	51
66	0	1	2	3	4	8	52
67	0	1	2	3	4	8	53
68	0	1	2	3	4	8	54
69	0	1	2	3	4	8	55
70	0	1	2	3	4	8	56
71	0	1	2	3	4	8	57
72	0	1	2	3	4	8	58
73	0	1	2	3	4	8	59
74	0	1	2	3	4	8	60
75	0	1	2	3	4	8	61
76	0	1	2	3	4	8	62
77	0	1	2	3	4	8	63
78	0	1	2	3	4	8	64
79	0	1	2	3	4	8	65
80	0	1	2	3	4	8	66
81	0	1	2	3	4	8	67
82	0	1	2	3	4	8	68
83	0	1	2	3	4	8	69
84	0	1	2	3	4	8	70
85	0	1	2	3	4	8	71
86	0	1	2	3	4	8	72
87	0	1	2	3	4	8	73
88	0	1	2	3	4	8	74
89	0	1	2	3	4	8	75
90	0	1	2	3	4	8	76
91	0	1	2	3	4	8	77
92	0	1	2	3	4	8	78
93	0	1	2	3	4	8	79
94	0	1	2	3	4	8	80
95	0	1	2	3	4	8	81
96	0	1	2	3	4	8	82
97	0	1	2	3	4	8	83
98	0	1	2	3	4	8	84
99	0	1	2	3	4	8	85
100	0	1	2	3	4	8	86
101	0	1	2	3	4	8	87
102	0	1	2	3	4	8	88
103	0	1	2	3	4	8	89
104	0	1	2	3	4	8	90
105	0	1	2	3	4	8	91
106	0	1	2	3	4	8	92
107	0	1	2	3	4	8	93
108	0	1	2	3	4	8	94
109	0	1	2	3	4	8	95
110	0	1	2	3	4	8	96
111	0	1	2	3	4	8	97
112	0	1	2	3	4	8	98
113	0	1	2	3	4	8	99
114	0	1	2	3	4	8	100
115	0	1	2	3	4	8	101
116	0	1	2	3	4	8	102
117	0	1	2	3	4	8	103
118	0	1	2	3	4	8	104
119	0	1	2	3	4	8	105
120	0	1	2	3	4	8	106
121	0	1	2	3	4	8	107
122	0	1	2	3	4	8	108
123	0	1	2	3	4	8	109
124	0	1	2	3	4	8	110
125	0	1	2	3	4	8	111
126	0	1	2	3	4	8	112
127	0	1	2	3	4	8	113
128	0	1	2	3	4	8	114
129	0	1	2	3	4	8	115
130	0	1	2	3	4	8	116
131	0	1	2	3	4	8	117</

```
Queue<Pair> q = new ArrayDeque<>();
q.add(new Pair(n, target, ""));

while(q.size() > 0){
    Pair curr = q.remove();
    if(curr.row == 0){
        System.out.println(curr.psf);
        continue;
    }

    int row = curr.row;
    int item = arr[row - 1];
    int col = curr.col;

    // NO
    if(dp[row - 1][col] > 0){
        q.add(new Pair(row - 1, col, curr.psf));
    }

    // YES
    if(col >= item && dp[row - 1][col - item] > 0)
        q.add(new Pair(row - 1, col - item, (row - 1)
    }
}
```

	0	1	2	3	4	5	6 (Target)
0	0	0	0	0	0	0	0
1 [2]	1 {2}	0	1 {2,3}	0	0	0	0
2 [4]	1 {2}	0	1 {2,3}	0	1 {4,5}	0	1 {7,4,5}
3 [0]	2 {0,3,4,5}	0	2 {2,3,4,5}	0	2 {4,5,7,4,5}	0	2 {4,5,7,4,5,0,3}
4 [0]	4 {0,3,4,5}	0	4 {2,3,4,5}	0	4 {4,5,6,7,4,5}	0	4 {7,4,5,8,4,5,0,3}
5 [2]	4	0	8	0	8	0	8

```

graph TD
    A["5, 6"] --> B["9, 6"]
    A --> C["4, 4, \"4\""]
    B --> D["3, 6"]
    B --> E["3, 6, \"3\""]
    D --> F["2, 6"]
    D --> G["2, 6, \"2\""]
    E --> H["1, 3"]
    E --> I["1, 3, \"3\""]
    E --> J["1, 3, \"3\""]
    C --> K["3, 4"]
    C --> L["3, 4, \"4\""]
    K --> M["2, 4"]
    K --> N["2, 4, \"2\""]
    L --> O["2, 4"]
    L --> P["2, 4, \"2\""]

```

```
public static void DFS(int row, int col, String psf, int[] arr, int[][] dp){  
    if(row == 0){  
        System.out.println(psf);  
        return;  
    }  
  
    int item = arr[row - 1];  
  
    if(dp[row - 1][col] > 0){  
        DFS(row - 1, col, psf, arr, dp);  
    }  
  
    if(col >= item && dp[row - 1][col - item] > 0){  
        DFS(row - 1, col - item, (row - 1) + " " + psf, arr, dp);  
    }  
}
```

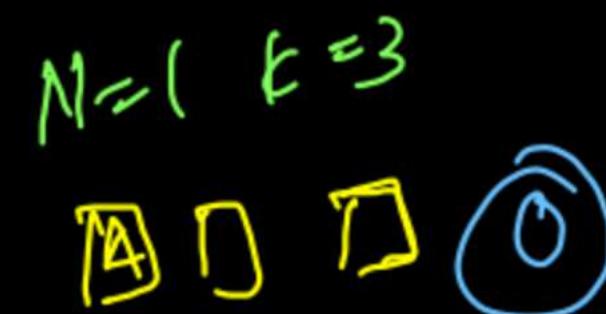
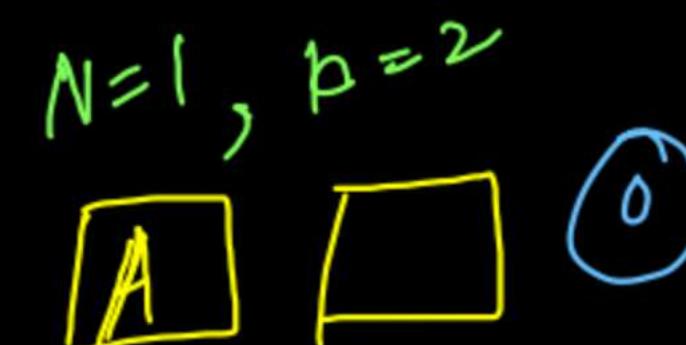
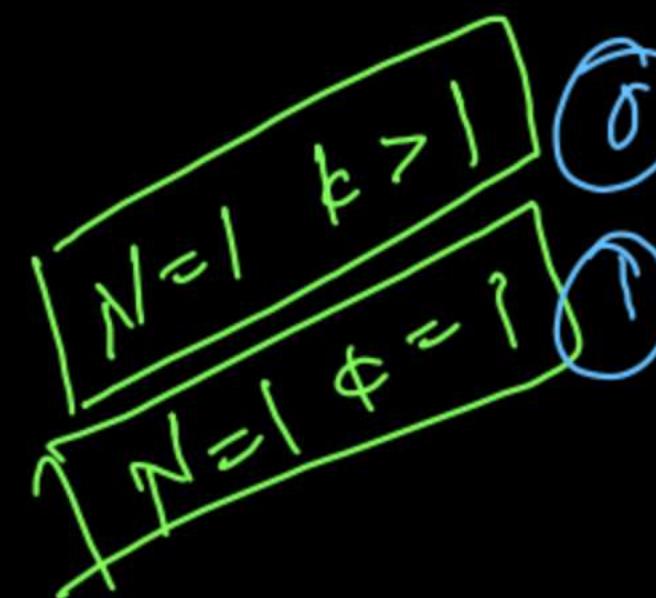
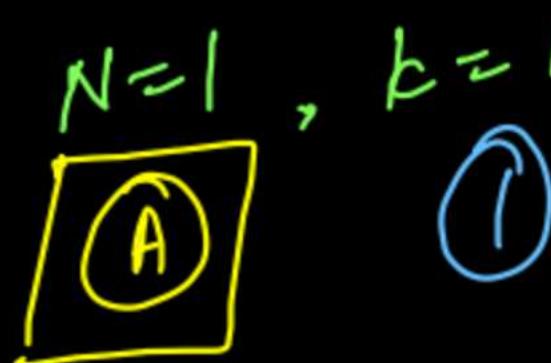
```
DFS(n, target, "", arr, dp);
```

Print All ways to partition {Recursion}
n elements into K non-empty

No permutations
should be printed

Subsets

eg

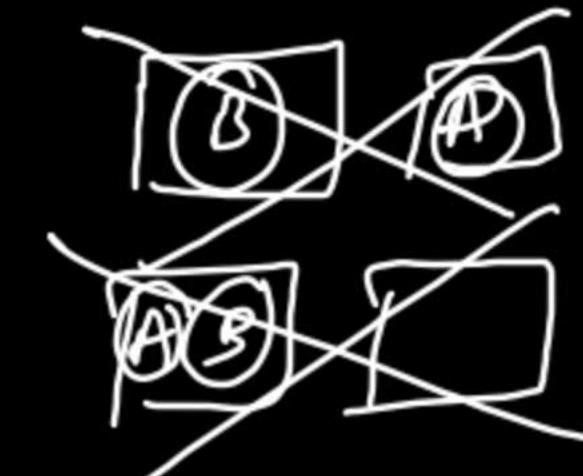


eg²

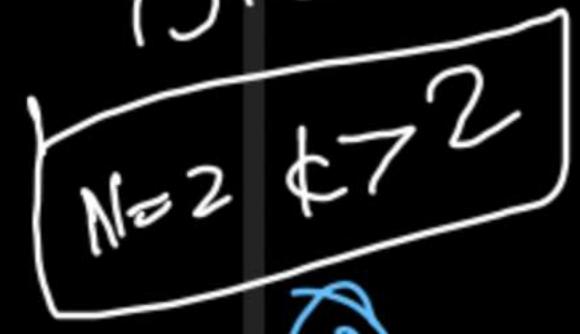
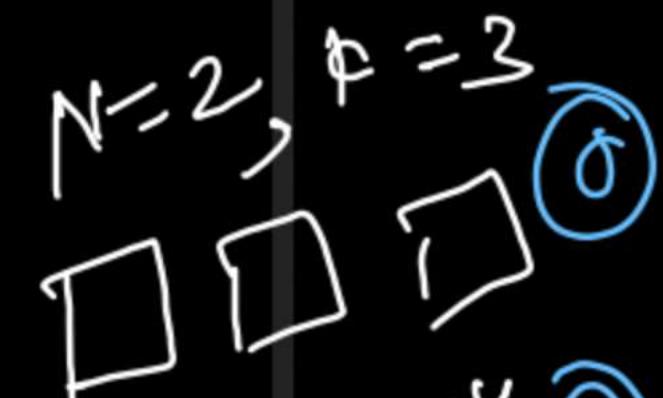
$N=2, k=1$



$N=2, \phi=2$



$N=2, k=2$



N=3

$\Rightarrow k=1$

$\boxed{A, B, C}$ ①

$\Rightarrow f=2$

\boxed{A} $\boxed{B, C}$

$\boxed{A, B}$ \boxed{C} ③

$\boxed{A, C}$ \boxed{B}

~~$\boxed{A, B, C}$~~

~~\boxed{B}~~ ~~\boxed{AC}~~ ~~$\boxed{B, C}$~~ ~~\boxed{C}~~

$\Rightarrow k=3$ ①

\boxed{A} \boxed{B} \boxed{C}

$\Rightarrow k > 3 (4, 5, 6 \dots)$ ②

N=4

$f=1$

~~$\boxed{A, B, C, D}$~~ ①

$t=2$

\boxed{A} \boxed{BCD}

\boxed{AB} \boxed{CD}

\boxed{AC} \boxed{BD}

\boxed{AD} \boxed{BC}

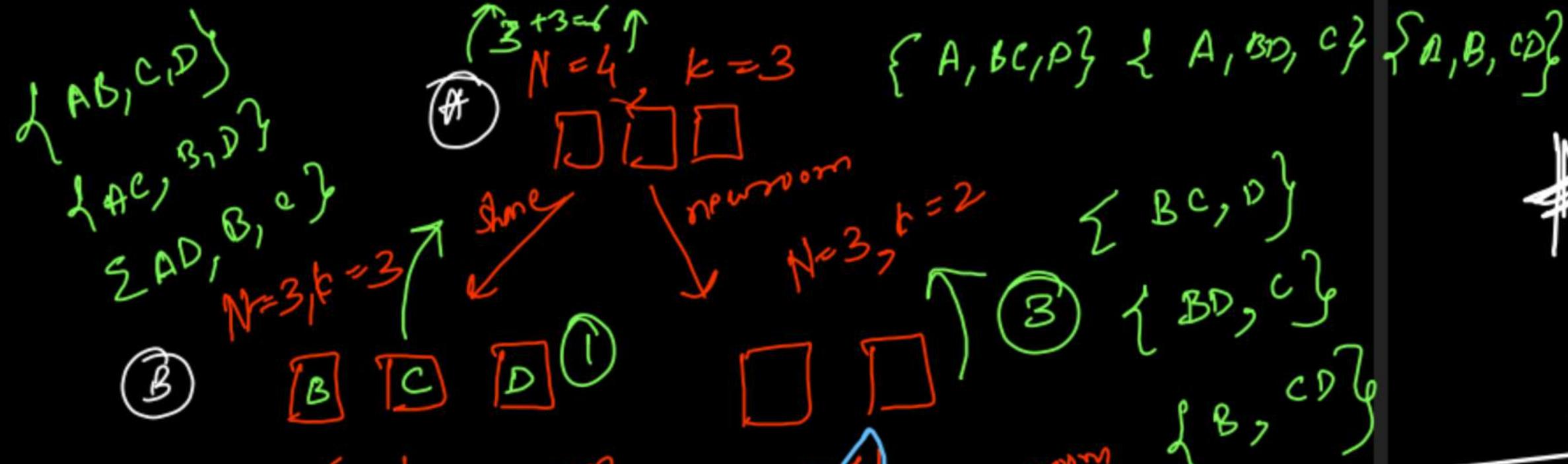
\boxed{ABC} \boxed{D}

\boxed{ABD} \boxed{C}

\boxed{ACD} \boxed{B}

$t=4$ ①

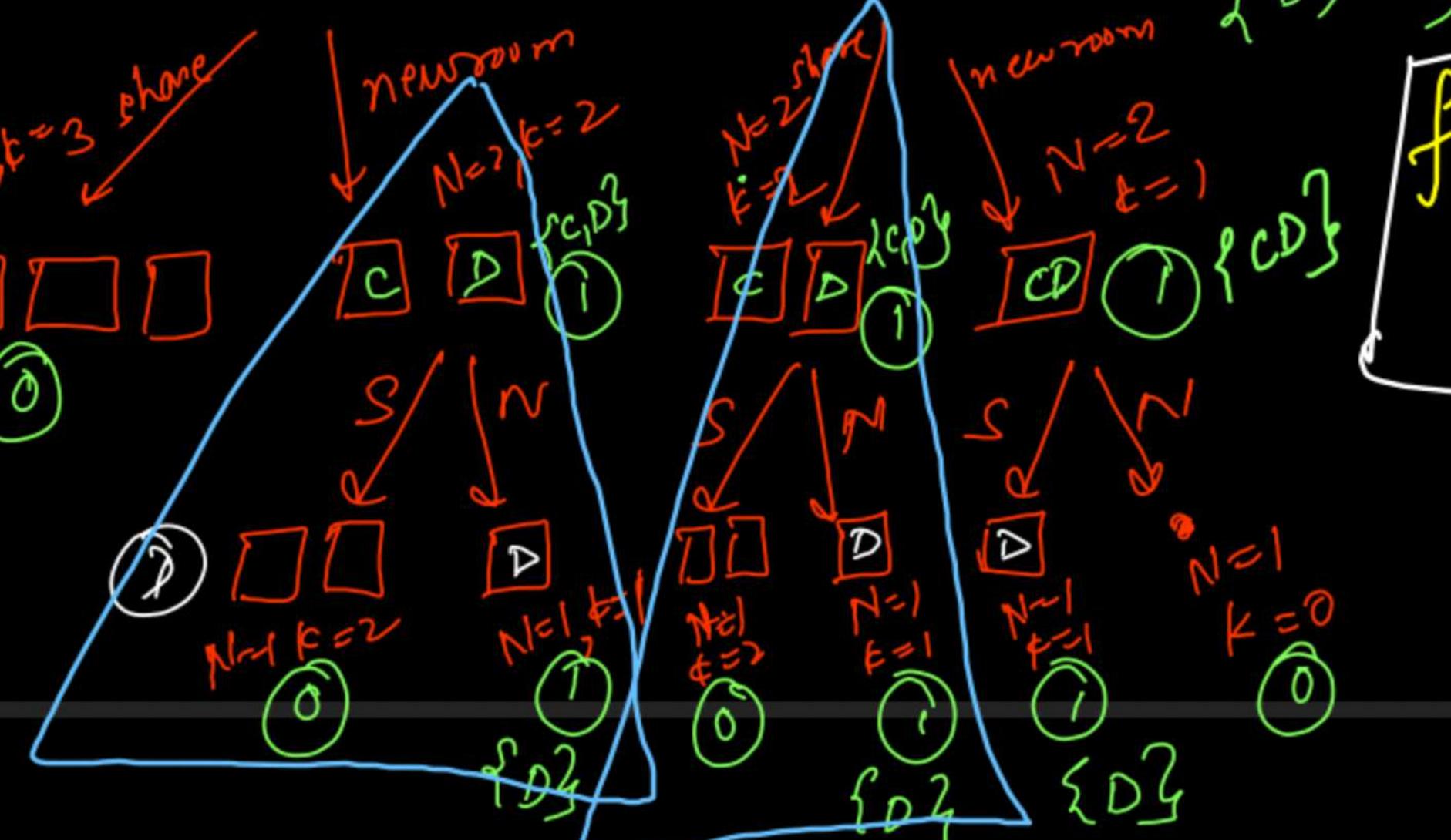
\boxed{A} $\boxed{B, C, D}$



DP State
 $[N, k]$

Recurrence Relation

$$f(N, k) = f(N-1, k) * k + f(N-1, k-1)$$



```
// All Ways -> All Rooms -> All Persons
public static ArrayList<ArrayList<ArrayList<Integer>>> solution(int currPerson, int n, int k) {
    ArrayList<ArrayList<ArrayList<Integer>>> resWays = new ArrayList<>();
    if(n < k || k <= 0){
        return resWays;
    }
    if(currPerson == n){
        if(k == 1){
            resWays.add(new ArrayList<>());
            resWays.get(0).add(new ArrayList<>());
            resWays.get(0).get(0).add(currPerson);
        }
        return resWays;
    }
}
```

```
// Share with Remaining People
ArrayList<ArrayList<ArrayList<Integer>>> ways1 = solution(currPerson + 1, n, k);

for(ArrayList<ArrayList<Integer>> way: ways1){
    for(int i=0; i<way.size(); i++){
        ArrayList<ArrayList<Integer>> newWay = deepCopy(way);
        newWay.get(i).add(0, currPerson);
        resWays.add(newWay);
    }
}
```

```
// Private New Room
ArrayList<ArrayList<ArrayList<Integer>>> ways2 = solution(currPerson + 1, n, k - 1);

for(ArrayList<ArrayList<Integer>> way: ways2){
    ArrayList<ArrayList<Integer>> newWay = deepCopy(way);
    ArrayList<Integer> newRoom = new ArrayList<>();
    newRoom.add(currPerson);
    newWay.add(0, newRoom);
    resWays.add(newWay);
}

return resWays;
}
```

Count ways to partition N people in
K non-empty rooms

```
public static long partitionKSubset(int n, int k, long[][] dp) {  
    if(n < k || k == 0) return 0l;  
    if(n == 1){  
        if(k == 1) return 1l;  
        return 0l;  
    }  
    if(dp[n][k] != -1) return dp[n][k];  
  
    long share = partitionKSubset(n - 1, k, dp);  
    long newRoom = partitionKSubset(n - 1, k - 1, dp);  
  
    return dp[n][k] = ((share * k) + newRoom);  
}
```

Total \Rightarrow $O(N \times K)$
Without space optimization
 $\rightarrow O(N \times K)$ time
 \rightarrow Space $\rightarrow 2D DP$

With space optimization
 $\rightarrow O(N \times K)$ time
 \rightarrow Space $\rightarrow O(K)$
 \rightarrow DP DP

Time $\rightarrow O(N \times K)$
Space $\rightarrow O(N)$ Recursion call stack
 $O(N \times K)$ 2D DP

Remaining Questions

- # Tug of war \rightarrow same size
- # Buy & sell stocks \rightarrow k transactions

#VVIMP
DP on Grids → Dynamic Programming Lecture 17
15th May Sunday, 9 AM to 12 PM

- Minimum Path sum → Leetcode 67
- Triangle Path Sum → Leetcode 120
- Falling Path sum (Goldmine) → Leetcode 931
- Unique Paths
 - Version I - Leetcode 62
 - Version II - Leetcode 63

Minimum Path Sum (LC 66)

Source	0	1	2
0	5	22	17
1	2	18	6
2	7	9	3
3	5	3	7

destination

$$\# \text{ rows} = 4, \text{ cols} = 3$$

① → Greedy fail

② → Recursion → $O(2^{mn+n-2})$

$\binom{n-1}{m-1}$ → down
 $\binom{n-1}{m-1}$ → right

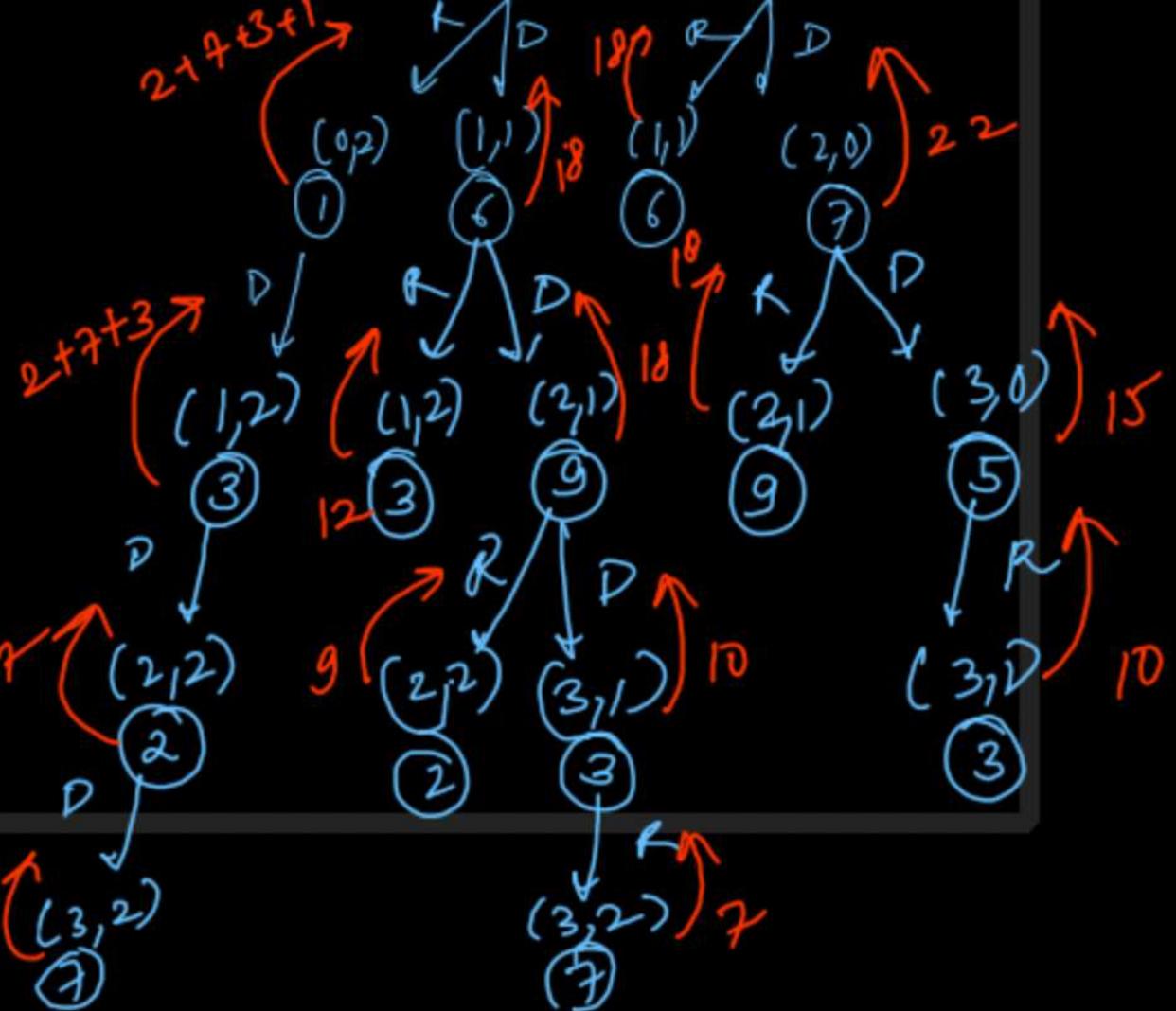
Possible ways
 (i, j)

1 unit
 horizontally
 right

$(i, j+1)$

1 unit
 vertically
 down

$(i+1, j)$



count }
 min }
 max } mid

```

public int helper(int row, int col, int[][] grid, int[][] dp){
    // Negative Base Case: Out of the Grid
    if(row >= grid.length || col >= grid[0].length)
        return Integer.MAX_VALUE;

    // Positive Base Case: Destination is Reached
    if(row == grid.length - 1 && col == grid[0].length - 1)
        return grid[row][col];

    if(dp[row][col] != -1) return dp[row][col];

    int horizontal = helper(row, col + 1, grid, dp);
    int vertical = helper(row + 1, col, grid, dp);

    return dp[row][col] = Math.min(horizontal, vertical) + grid[row][col];
}

public int minPathSum(int[][] grid) {
    int[][] dp = new int[grid.length + 1][grid[0].length + 1];
    for(int i=0; i<=grid.length; i++){
        for(int j=0; j<=grid[0].length; j++){
            dp[i][j] = -1;
        }
    }

    return helper(0, 0, grid, dp);
}

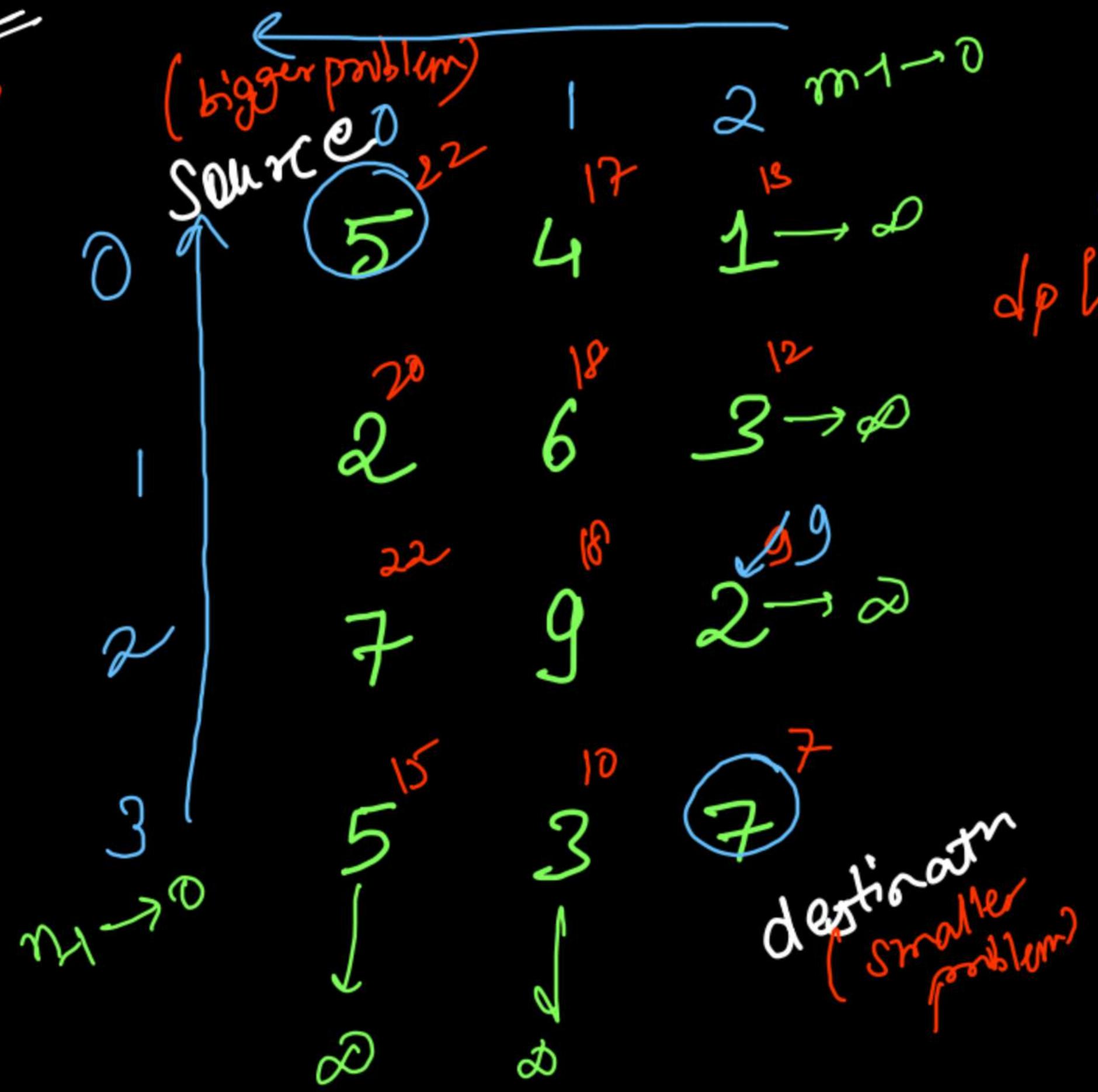
```

Memoization

Time $\rightarrow O(N \times M)$

Space $\rightarrow O(N+M-2)$ R.C.S-
 $O(N \times M)$ 2D DP

Tabulation
Bottom up



Recurrence Relat'

$$dp[i][j] = \min(dp[i+1][j], dp[i][j+1]) + grid[i][j]$$

```

public int minPathSum(int[][] grid) {
    int[][] dp = new int[grid.length + 1][grid[0].length + 1];
    for(int i=0; i<=grid.length; i++){
        for(int j=0; j<=grid[0].length; j++){
            dp[i][j] = Integer.MAX_VALUE;
        }
    }

    for(int i=grid.length-1; i>=0; i--){
        for(int j=grid[0].length-1; j>=0; j--){
            if(i == grid.length - 1 && j == grid[0].length - 1){
                dp[i][j] = grid[i][j];
                continue;
            }

            dp[i][j] = Math.min(dp[i + 1][j], dp[i][j + 1]) + grid[i][j];
        }
    }

    return dp[0][0];
}

```

Tabulation

$O(N \times M)$ Time

$O(N \times M)$ Space



```

public int minPathSum(int[][] grid) {
    int[] dp = new int[grid[0].length + 1];
    for(int j=0; j<=grid[0].length; j++){
        dp[j] = Integer.MAX_VALUE;
    }

    for(int i=grid.length-1; i>=0; i--){
        for(int j=grid[0].length-1; j>=0; j--){
            if(i == grid.length - 1 && j == grid[0].length - 1){
                dp[j] = grid[i][j];
                continue;
            }

            dp[j] = Math.min(dp[j], dp[j + 1]) + grid[i][j];
        }
    }

    return dp[0];
}

```

Space Optimization

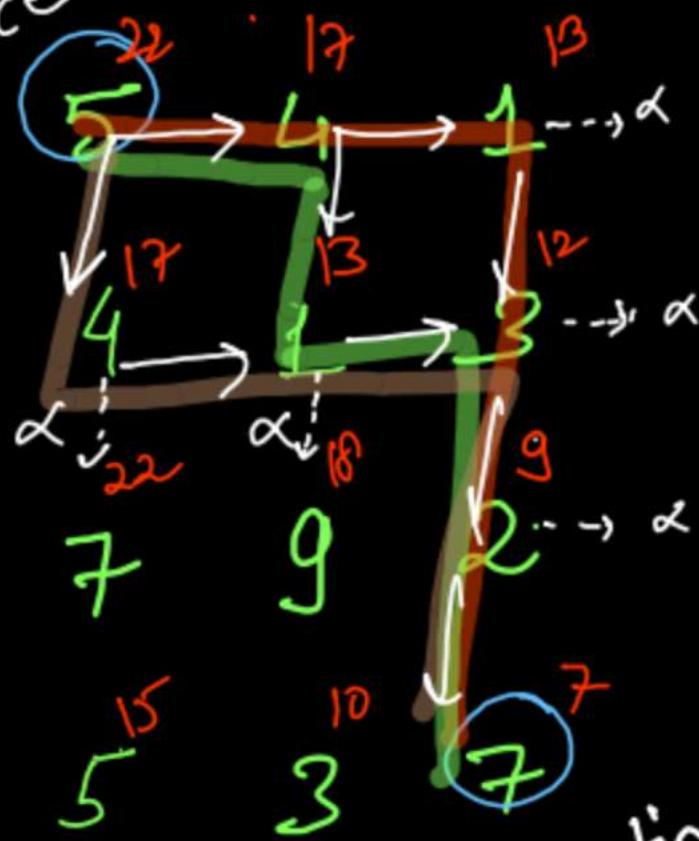
$O(N \times M)$ time

$O(M)$ space



Point All paths with
min cost

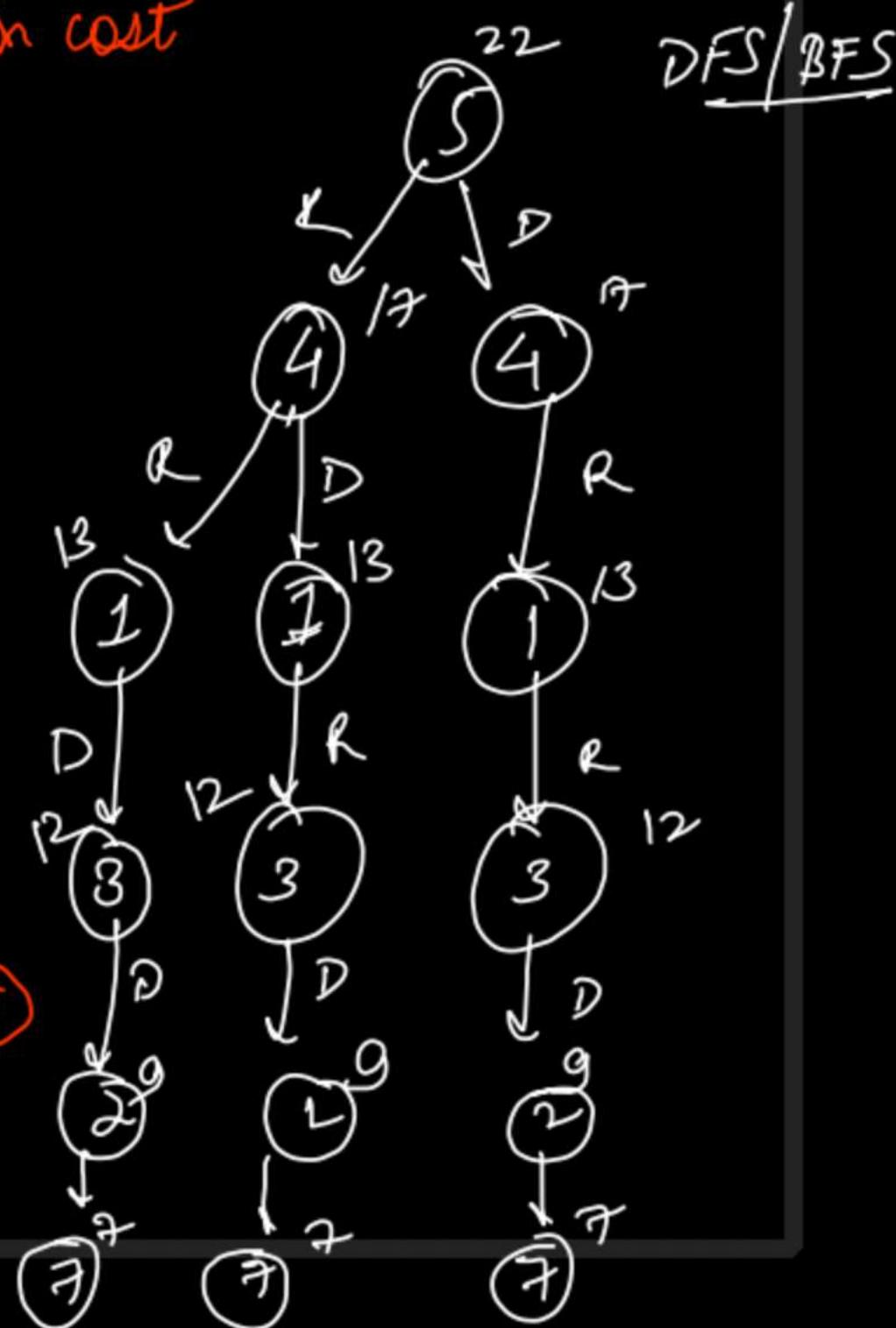
(larger problem)
source



destination
(smaller problem)

Time $\rightarrow O(\text{Polynomial})$

Space $\sum O(N+M-2)^R \cdot C.S$
 $O(N \cdot M)$ 2D DP table



DFS/BFS

11:10

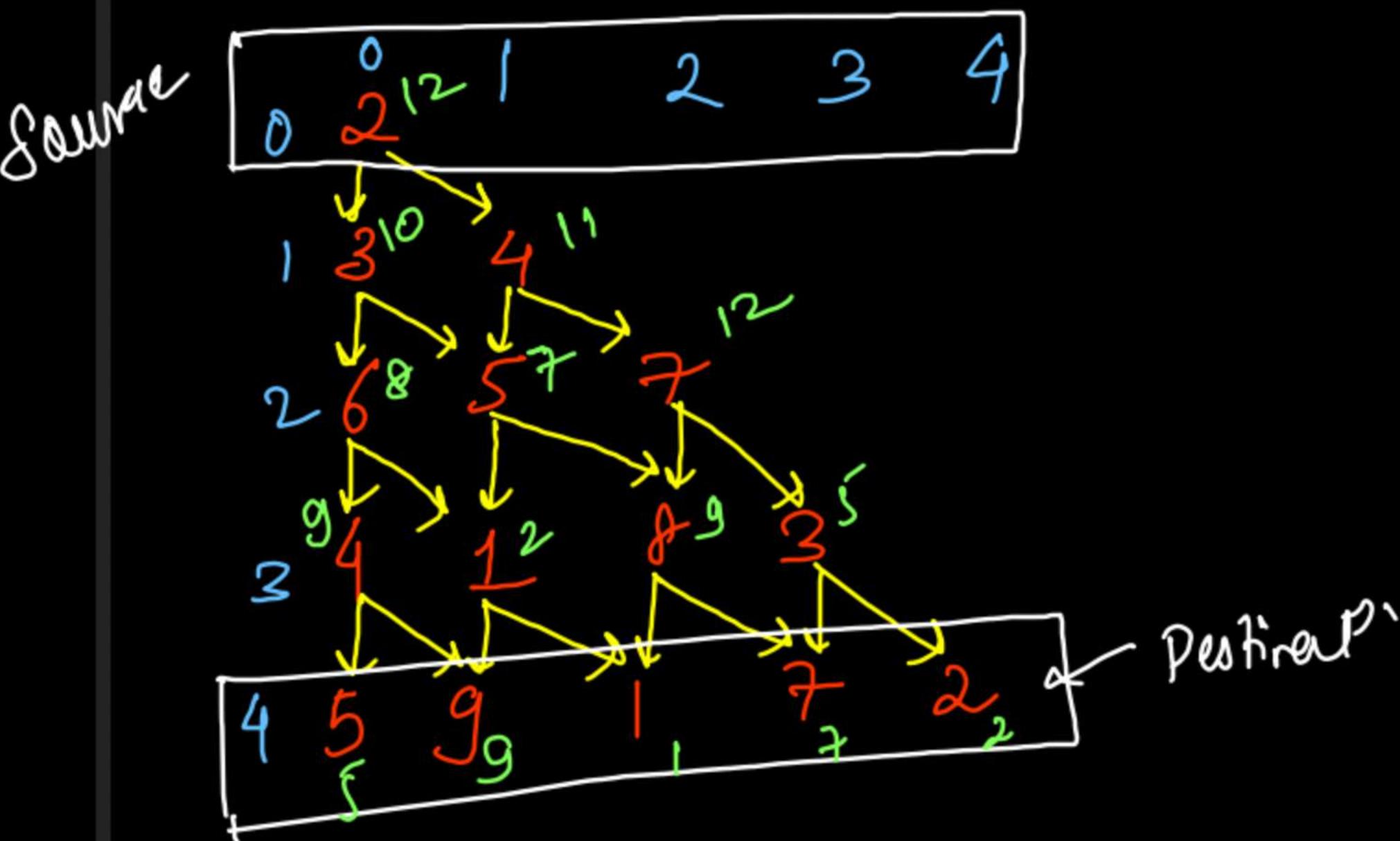
```
public static void DFS(int row, int col, int[][] grid, int[][] dp, String psf){  
    if(row == grid.length - 1 && col == grid[0].length - 1){  
        System.out.println(psf);  
        return;  
    }  
  
    int min = Math.min(dp[row + 1][col], dp[row][col + 1]);  
  
    if(dp[row + 1][col] == min){  
        DFS(row + 1, col, grid, dp, psf + "V");  
    }  
  
    if(dp[row][col + 1] == min){  
        DFS(row, col + 1, grid, dp, psf + "H");  
    }  
}
```

```
//write your code here  
int[][] dp = minPathSum(arr);  
System.out.println(dp[0][0]);  
DFS(0, 0, arr, dp, "");
```

Triangle (LC 120)

Given a triangle array, return the minimum path sum from top to bottom.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index $i + 1$ on the next row.



$$dp[i][j] = \min(dp[i+1][j], dp[i+1][j+1]) + grid[i][j]$$

single source (0,0) \curvearrowleft first row
multiple destination \curvearrowleft last row
④ Base case

```

class Solution {
    public int helper(List<List<Integer>> triangle, int row, int col, Integer[][] dp){
        if(row == triangle.size() - 1){
            // Entire Last Row is the Destination
            return triangle.get(row).get(col);
        }

        if(dp[row][col] != null) return dp[row][col];

        int down = helper(triangle, row + 1, col, dp);
        int rightdown = helper(triangle, row + 1, col + 1, dp);

        return dp[row][col] = (triangle.get(row).get(col) + Math.min(down, rightdown));
    }

    public int minimumTotal(List<List<Integer>> triangle) {
        int n = triangle.size();
        Integer[][] dp = new Integer[n + 1][n + 1];
        return helper(triangle, 0, 0, dp);
    }
}

```

Recursion

→ $O(2^{\text{Rows}})$ time
 $O(\text{Rows}) R \cdot C \cdot S$

Memoization

$O(\text{Rows} \cdot \text{Rows})$ time

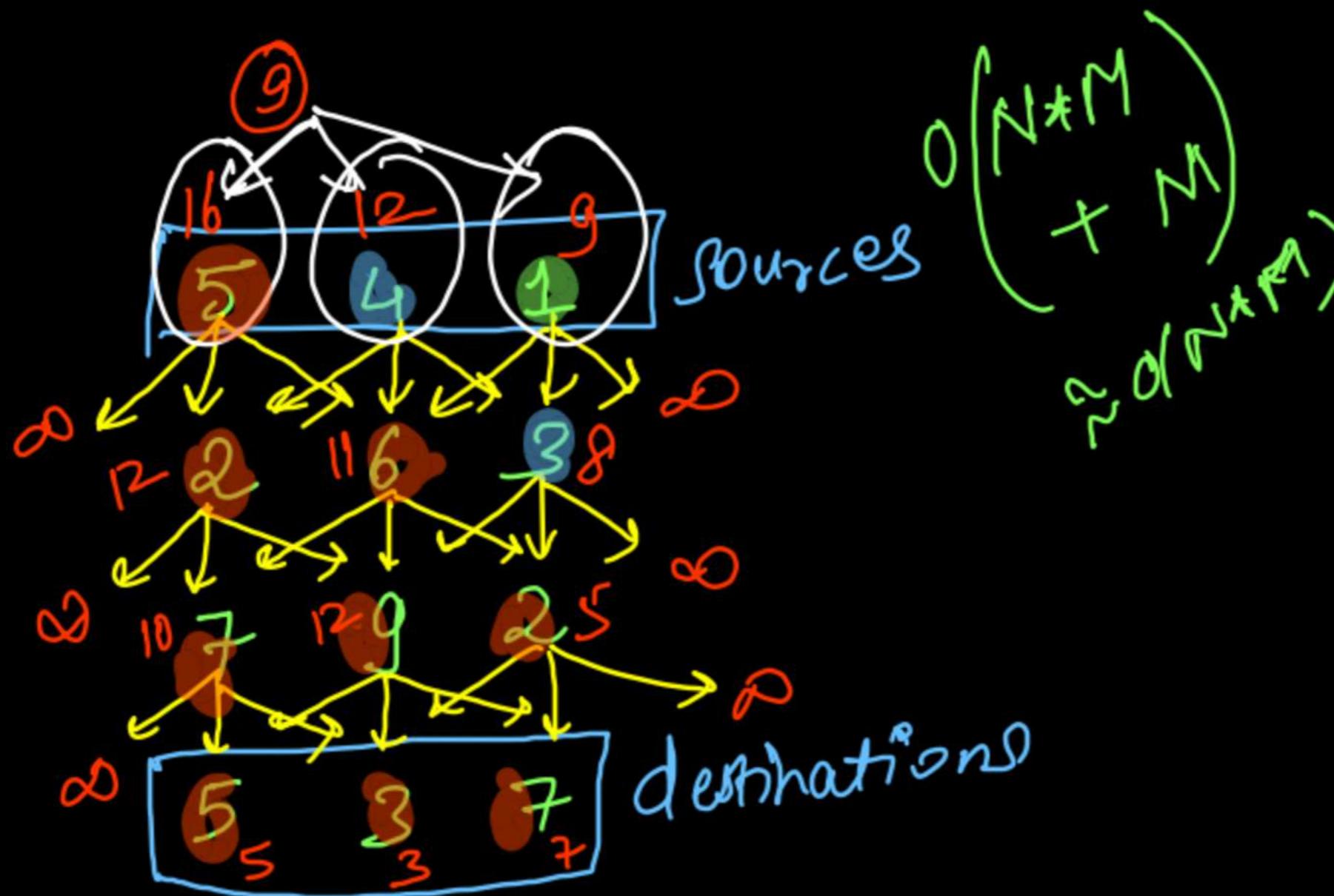
Tabulation

→ W/o space opt $\leftrightarrow O(R \times R)$ time
 $O(R \times R)$ 2D DP

→ with space opt $\leftrightarrow O(R \times R)$ time
 $O(2 \times R)$ 2 rows of 1D DP
 $O(\text{Rows}) R \cdot C \cdot S$

931

Minimum Falling Path Sum



Greedy → fail?

multiple source
↓
multiple destinations

possible move

right down
 $(i+1, j+1)$

down
 $(i+1, j)$

left down
 $(i+1, j-1)$

```

public int memo(int row, int col, int[][] grid, int[][] dp){
    if(col < 0 || col >= grid[0].length){
        // Out of Matrix: Negative Base Case
        return Integer.MAX_VALUE;
    }

    if(row == grid.length - 1){
        // Entire last row is the destination
        return grid[row][col];
    }

    if(dp[row][col] != -1) return dp[row][col];

    int leftDown = memo(row + 1, col - 1, grid, dp);
    int down = memo(row + 1, col, grid, dp);
    int rightDown = memo(row + 1, col + 1, grid, dp);

    return dp[row][col] = Math.min(down, Math.min(leftDown, rightDown)) + grid[row][col];
}

```

} Multiple Definition

DP table → only filled only once
Time → $O(N * M)$

Space → $O(N * M)$
2D DP

```

public int minFallingPathSum(int[][] grid) {
    int[][] dp = new int[grid.length + 1][grid[0].length + 1];
    for(int i=0; i<dp.length; i++){
        for(int j=0; j<dp[0].length; j++){
            dp[i][j] = -1;
        }
    }

    int min = Integer.MAX_VALUE;

    // Call for Each Element of the first row as the source node
    // DP table is going to be filled only once: Overall Time:  $O(N * M)$ 
    for(int src=0; src<grid[0].length; src++){
        int path = memo(0, src, grid, dp);
        min = Math.min(min, path);
    }

    return min;
}

```

Multiple sources {

Dynamic Programming

Lecture - 18

17th May, 10 PM to 12 AM

⑥ Unique Paths - ①

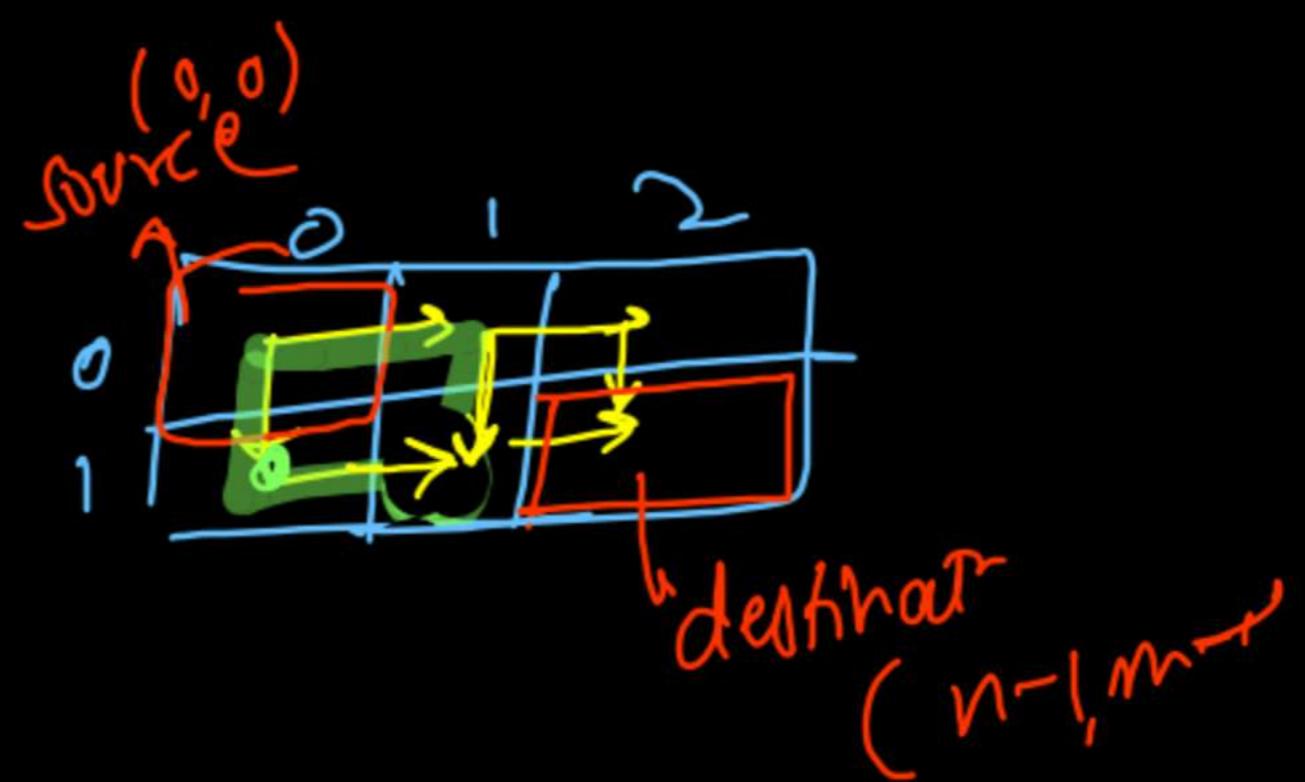
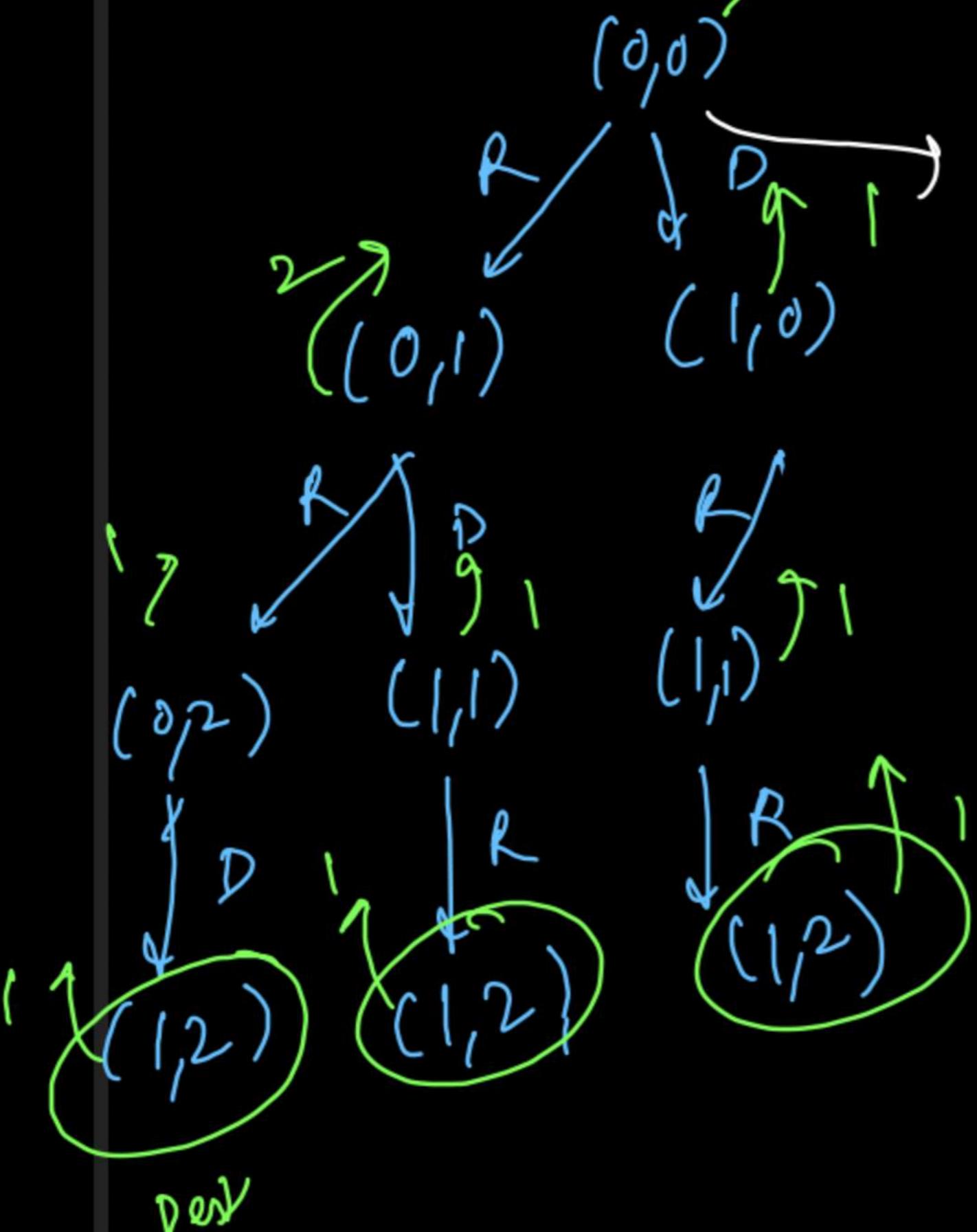
Count of ways from source to destination

↓
top left

↓
bottom right

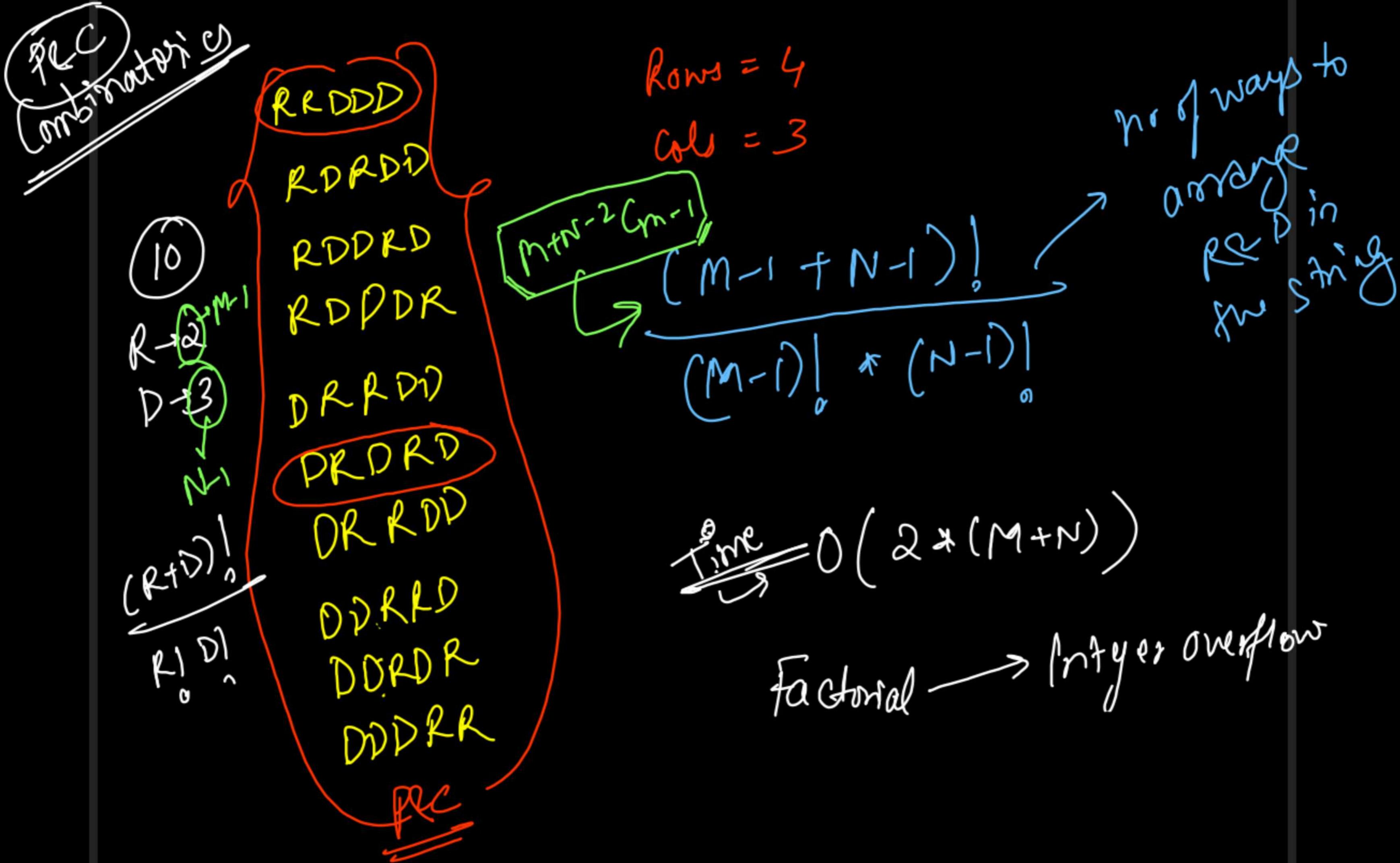
Possible moves → horizontally right
→ vertically down

WWS = 2, cols = 3



Time
Recursion $\rightarrow O(2^{n+m-2})$ expo

DP $\rightarrow O(N \cdot M)$ quadratic

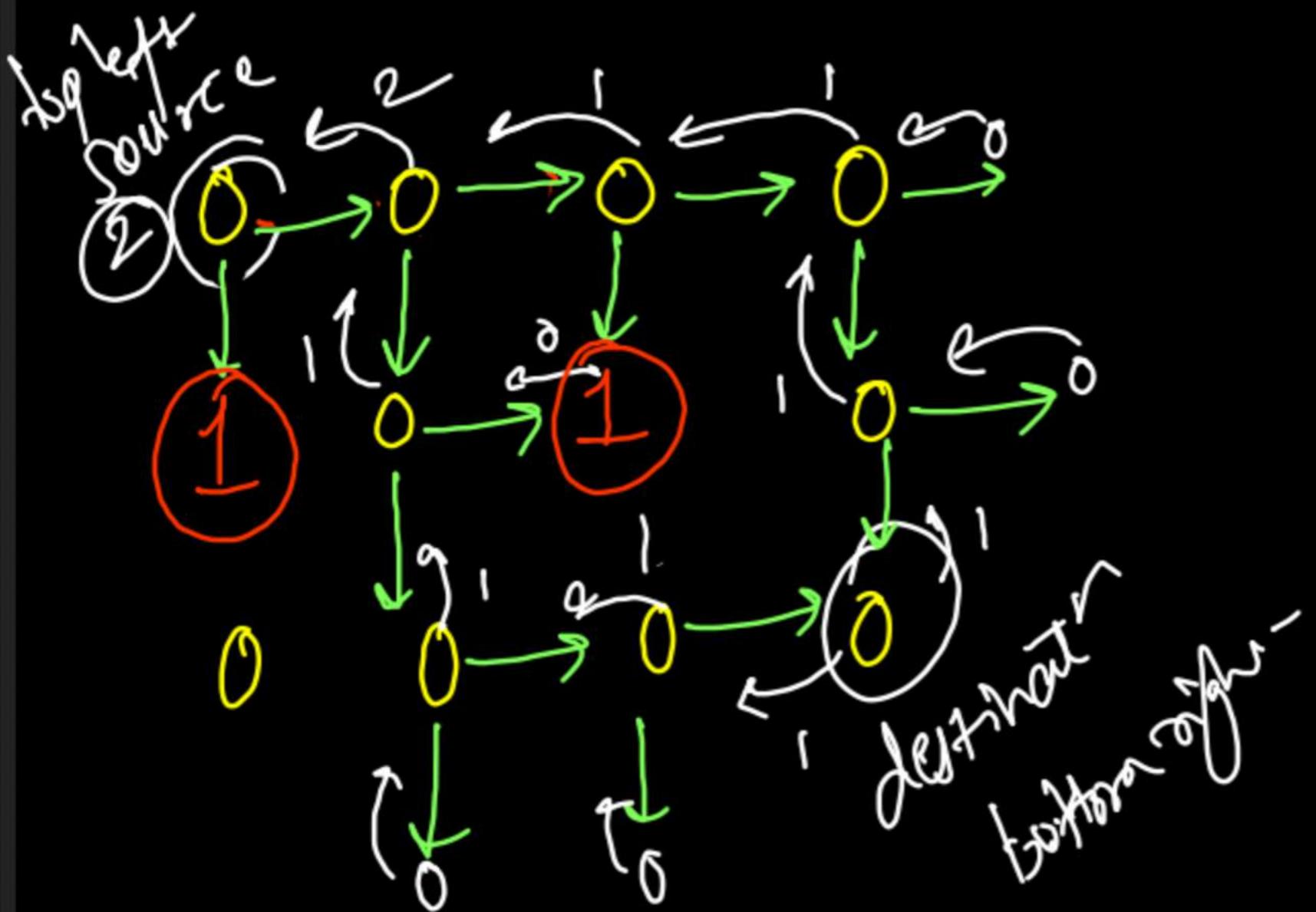


Unique Paths - I

```
public int memo(int sr, int sc, int dr, int dc, int[][] dp){  
    if(sr > dr || sc > dc) return 0;  
    if(sr == dr && sc == dc) return 1;  
    if(dp[sr][sc] != -1) return dp[sr][sc];  
  
    return dp[sr][sc] = (memo(sr + 1, sc, dr, dc, dp)  
                        + memo(sr, sc + 1, dr, dc, dp));  
}
```

```
public int uniquePaths(int m, int n) {  
    int[][] dp = new int[m + 1][n + 1];  
    for(int i=0; i<=m; i++)  
        for(int j=0; j<=n; j++)  
            dp[i][j] = -1;  
  
    return memo(0, 0, m - 1, n - 1, dp);  
}
```

Memoization
Time $\rightarrow O(N \times M)$
Space $\rightarrow O(N \times M)$



Unique Paths - 11

Possible ways

- ✓ horizontally
- ↓ vertically
- ↙ down

blockage node → dest
0 ways

```

public int memo(int sr, int sc, int dr, int dc, int[][][] grid, int[][] dp){
    if(sr > dr || sc > dc) return 0;
    if(grid[sr][sc] == 1) return 0;
    if(sr == dr && sc == dc) return 1;
    if(dp[sr][sc] != -1) return dp[sr][sc];

    return dp[sr][sc] = (memo(sr + 1, sc, dr, dc, grid, dp)
                         + memo(sr, sc + 1, dr, dc, grid, dp));
}

public int uniquePathsWithObstacles(int[][] grid) {
    // Source is blocked
    if(grid[0][0] == 1) return 0;

    int m = grid.length, n = grid[0].length;
    // Destination is blocked
    if(grid[m - 1][n - 1] == 1)
        return 0;

    int[][] dp = new int[m + 1][n + 1];
    for(int i=0; i<=m; i++)
        for(int j=0; j<=n; j++)
            dp[i][j] = -1;

    return memo(0, 0, m - 1, n - 1, grid, dp);
}

```

Time $\rightarrow O(M \times N)$

Space $\rightarrow O(M \times N)$
 $(O(DP))$

#GFG Reypad Problem

N = numbers of keys to be pressed

$$\underline{\underline{N=1}}$$

1, 2, 3, 4, 5, 6, 7, 8, 9, 0 \rightarrow 10 ways

$$\underline{\underline{N=2}}$$

\rightarrow 36 ways

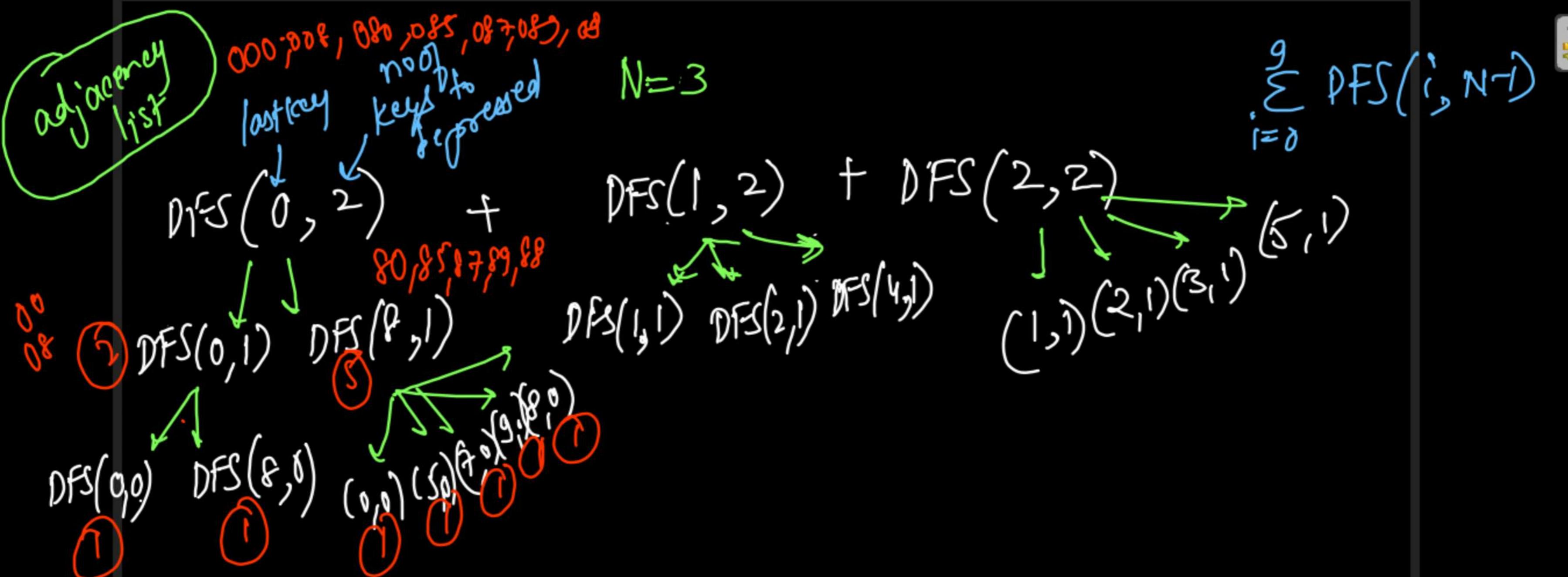
11, 12, 14, 22, 21, 23, 25,
33, 32, 36, 44, 41, 45, 47, 55, 52, 54,
56, 58, 66, 63, 65, 69, 77,
79, 78, 88, 85, 87, 89, 80
99, 90, 98, 00, 08

After pressing any key

Next move is \rightarrow pressing the same key again

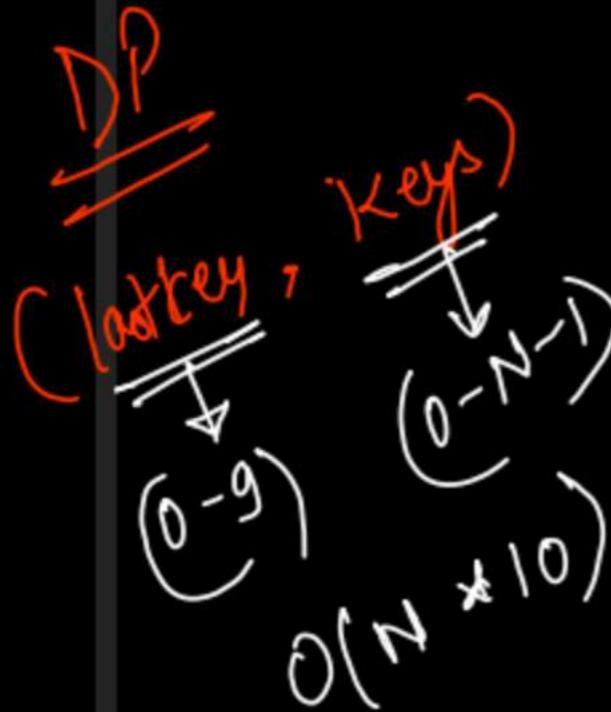
press any one of its 4 neighbours (T, L, D, R)





$\text{DFS}(3, 2) + \text{DFS}(4, 2) + \text{DFS}(5, 2) + \text{DFS}(6, 2) + \text{DFS}(7, 2) + \text{DFS}(8, 2) + \text{DFS}(9, 2)$

~~Recursion~~
Worst case $\rightarrow O(5^N)$



Space $\hookrightarrow O(N * 10)$ DP table

```
// For Each Key, store the neighbouring keys and the same key itself.  
int[][] adj = {{0, 8}, {1, 2, 4}, {1, 2, 3, 5}, {2, 3, 6}, {1, 4, 5, 7},  
               {2, 4, 5, 6, 8}, {3, 5, 6, 9}, {4, 7, 8}, {0, 5, 7, 8, 9}, {6, 8, 9}};  
  
public long DFS(int lastKey, int keys, long[][] dp){  
    if(keys == 0) return 1L;  
    if(dp[lastKey][keys] != -1L) return dp[lastKey][keys];  
  
    long ans = 0;  
    for(int nbr: adj[lastKey]){  
        ans = ans + DFS(nbr, keys - 1, dp);  
    }  
  
    return dp[lastKey][keys] = ans;  
}  
  
public long getCount(int N)  
{  
    long[][] dp = new long[10][N];  
    for(int i=0; i<10; i++){  
        for(int j=0; j<N; j++){  
            dp[i][j] = -1;  
        }  
    }  
  
    long ans = 0;  
    for(int i=0; i<10; i++){  
        ans = ans + DFS(i, N - 1, dp);  
    }  
  
    return ans;  
}
```

S_1, S_2 \rightarrow min diff

~~Tug of War~~ \rightarrow Equal size

1. You are given an array of n integers.
2. You have to divide these n integers into 2 subsets such that the difference of sum of two subsets is as minimum as possible.
3. If n is even, both set will contain exactly $n/2$ elements. If is odd, one set will contain $(n-1)/2$ and other set will contain $(n+1)/2$ elements.
3. If it is not possible to divide, then print "-1".

mutually exclusive
exhaustive
Equal size



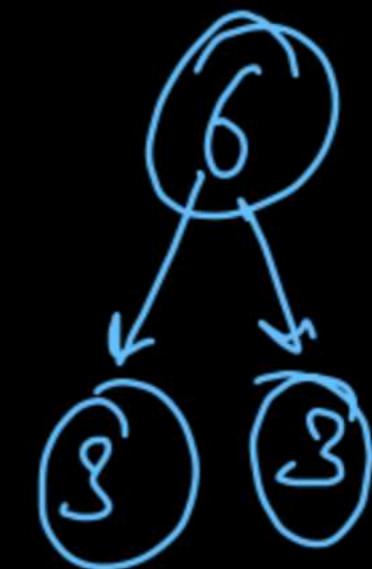
Target sum
subset

{1, 2, 3, 10, 16}

Diff size $S_1 - S_2 = \min \text{diff}$

{1, 2, 3, 10} {16}

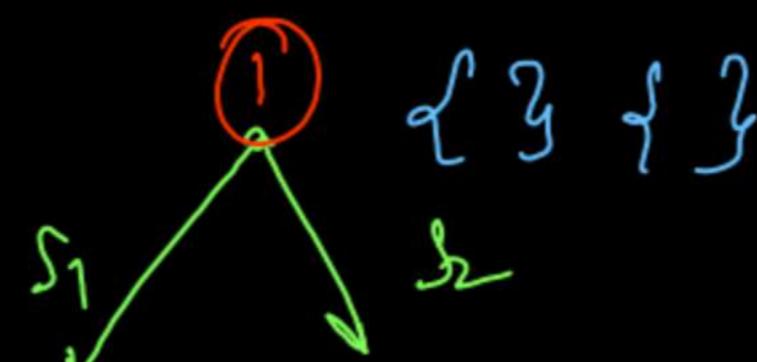
$$16 - 16 = 0$$



$\{1, 2, 3\} \rightarrow 10$

```

    1   2   3
  / \ / \ / \
  0   1   0   1   0
  
```



$\{1, 3\} \rightarrow 2$



$\{3\} \rightarrow 3$ (2)



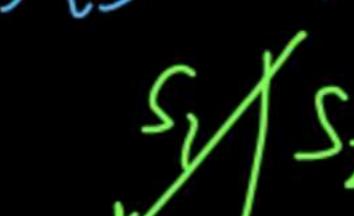
$\{1, 2\} \rightarrow 3$ (3)



$\{2\} \rightarrow 3$ (3)



10 $\{1, 2, 3\} \rightarrow \{\}$



$\{\}$
1, 2, 3, 10

α
 $\{1, 2, 3, 6\}$
 $\{3\}$

1, 2, 3
10

1, 2, 10
3
1, 2

1, 3, 10
2
1, 3

1, 10
2, 10
2, 10
1, 10

1
2, 3, 10
1, 3
1, 3

1, 2, 3
2, 10
2, 10
1, 3

3
1, 2, 10
1, 2

1, 2, 3
10

$\{\}$
1, 2, 3, 10

α
 $\{3\}$

1, 2, 3
10

1, 2, 10
3
1, 2

1, 3, 10
2
1, 3

1, 10
2, 10
2, 10
1, 10

1
2, 3, 10
1, 3
1, 3

1, 2, 3
2, 10
2, 10
1, 3

3
1, 2, 10
1, 2

1, 2, 3
10

$\{\}$
1, 2, 3, 10

α
 $\{3\}$

1, 2, 3
10

1, 2, 10
3
1, 2

1, 3, 10
2
1, 3

1, 10
2, 10
2, 10
1, 10

1
2, 3, 10
1, 3
1, 3

1, 2, 3
2, 10
2, 10
1, 3

3
1, 2, 10
1, 2

1, 2, 3
10

$\{\}$
1, 2, 3, 10

α

1, 2, 3
10

1, 2, 10
3
1, 2

1, 3, 10
2
1, 3

1, 10
2, 10
2, 10
1, 10

1
2, 3, 10
1, 3
1, 3

1, 2, 3
2, 10
2, 10
1, 3

3
1, 2, 10
1, 2

1, 2, 3
10

$\{\}$
1, 2, 3, 10

- $N \leq 20$ → Recursion → Exponential time
 - $N \leq 10^3$ → $O(N^3)$ ↗
Floyd Warshall
Bellman Ford
 - $N \leq 10^4$ → $O(N^2)$ ↗
Dijkstra
Prim's, Kruskal's
 - $N \leq 10^6$ → $O(N \log N)$ → Sorting (Merge/Quick)
 - $N \leq 10^8$ → $O(N)$ → DFS, BFS
- 3D DP
- 2D DP
- 1D DP

$O(2^N)$

Time Complexity

```
        helper(arr, 0, "", 0, 0, "", 0, 0);
        System.out.println("[" + s1Ans + "] [" + s2Ans + "]");
    }

    static int mindiff = Integer.MAX_VALUE;
    static String s1Ans = "", s2Ans = "";

    public static void helper(int[] arr, int idx, String s1, int s1Sum, int s1Len, String s2, int s2Sum, int s2Len){
        if(idx == arr.length){
            int diff = Math.abs(s1Sum - s2Sum);

            if(arr.length % 2 == 1 && (s1Len == s2Len + 1 || s2Len == s1Len + 1) && diff < mindiff){
                s1Ans = s1.substring(2); s2Ans = s2.substring(2);
                mindiff = diff; ↳ to remove the leading space & zero
            }
            else if(arr.length % 2 == 0 && s1Len == s2Len && diff < mindiff){
                s1Ans = s1.substring(2); s2Ans = s2.substring(2);
                mindiff = diff;
            }
            return;
        }

        // Element is inserted in S1
        helper(arr, idx + 1, s1 + ", " + arr[idx], s1Sum + arr[idx], s1Len + 1, s2, s2Sum, s2Len);

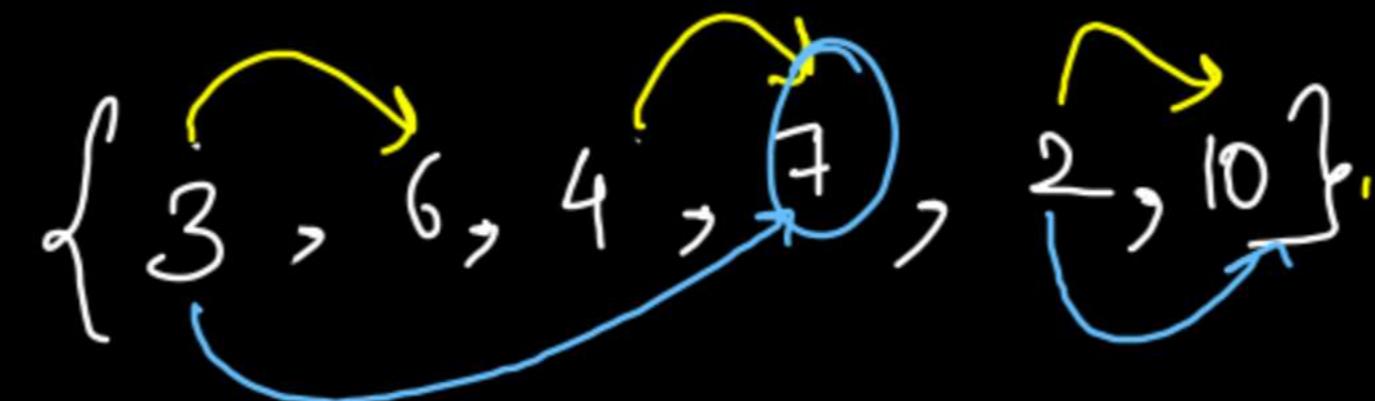
        // Element is inserted in S2
        helper(arr, idx + 1, s1, s1Sum, s1Len, s2 + ", " + arr[idx], s2Sum + arr[idx], s2Len + 1);
    }
}
```

① equal length
② min diff
③ max diff
ME & E

~~#
Buy & Sell Stocks
Leetcode~~

Buy & sell stocks \rightarrow K Transactions

↳ Leetcode 188



$$K = 3 \text{ transactions} \Rightarrow (6-3) + (7-4) + (10-2) = 3 + 3 + 8 = 14$$

$$K = 2 \text{ transactions} \Rightarrow (7-3) + (10-2) = 4 + 8 = 12$$

$$K = 0 \text{ transactions} \Rightarrow (10-2) = 8$$

$10^{-2} + 6$	$2^{-7} + 4 = -1$
$10^{-4} + 3$	$2^{-4} + 3 = 1$
$10^{-6} + 3$	$2^{-6} + 3 = -1$
$R=3$	$2 \times 3 + 0 \rightarrow$
<i>overlapping transaction cannot occur</i>	
<i>transactions</i>	
<i>overlap</i>	
<i>stocks</i>	
$dp[i][t] = \max_{j \in i} dp[j][t-1] + (\text{arr}[i] - \text{arr}[j])$	
$(3) \leftarrow (6) \leftarrow (4) \leftarrow (7) \leftarrow (2) \leftarrow (10)$	
0	0 0 0 0 0 0
1	$3-3=0$ $6-3=3$ $4-3 \text{ vs } 4-6$ $\text{vs } 6-3 = 3$ $7-4 \text{ vs } 7-6$ $\text{vs } 3-3 = 0$ $10-2=8$
2	0 $6-3$ $4-6+3 \text{ vs } 4-7$ $6-3$ 3 $(7-4)+(6-3)$ $= 6$ 6 12
3	0 3 3 6 6 $10-2+6$ -14
	$2-7+6$ $2-4+3$ $2-6+3$ $2-3+0$
	$2-4+3 \text{ vs } 7-6+3$ $\text{vs } 7-3+0$

$dp[t][i] = \text{Amount T transactions up to i-th day}$

```
class Solution {
    public int maxProfit(int k, int[] prices) {
        if(prices.length == 0 || k == 0) return 0;

        int[][] dp = new int[k + 1][prices.length];

        for(int t=1; t<=k; t++){
            for(int i=0; i<prices.length; i++){
                dp[t][i] = (i - 1 >= 0) ? dp[t][i - 1] : 0;

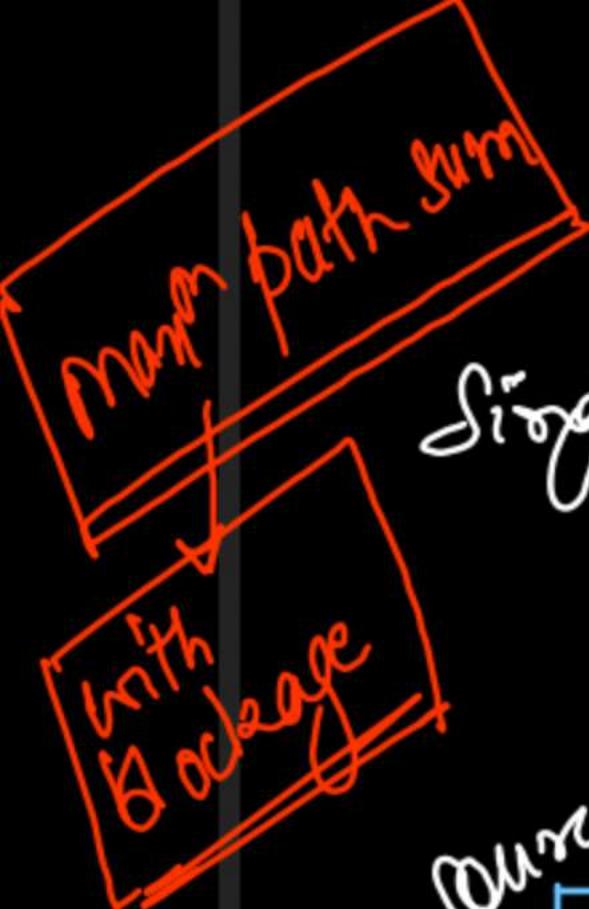
                for(int j=i-1; j>=0; j--){
                    // Last Transaction is between j and i, and remaining (t - 1)
                    // transactions are before the jth day (Non-overlapping).

                    dp[t][i] = Math.max(dp[t][i], (prices[i] - prices[j]) + dp[t - 1][j]);
                }
            }
        }

        return dp[k][prices.length - 1];
    }
}
```

Time $\rightarrow O(K * N^2)$

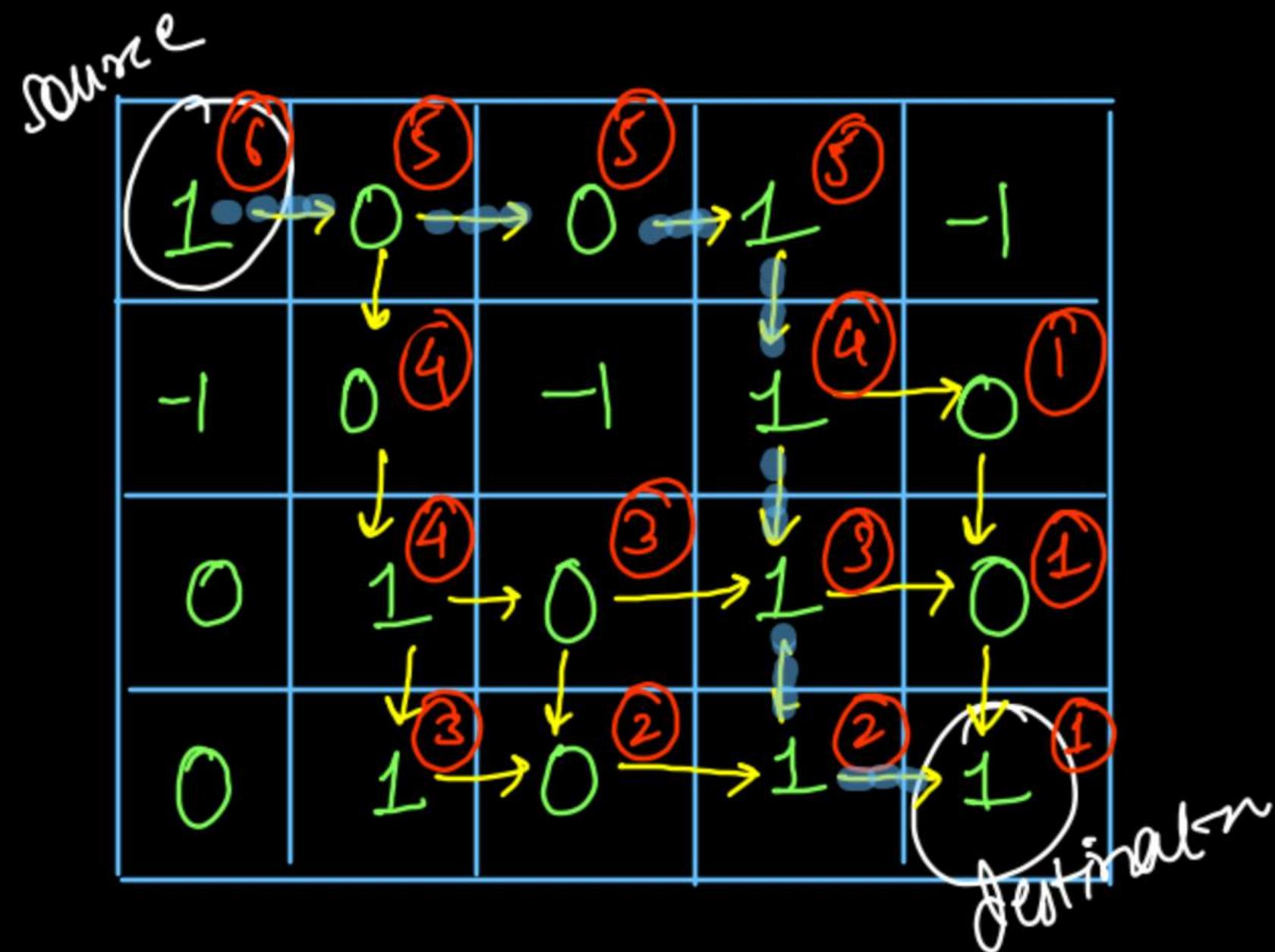
↳
max profit



DP on Grids

Single source \rightarrow Top left Single Dest \rightarrow Bottom Right

Possible move \rightarrow Right or Down



src
↓
intermediate
dp[int] -
definition

```

public int helper(int row, int col, int[][] grid, int[][] dp){
    if(row >= grid.length || col >= grid[0].length || grid[row][col] == -1){
        // Negative Base Case (Out of Matrix or Blockage Node)
        return 0;
    }

    if(row == grid.length - 1 && col == grid[0].length - 1){
        // Positive Base Case: Destination Cell |
        return grid[row][col];
    }

    if(dp[row][col] != -1) return dp[row][col];

    int right = helper(row, col + 1, grid, dp);
    int down = helper(row + 1, col, grid, dp);

    return dp[row][col] = Math.max(right, down) + grid[row][col];
}

int[][] dp = new int[grid.length][grid[0].length];
for(int i=0; i<dp.length; i++){
    for(int j=0; j<dp[0].length; j++){
        dp[i][j] = -1;
    }
}

return helper(0, 0, grid, dp);

```

Time $\rightarrow O(N \times M)$

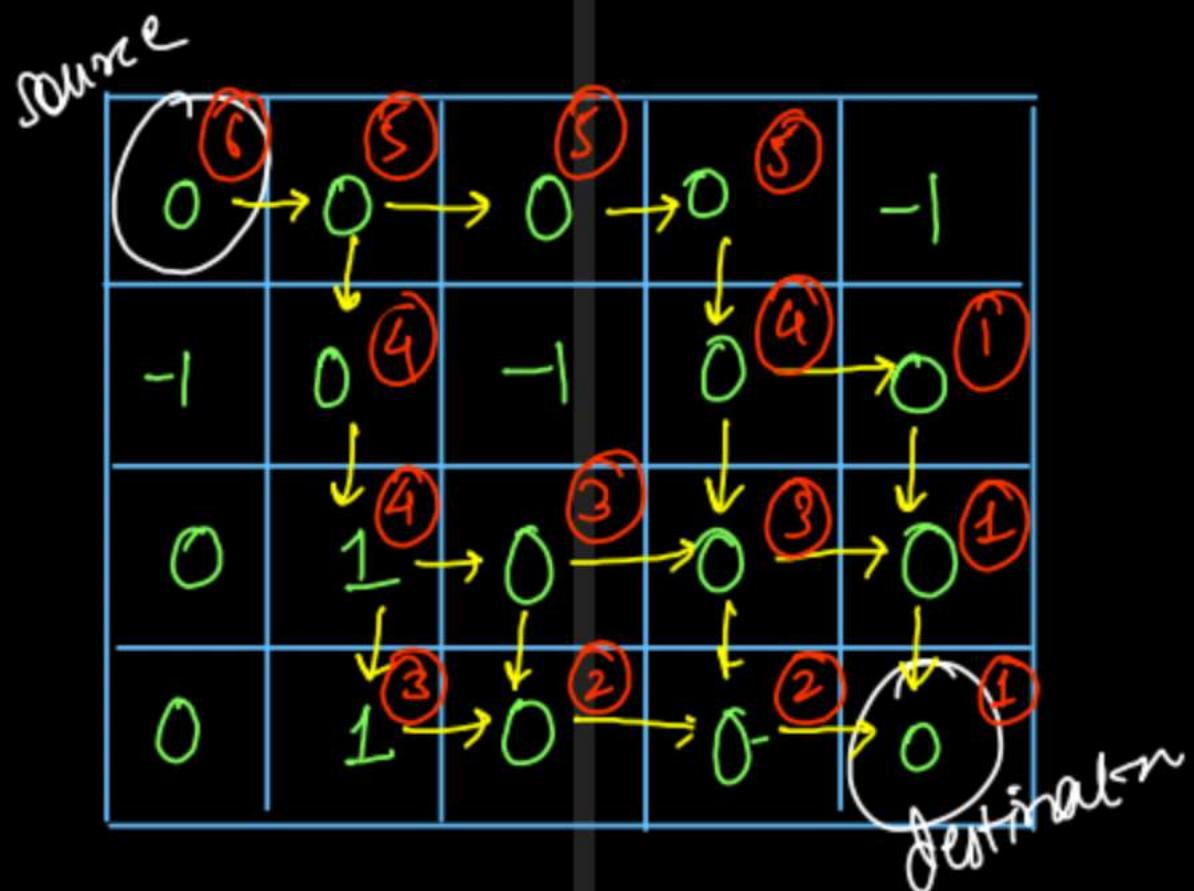
Space $\rightarrow O(N \times M)$

Cherry Pickup - I

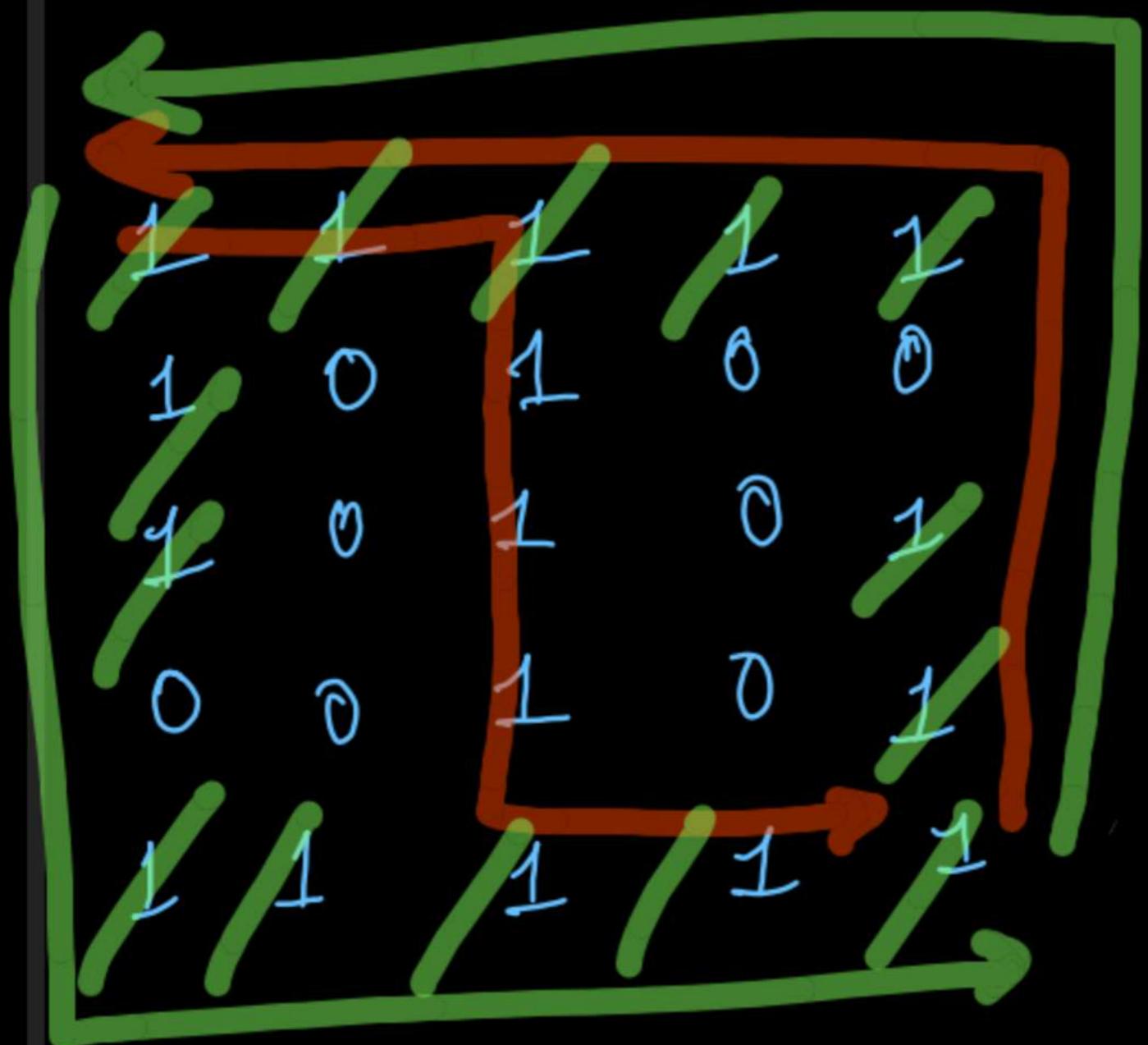
Leetcode 741

Ist Iteration → Source → Dest
Top Left Bottom Right

IInd Iteration → Dest → Source
Bottom right Top left



Both iterations are not
independent, rather they are inter-related



your answer

$TL \rightarrow BR$

⑨

$TL \rightarrow BR$

④

$= ⑬$

expected output

$TL \rightarrow BR$

$TL \rightarrow BR$

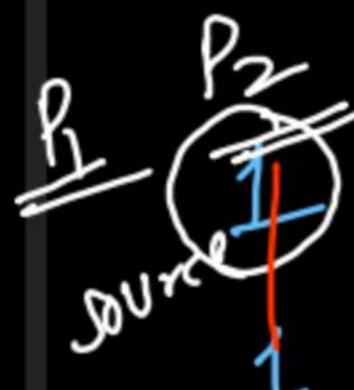
⑧

$TL \rightarrow BR$

⑥

~~P₁ & P₂~~
destination

Single source, single destination
two simultaneous paths



	1	1	1	1
1	0	1	0	0
1	-1	1	-1	1
0	0	1	0	1
1	1	1	1	1



(0,0), (0,0)

grnd[i][j]

①
det

(0,1)
(0,1)

(0,0), (0,0)

(0,1)
(1,0)

(1,0)
(0,1)

grnd[i][j]

(1,0)
(1,0)

grnd[i][j]

(1,1)
(1,1)

grnd[i][j]

P₁
P₂

P₁

P₁

P₁

P₂

P₂

P₂

②

③

④

~~Negative base case~~ $P_1 \rightarrow$ Out of matrix $P_2 \rightarrow$ Out of matrix
 $P_1 \rightarrow$ blockage $P_2 \rightarrow$ blockage

~~Positive base case~~

$P_1 = P_2 =$ destination

$P_1 \& P_2 \rightarrow$ starting from same source
calls (steps distance)

\Rightarrow Total dist by $P_1 =$ Total Dist by P_2
 $i_1 + j_1 = i_2 + j_2$

```

public int helper(int row1, int col1, int row2, int col2, int[][] grid, int[][][] dp){
    // int col2 = row1 + col1 - row2; Col2 is not a variable

    if(row1 >= grid.length || col1 >= grid[0].length || grid[row1][col1] == -1
    || row2 >= grid.length || col2 >= grid[0].length || grid[row2][col2] == -1){
        // Negative Base Case (Out of Matrix or Blockage Node)
        return Integer.MIN_VALUE;
    }

    if(row1 == grid.length - 1 && col1 == grid[0].length - 1){
        // Positive Base Case: Destination Cell
        return grid[row1][col1];
    }

    if(dp[row1][col1][row2] != -1) return dp[row1][col1][row2];

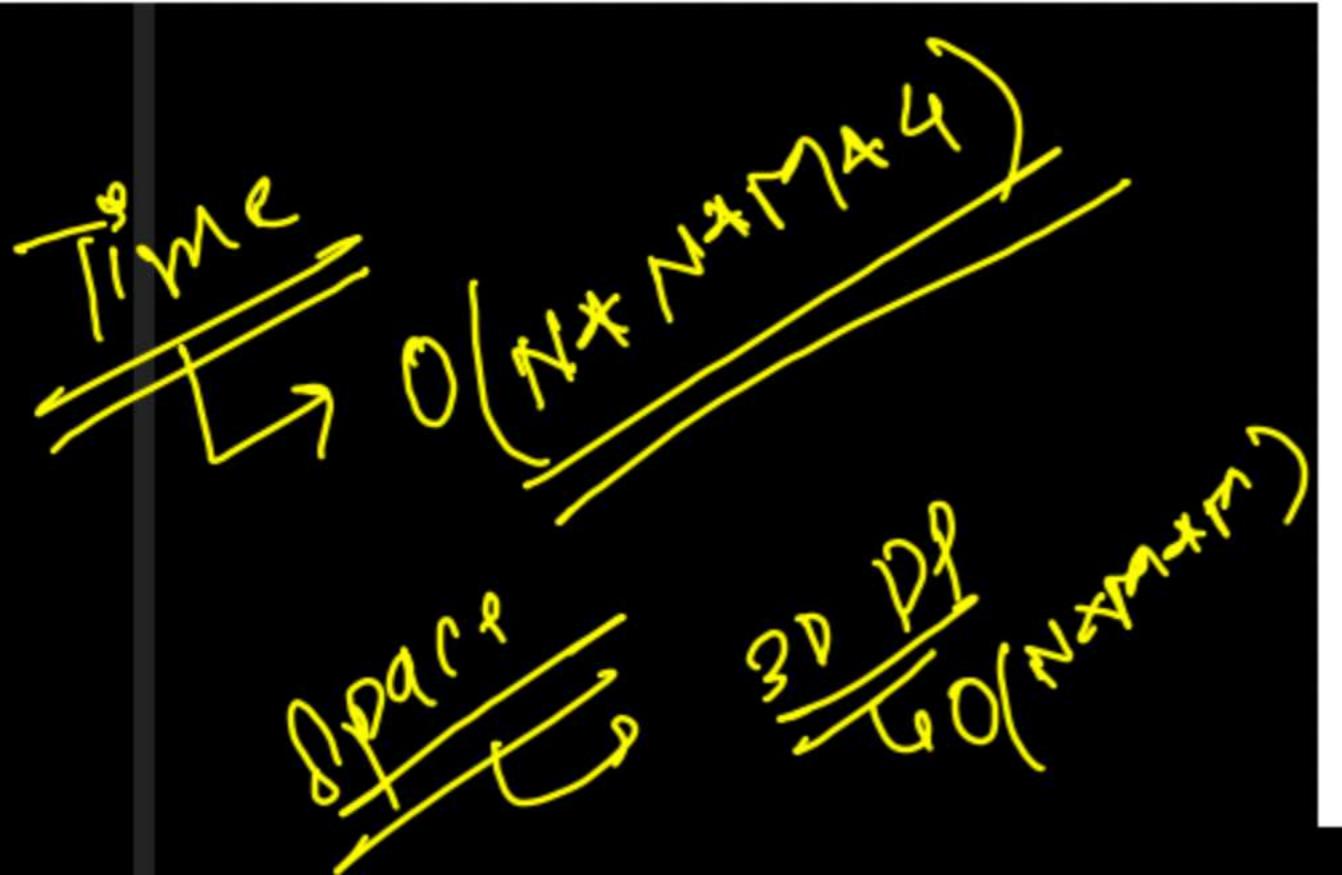
    int ans = grid[row1][col1] + grid[row2][col2];
    if(row1 == row2 && col1 == col2) ans -= grid[row1][col1];

    int RR = helper(row1, col1 + 1, row2, col2 + 1, grid, dp);
    int RD = helper(row1, col1 + 1, row2 + 1, col2, grid, dp);
    int DR = helper(row1 + 1, col1, row2, col2 + 1, grid, dp);
    int DD = helper(row1 + 1, col1, row2 + 1, col2, grid, dp);

    return dp[row1][col1][row2] = Math.max(Math.max(RR, RD), Math.max(DR, DD)) + ans;
}

```

- ① Independent
- ② Same - Simult
- ③ 4D → 3D



```

public int cherryPickup(int[][] grid) {
    int[][][] dp = new int[grid.length][grid[0].length][grid[0].length];
    for(int i=0; i<dp.length; i++){
        for(int j=0; j<dp[0].length; j++){
            for(int k=0; k<dp[0][0].length; k++){
                dp[i][j][k] = -1;
            }
        }
    }

    int ans = helper(0, 0, 0, 0, grid, dp);
    if(ans <= 0) return 0;
    return ans;
}

```

$$\text{LIS}(0, -1)$$

longest increasing \hookrightarrow strictly increasing Subsequence \hookrightarrow subset $\rightarrow 2^N$ total subsets

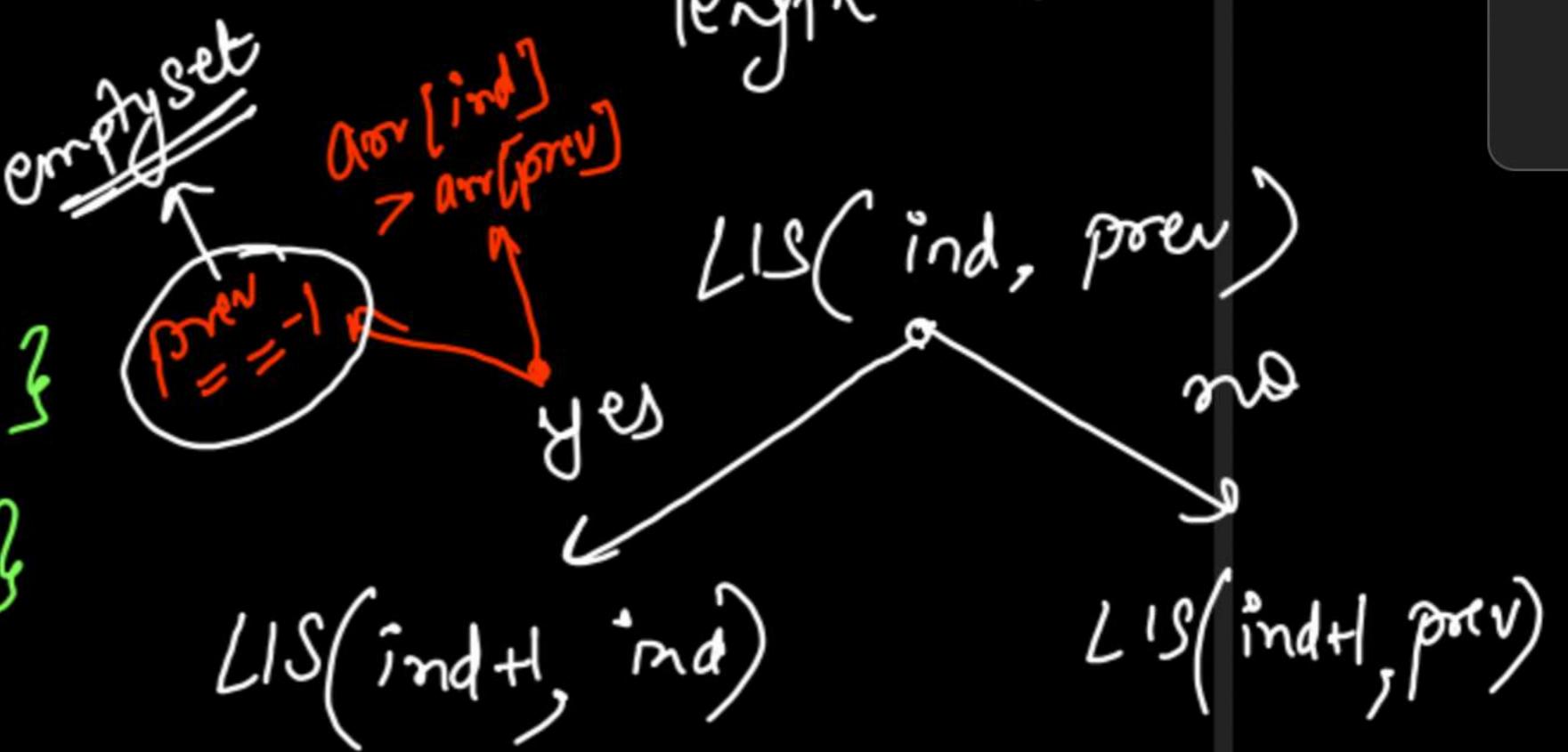
$$\{ 4, \textcircled{2}, 6, \textcircled{3}, \textcircled{5}, 7, 1, \textcircled{6}, \textcircled{8}, \textcircled{9}, 2 \}$$

The diagram illustrates a recursive partitioning of the set $\{1, 2, 3, 4, 5, 6\}$ into two disjoint subsets. The root node is labeled '#inden' with a crossed-out '#Dnen'. The tree structure is as follows:

- Level 1:** The root node branches into sets of size 1 and 5, and sets of size 2 and 4.
- Level 2:**
 - Set of size 1: branches into sets $\{1\}$ and $\{2, 3, 4, 5, 6\}$.
 - Set of size 5: branches into sets $\{1, 2, 3, 4, 5\}$ and $\{6\}$.
 - Set of size 2: branches into sets $\{2\}$ and $\{1, 3, 4, 5, 6\}$.
 - Set of size 4: branches into sets $\{2, 3\}$ and $\{1, 4, 5, 6\}$.
- Level 3:**
 - From $\{1, 2, 3, 4, 5\}$: branches into $\{1, 2\}$ and $\{3, 4, 5, 6\}$.
 - From $\{6\}$: branches into $\{6\}$ and $\{\}$.
 - From $\{2\}$: branches into $\{2\}$ and $\{\}$.
 - From $\{1, 3, 4, 5, 6\}$: branches into $\{1, 3\}$, $\{4, 5\}$, and $\{6\}$.
 - From $\{2, 3\}$: branches into $\{2, 3\}$ and $\{\}$.
 - From $\{1, 4, 5, 6\}$: branches into $\{1, 4\}$, $\{5, 6\}$, and $\{\}$.
- Level 4:**
 - From $\{1, 2\}$: branches into $\{1, 2\}$ and $\{\}$.
 - From $\{3, 4, 5, 6\}$: branches into $\{3, 4, 5\}$ and $\{6\}$.
 - From $\{2\}$: branches into $\{2\}$ and $\{\}$.
 - From $\{1, 3\}$: branches into $\{1, 3\}$ and $\{\}$.
 - From $\{4, 5\}$: branches into $\{4, 5\}$ and $\{\}$.
 - From $\{6\}$: branches into $\{6\}$ and $\{\}$.
 - From $\{1, 3\}$: branches into $\{1, 3\}$ and $\{\}$.
 - From $\{4, 5\}$: branches into $\{4, 5\}$ and $\{\}$.
 - From $\{6\}$: branches into $\{6\}$ and $\{\}$.
- Level 5:**
 - From $\{3, 4, 5\}$: branches into $\{3, 4\}$ and $\{5\}$.
 - From $\{1, 3\}$: branches into $\{1\}$ and $\{3\}$.
 - From $\{4, 5\}$: branches into $\{4\}$ and $\{5\}$.
 - From $\{6\}$: branches into $\{6\}$ and $\{\}$.

Red 'X' marks are placed under several nodes to indicate invalid partitions, such as overlapping sets or sets containing empty sets.

longest LIS: $\{2, 3, 5, 6, 8, 9\}$
length = 6



```

class Solution {
    public int memo(int curr, int prev, int[] nums, int[][] dp){
        if(curr == nums.length) return 0;
        if(dp[curr][prev + 1] != -1) return dp[curr][prev + 1];
        int yes = (prev == -1 || nums[prev] < nums[curr])
                  ? memo(curr + 1, curr, nums, dp) + 1 : 0;
        int no = memo(curr + 1, prev, nums, dp);
        return dp[curr][prev + 1] = Math.max(yes, no);
    }

    public int lengthOfLIS(int[] nums) {
        int n = nums.length;
        int[][] dp = new int[n + 1][n + 1];
        for(int i=0; i<=n; i++){
            for(int j=0; j<=n; j++){
                dp[i][j] = -1;
            }
        }
        return memo(0, -1, nums, dp);
    }
}

```

to
base
case: -1
also in
Xe
DP

Memoization

curr prev
 \downarrow \downarrow
 Time $\rightarrow O(N \times N)$

current element
 is included
 \Rightarrow length increased by 1

Space $\rightarrow O(N \times N)$

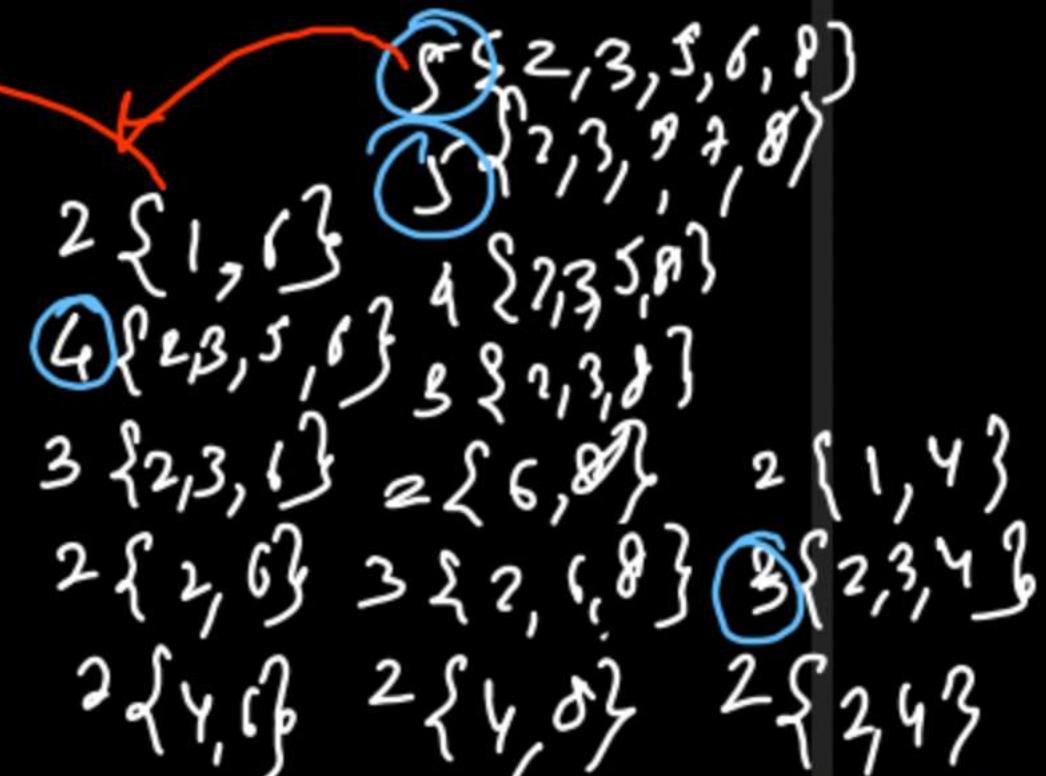
2D DP

$\{1, 2, 3, 4\}$

Overlapping subproblems

 $\{1\}$ $\{1, 2\}$ $\{3\}$ $\{1, 2\}$ $\{2\}$ $\{1\}$ $\{3\}$ $\{1, 2\}$ 3^{rd} 3^{rd} $\{1, 2\}$ 3^{rd} 3^{rd}

Tabulation



1 $\{4\}$	1 $\{2\}$	1 $\{6\}$	1 $\{3\}$	1 $\{5\}$	1 $\{7\}$	1 $\{1\}$	1 $\{6\}$	1 $\{8\}$	1 $\{4\}$
--------------	--------------	--------------	--------------	--------------	--------------	--------------	--------------	--------------	--------------

4 ② 6 ③ ⑤ 7 1 ⑥, ⑧ 9

$dp[i] = \text{longest increasing subsequence length}$
Ending at index i

```
public int lengthOfLIS(int[] nums) {  
    int n = nums.length;  
    int[] dp = new int[n];  
    int maxLIS = 0;  
    for(int i=0; i<nums.length; i++){  
        dp[i] = 1; // If Prev Does not Exist, then current element can have yes  
  
        for(int j=0; j<i; j++){  
            if(nums[j] < nums[i]){  
                dp[i] = Math.max(dp[i], dp[j] + 1);  
            }  
        }  
  
        maxLIS = Math.max(maxLIS, dp[i]);  
    }  
  
    return maxLIS;  
}
```

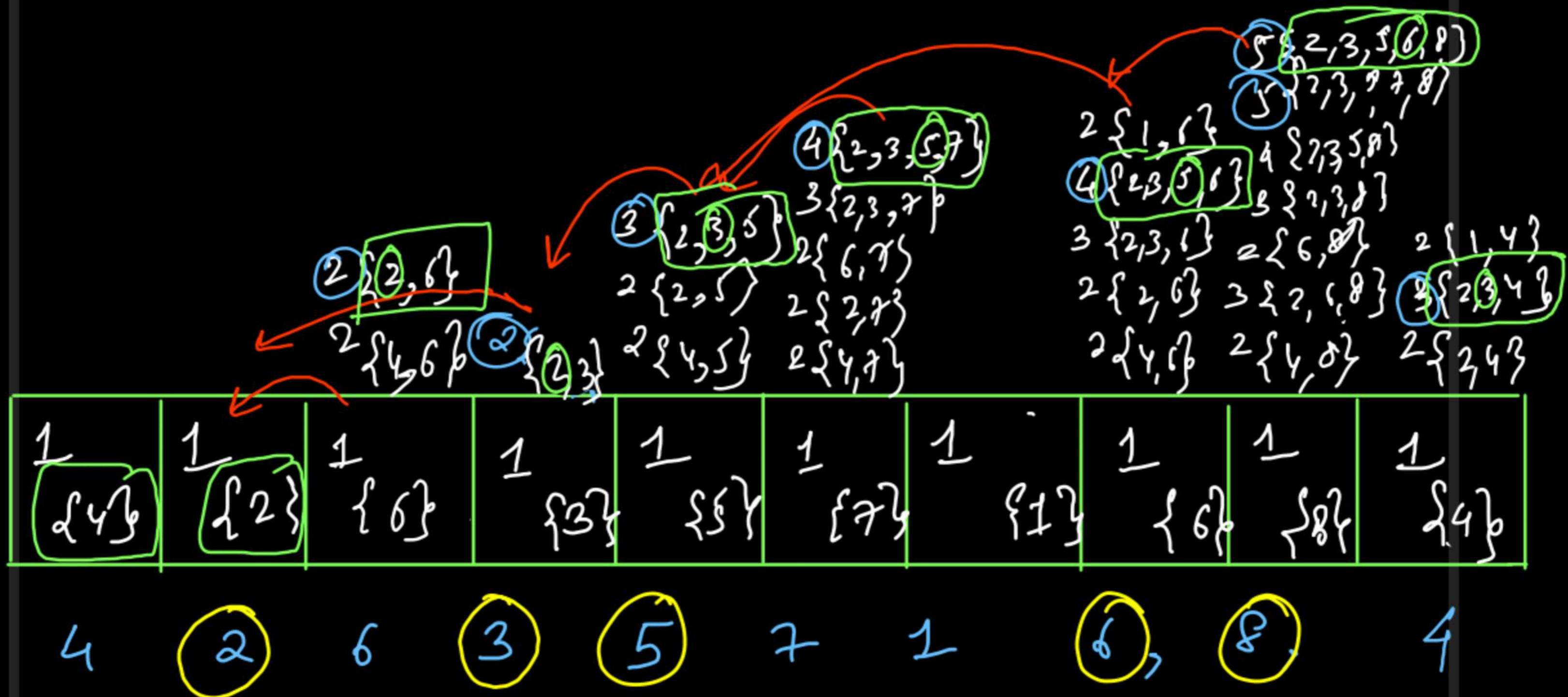
Tabulation

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

Space $\rightarrow O(N)$

[1D DP]

Pointing LIS → Any one → ArrayList LIS stored at each index
 Point All → Backtracking



Point Any one LIS

```
public static void solution(int[] nums){  
    int n = nums.length;  
    ArrayList<Integer>[] dp = new ArrayList[n];  
  
    int maxLIS = 0, maxLISind = 0;  
  
    for(int i=0; i<nums.length; i++){  
        dp[i] = new ArrayList<>();  
        dp[i].add(nums[i]);  
  
        for(int j=0; j<i; j++){  
            if(nums[j] < nums[i]){  
                if(dp[j].size() + 1 > dp[i].size()){  
                    dp[i] = new ArrayList<>(dp[j]);  
                    dp[i].add(nums[i]);  
                }  
            }  
        }  
  
        if(dp[i].size() > maxLIS){  
            maxLIS = dp[i].size();  
            maxLISind = i;  
        }  
    }  
  
    for(int val: dp[maxLISind]){  
        System.out.print(val + " ");  
    }  
}
```

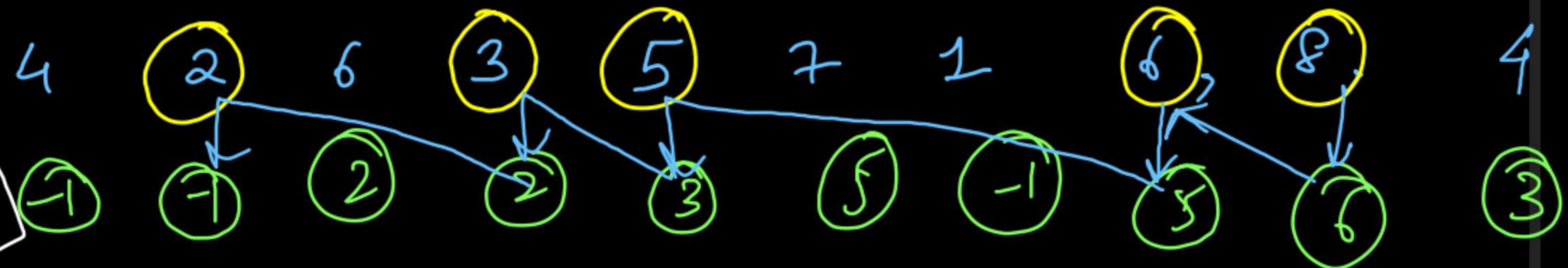
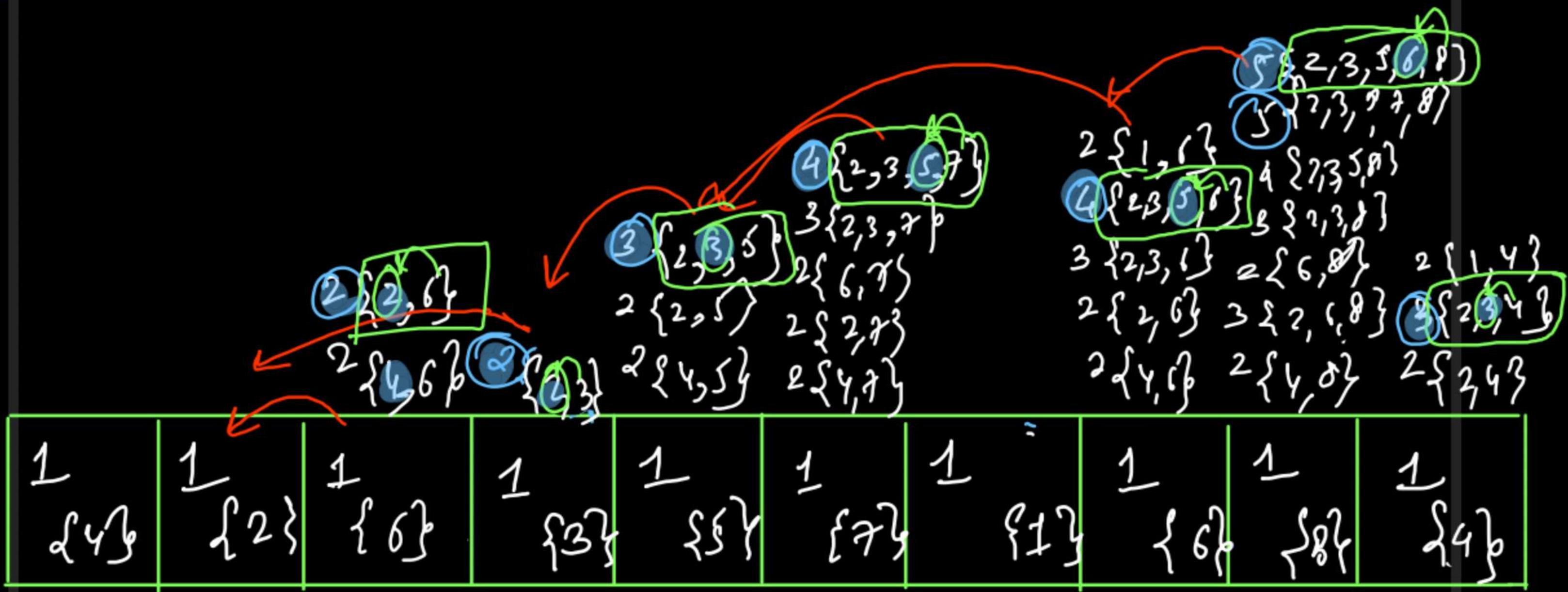
✓ Time
 $O(N \times N \times N)$
 $= O(N^3)$

✓ Space
 $O(N \times N)$
Storing AL
at each index

* deep copy $\rightarrow O(N)$

temping prev state

$\{2, 3, 5, 6, 7, 8\}$



```

public static void solution(int[] nums){
    int n = nums.length;
    int[] dp = new int[n];
    int[] prev = new int[n];

    int maxLIS = 0, lisidx = 0;

    for(int i=0; i<nums.length; i++){
        dp[i] = 1; // Length
        prev[i] = -1; // Empty Subset -> Current Element

        for(int j=0; j<i; j++){
            if(nums[j] < nums[i]){
                if(dp[j] + 1 > dp[i]){
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
                }
            }
        }

        if(dp[i] > maxLIS){
            maxLIS = dp[i];
            lisidx = i;
        }
    }

    ArrayList<Integer> LIS = new ArrayList<>();

    while(lisidx != -1){
        LIS.add(nums[lisidx]);
        lisidx = prev[lisidx];
    }

    Collections.reverse(LIS);
    System.out.println(LIS);
}

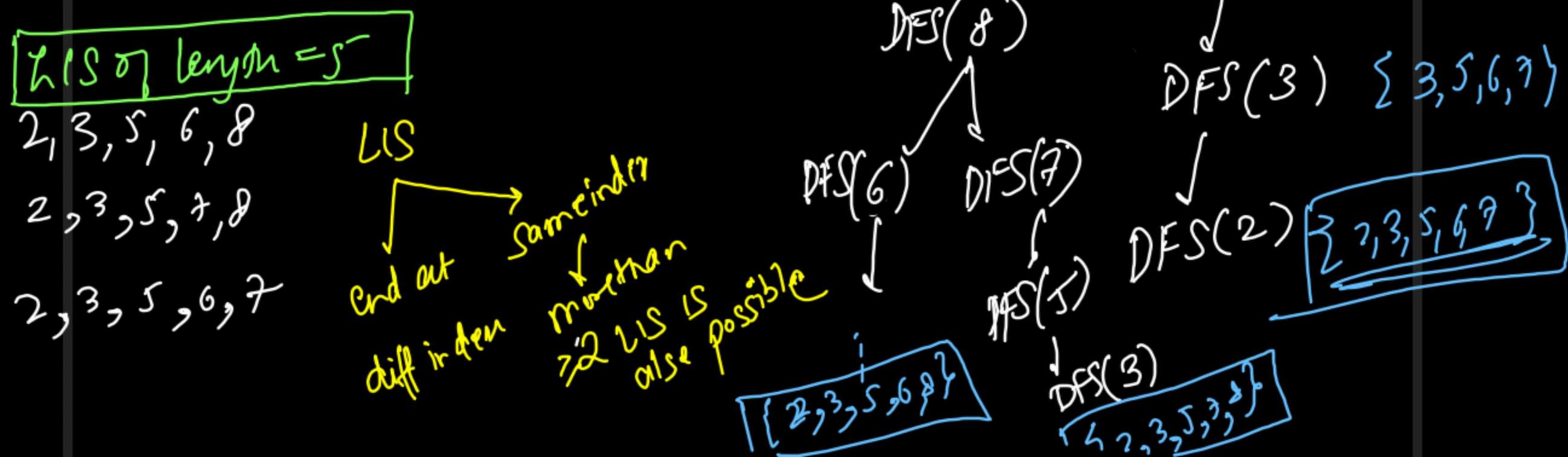
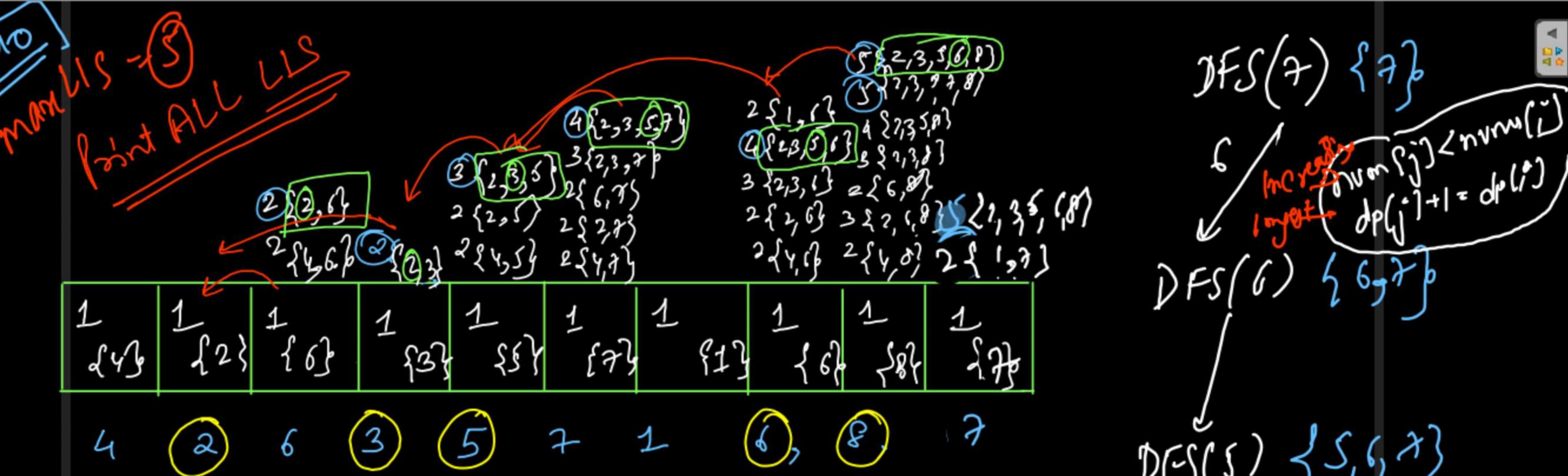
```

$$\text{Time} \geq O(N^2 + N) \\ \approx O(N^2)$$

$$\text{Space} \rightarrow O(2^N) \\ + O(N) \\ = O(N)$$

} Backtracking $\rightarrow O(LIS \text{ length})$

$\approx O(N)$ in
worst case



```
public static void DFS(int curr, int[] nums, int[] dp, String psf)
    if (dp[curr] == 1) {
        System.out.println(psf);
        return;
    }
    for (int prev = curr - 1; prev >= 0; prev--) {
        if (nums[prev] < nums[curr] && dp[curr] == dp[prev] + 1) {
            DFS(prev, nums, dp, nums[prev] + " -> " + psf);
        }
    }
}
```

Increasing

longest

~~Time~~
~~# Worst case~~ $\hookrightarrow O(\text{exponential})$

Any case $\rightarrow O(\text{polynomial})$

```
public static void solution(int[] nums) {  
    int n = nums.length;  
    int[] dp = new int[n];  
  
    int maxLIS = 0;  
  
    for (int i = 0; i < nums.length; i++) {  
        dp[i] = 1; // Length  
  
        for (int j = 0; j < i; j++) {  
            if (nums[j] < nums[i]) {  
                dp[i] = Math.max(dp[i], dp[j] + 1)  
            }  
        }  
  
        if (dp[i] > maxLIS) {  
            maxLIS = dp[i];  
        }  
    }  
  
    System.out.println(maxLIS);
```

```
for (int i = n - 1; i >= 0; i--) {
    // Start DFS from each node at which Increasing Subset is of
    // Longest Length

    if (dp[i] == maxLIS) {
        DFS(i, nums, dp, "" + nums[i]);
    }
}
```

*{ multiple sources
LIS can
diff:*

1); } multiple Sources
↳ US can end at diff indicat

~~LIS~~ → LIS → Time optimization → $O(N \log N)$
(Binary Search)

→ Count
LIS

Dynamic Programming

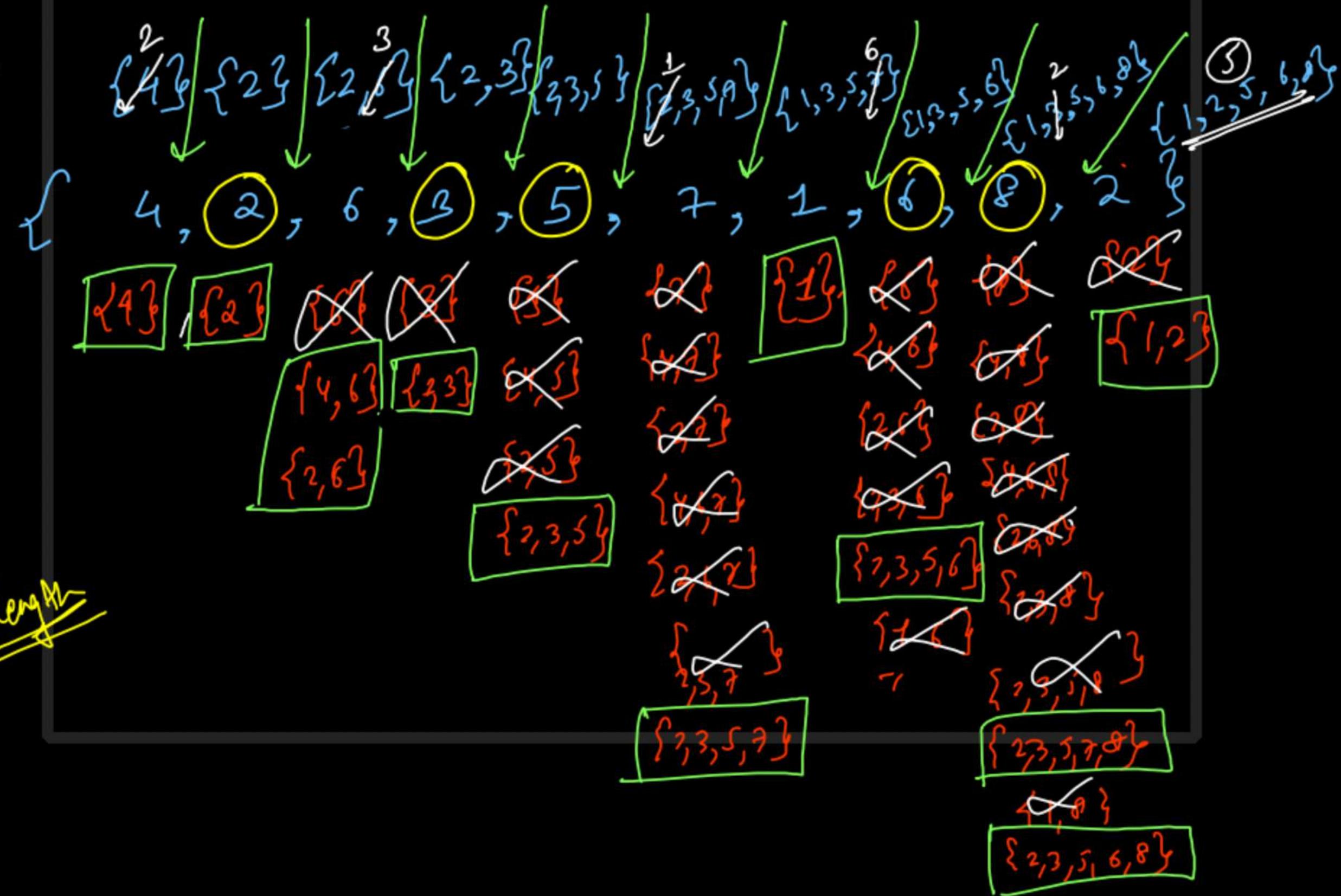
Lecture 22

→ Length of LIS $\rightarrow O(N \log N)$ optimized BS approach

→ Count of LIS

Longest Increasing Subsequence
Variations

longest increasing subsequence \rightarrow BS Approach



```

public int lowerBound(ArrayList<Integer> nums, int target){
    int low = 0, high = nums.size() - 1;
    int idx = nums.size();

    while(low <= high){
        int mid = low + (high - low) / 2;

        if(nums.get(mid) < target){
            low = mid + 1;
        } else {
            high = mid - 1;
            idx = mid;
        }
    }

    return idx;
}

```

Lower Bound $\rightarrow O(1 \log_2 N)$

```

public int lengthOfLIS(int[] nums) {
    int n = nums.length;

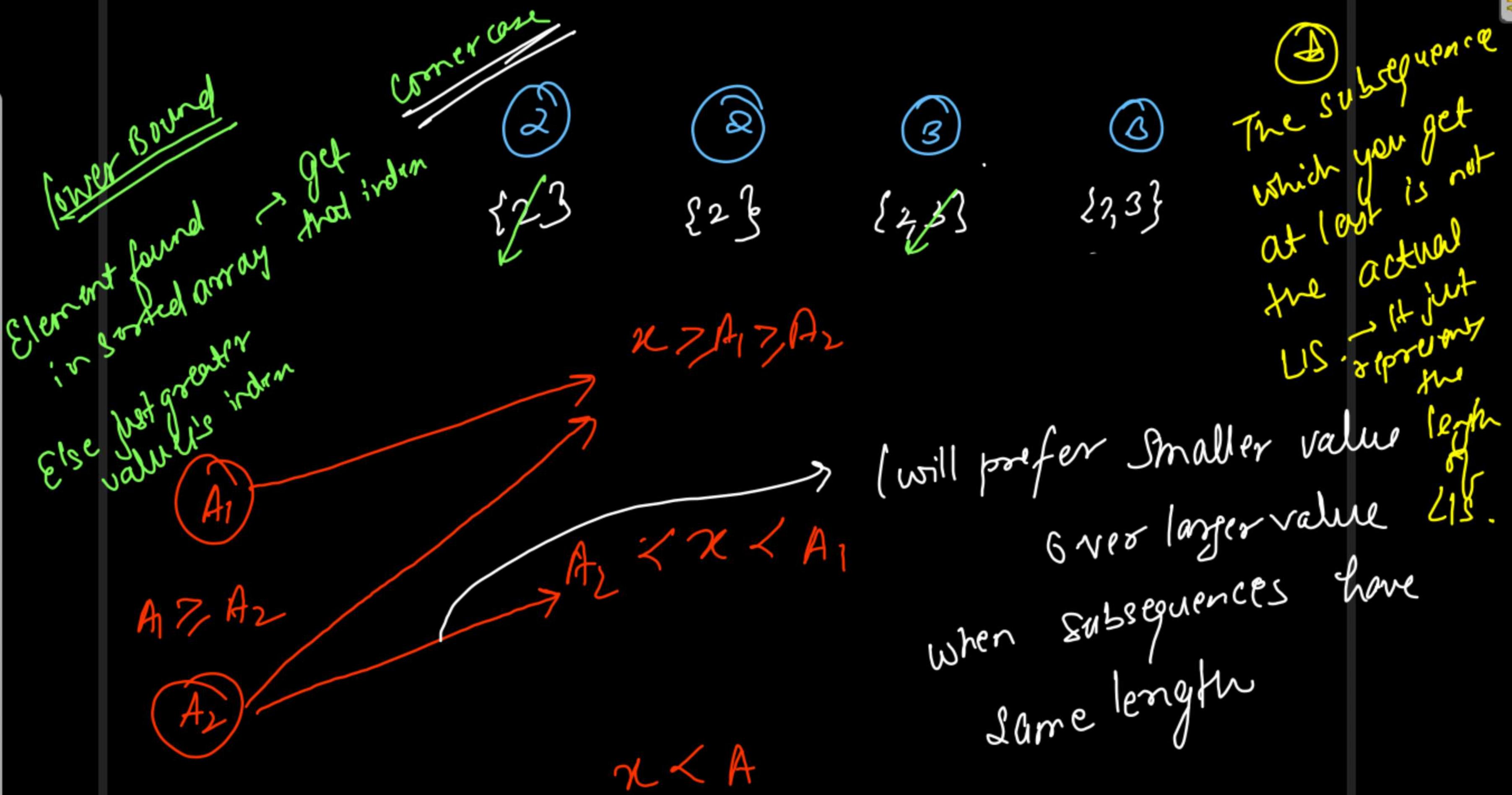
    ArrayList<Integer> sorted = new ArrayList<>();

    for(int i=0; i<nums.length; i++){
        int lb = lowerBound(sorted, nums[i]);
        if(lb == sorted.size()){
            sorted.add(nums[i]);
            // Current Element larger than the largest
            // LIS of one length more
        } else {
            sorted.set(lb, nums[i]);
        }
    }

    return sorted.size(); // This Sorted Array has same size as LIS
}

```

$O(N * \log_2 N)$
 Binary search
 Based
 Time optimization!



→ There can be
 more than
 1 LIS ending
 at given index
 or
 → There can be
 more than
 such indices

LC 673

Count
(DP)

Count LIS

$$\text{Count}[i] = \sum_j \text{count}[j]$$

where $\text{dp}(i) = \text{dp}(j) + 1$

LL
min(dp[i]) increasing
 $\text{dp}[i] > \text{min(dp)}$
 $\text{dp}[i] + 2$

$$5+5 = 10 \text{ LIS}$$

$$\{2, 3, 4, 7, 8\}$$

$$\{1, 3, 5, 7, 8\}$$

$$\{2, 3, 6, 7, 8\}$$

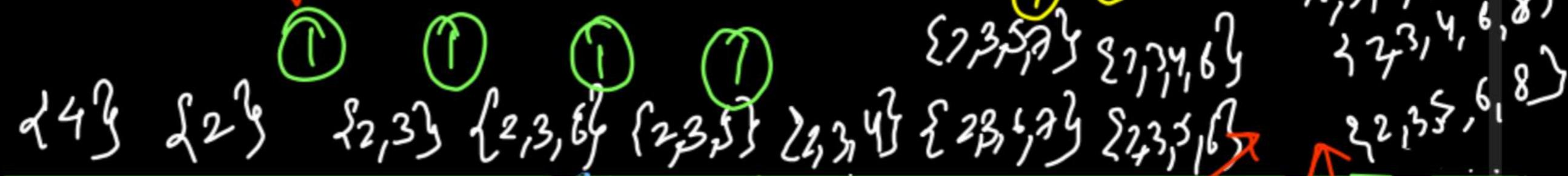
$$\{2, 3, 4, 6, 8\}$$

$$\{2, 3, 4, 6, 8\}$$

$$\{2, 3, 5, 6, 8\}$$

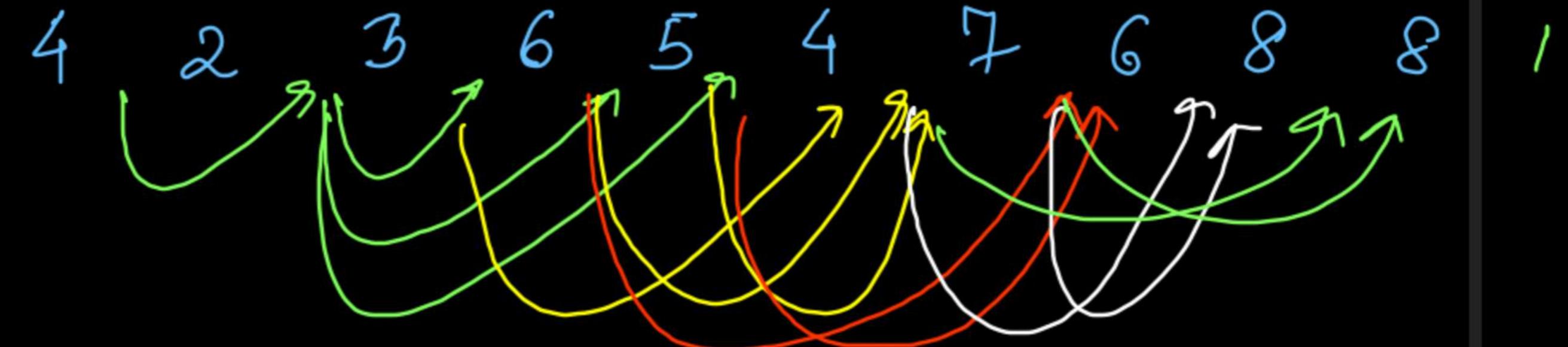
$$\{2, 3, 4, 5, 6\}$$

$$\{2, 3, 5, 6, 7\}$$



length
(DP)

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



```

int n = nums.length;
int[] dp = new int[n]; // Length of LIS ending at index i
Arrays.fill(dp, 1);

int[] count = new int[n]; // Count of LIS ending at index i
Arrays.fill(count, 1);

int maxLIS = 0;

for(int i=0; i<n; i++){
    for(int j=0; j<i; j++){
        if(nums[j] < nums[i] && dp[i] <= dp[j] + 1){
            if(dp[i] < dp[j] + 1)
                count[i] = 0;
            dp[i] = Math.max(dp[i], dp[j] + 1);
            count[i] += count[j];
        }
    }
    maxLIS = Math.max(maxLIS, dp[i]);
}

int countLIS = 0;
for(int i=0; i<n; i++){
    if(dp[i] == maxLIS)
        countLIS += count[i];
}

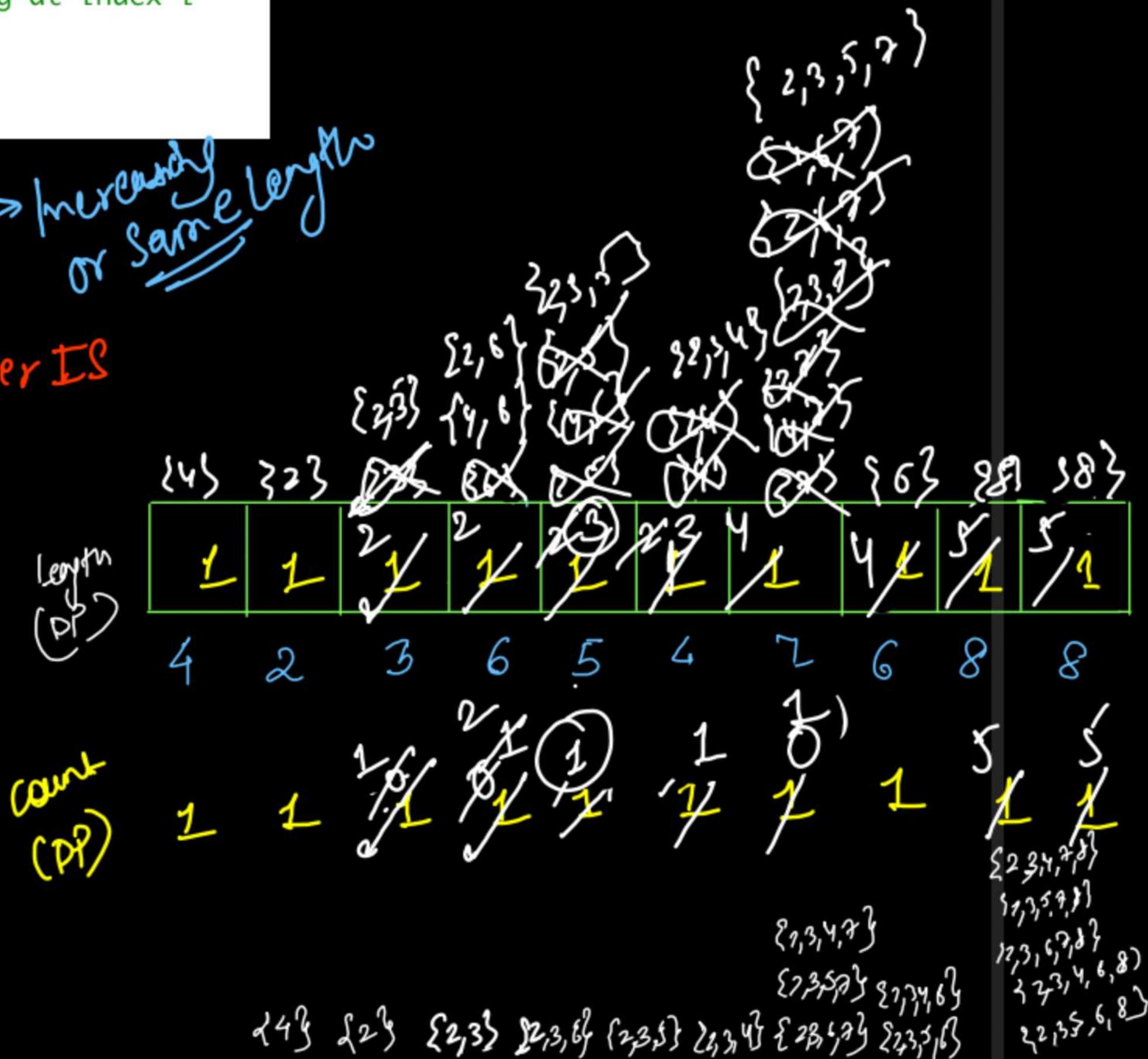
return countLIS;

```

*increasing
length
or same length*

[rejecting smaller LIS]

{add count of all LIS}

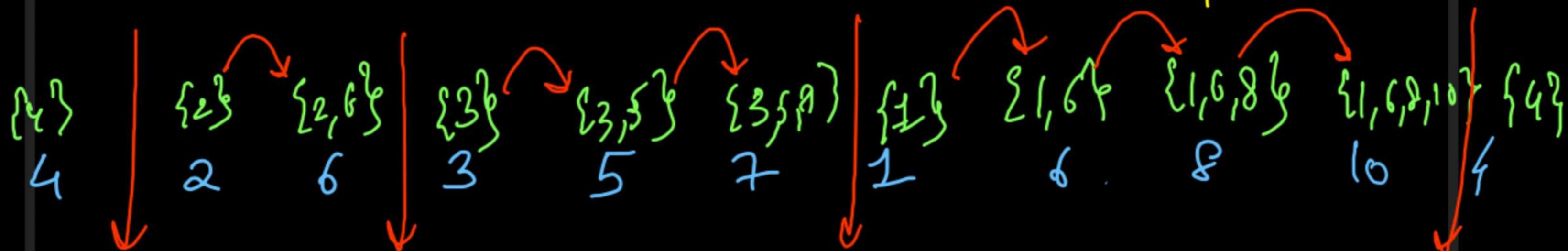


nc 674

longest Increasing Subarray

LIS + Kadane's

Subarray/Substring
↑ contiguous
Subsequence/Subset



Current Length = 1 1 2 3 4 5 6 7 8 9 10

max LIS = 1 2 3 4

```
public int findLengthOfLCIS(int[] nums) {  
    int curr = 0, max = 0;  
  
    for(int i=0; i<nums.length; i++){  
  
        if(i > 0 && nums[i-1] < nums[i]){  
            curr++; // Extend the Previous Subarray  
        } else {  
            curr = 1; // Start New Increasing Subarray  
        }  
  
        max = Math.max(max, curr);  
    }  
  
    return max;  
}
```

Time $\rightarrow O(n)$
Space $\rightarrow O(1)$
Linear
Constant

~~GFG~~

Max sum

Increasing subsequence

~~Logan~~

~~Sum~~

$\{4\}$	$\{2\}$	$\{4, 6\}$	$\{1, 3\}$	$\{1, 3, 5\}$	$\{4, 5, 7\}$	$\{1\}$	$\{1, 3, 5, 7\}$	$\{1, 2\}$
4	2	6	3	5	7	1	6	2
④	②	⑩	⑤	⑩	⑨	①	⑯	③

115
① Ordering
Increasing Progression,
Inc-Dec (Peak-valley)
Previous states
or property
②

```

public int maxSumIS(int nums[], int n)
{
    int[] dp = new int[n];
    int maxSum = 0;
    for (int i = 0; i < nums.length; i++) {
        dp[i] = nums[i]; // If Prev Does not Exist, then
          
        for (int j = 0; j < i; j++) {
            if (nums[j] < nums[i]) {
                dp[i] = Math.max(dp[i], dp[j] + nums[i]);
            }
        }
          
        maxSum = Math.max(maxSum, dp[i]);
    }

    return maxSum;
}

```

Time $\rightarrow O(N \times N)$

Space $\rightarrow O(N)$

1D DP

1671. Minimum Number of Removals to Make Mountain Array

feeling α
7/3 longer than

Lotto 113

15

{ longest Bitonic subsequence }
 ↗ {4} ↗ {2} ↗ {2,6} ↗ {1,3} ↗ {1,3} ↗ {2,3,5} ↗ {1,3,5,1} ↗ {1,3,5,1,3} ↗ {1,3,5,1,3,1}

4 2 6 3 5 7 1
 {4,5,2,1,3} {2,1,3} {6,5,4,2,1} {3,2,1,3} {5,4,2,1,3} {3,4,2,1,3} {1,3,5,1,3,1}

(6) (4) 7 (6) (7)

Bihmonic

Increasing Decreasing

$\{3,4,3\}$ $\{1,2,3\}$ $\{1,2,3\}$ $\{1,3\}$

4 2 3 1

$\{1,3\}$ $\{2,1\}$ $\{3,1,3\}$ $\{1,3\}$ $\{1,2,3\}$

(3) (3) (4) (4) 2

A diagram illustrating a path graph with 6 vertices. The vertices are represented by blue circles with green dots. The edges are drawn as green lines. There are two red edges: one connecting vertex 2 to vertex 3, and another connecting vertex 6 to vertex 5. The vertices are labeled with yellow numbers: 1, 2, 3, 4, 5, and 6. Vertex 1 is at the top left, vertex 6 is at the top right, vertex 2 is below vertex 1, vertex 3 is below vertex 2, vertex 4 is below vertex 5, and vertex 5 is above vertex 6.

```

int[] left = new int[nums.length];
// Longest Increasing Subsequence ending at index i

for(int i=0; i<nums.length; i++){
    left[i] = 1;

    for(int j=0; j<i; j++){
        if(nums[j] < nums[i]){
            left[i] = Math.max(left[i], left[j] + 1);
        }
    }
}

int[] right = new int[nums.length];
// Longest Decreasing Subsequence starting at index i

for(int i=nums.length-1; i>=0; i--){
    right[i] = 1;

    for(int j=nums.length-1; j>i; j--){
        if(nums[j] < nums[i]){
            right[i] = Math.max(right[i], right[j] + 1);
        }
    }
}

```

```

// Longest Bitonic Subsequence with peak at index i
// = LIS ending at i + LDS starting at i - 1
// -1 represents the peak element occurring in both LIS and LDS
// Constraint: LIS > 1 && LDS > 1 (Atleast 1 element to the left of peak
// and atleast 1 element to the right of peak)
int maxBitonic = 0;
for(int i=0; i<nums.length; i++){
    int curr = left[i] + right[i] - 1;
    if(left[i] > 1 && right[i] > 1){
        maxBitonic = Math.max(maxBitonic, curr);
    }
}

return nums.length - maxBitonic;

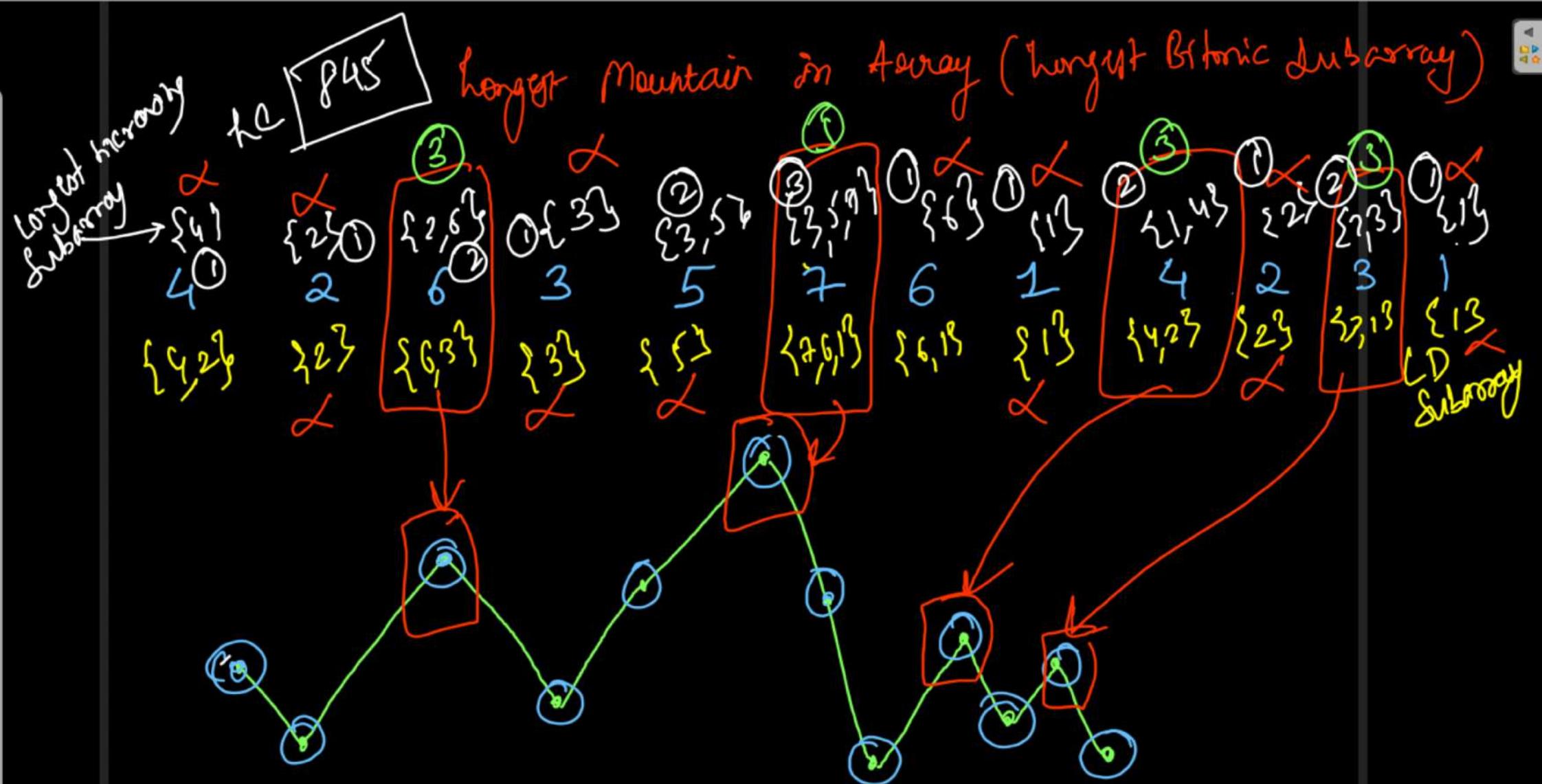
```

$\Theta(n^2)$

$\Theta(n^2)$

}

Return the no of elements
to be removed to
form longest
bitonic subsequence



```

public int longestMountain(int[] nums) {
    int[] left = new int[nums.length];
    // Longest increasing subarray ending at index i
    Arrays.fill(left, 1);

    for(int i=1; i<nums.length; i++){
        if(nums[i - 1] < nums[i]){
            left[i] = left[i - 1] + 1;
        }
    }

    int[] right = new int[nums.length];
    // Longest decreasing subarray starting at index i
    Arrays.fill(right, 1);

    for(int i=nums.length-2; i>=0; i--){
        if(nums[i] > nums[i + 1]){
            right[i] = right[i + 1] + 1;
        }
    }
}

```

```

// Longest Bitonic Subarray = LIS + LDS - 1
int max = 0;
for(int i=0; i<nums.length; i++){
    if(left[i] > 1 && right[i] > 1){
        max = Math.max(max, left[i] + right[i] - 1);
    }
}

return max;

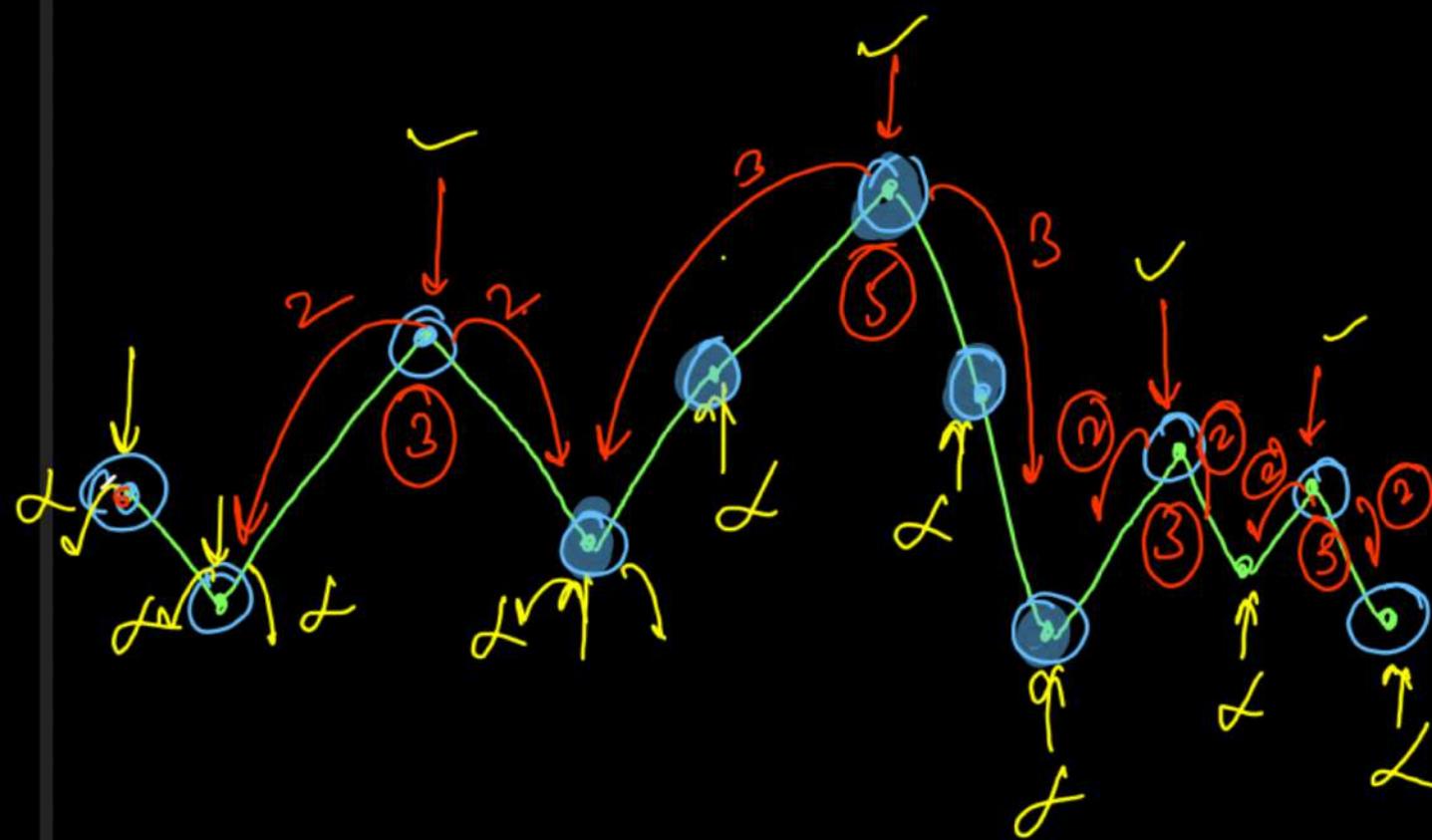
```

$\left\{ \begin{array}{l} \rightarrow O(N) \\ + \\ \left\{ \begin{array}{l} \rightarrow O(N) \\ + \\ \left\{ \begin{array}{l} \rightarrow O(N) \end{array} \right. \end{array} \right. \end{array} \right.$

longest Bitonic
 Subarray ($3N$)
 Time $\rightarrow O(N)$
 Space $\rightarrow O(N)$ ($2N$)

$$man = \phi \beta S$$

~~Greedy Soln~~ \Rightarrow Two Pointers
Technique

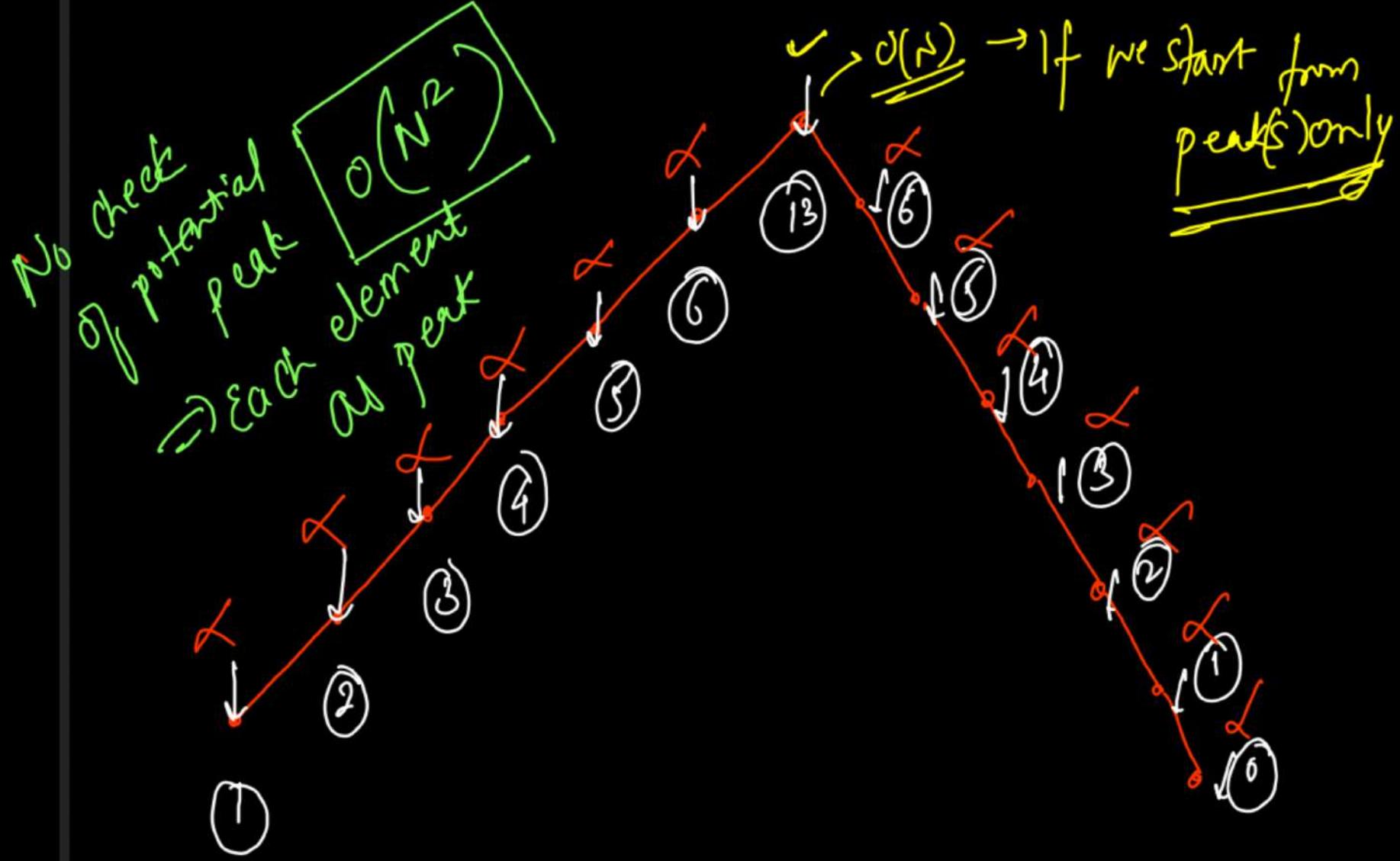


```
int max = 0;

for(int i=0; i<nums.length; i++){
    // Potential Peak Element
    if(i > 0 && i < nums.length - 1 && nums[i] > nums[i - 1] && nums[i] > nums[i + 1]){
        int curr = 1;
        int left = i - 1;
        while(left >= 0 && nums[left] < nums[left + 1]){
            left--;
            curr++;
        }
        int right = i + 1;
        while(right < nums.length && nums[right] < nums[right - 1]){
            right++;
            curr++;
        }
        max = Math.max(max, curr);
    }
}
```

Time $\rightarrow O(N)$
Linear

Space $\rightarrow O(1)$
Two pointers



distinct
positive

LC(368) Largest Divisible Subset

e.g. { 10, 2, 8, 3, 4, 1, 5, 16, 20, 9 }

$$(A(i), A(j))$$

sorted
↓
i < j

$$\Leftrightarrow A(i) \cdot A(j) = 0$$

$$j > i \Rightarrow A(j) \cdot A(i) = 0$$

If any subset is valid,
all its permutations
are also valid

$$(1, 5) \quad 10 \cdot 1 \cdot 5 = 0 \quad \{10, 20, 5\}$$

$$(5, 20) \quad 20 \cdot 1 \cdot 5 = 0 \quad \{10, 5, 20\} \quad \textcircled{3}$$

$$(10, 20) \quad 20 \cdot 1 \cdot 10 = 0 \quad \{5, 20, 10\}$$

$$\{20, 5, 10\}$$

$$\{20, 10, 5\}$$

$$\{5, 10, 20\}$$

$$\{2, 8, 4, 1, 16\}$$

$$\boxed{\{1, 2, 4, 8, 16\}}$$

↓

$\{ 10, 2, 8, 3, 4, 1, 5, 16, 9, 17 \}$

Sorted ~~for LIS to work~~

$\{ 1, 2, 3, 4, 5, 8, 9, 10, 16, 17 \}$

① {1} {2} {3} {4} {5} {8} {9} {10} {16} {17}
 ② {1,2} {1,3} {1,4} {1,5} {1,8} {1,9} {1,10} {1,16} {1,17}
 ③ {1,2,3} {1,2,4} {1,2,5} {1,2,8} {1,2,9} {1,2,10} {1,2,16} {1,2,17}

10ⁿ

②

②

{1,2,4}

②

{1,2,8}

{1,2,9}

{1,2,10}

{1,2,16}

{1,2,17}

③

③

③

$\begin{array}{|l} \hline \textcircled{1} \\ \hline \end{array} \left\{ \begin{array}{l} A_1/B = 0 \\ B_1/C = 0 \end{array} \right. \rightarrow A_1/C = 0$

10ⁿ

③

④

③

③

③

③

③

③

③

③

③

③

10ⁿ

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

10ⁿ

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

10ⁿ

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

10ⁿ

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

③

```

public List<Integer> largestDivisibleSubset(int[] nums) {
    // Sort the Array (Any Permutation of a valid subset is also valid)
    Arrays.sort(nums);

    int[] dp = new int[nums.length];
    Arrays.fill(dp, 1);

    int[] prev = new int[nums.length];
    Arrays.fill(prev, -1);

    int maxlen = 0, idx = 0;

    for(int i=1; i<nums.length; i++){
        for(int j=i-1; j>=0; j--){
            if(nums[i] % nums[j] == 0 && dp[j] + 1 > dp[i]){
                prev[i] = j;
                dp[i] = dp[j] + 1;
            }
        }

        if(dp[i] > maxlen){
            maxlen = dp[i];
            idx = i;
        }
    }
}

```

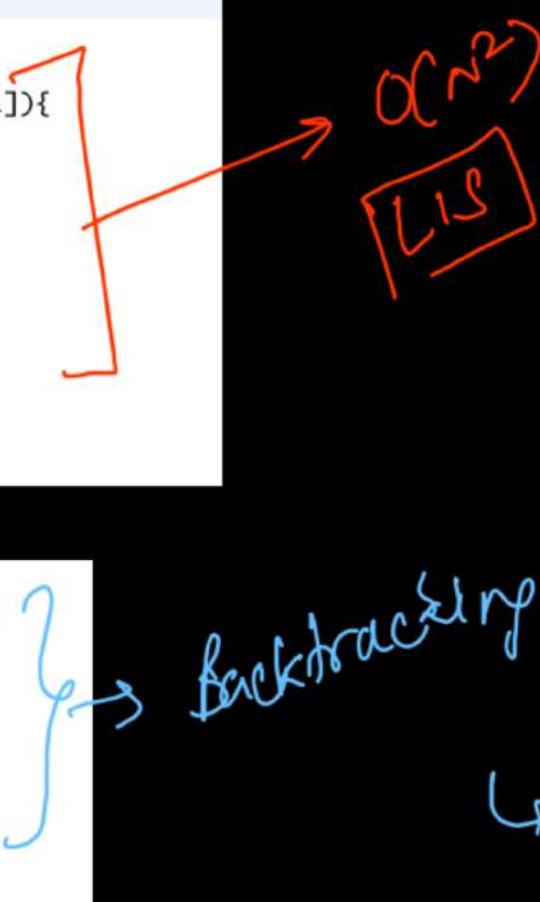
Time $\rightarrow O(N^2)$
 Space $\rightarrow O(N)$

```

List<Integer> subset = new ArrayList<>();
while(idx != -1){
    subset.add(nums[idx]);
    idx = prev[idx];
}

return subset;
}

```



backtracking \rightarrow Generating
 the subset
 $\hookrightarrow O(N)$

Tuesday, Thursday

9:45 PM

to

11:45 PM

Saturday

9 AM - 12 PM

3 PM - 5 PM

Sunday

9 AM - 12 PM

Dynamic Programming Lecture 23

25th May Wednesday 10 PM - 12 AM

LIS Variations

→ Perfect Squares (279)

↳ subarray (413)

→ Arithmetic Slices

↳ subset (446)

→ Wiggle Subset

↳ Russian Doll Envelope (354)

→ 2D/3D LIS

↳ Box Stacking (1691)

Today Tuesday	Wednesday	Thursday
3 of 100	3 of 100	3 of 100
11:11 AM	11:11 AM	11:11 AM

Perfect Squares

$$1 = 1^2 \quad \textcircled{1}$$

$$2 = 1^2 + 1^2 \quad \textcircled{2}$$

$$3 = 1^2 + 1^2 + 1^2 \quad \textcircled{3}$$

$$4 = 2^2 \quad \textcircled{1}$$

$$5 = 1^2 + 2^2 \quad \textcircled{2}$$

$$6 = 1^2 + 1^2 + 2^2 \quad \textcircled{3}$$

$$7 = 1^2 + 1^2 + 2^2 + 2^2 \quad \textcircled{4}$$

$$8 = 2^2 + 2^2 \quad \textcircled{2}$$

$$9 = 3^2 \quad \textcircled{1}$$

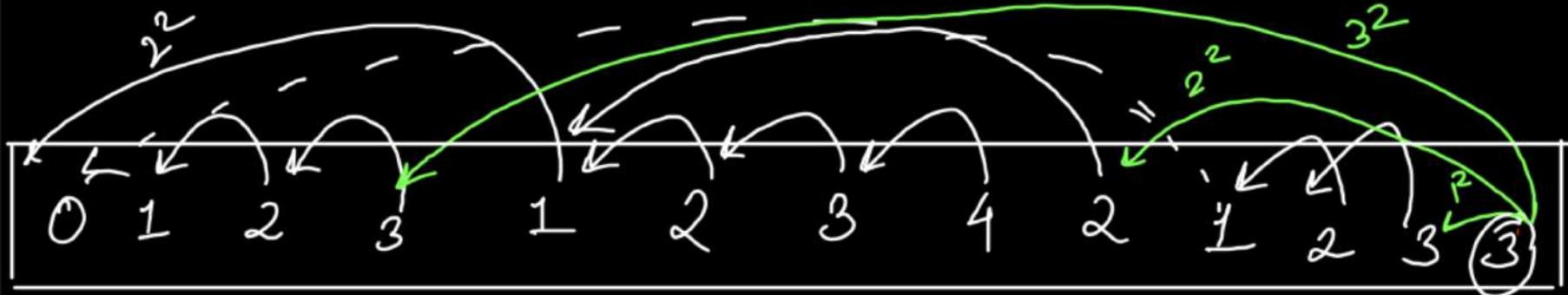
$$10 = 3^2 + 1^2 \quad \textcircled{2}$$

$$11 = 1^2 + 2^2 + 3^2 \quad \textcircled{3}$$

$$12 = 2^2 + 2^2 + 2^2 \quad \textcircled{3}$$

$$13 = 2^2 + 3^2 \quad \textcircled{2}$$

~~Greedy fails here~~
 $12 = 2^2 + 1^2 + 1^2 + 1^2$



0	1	2	3	4	5	6	7	8	9	10	11	12
1^2	$1^2 + 1^2$	2^2	$1^2 + 2^2$	$1^2 + 2^2 + 1^2$	$1^2 + 2^2$	$2^2 + 1^2$	3^2	$3^2 + 1^2$	$3^2 + 1^2$	7^2		

Time $\rightarrow O(N^2)$

Space

$O(N)$

1D DP

```

public int numSquares(int n) {
    int[] dp = new int[n + 1];

    for(int i=1; i<=n; i++){
        dp[i] = i;
        for(int j=1; j*j<=i; j++){
            dp[i] = Math.min(dp[i], dp[i - j * j] + 1);
        }
    }

    return dp[n];
}

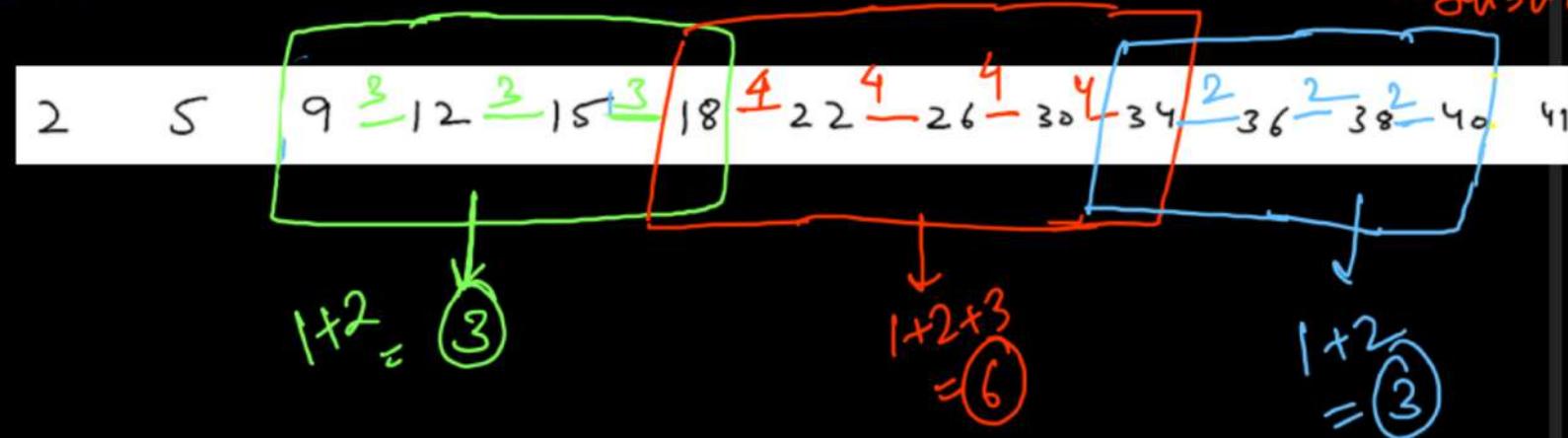
```

$1^2 + 1^2 + 3^2$
 $2^2 + 2^2 + 2^2$
 $3^2 + 1^2 + 1^2 + 1^2$

$\frac{7}{2}$
 Arithmetic progression
 formula: $a + (n-1)d$; $a + (n-1)d = a + (n-1)d$
 E.g.

$\frac{7}{2}$ Arithmetic slices
 I (Subarray) II (Subset)

Longest AP Subarray = \emptyset
 β
 γ
 δ
 ϵ



```
public int numberOfArithmeticSlices(int[] nums) {
    if(nums.length < 3) return 0;

    int count = 0;
    int currLength = 2;
    int diff = nums[1] - nums[0];

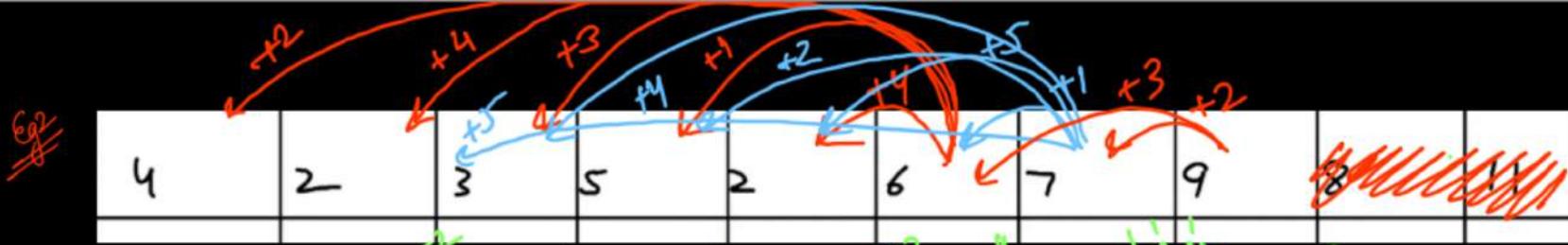
    for(int i=2; i<nums.length; i++){
        int newdiff = nums[i] - nums[i - 1];

        if(newdiff == diff){
            currLength++;
        } else {
            currLength = 2;
            diff = newdiff;
        }

        if(currLength >= 3)
            count = count + (currLength - 2);
    }

    return count;
}
```

④ ~~Greedy~~ | Kadane
Time $\rightarrow O(N)$
Space $\rightarrow O(1)$



2D DP
state

diff
9DP
1DDP

Hashmap

to store
specific
cdiff !!
no store
-ve diff

$\{4, 2\}$ $\{2, 3\}$ $\{3, 5\}$ $\{5, 2\}$ $\{2, 6\}$ $\{6, 1\}$ $\{5, 3\}$ $\{3, 7\}$ $\{7, 9\}$ $\{9, 3\}$ $\{3, 2\}$ $\{2, 5\}$ $\{5, 6\}$ $\{6, 3\}$ $\{3, 8\}$ $\{8, 1\}$ $\{1, 3\}$ $\{3, 2\}$ $\{2, 4\}$ $\{4, 5\}$ $\{5, 6\}$ $\{6, 4\}$ $\{4, 7\}$ $\{7, 5\}$ $\{5, 9\}$ $\{9, 6\}$ $\{6, 8\}$ $\{8, 7\}$ $\{7, 1\}$ $\{1, 5\}$ $\{5, 2\}$ $\{2, 9\}$ $\{9, 7\}$

Count = 124878

```

public int numberOfArithmeticSlices(int[] nums) {
    if(nums.length < 3) return 0;

    // DP[IDX][COMMON DIFF] = COUNT OF AP SUBSETS
    HashMap<Long, Long>[] dp = new HashMap[nums.length];
    for(int i=0; i<nums.length; i++){
        dp[i] = new HashMap<>();
    }

    long count = 0;
    for(int i=1; i<nums.length; i++){
        for(int j=i-1; j>=0; j--){
            long diff = 1L * nums[i] - nums[j];

            long oldVal = dp[i].getOrDefault(diff, 0L);
            long newVal = oldVal + dp[j].getOrDefault(diff, 0L) + 1L;

            dp[i].put(diff, newVal);
            count += dp[j].getOrDefault(diff, 0L);
        }
    }

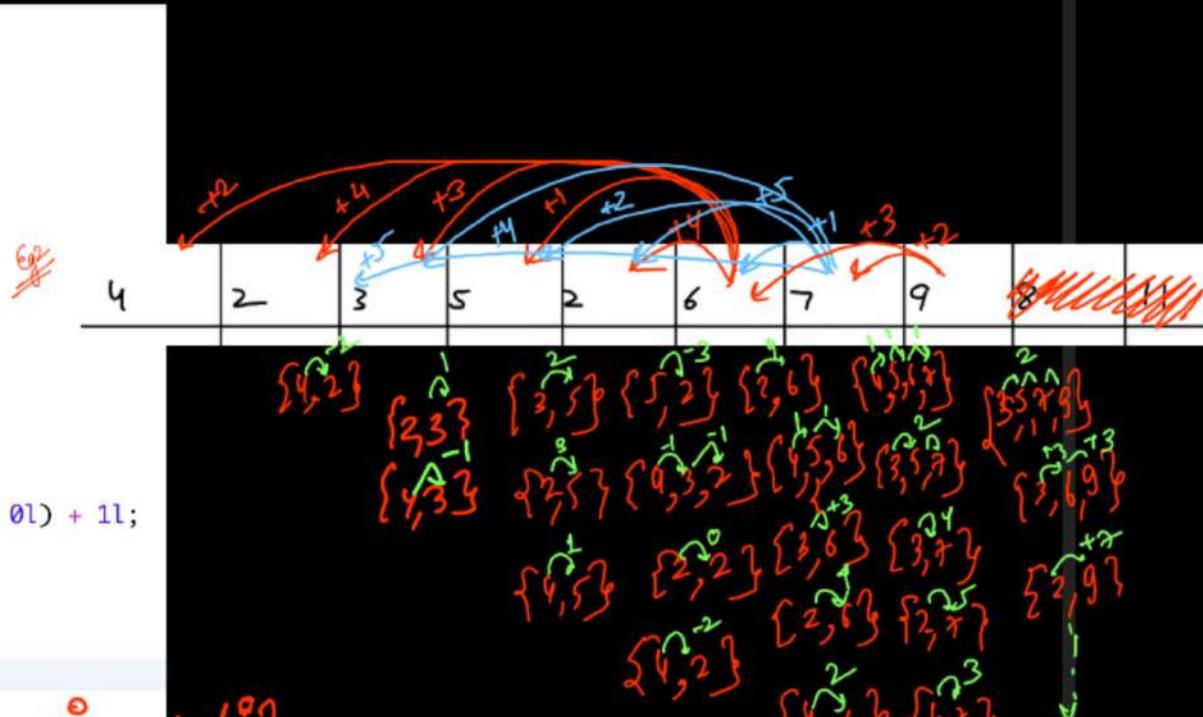
    return (int)count;
}

```

Instead of adding $dp[i]$,

we are adding
 $dp[j] \Rightarrow$ we are
able to exclude
subsets of
length 2

$$\text{Count} = \emptyset / 2 - \dots$$



Time Complexity

Wiggle Subsequence



[1, 17, 5, 10, 13, 15, 10, 5, 16, 8]

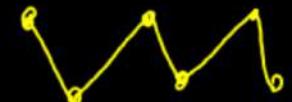
{1} {1, 5} {1, 13} {1, 17, 5} {1, 17, 13} {1, 17, 5, 13} {1, 10} {1, 17, 10} {1, 13, 10} {1, 17, 13, 10} {1, 17, 13, 15} {1, 17, 5, 15} {1, 17, 10, 15} {1, 17, 13, 10, 15} {1, 17, 13, 10, 15, 8}

longest = 5

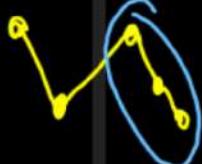
{1, 17, 5, 10, 5, 16, 8} ✓



{17, 5, 15, 10, 16, 8} ✓



{1, 17, 5, 10, 13} {17, 5, 15, 10, 8} ✗



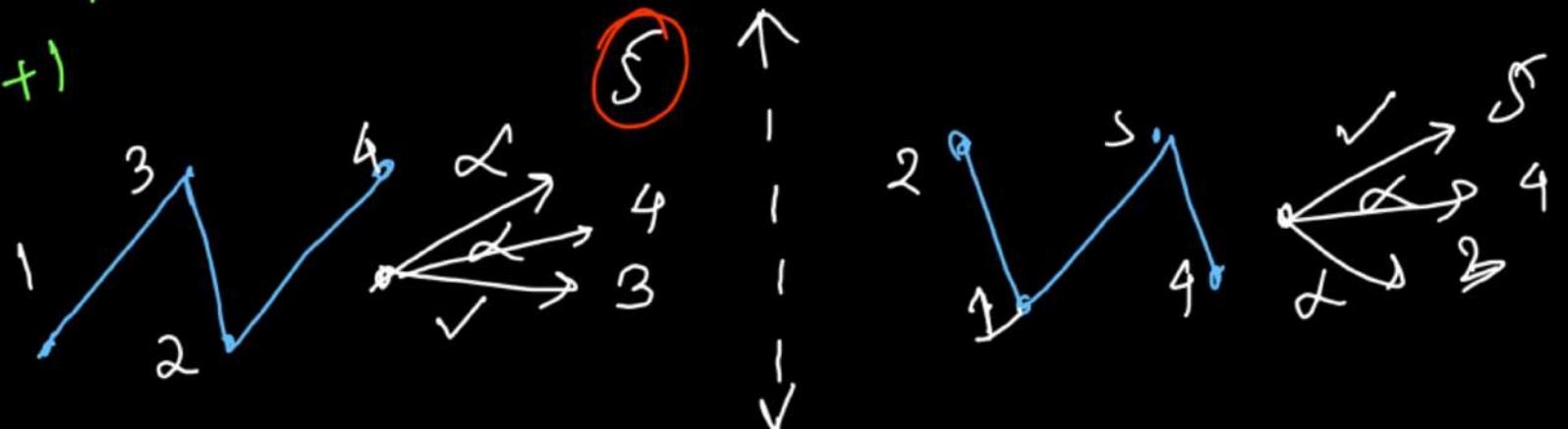
$\text{length} = \cancel{7}$
 $\cancel{4}, 5$

$$\begin{aligned} \text{inc}[i] \\ = \text{dec}[j] + 1 \\ \xleftarrow{\quad \text{dec}[i] \\ = \text{inc}[j] + 1 \quad} \end{aligned}$$

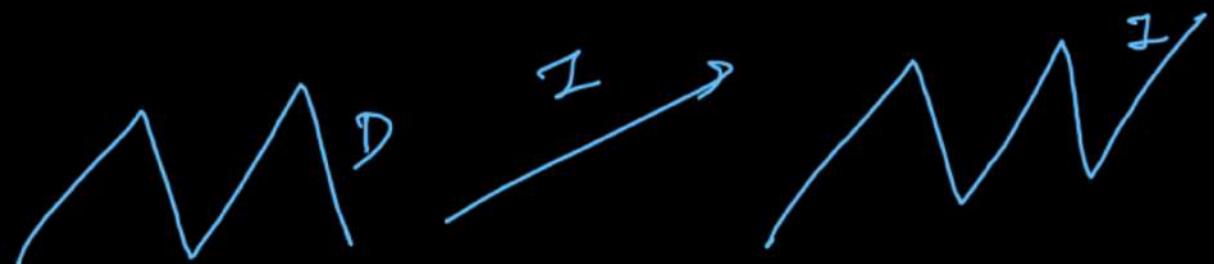
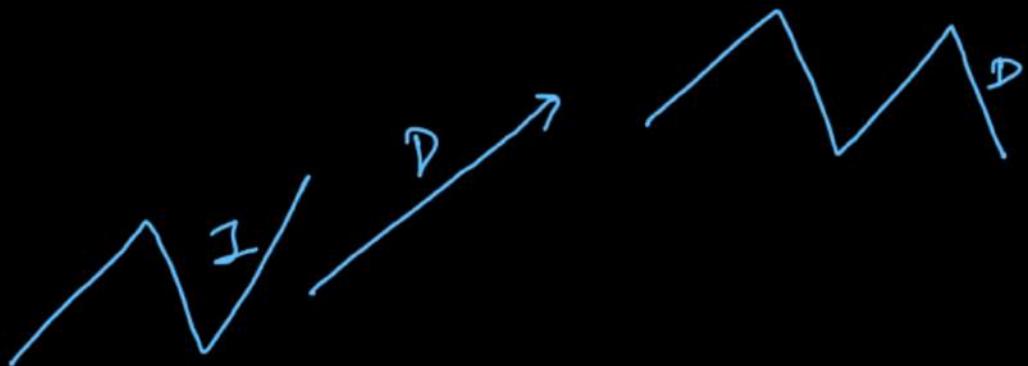
$$\begin{array}{l} I \rightarrow \{1, 17\} \\ D \rightarrow \{17\} \end{array} \quad \begin{array}{l} I \rightarrow \{1, 17, 5, 10\} \\ D \rightarrow \{1, 17, 10\} \end{array} \quad \begin{array}{l} I \rightarrow \{1, 17, 5, 15\} \\ D \rightarrow \{17, 15\} \end{array}$$

[1, 17, 5, 10, 13, 15, 10, 5, 16, 8]

$$\begin{array}{l} I \rightarrow \{1\} \\ D \rightarrow \{1\} \end{array} \quad \begin{array}{l} I \rightarrow \{1, 5\} \\ D \rightarrow \{1, 5\} \end{array} \quad \begin{array}{l} I \rightarrow \{1, 17, 5, 13\} \\ D \rightarrow \{1, 17, 5, 13\} \end{array} \quad \begin{array}{l} I \rightarrow \{1, 17, 5, 10\} \\ D \rightarrow \{1, 17, 5, 10\} \end{array}$$



$$\text{dec}(i) = \text{inc}(j) + 1$$



$$\begin{aligned}\text{inc}(i) \\ = \text{dec}(j) + 1\end{aligned}$$

```

public int wiggleMaxLength(int[] nums) {
    int[] inc = new int[nums.length];
    Arrays.fill(inc, 1);

    int[] dec = new int[nums.length];
    Arrays.fill(dec, 1);

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

        for(int j=0; j<i; j++){
            if(nums[i] > nums[j]){
                // We are Increasing
                inc[i] = Math.max(inc[i], dec[j] + 1);
            } else if(nums[i] < nums[j]){
                // We are Decreasing
                dec[i] = Math.max(dec[i], inc[j] + 1);
            }
        }

        maxLength = Math.max(maxLength, Math.max(inc[i], dec[i]));
    }

    return maxLength;
}

```

Time $\rightarrow O(2 \cdot n^2)$
 $= O(n^2)$

Space $\rightarrow O(2 \cdot n)$

2 rows ↴ 1D DP

LIS on 2D Coordinate

Overlapping Boxes
Practice - GFG

Russian Doll

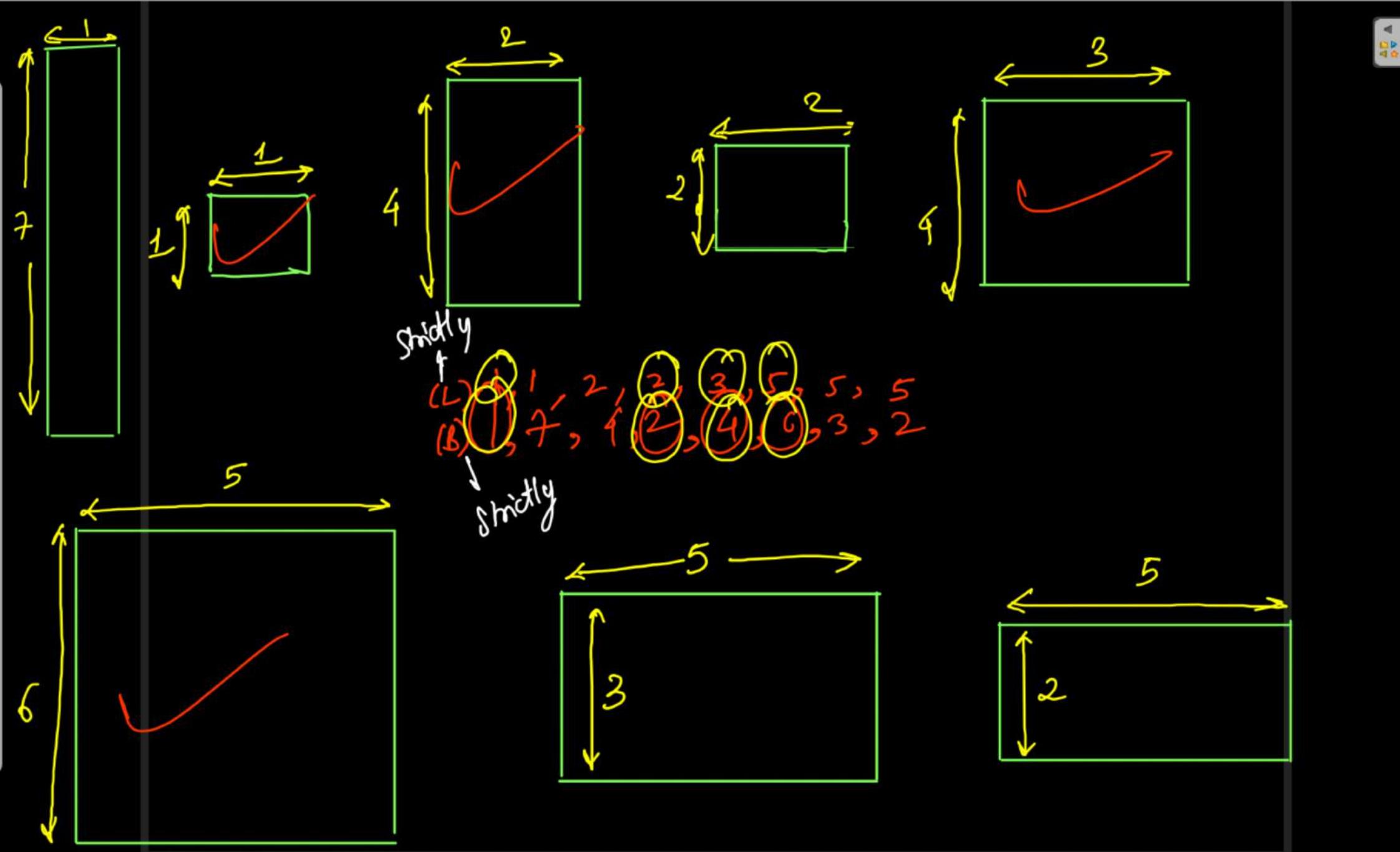


Box Stacking

GFG
IV-1

Leetcode
II

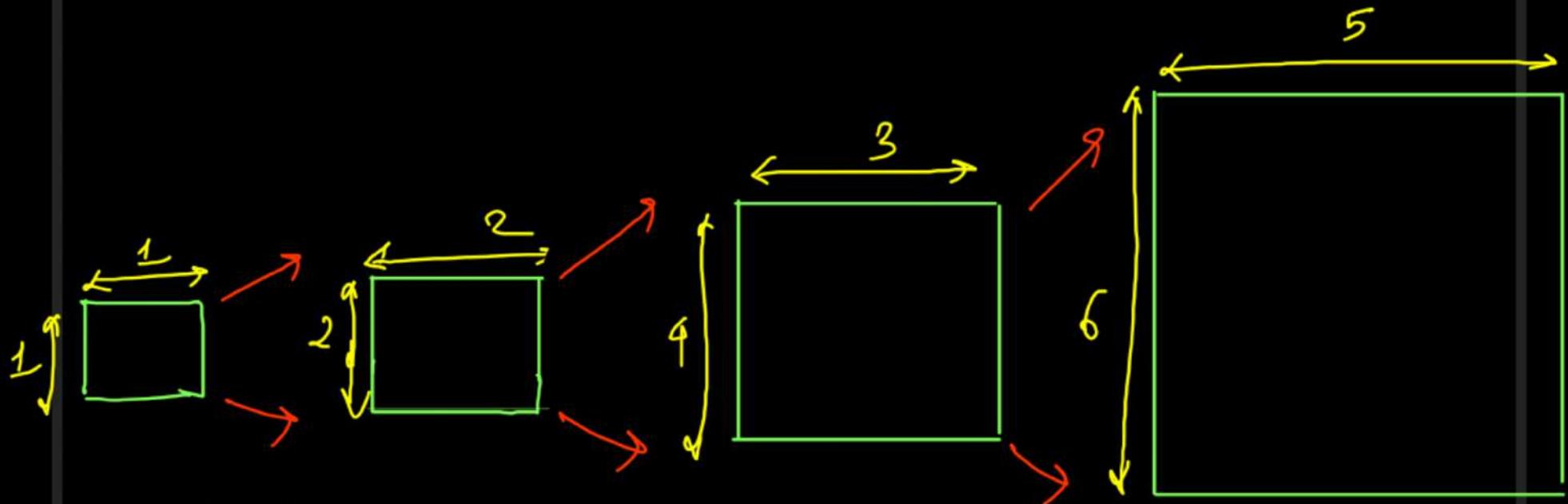
One envelope can fit into another if and only if both the width and height of one envelope are greater than the other envelope's width and height.



Time $O(N \log N + N \log N)$
Space $\Theta(N)$

```
public int lowerBound(ArrayList<Integer> nums, int target){  
    int low = 0, high = nums.size() - 1;  
    int idx = nums.size();  
  
    while(low <= high){  
        int mid = low + (high - low) / 2;  
  
        if(nums.get(mid) < target){  
            low = mid + 1;  
        } else {  
            high = mid - 1;  
            idx = mid;  
        }  
    }  
  
    return idx;  
}
```

```
public int maxEnvelopes(int[][] envelopes) {  
    Arrays.sort(envelopes, (int[] first, int[] second) -> {  
        return (first[0] != second[0]) ? first[0] - second[0]  
            : second[1] - first[1];  
    });  
  
    int n = envelopes.length;  
    ArrayList<Integer> sorted = new ArrayList<>();  
  
    for(int i=0; i<envelopes.length; i++){  
        int lb = lowerBound(sorted, envelopes[i][1]);  
        if(lb == sorted.size()){  
            sorted.add(envelopes[i][1]);  
            // Current Element larger than the largest  
            // LIS of one length more  
        } else {  
            sorted.set(lb, envelopes[i][1]);  
        }  
    }  
  
    return sorted.size(); // This Sorted Array has same size as LIS  
}
```



LIS on length

LIS on breadth

1, 2, 3, 5

1, 2, 4, 6

2D LIS

$\begin{cases} \rightarrow 1^{\text{st}} \text{ coordinate} \rightarrow \text{Sorting} \\ \downarrow 2^{\text{nd}} \text{ coordinate} \rightarrow \text{LIS} \end{cases}$

longest common subsequence

Lecture-1 Saturday (28 May)

9 AM to 12 PM

→ length of LCS

→ Print LCS

Any one

All

→ longest repeated subset (LRS)

→ Palindromic
subsequence

length

Count

Highest Common Subsequence

↑
is order of characters

e.g.

"a b c d e" , "i d c a"

Brute force

$O(2^n \times 2^m)$

Compare all
subsequences

The diagram illustrates the recursive step of finding the Longest Common Subsequence (LCS) between two strings, s and t .

On the left, a portion of string s is shown with indices $i-1$ and i . A double-headed arrow between these indices is labeled $s[i:i] = s[i:j]$. Below this, the character a is circled in green and labeled "char taken".

In the center, the label $LCS(i, j)$ is written above a vertical line.

On the right, a portion of string t is shown with indices $j-1$ and j . A double-headed arrow between these indices is labeled $t[j:j]$. Below this, the character v is circled in green and labeled "char taken".

Below the central labels, the expression " a " + $LCS(i+1, j+1)$ " is written.

At the bottom, a sequence of characters is shown in a grid:

a	v	a	v	$\rightarrow n$
a	v	a	d	length ↓
a	d	a	v	
a	d	a	d	

$S[i:j] = S^2[j]$

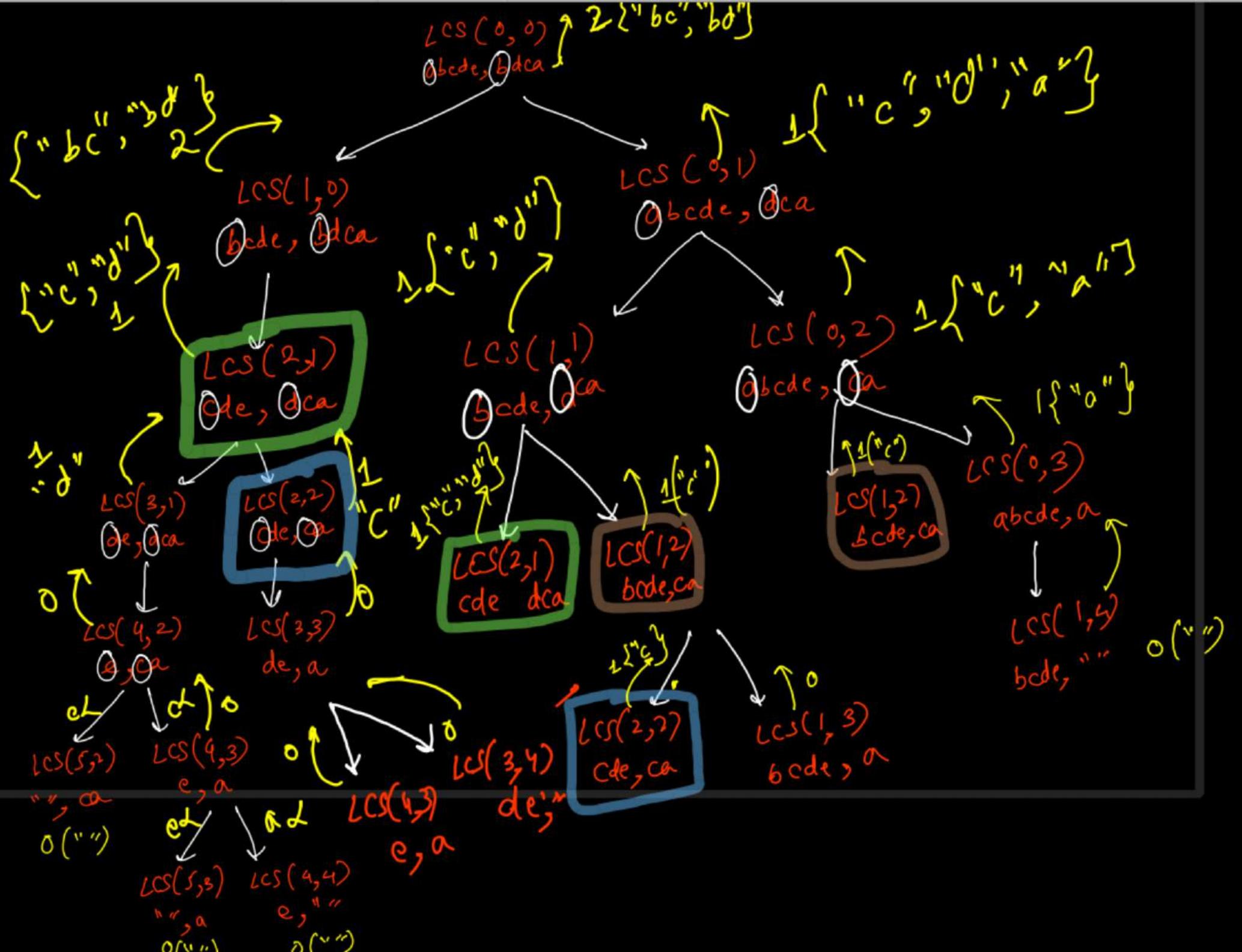
LCS(i:j)

char not taken = $\delta + LCS(i+1, j)$ $\delta + LCS(i, j+1)$

"rC", δ , "Cr"
 "Cr", δ , "r"

r v i v C → common ↕
 rC iC C → Length ↕

$LCS(i:j)$
 $\delta + LCS(i+1, j+1)$



```

public int LCS(int i, int j, String s1, String s2, int[][] dp){
    if(i == s1.length() || j == s2.length())
        return 0; // LCS of Empty String with Other String is Empty String only

    if(dp[i][j] != -1) return dp[i][j];

    char ch1 = s1.charAt(i);
    char ch2 = s2.charAt(j);

    if(ch1 == ch2) // If characters are same, take the common from both of them
        return dp[i][j] = 1 + LCS(i + 1, j + 1, s1, s2, dp);

    // If character is uncommon, either not take s1[i] or not take s2[j]
    int option1 = LCS(i + 1, j, s1, s2, dp);
    int option2 = LCS(i, j + 1, s1, s2, dp);
    return dp[i][j] = 0 + Math.max(option1, option2);
}

```

```

public int longestCommonSubsequence(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=0; i<dp.length; i++){
        for(int j=0; j<dp[0].length; j++){
            dp[i][j] = -1;
        }
    }

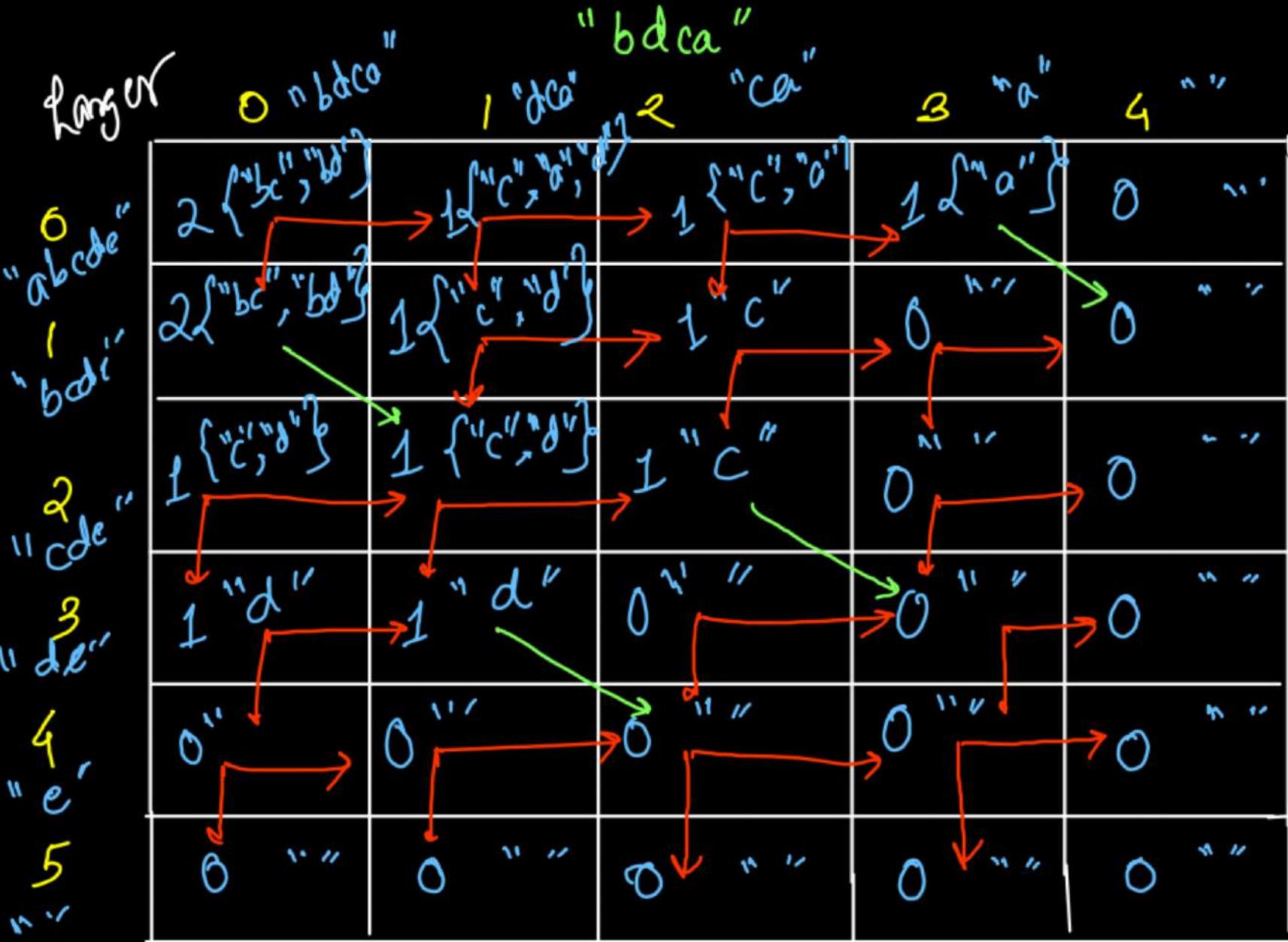
    return LCS(0, 0, s1, s2, dp);
}

```

Time $\rightarrow O(N \times M)$

Space $\rightarrow 2D DP$
 $O(N \times M)$

eg "abcde", "bdca"



Smallest

```

public int longestCommonSubsequence(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];

    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];
            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    return dp[0][0];
}

```

Time $\rightarrow O(N \times M)$

Space $\rightarrow O(N \times M)$

$\boxed{2D DP}$

```

int[] next = new int[s2.length() + 1];

for(int i=s1.length()-1; i>=0; i--){
    int[] curr = new int[s2.length() + 1];

    for(int j=s2.length()-1; j>=0; j--){
        char ch1 = s1.charAt(i);
        char ch2 = s2.charAt(j);

        if(ch1 == ch2)
            curr[j] = 1 + next[j + 1];
        else curr[j] = Math.max(next[j], curr[j + 1]);
    }

    next = curr;
}

return next[0];

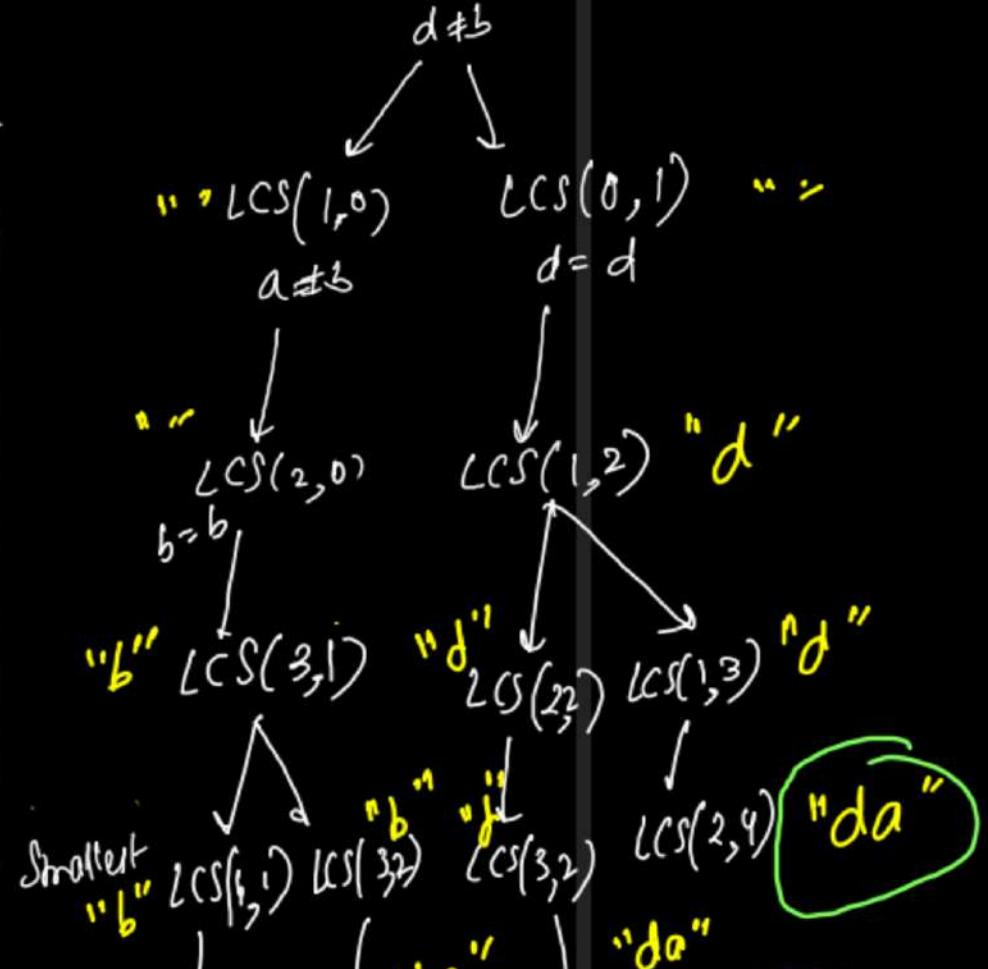
```

Time $\rightarrow O(N \times M)$

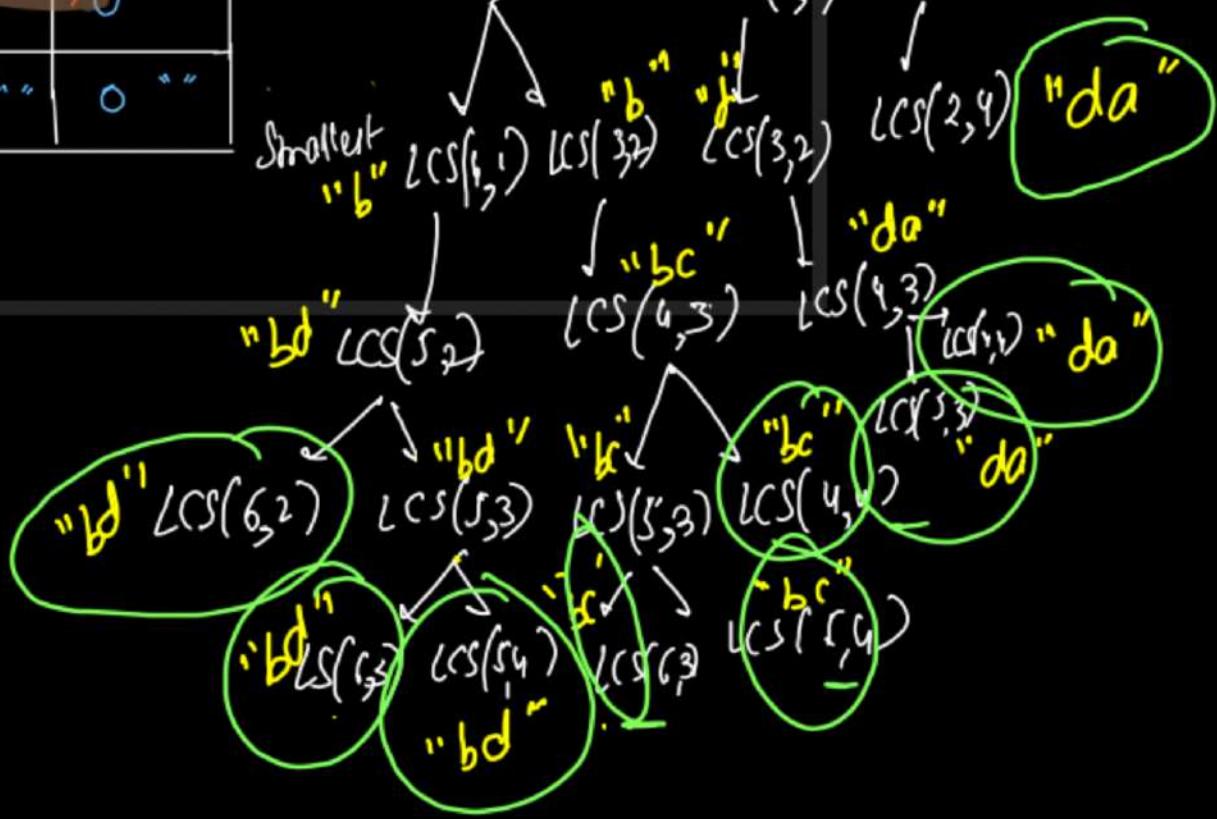
Space $\rightarrow O(2 \times M)$

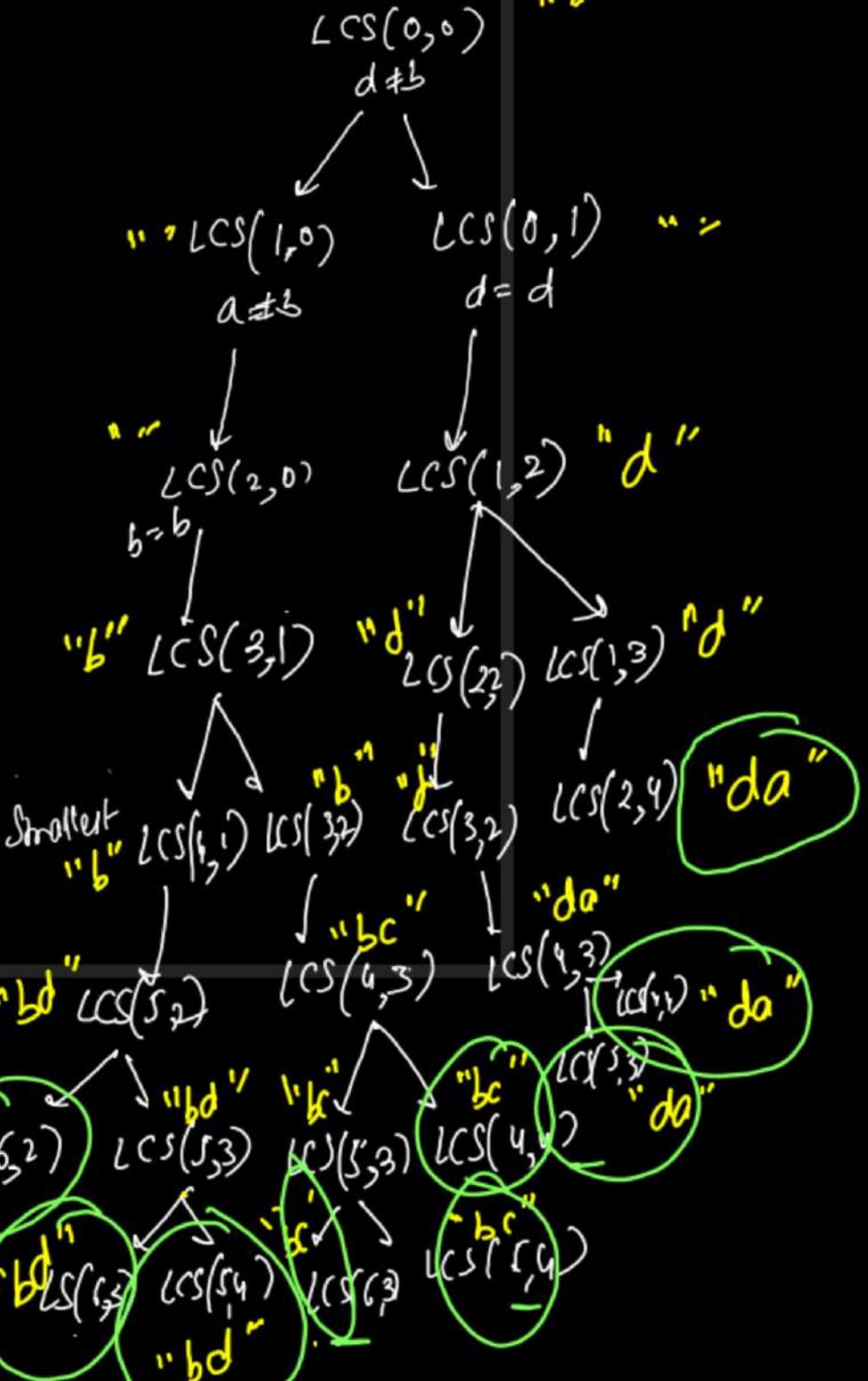
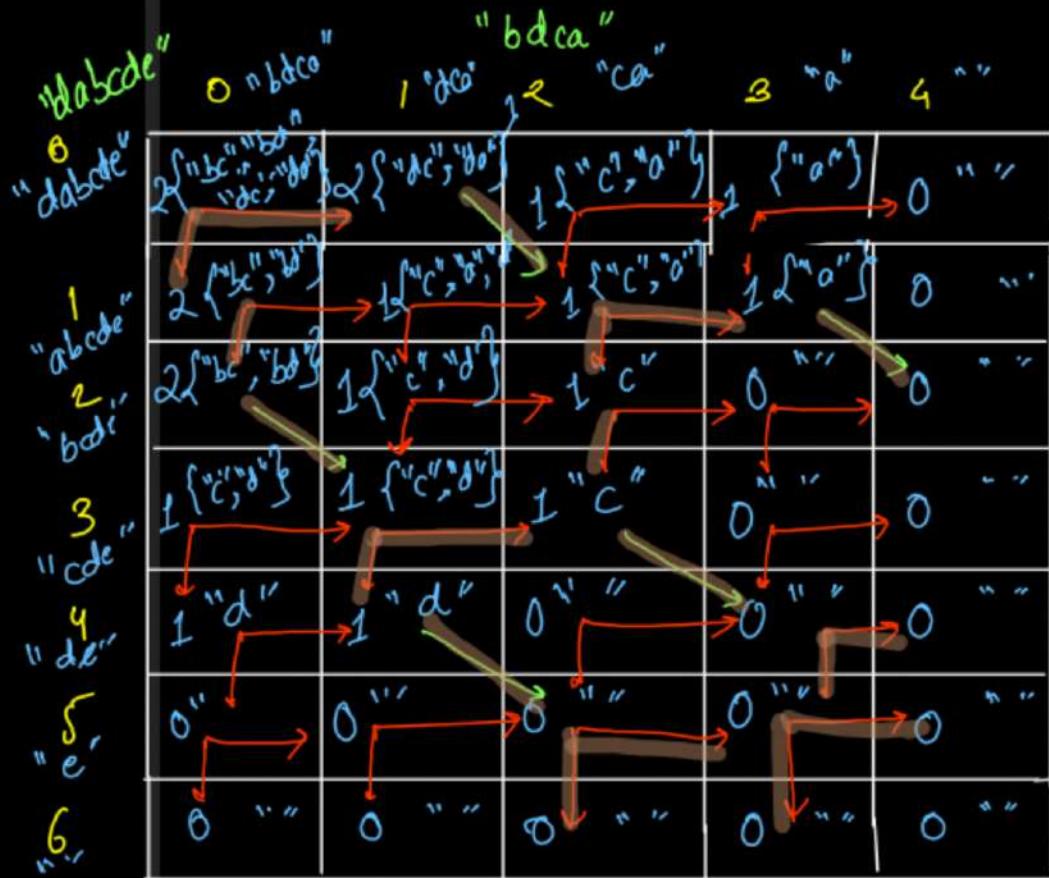
$\approx \boxed{1D DP}$

	0 "bdc"	1 "dc"	2 "ca"	3 "a"	4 "
0 "dabce"	"bdca"				
1 "abcde"		"ca"			
2 "bde"			"a"		
3 "de"				"a"	
4 "e"					"a"
5 "					
6 "					



- ✗ ArrayList → Sorted ↗
- ✗ ArrayList → Unique ↗
- ✗ HashSet → Unique ↗
- ✓ TreeSet → Unique ↗





```

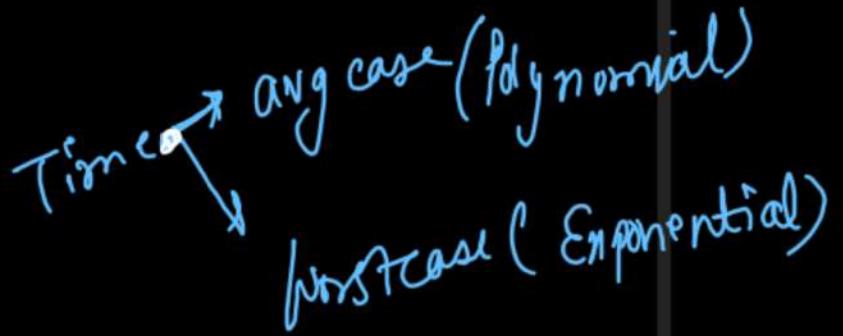
TreeSet<String> answers; // Both Ordered (Lexicographical Order), Unique

public void helper(int i, int j, String s1, String s2, int[][] dp, String lcs){
    if(i == s1.length() || j == s2.length()){
        answers.add(lcs);
        return;
    }

    char ch1 = s1.charAt(i);
    char ch2 = s2.charAt(j);

    if(ch1 == ch2){
        // Character Taken (Same)
        helper(i + 1, j + 1, s1, s2, dp, lcs + ch1);
    } else {
        // Character Not Taken
        if(dp[i][j] == dp[i + 1][j]){
            helper(i + 1, j, s1, s2, dp, lcs);
        }
        if(dp[i][j] == dp[i][j + 1]){
            helper(i, j + 1, s1, s2, dp, lcs);
        }
    }
}

```


 Time
 Best Case (Exponential)
 Worst Case (Polynomial)

```

public List<String> all_longest_common_subsequences(String s1, String s2)
{
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];

    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];
            else
                dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    answers = new TreeSet<>();
    helper(0, 0, s1, s2, dp, "");
    List<String> res = new ArrayList<>();
    for(String str: answers){
        res.add(str);
    }
    return res;
}

```

718

longest Common Substring

~~LCSS~~

Contiguous

~~eg~~

"dabcde" vs "bdca"

Substitution

bc → ✓ ✗

bd → ✗ ✓

dc → ✗ ✓

da → ✓ ✗

longest common substring
→ "d", "a", "b", "c"~~eg~~"ab~~cd~~abc"

$$\delta(S_i) = S_j$$

LCSS(i, j)

→ LCSS(i+1, j+1)

$$S_i \neq S_j$$

~~LCSS(i+1, j)~~
~~LCSS(i, j+1)~~

	b	c	d	b c d b c a	c	a	..
a	0	0	0	0	0	1	0
b	3 "bcd"	0	0	2 "bc"	0	0	0
c	0	2 "cd"	0	0	1 "c"	0	0
d	0	0	1 "d"	0	0	0	0
a	0	0	0	0	0	1 "a"	0
b	2 "bc"	0	0	2 "bc"	0	0	0
c	0	1 "c"	0	0	1 "c"	0	0
..	0	0	0	0	0	0	0

```

class Solution {
    public int findLength(int[] s1, int[] s2) {
        int[][] dp = new int[s1.length + 1][s2.length + 1];

        int maxLen = 0;
        for(int i=s1.length-1; i>=0; i--){
            for(int j=s2.length-1; j>=0; j--){
                if(s1[i] == s2[j])
                    dp[i][j] = 1 + dp[i + 1][j + 1];

                // else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
                maxLen = Math.max(maxLen, dp[i][j]);
            }
        }

        return maxLen;
    }
}

```

Time $\rightarrow O(N \times M)$

Space $\rightarrow O(N \times M)$



④ It can be ~~Space~~
optimized
to 1D DP

Longest Common Subsequence - II

29th May, Sunday, 9 AM - 12 PM

- Longest Repeating / Duplicate Subsequence (QFG)
 - I (104 LC)
- Longest Duplicate String
 - II (QFG)
- Longest Palindromic Subsequence (LC 516)
 - Insert (QFG)
- Min^m Steps to Make Palindrome
 - Delete (1312 LC)
- Palindromic Substrings
 - longest (5 LC)
 - Count (647 LC)



Hongut Repeating Subsequence

LCS

"abac**b**c"

	aback	backc	acbc	c ^b c	bc	c	" "
abacbc	3 "abc"	3 "abc"	3 "abc"	2 "bu"	2 "bl"	1 "c"	0
bachbc	3 "blk"	2 "bc"	2 "bc"	2 "be"	2 "bl"	1 "c"	0
acbc	3 "abc"	2 "bc"	1 "c"	1 "c"	1 "c"	1 "c"	0
abc	2 "bc"	2 "be"	1 "c"	1 "c"	1 "c"	1 "c"	0
bc	2 "bc"	2 "bc"	1 "c"	1 "c"	0	0	0
c	1 "c"	1 "c"	1 "c"	1 "c"	0	0	0
" "	0	0	0	0	0	0	0

$$\begin{aligned} \text{diff char} \\ dp[i][j] \\ = dp[i+1][j] \\ dp[i][j+1] \end{aligned}$$

$$\begin{aligned} \text{Same char} \\ 2 & \& 1 = 0 \\ dp[i][j] &= dp[i+1][j+1] \\ \neq 1 \end{aligned}$$

```

public int LongestRepeatingSubsequence(String s)
{
    int[][] dp = new int[s.length() + 1][s.length() + 1];

    for(int i=s.length()-1; i>=0; i--){
        for(int j=s.length()-1; j>=0; j--){
            char ch1 = s.charAt(i);
            char ch2 = s.charAt(j);

            if(ch1 == ch2 && i != j) → longest repeating subset
                dp[i][j] = 1 + dp[i + 1][j + 1];
            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    return dp[0][0];
}

```

with same
LCS + 1

Longest Repeating Subsequence									
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	3	4	5	6	7	8	9	10
3	3	4	5	6	7	8	9	10	11
4	4	5	6	7	8	9	10	11	12
5	5	6	7	8	9	10	11	12	13
6	6	7	8	9	10	11	12	13	14
7	7	8	9	10	11	12	13	14	15
8	8	9	10	11	12	13	14	15	16
9	9	10	11	12	13	14	15	16	17

Longest Repeating Substring

Lecture 1044

Non-overlapping (GFG)

Overlapping (Leetcode)

e.g. "banana"

	b	a	n	a	n	a	..
b	0	0	0	0	0	0	0
a	0	0	0	3 "ana"	0	1 "a"	0
n	0	0	0	0	2 "na"	0	0
a	0	3 "ana"	0	0	0	1 "a"	0
n	0	0	2 "na"	0	0	0	0
a	0	1 "a"	0	1 "a"	0	0	0
..	0	0	0	0	0	0	0

Longest common
substring

On 2 same
strings

$\text{dp}(i)(j) = \text{LCS } \sigma$

$\text{S1.substr}(i)$

$\text{S2.substr}(j)$

meaning

1044

```
public String longestDupSubstring(String s) {  
    int[][] dp = new int[s.length() + 1][s.length() + 1];  
  
    int idx = s.length(), len = 0;  
  
    for(int i=s.length()-1; i>=0; i--){  
        for(int j=s.length()-1; j>=0; j--){  
            char ch1 = s.charAt(i);  
            char ch2 = s.charAt(j);  
  
            if(ch1 == ch2 && i != j)  
                dp[i][j] = 1 + dp[i + 1][j + 1];  
  
            if(dp[i][j] > len){  
                idx = i;  
                len = dp[i][j];  
            }  
  
            // else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);  
        }  
  
        return s.substring(idx, idx + len);  
    }  
}
```

(TLE) → Expected Rolling Hash

Time → $O(N^2)$
Space → $O(N^2)$ 2D DP



Largest Repeated / Duplicate Substrings
with non-overlapping (QFG)

"banana" length = 2 ("an")

b	0	0	0	0	0	0	0
a'	0	0	0	3 "ana"	0	1 "a"	0
n'	0	0	0	0	2 "na"	0	0
a''	0	3 "na"	0	0	0	1 "a"	0
n''	0	0	2 "na"	0	0	0	0
a'''	0	1 "a"	0	1 "a"	0	0	0
..	0	0	0	0	0	0	0

$\min(i,j) + \text{dplis}(j)$
 $< \max(i,j)$

```

int[][] dp = new int[s.length() + 1][s.length() + 1];
int idx = s.length(), len = 0;

for(int i=s.length()-1; i>=0; i--){
    for(int j=s.length()-1; j>=0; j--){
        char ch1 = s.charAt(i);
        char ch2 = s.charAt(j);

        if(ch1 == ch2 && i != j)
            dp[i][j] = 1 + dp[i + 1][j + 1];

        if(Math.min(i, j) + dp[i][j] >= Math.max(i, j)){
            // Overlapping Substrings
            dp[i][j] = 0;
        }

        if(dp[i][j] >= len){
            idx = i;
            len = dp[i][j];
        }
    }
}

```

```

if(len == 0) return "-1";
return s.substring(idx, idx + len);

```

$\text{abs}(i-j) \leq dp(i,j)$
 Only non-overlapping substrings
 To compute
 Time $\rightarrow O(n^2)$
 Space $\rightarrow O(n^2)$

Longest Palindromic Substring									
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

"abba" "aaa" "ababa" "aca"

"aaa"

"ababa" "aca"

Longest Palindromic Subsequence

$$\text{Rev}("abcacba") = \underline{abca} \underline{cba}$$

acaca, ababa

" $a_{bc}a_c{}^b - a$

a
b
c
a''
b''

C

a''

```

public int LCS(String s1, String s2){
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];

    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];
            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    return dp[0][0];
}

public int longestPalindromeSubseq(String s) {
    StringBuilder rev = new StringBuilder(s);
    rev = rev.reverse();
    String s2 = new String(rev);

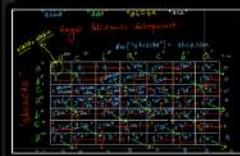
    return LCS(s, s2);
}

```

LPS
 $= LCS(s, rev(s))$

Time $\rightarrow O(n^2)$

Space $\approx O(n^2)$



The screenshot shows a mobile application interface with a title bar at the top. Below the title, there is a large grid area containing a 2D matrix. The matrix has rows and columns labeled with letters from the string 's'. The cells are filled with various characters, likely representing the state or value of each cell in the dynamic programming table used for the LCS computation. The overall layout is clean and professional, typical of a developer's tool or educational app.

"abcabca"

min Deletions
to make Palindrome

min Insertions
to make Palindrome

"abcabca"

 * * * * *

"a b c ^b↓ a b c ^b↓ a"

 * * * * *

$$ans = \text{str.length} - LPS$$

$$ans = \text{str.length} - LPS$$

```
10
"abcabca"
ans = str.length - LPS
ans = 7 - 3
ans = 4
```

```

class Solution {
    public int LCS(String s1, String s2){
        int[][] dp = new int[s1.length() + 1][s2.length() + 1];

        for(int i=s1.length()-1; i>=0; i--){
            for(int j=s2.length()-1; j>=0; j--){
                char ch1 = s1.charAt(i);
                char ch2 = s2.charAt(j);

                if(ch1 == ch2)
                    dp[i][j] = 1 + dp[i + 1][j + 1];

                else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
            }
        }

        return dp[0][0];
    }

    public int minInsertions(String s) {
        StringBuilder rev = new StringBuilder(s);
        rev = rev.reverse();
        String s2 = new String(rev);

        return s.length() - LCS(s, s2);
    }
}

```

Min insertions /
 Deletions
 to make
 Palindrome



longest Palindromic Substring

"abcabac" LPS = "aba" → ③

$$LPS = LCS(s, \text{rev}(s))$$

↑ ↗
Longest Palindromic longest Common
Substring Subsequence

Time → $O(N^2)$

Space → $O(N^2)$

```
public String longestCommonSubstring(String s1, String s2){  
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];  
  
    String res = "";  
  
    for(int i=s1.length()-1; i>=0; i--){  
        for(int j=s2.length()-1; j>=0; j--){  
            char ch1 = s1.charAt(i);  
            char ch2 = s2.charAt(j);  
  
            if(ch1 == ch2)  
                dp[i][j] = 1 + dp[i + 1][j + 1];  
  
            if(dp[i][j] >= res.length()){  
                String curr = s1.substring(i, i + dp[i][j]);  
  
                if(isPalindrome(curr) == true){  
                    res = curr;  
                }  
            }  
        }  
    }  
  
    return res;  
}
```

```
public String longestPalindrome(String s) {  
    StringBuilder rev = new StringBuilder(s);  
    rev = rev.reverse();  
    String s2 = new String(rev);  
  
    return longestCommonSubstring(s, s2);  
}
```

This type of DP might go wrong for some test cases

```
class Solution {  
    public String longestPalindrome(String s) {  
        String rev = new StringBuilder(s).reverse().toString();  
        return longestCommonSubstring(s, rev);  
    }  
}
```

Expand Around

Center

Greedy
odd length
palindromic substrings

"abaabacca"

↳ Treat each character as center for odd length palindromes

even length
palindromic
substrings

"abaababaacca"

↳ Take in $2w^2$ characters
as middle for even length palindromes

(odd)

LPSS = "

"a"/"b"/"c"

"aba"

"cabac"

(even)

LPSS = "

"abb"

"abba"

"ababa"



Time $\rightarrow O(N^2)$ $\rightarrow N \times O(N)$ *expand time in worst case*, Space $\rightarrow O(1)$

```

String res = "";
for(int i=0; i<s.length(); i++){
    // Odd Length Palindromes
    int left = i - 1, right = i + 1, len = 1;
    while(left >= 0 && right < s.length()){
        if(s.charAt(left) == s.charAt(right)){
            len = len + 2;
            left--; right++;
        } else {
            break;
        }
    }

    if(len > res.length()){
        res = s.substring(left + 1, left + 1 + len);
    }
}

```

```

for(int i=0; i<s.length(); i++){
    // Even Length Palindromes
    if(i + 1 == s.length() || s.charAt(i) != s.charAt(i + 1)){
        continue;
    }

    int left = i - 1, right = i + 2, len = 2;
    while(left >= 0 && right < s.length()){
        if(s.charAt(left) == s.charAt(right)){
            len = len + 2;
            left--; right++;
        } else break;
    }

    if(len > res.length()){
        res = s.substring(left + 1, left + 1 + len);
    }
}

return res;

```

Diagram illustrating the expansion of a palindrome centered at index 5 ('b') in the string 'abbaabacca'. The center is highlighted in green. The left boundary is at index 1 ('a'), and the right boundary is at index 7 ('c'). The length of the palindrome is 6.



Expression Matching Problems

Recursion
Memoization
Tabular
Space Optimization

- Shortest Common Supersequence [LC 1092]
- Distinct subsequences [LC 115] Hard Hard
- Edit Distance [LC 72] Hard
- Wildcard Matching [LC 44] Hard

- RegEx Matching [LC 10] Hard

- Min Operations to make $A \xrightarrow{\beta} B \rightarrow$ Min ASCII
 - Interleaving String
 - Min Insertion/Deletion
 - Sum
 - Delete

contains both S1 & S2
as a subsequence

Shortest Common Supersequence
 \log_2 , hand

Common Supersequence

architrophihi

architrophihi

Shortest Common Subsequence

There may be SCS

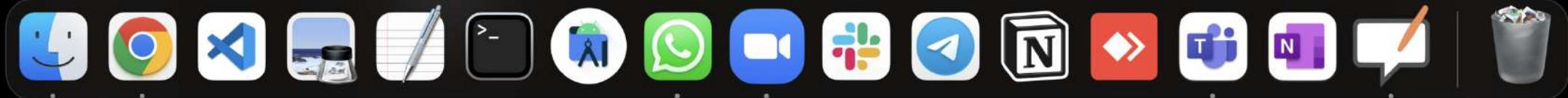
"architrophihi"

"ghi"
LCS

"rophihiarchit", "arrocochhiitnit"

"arcophitni"

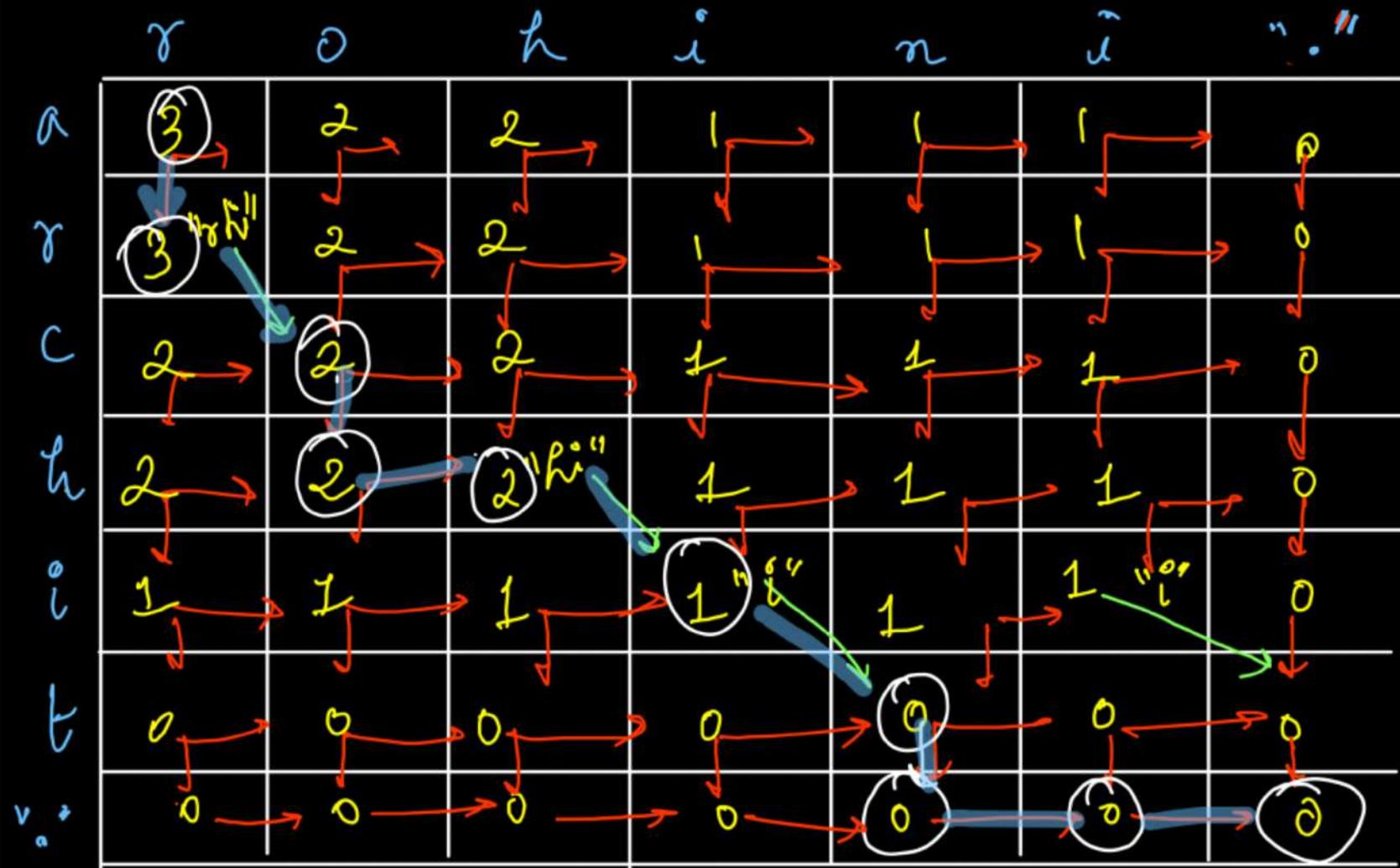
$SCS = n + m - LCS$



LCS table

Backtrack(SCS)

yes → In both cells,
 add character
 in SCS.
 "in SCS".



"arcohitni"

← ↓ →

SCS

```

String scs = "";

public void backtrack(int i, int j, String s1, String s2, int[][] dp, String ans){
    if(i == s1.length() && j == s2.length()){
        scs = ans;
        return;
    }
}

char ch1 = (i < s1.length()) ? s1.charAt(i) : 'A';
char ch2 = (j < s2.length()) ? s2.charAt(j) : 'B';

if(ch1 == ch2){
    // Yes - Diagonal
    backtrack(i + 1, j + 1, s1, s2, dp, ans + ch1);
}
else if(i + 1 <= s1.length() && dp[i][j] == dp[i + 1][j]){
    // No - Down
    backtrack(i + 1, j, s1, s2, dp, ans + ch1); DP
}
else {
    // No - Right
    backtrack(i, j + 1, s1, s2, dp, ans + ch2); DP
}
}

```

```

public String shortestCommonSupersequence(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];

    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];

            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    // If Length of SCS was asked, SCS = N + M - LCS
    backtrack(0, 0, s1, s2, dp, "");
    return scs;
}

```

last row & last col → base case → bottom right corner

DP
 $S(CS) \rightarrow O(m * n)$
 $+ (m + n))$
 ↗
 Backtracking
 (Any one SCS)

in terms of index & Distinct Subsequences

"b'a'b'g'b'ag'", "bag"
 s_1 s_2

Find all occurrences
of s_2 in s_1
in form of subsequences

$LCS(i, j) \rightarrow LCS(i+1, j+1)$

$s(i) = s(j) \rightarrow$

$s(i) \neq s(j) \rightarrow$

$LCS(i, j+1)$

$LCS(i+1, j)$

babgbag $\rightarrow b'a'g'$
babgbag $\rightarrow b'a'g''$
babgbag $\rightarrow b'a''g'''$
babgbag $\rightarrow b''a''g''$
babgbag $\rightarrow b'''a'''g'''$

count 5



1. Substring(i)
↓ count occurrences
2. Substring(j)

```

public int memo(int i, int j, String s1, String s2, int[][] dp){
    if(j == s2.length()) return 1; // Required String (s2) is completely found
    if(i == s1.length()) return 0; // Required is still left, actual string is empty

    if(dp[i][j] != -1) return dp[i][j];

    char actual = s1.charAt(i);
    char required = s2.charAt(j);

    if(actual == required) // both yes and no calls
        return dp[i][j] = (memo(i + 1, j + 1, s1, s2, dp)
            + memo(i + 1, j, s1, s2, dp));

    // if actual != required, only no call
    return dp[i][j] = memo(i + 1, j, s1, s2, dp);
}

public int numDistinct(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=0; i<dp.length; i++)
        for(int j=0; j<dp[0].length; j++)
            dp[i][j] = -1;

    return memo(0, 0, s1, s2, dp);
}

```

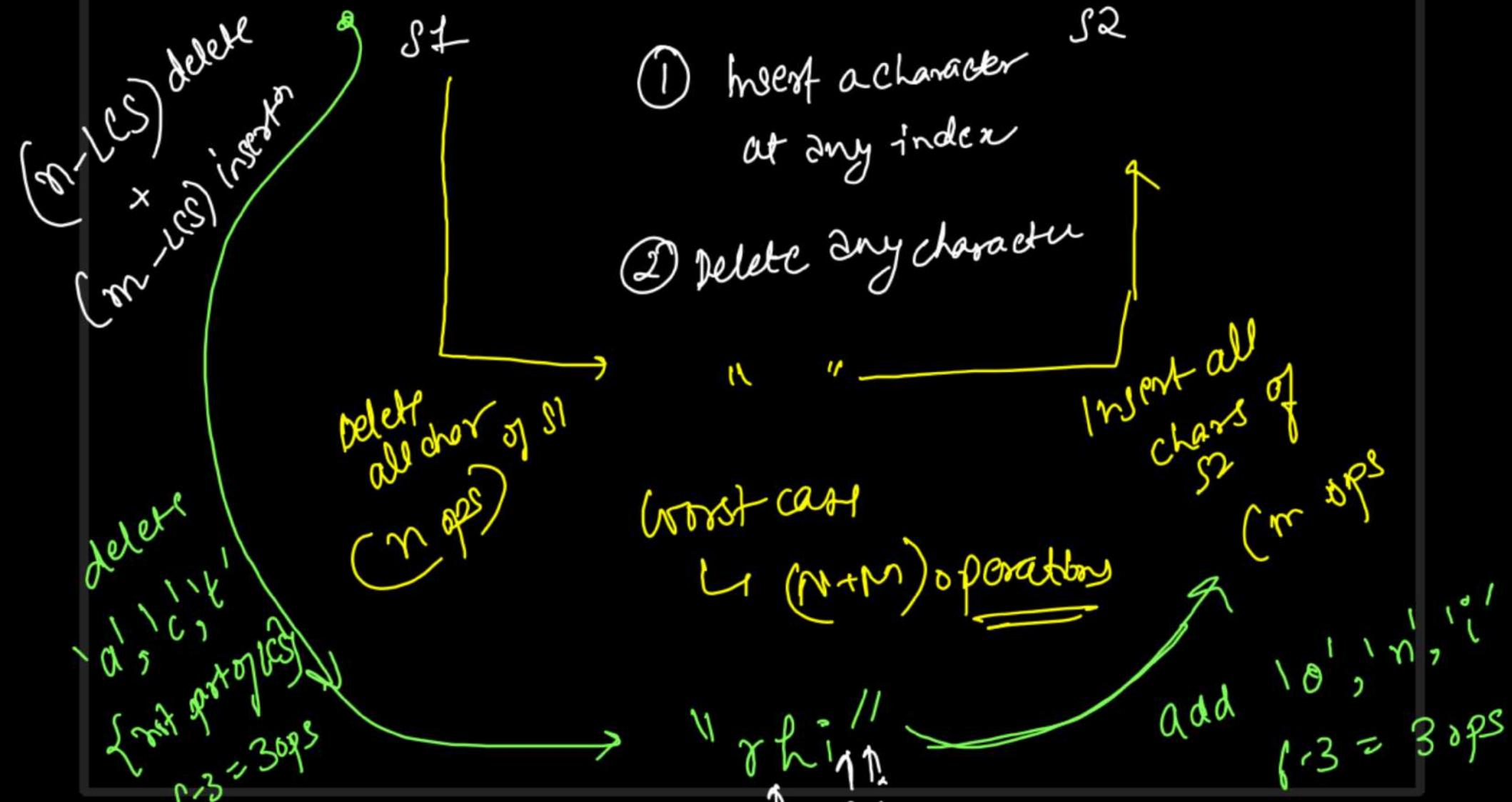
Distinct
subsequences

$O(N \times M)$ time
 $O(N \times M)$ space



Min Steps to Convert S1 to S2

"aachit" ————— Convert ————— "johni"



```

public int minOperations(String s1, String s2)
{
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=s1.length()-1; i>=0; i--){
        for(int j=s2.length()-1; j>=0; j--){
            char ch1 = s1.charAt(i);
            char ch2 = s2.charAt(j);

            if(ch1 == ch2)
                dp[i][j] = 1 + dp[i + 1][j + 1];

            else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
        }
    }

    int lcs = dp[0][0];
    int deletions = (s1.length() - lcs);
    int insertions = (s2.length() - lcs);
    return deletions + insertions;
}

```

~~lcs~~
 Time $\rightarrow O(N^2 M)$
 Space $\rightarrow \alpha N^2 M$

Expression Matching

Lecture ②

9:45 pm to 11:45 pm

2 June 2022, Thursday

- Wildcard Matching LC 44
- Regular Expression Matching LC 10
- Edit Distance LC 72
- Interleaving String LC 97
- Min operations to make
Strings Equal
 - QF9
 - LC 583
 - LC 712

wildcard
matching

"architagg"
String

$i = n \& j = m$

return true

$i == n \text{ || } j == m$

return false

? → single character

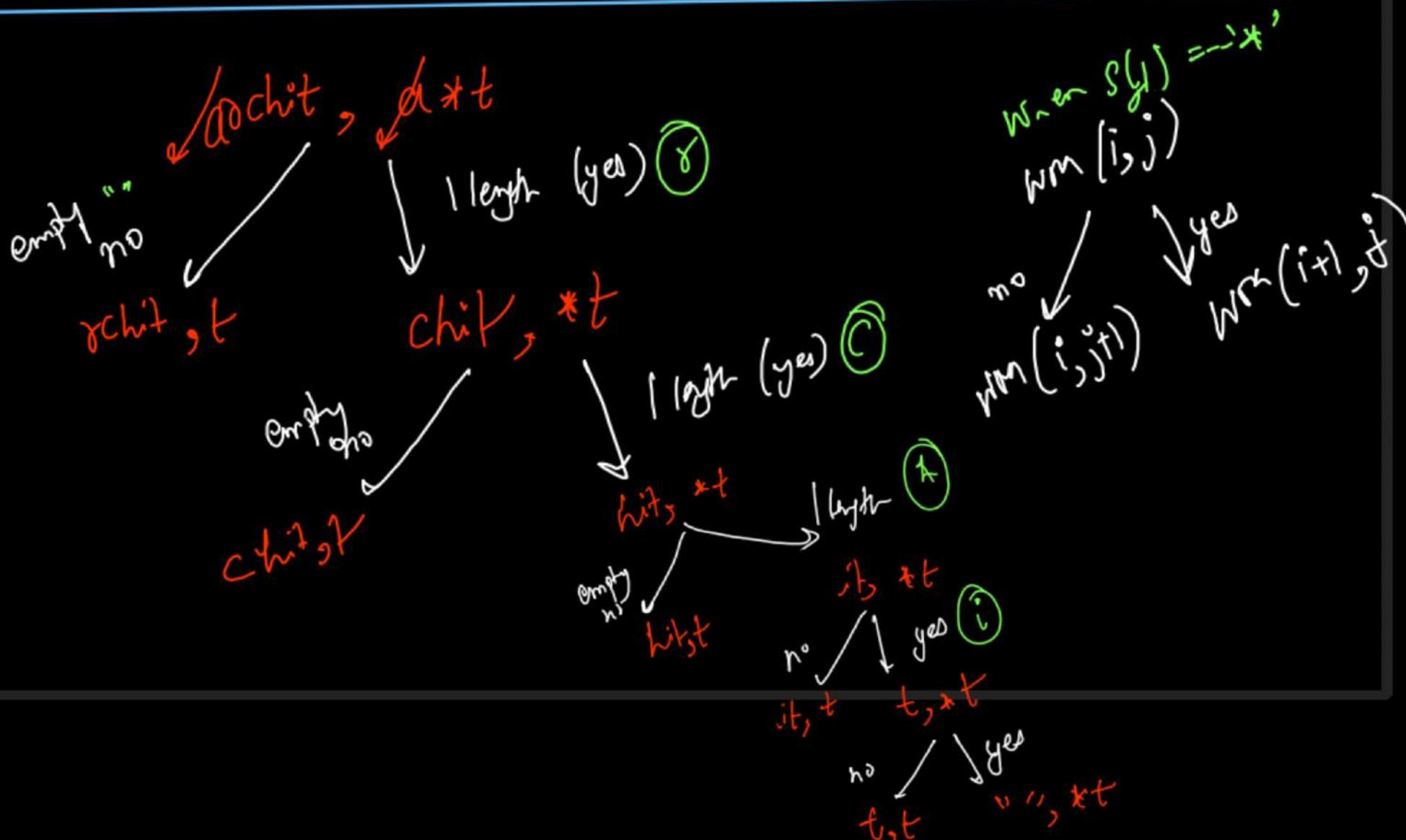
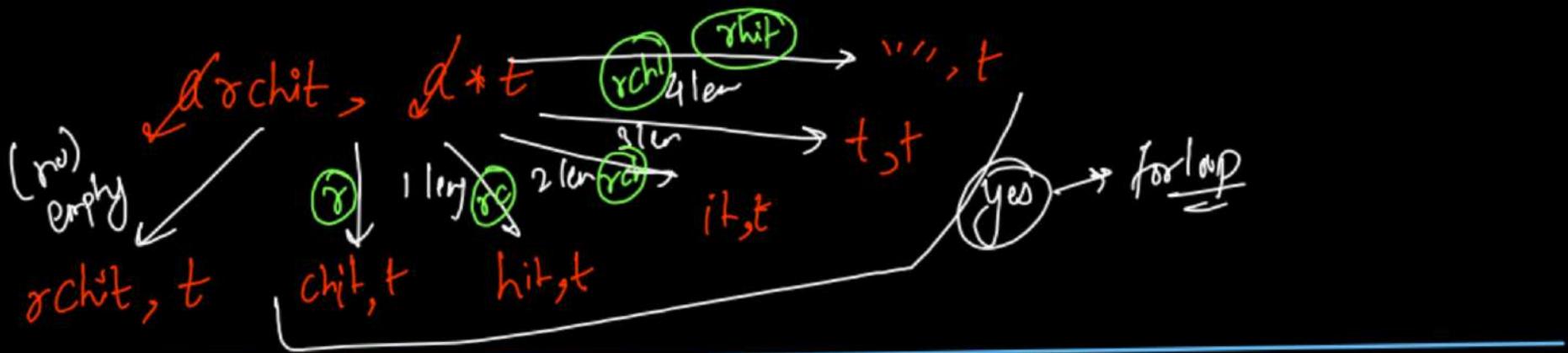
* → contiguous (diff)
group of characters

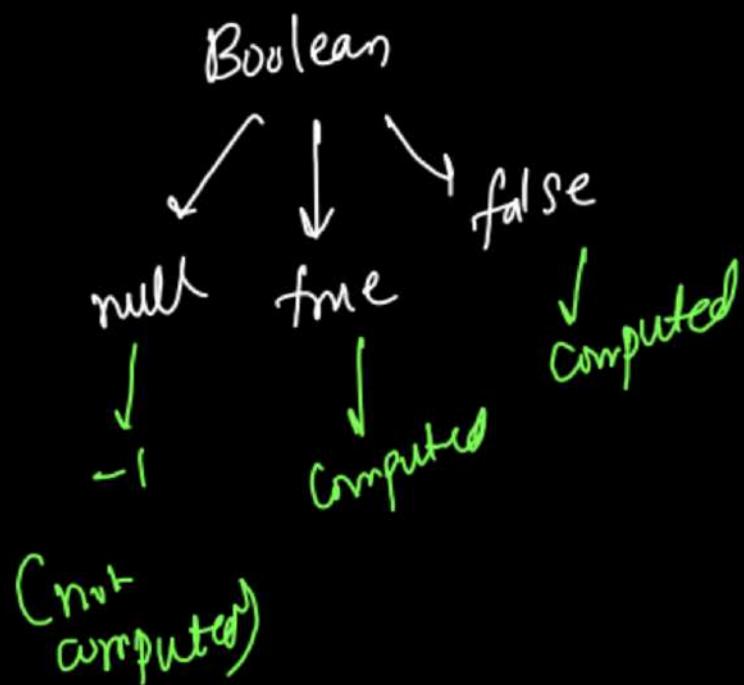
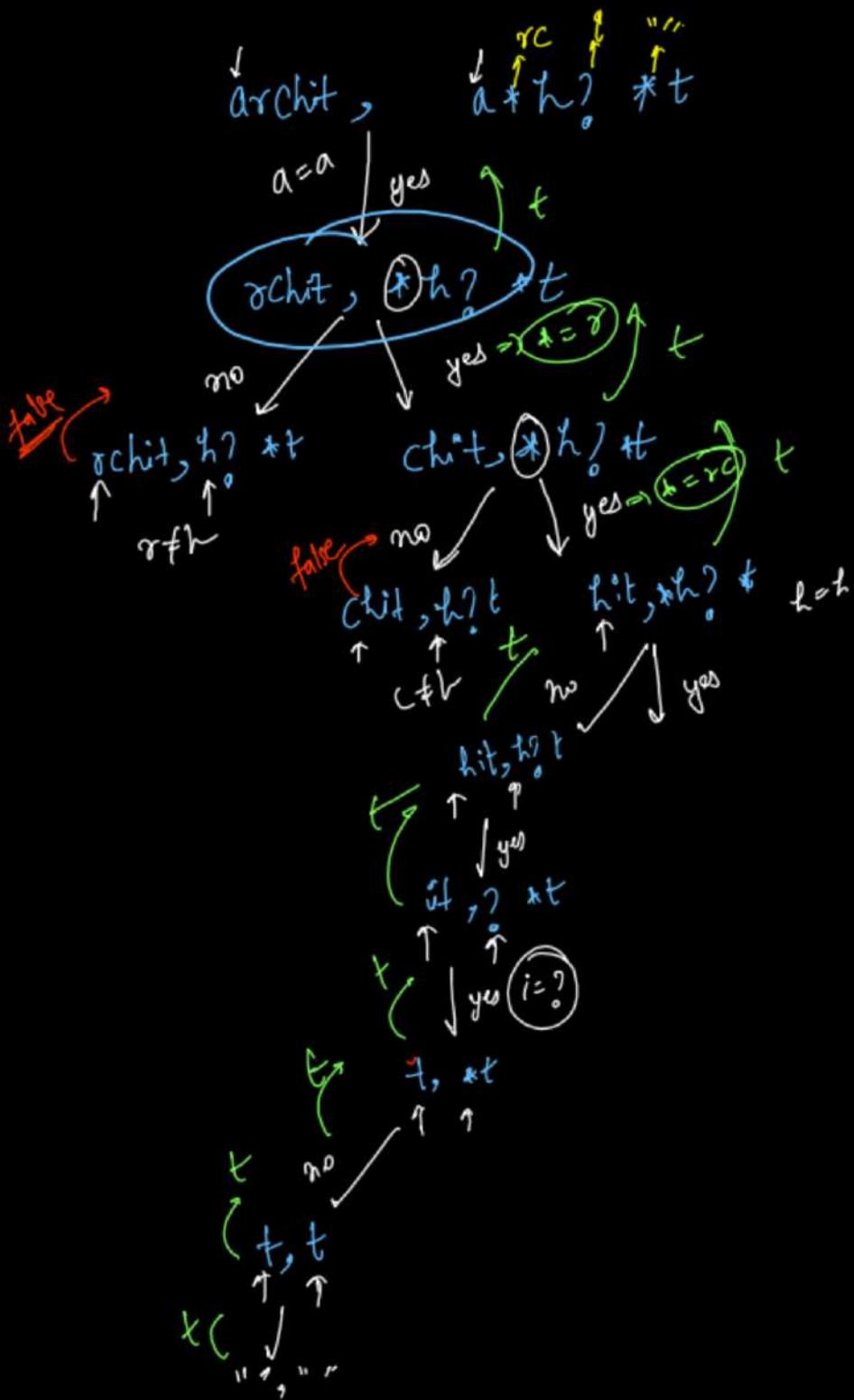
0, 1, 2, ..., length

" a^{*} h? t[?] a[?] g "

Pattern

$s[i] == s[j]$	$s[j] == ?$	$s[i] == *$	$s[i] != s[j]$
$WM(i, j)$ \downarrow Yes $WM(i+1, j+1)$ $a = a$	$WM(i, j)$ \downarrow $? \rightarrow s[i]$ $WM(i+1, j+1)$ $ar == a?$	$WM(i, j)$ \downarrow Yes $WM(i, j+1)$ $WM(i+1, j)$ $(\text{no}) \quad \underline{\text{or}} \quad (\text{yes})$	return false $ar != a$





① $i=n$, $j=m$
" ", " " → true

② " ", "a" → false
"*", "a", "*", "ad"

③ "a", " " → false

④ " ", " **" → true

corner
case

" ", "a"

" ", " **"

" ", " ***"

```

public boolean memo(int i, int j, String s1, String s2, Boolean[][] dp){
    if(i == s1.length() && j == s2.length()){
        return true;
    }

    if(i < s1.length() && j == s2.length()){
        // first string is not empty, second string is empty
        return false;
    }

    if(i == s1.length() && j < s2.length()){
        // first string is empty, second string is still left
        for(int k=j; k<s2.length(); k++){
            if(s2.charAt(k) != '*')
                return false;
        }
        return true;
    }
}

```

```

if(dp[i][j] != null) return dp[i][j];

char ch1 = s1.charAt(i);
char ch2 = s2.charAt(j);

if(ch1 == ch2 || ch2 == '?'){
    return dp[i][j] = memo(i + 1, j + 1, s1, s2, dp);
}

if(ch2 == '*'){
    boolean no = memo(i, j + 1, s1, s2, dp);
    if(no == true) return dp[i][j] = true;

    boolean yes = memo(i + 1, j, s1, s2, dp);
    return dp[i][j] = yes;
}

// unequal characters
return dp[i][j] = false;
}

```

```

public boolean isMatch(String s1, String s2) {
    Boolean[][] dp = new Boolean[s1.length() + 1][s2.length() + 1];
    return memo(0, 0, s1, s2, dp);
}

```

less variation

Time $\rightarrow O(N \times M)$

Space $\rightarrow O(N \times M)$

Edit Distance

① delete 'a'
archit
rchit

② replace 'c' with 'g'

gohit

③ replace t with n

gohin

④ Insert 'i'
johini

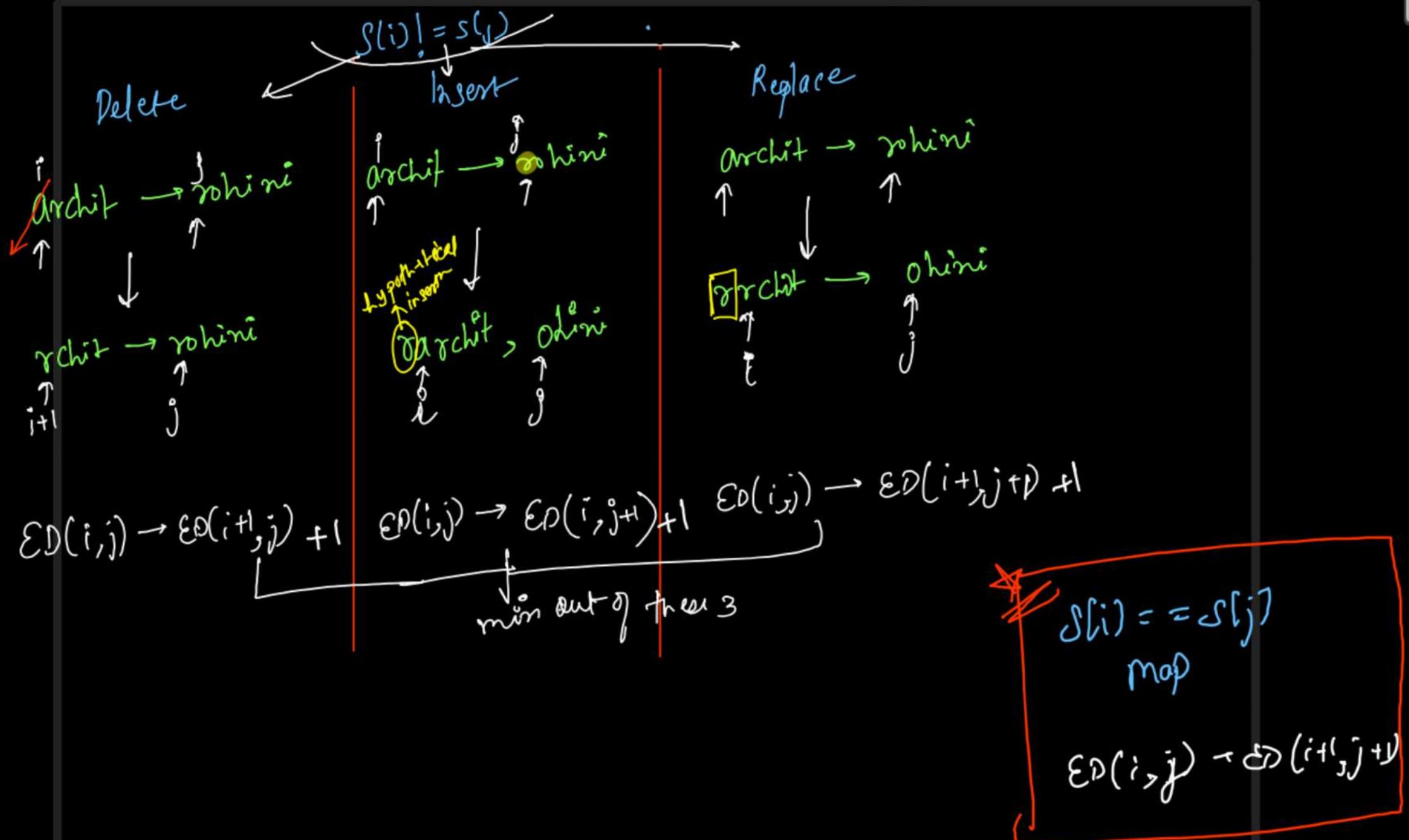
"archit" → "go*hi*n*i*"

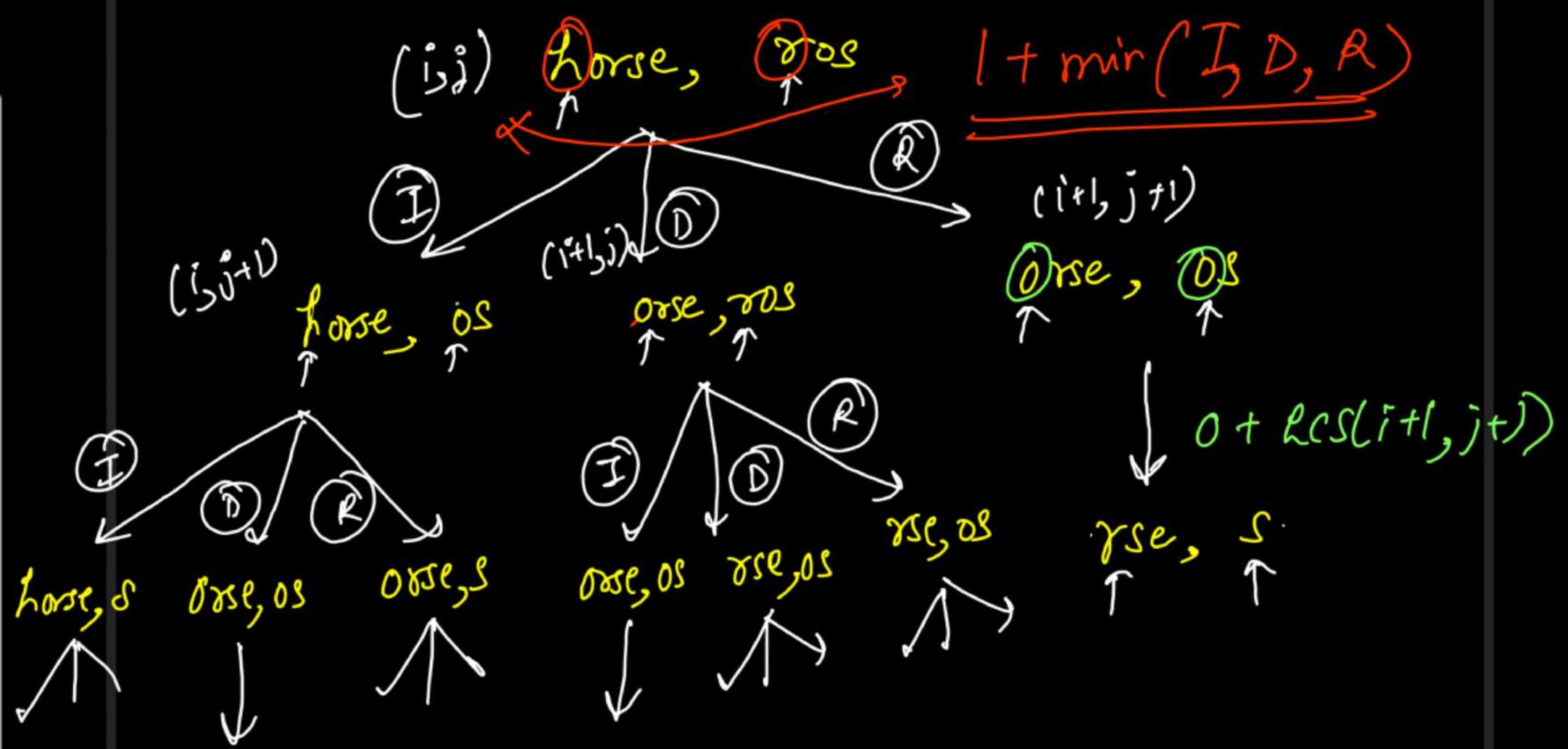
① delete any char

② Insert any char at any char

③ Replace any char by any char

This may even reduce the no. of operations





Base cases

① "", "" → 0 operations

② "abc", "" → n operations (delete all)

③ "", "pqrs" → m operations (insert all)

```

public int memo(int i, int j, String s1, String s2, int[][] dp){
    if(i == s1.length() && j == s2.length()) return 0;
    if(i == s1.length()) return s2.length() - j; // First String Empty, Insert All to make Second
    if(j == s2.length()) return s1.length() - i; // Second String Empty, Delete All Char in First

    if(dp[i][j] != -1) return dp[i][j];

    char ch1 = s1.charAt(i);
    char ch2 = s2.charAt(j);

    if(ch1 == ch2)
        return dp[i][j] = memo(i + 1, j + 1, s1, s2, dp);
    // no operation, since char is same

    int delete = memo(i + 1, j, s1, s2, dp);
    int insert = memo(i, j + 1, s1, s2, dp);
    int replace = memo(i + 1, j + 1, s1, s2, dp);
    return dp[i][j] = 1 + Math.min(delete, Math.min(insert, replace));
}

```

```

public int minDistance(String s1, String s2) {
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=0; i<dp.length; i++){
        for(int j=0; j<dp[0].length; j++){
            dp[i][j] = -1;
        }
    }

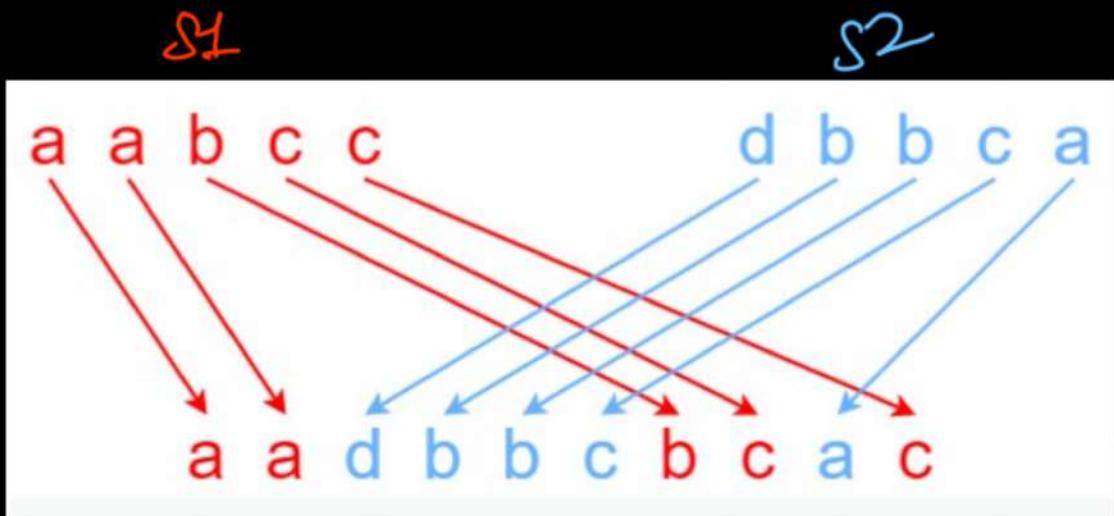
    return memo(0, 0, s1, s2, dp);
}

```

Time $\rightarrow O(N * M)$

Space $\rightarrow O(N * M)$

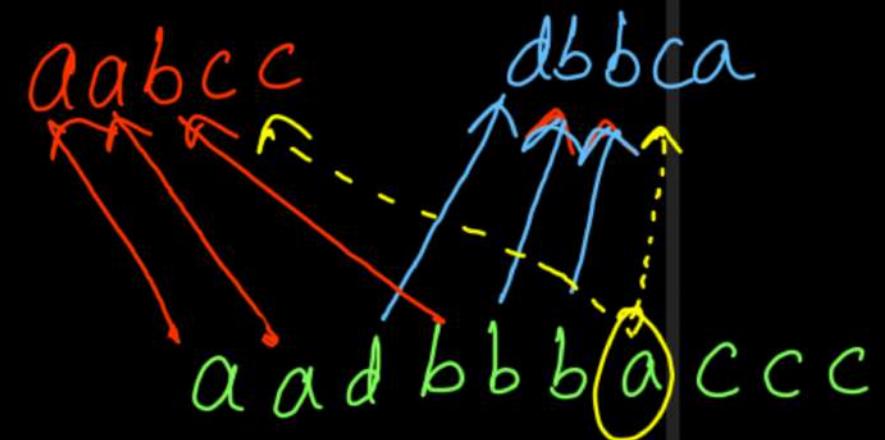
Interleaving String



$$S_3 = n+m$$

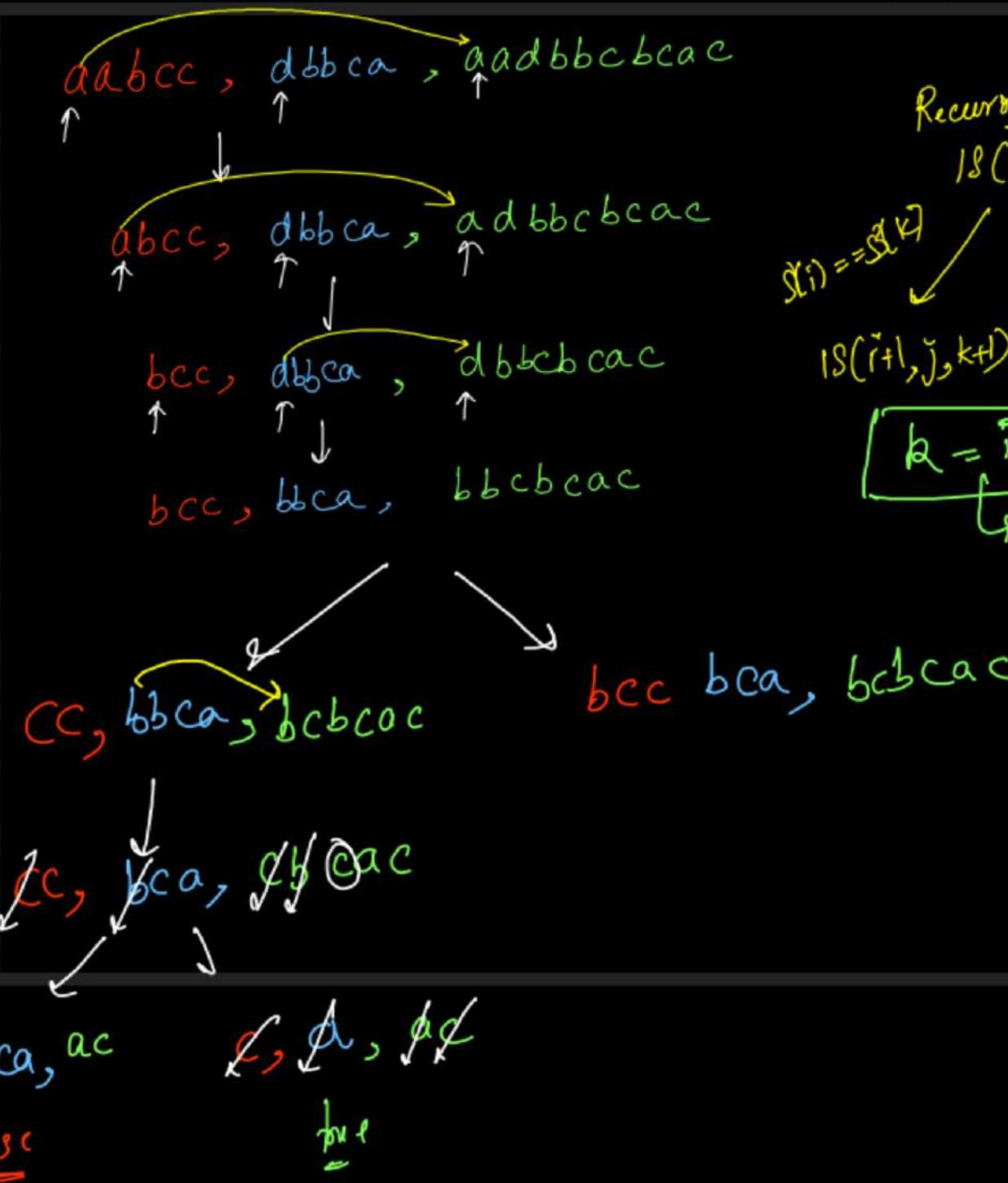
True

eg 1



eg 2

False



```

public boolean memo(int i, int j, int k, String s1, String s2, String s3, Boolean[][] dp){
    // Positive Base Case
    if(k == s3.length()){
        // Result is empty
        return true;
    }

    if(dp[i][j] != null) return dp[i][j];

    char ch1 = (i < s1.length()) ? s1.charAt(i) : 'A';
    char ch2 = (j < s2.length()) ? s2.charAt(j) : 'B';

    if(s3.charAt(k) == ch1 && memo(i + 1, j, k + 1, s1, s2, s3, dp) == true){
        return dp[i][j] = true;
    }

    if(s3.charAt(k) == ch2 && memo(i, j + 1, k + 1, s1, s2, s3, dp) == true){
        return dp[i][j] = true;
    }

    return dp[i][j] = false;
}

```

```

public boolean isInterleave(String s1, String s2, String s3) {
    if(s3.length() != s1.length() + s2.length()) return false;
    Boolean[][] dp = new Boolean[s1.length() + 1][s2.length() + 1];
    return memo(0, 0, 0, s1, s2, s3, dp);
}

```

Time $\Theta(n^2)$, Space $\Theta(n^2)$

$$\underline{k = i+j}$$

k is not a
DP State

false ($2+1 \neq 4$)

"ar" + "c" \neq "arch"

"as" + "ch" \neq "asc"

false ($2+2 \neq 3$)

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 → Using Dynamic Programming

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

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

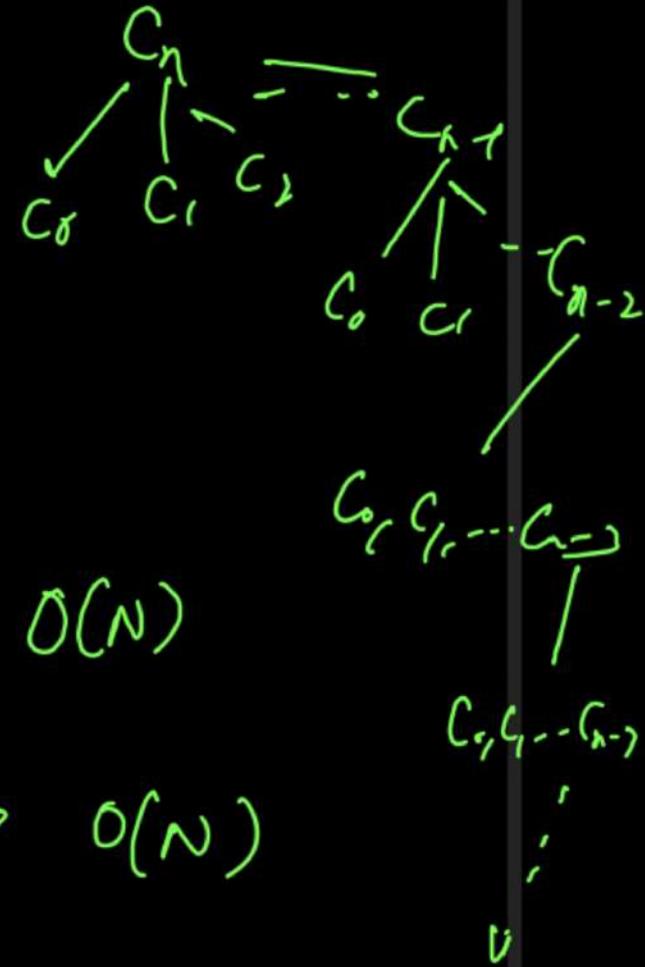
WDP

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	C_1	C_2	C_3	C_4	C_5	C_6

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

~~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~~

```

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 → $O(N^2)$

Space → $O(N)$ 1D DP

Recursion tree

Space → $O(N)$ 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 → $O(N^2)$

Space → $O(N)$ 1D DP

NM catalan No Using Binomial coeff

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

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

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

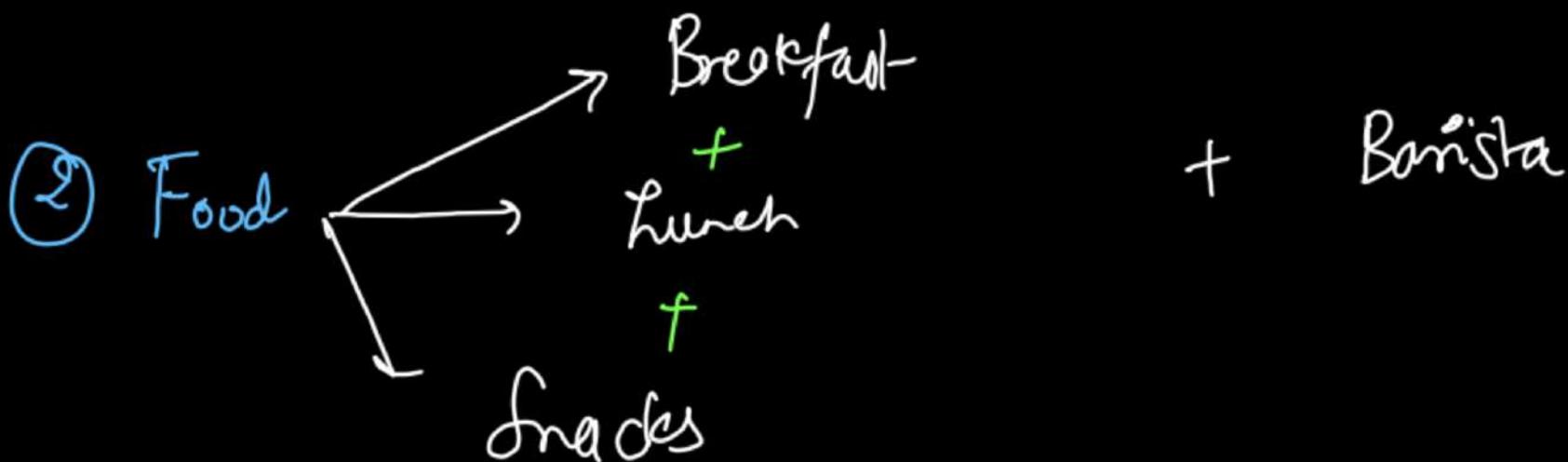
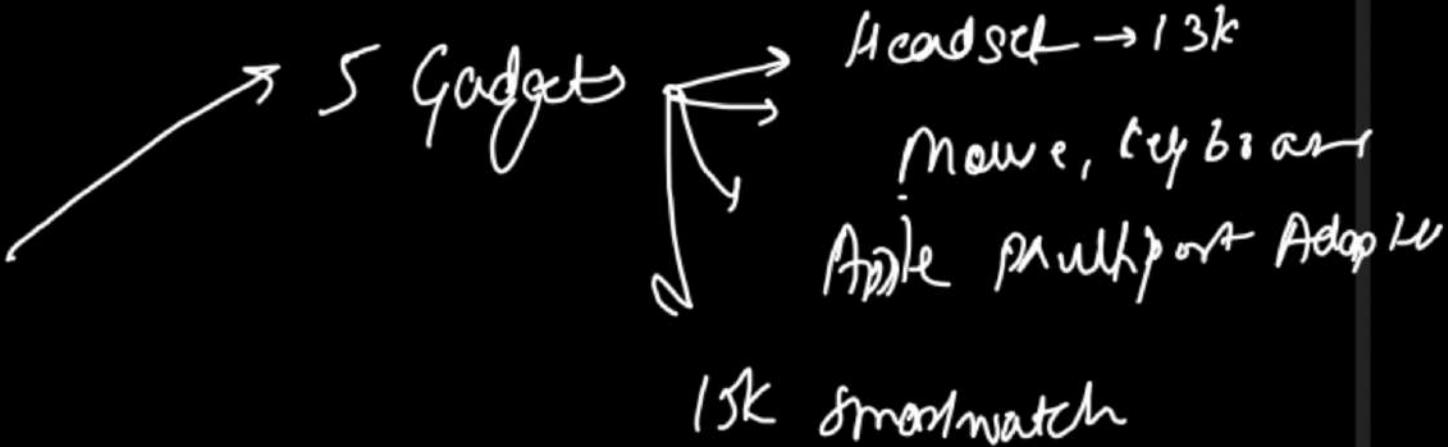
$$f(1) = \frac{^2 C_1}{2} = \frac{2!}{1! 1!} = 1$$

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

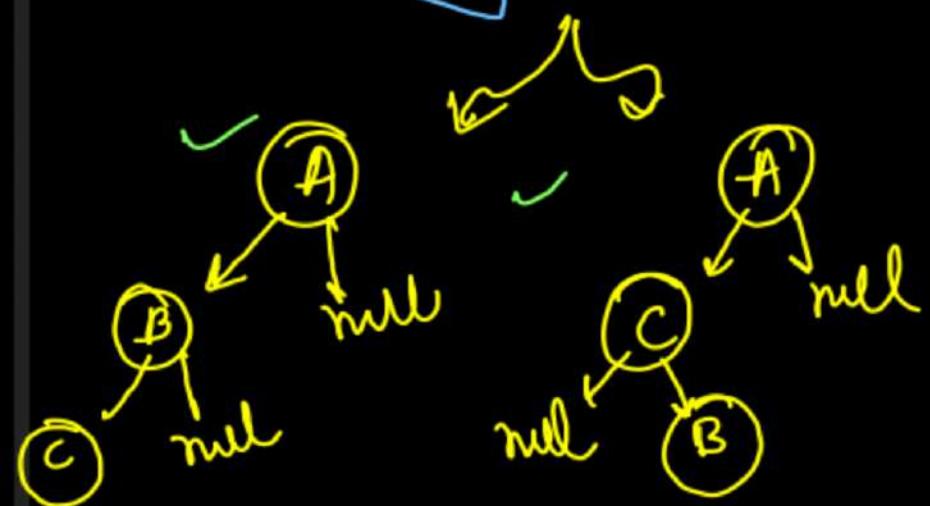
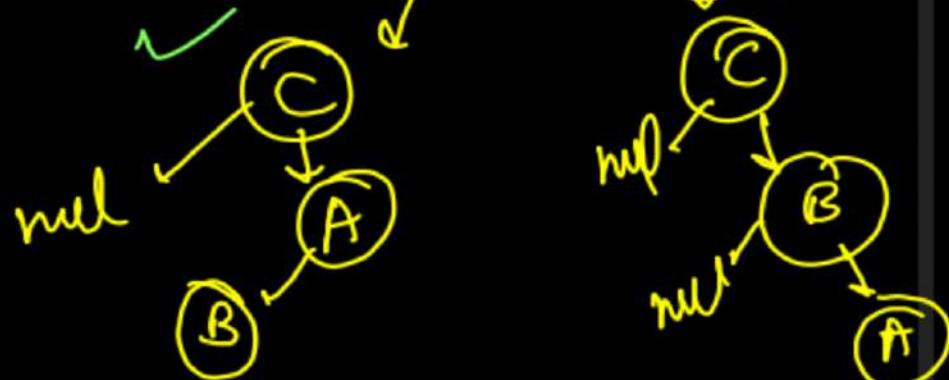
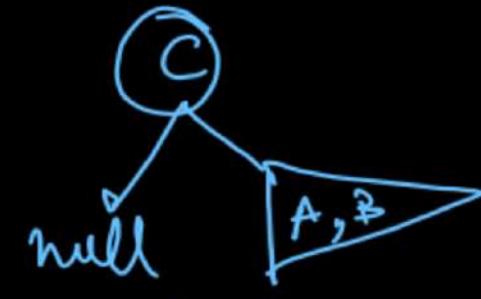
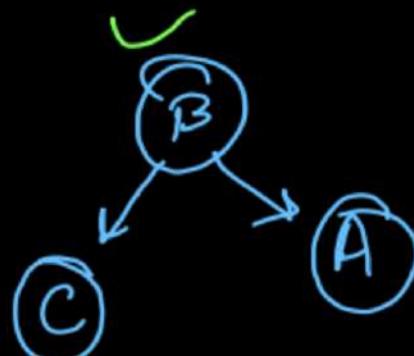
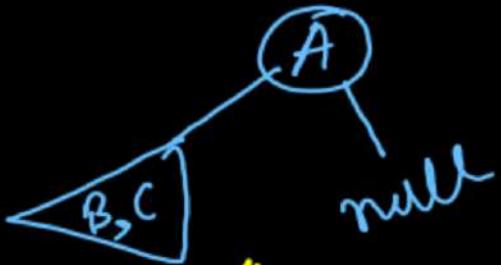
②

A > B

B → A

$N=3$

$20 \quad 15 \quad 10$
 $\underline{A > B > C}$

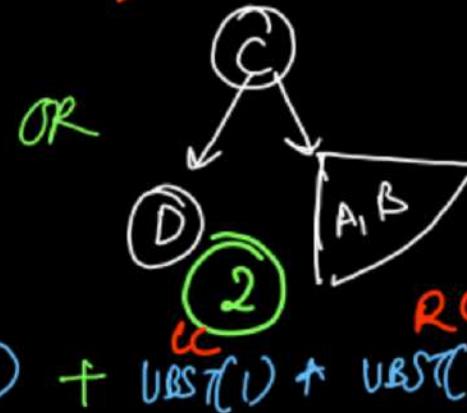
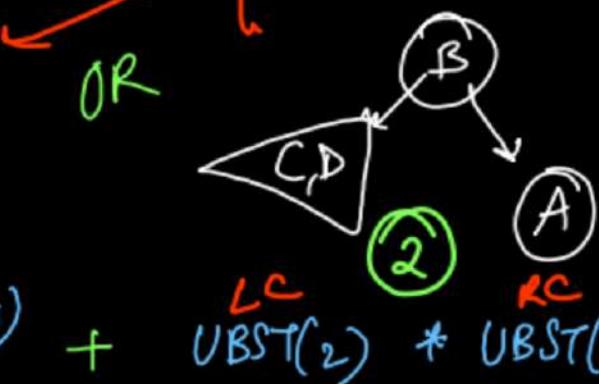
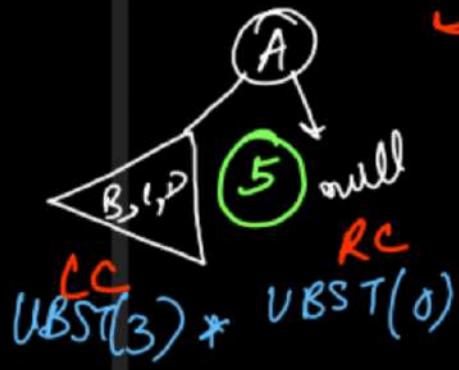


Unique $BST(n) = UBST(n)$ with k_1 as root + k_2 as root
+ k_3 as root + ... + k_n as root

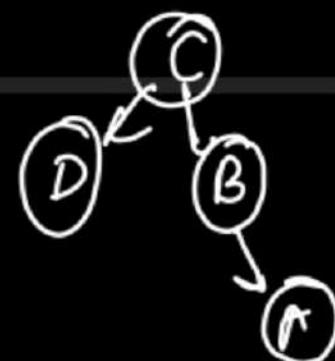
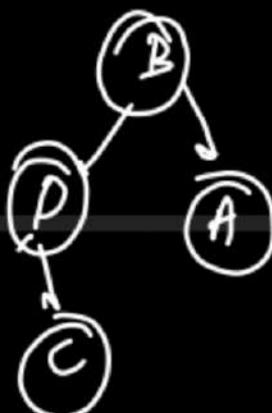
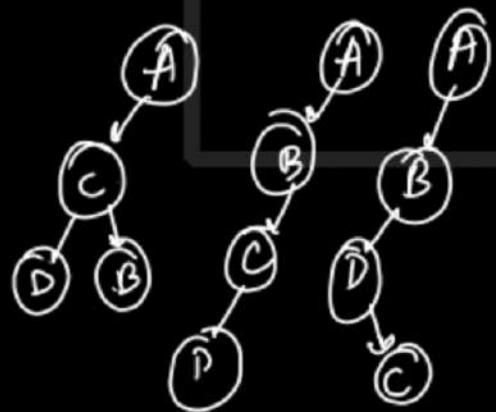
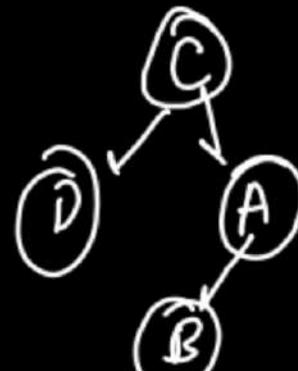
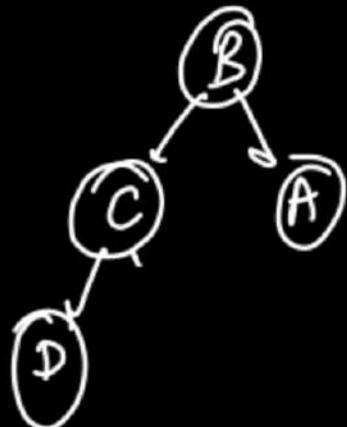
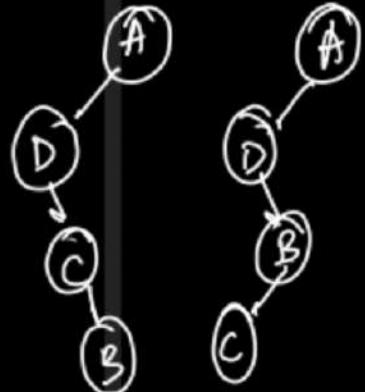
$N=4$

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

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



$$LC * RC$$



② Count Balanced Parentheses

$N \leftarrow$ opening brackets
closing brackets } pairs

$N=0$,

" $N=2$ "

$N=3$

①

" $()()$ "

②

$N=1$

①

$\textcircled{1} \textcircled{0}$ ↘ $\textcircled{0}, \textcircled{1}$
 $(())$ $()()$

$(())()$

$()(())$

$()(())$

$(((())))$

$(()(()))$

⑤

$(())$

$(\textcircled{2})\textcircled{0}$

↙ ↓

$(\textcircled{1})\textcircled{0}$

$(\textcircled{1})\textcircled{1}$

$(()())$

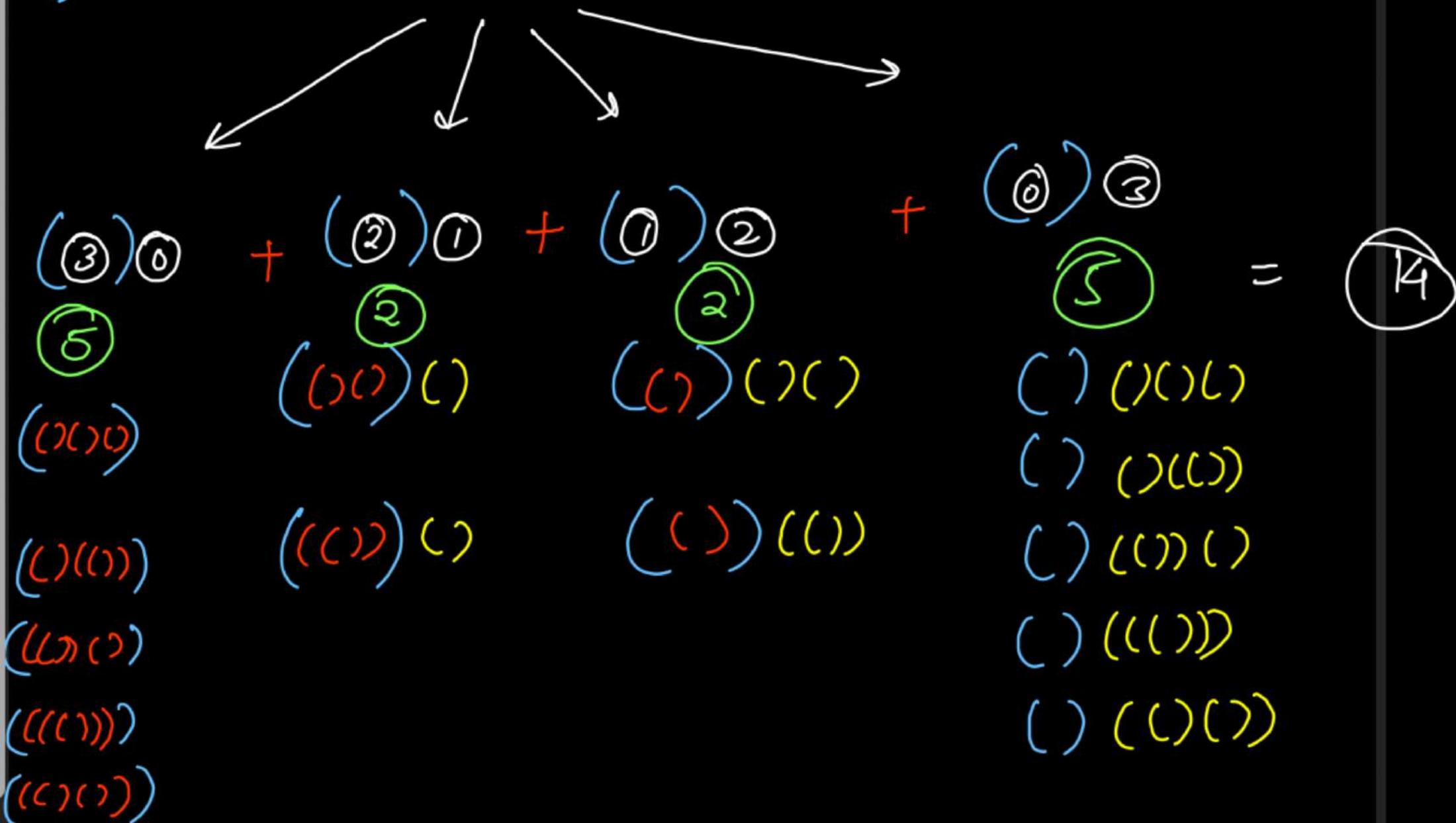
$(()())$

$(()())$

N=4

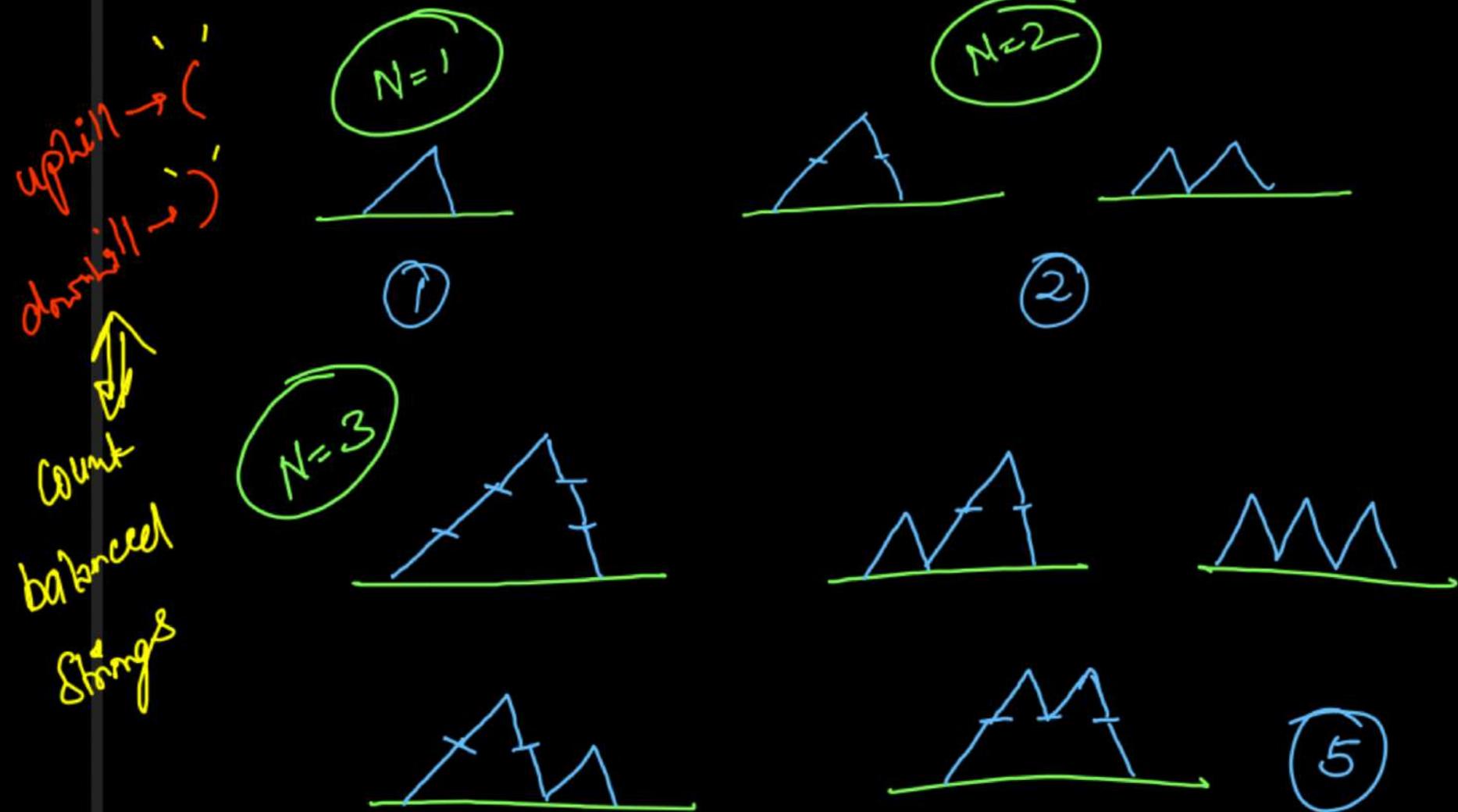
()

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



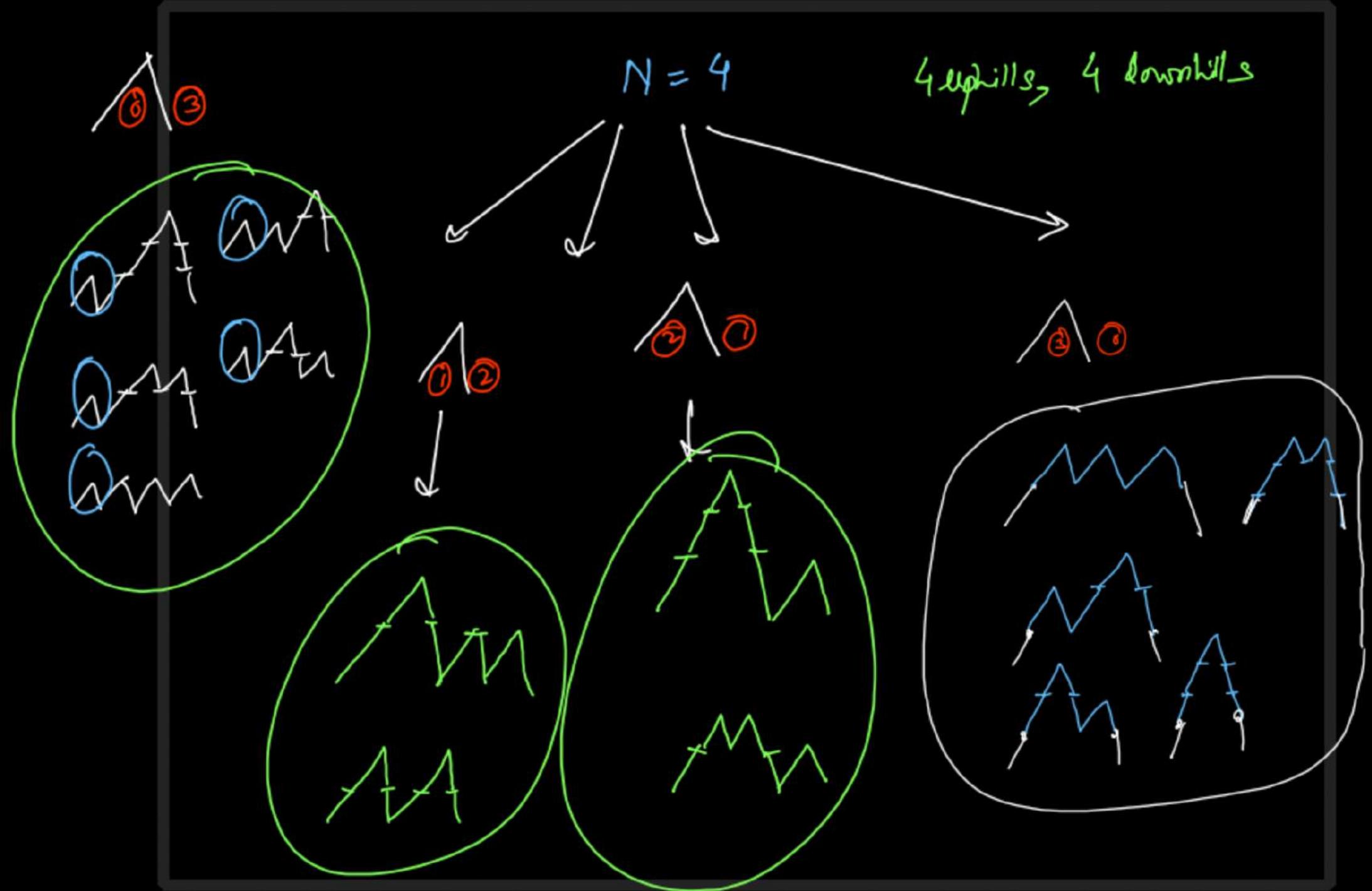
③ Count mountain ranges

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



$N = 4$

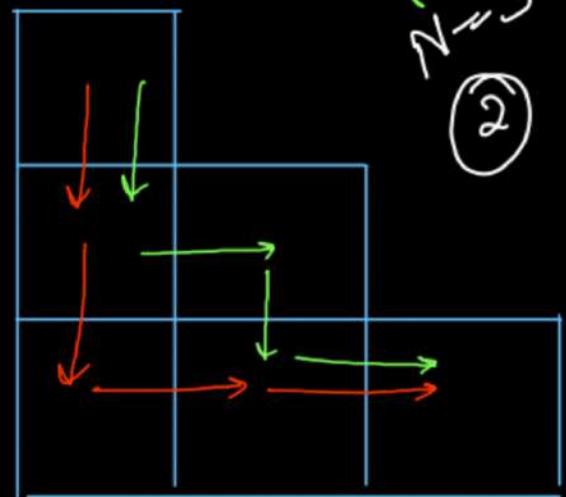
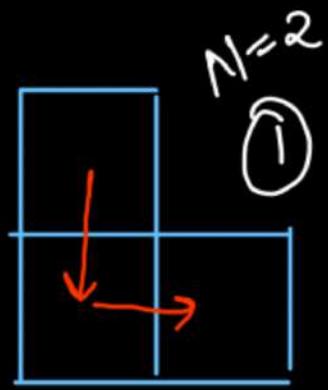
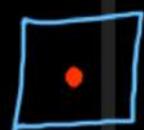
4 uphills, 4 downhills



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

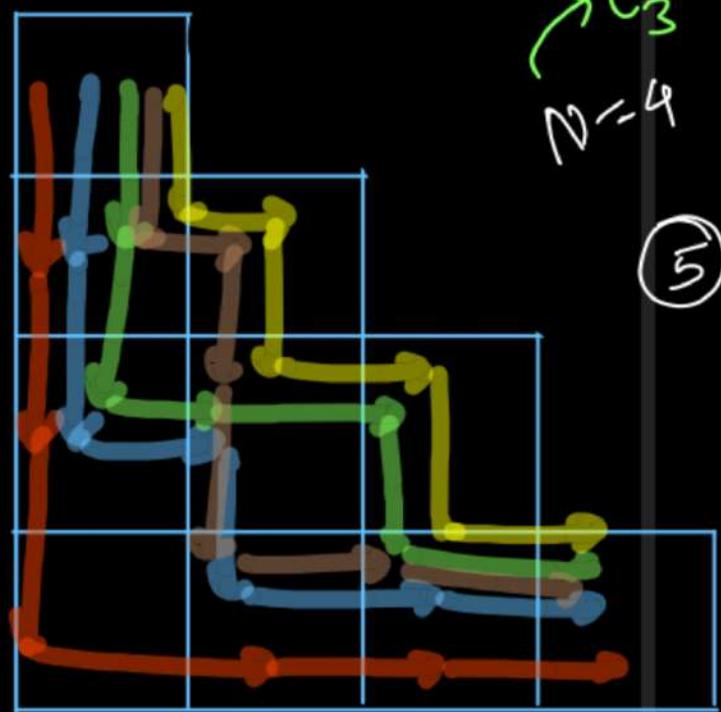
2 moves
right
down

$N=1$



C_2
 $N=3$

②



⑤

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

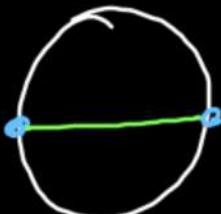
④ Count Paths in Upper-Right Triangle
or Lower-Left Triangle

⑤

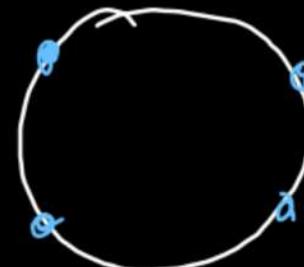
Handshakes / Non-intersecting chords

⊕

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

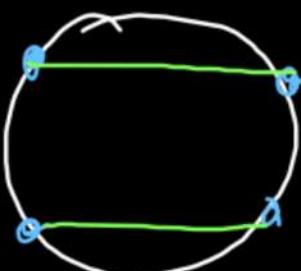


$$N = 2$$

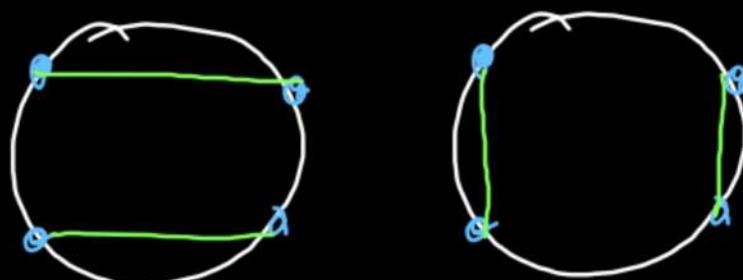


$$N = 4$$

①

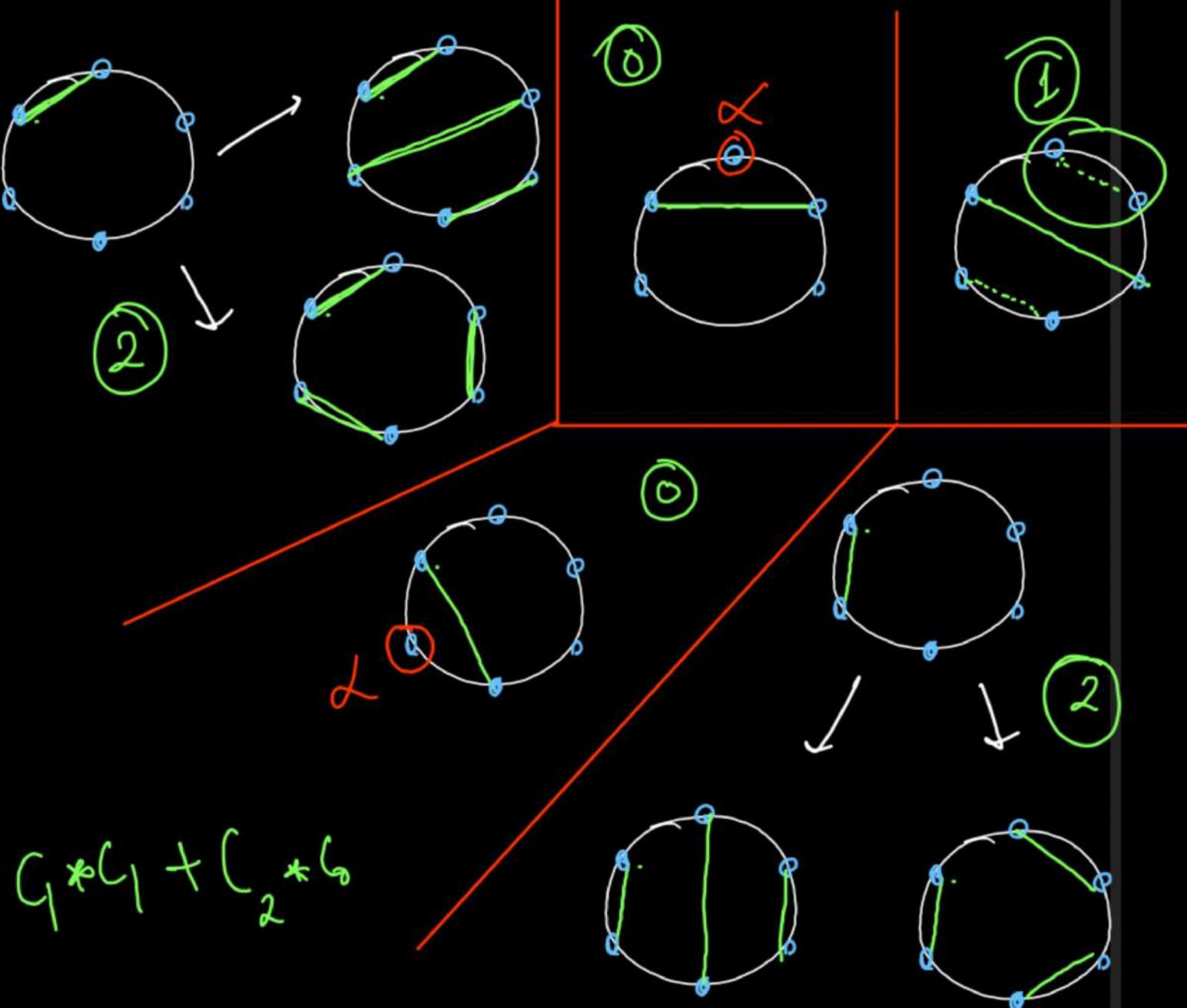
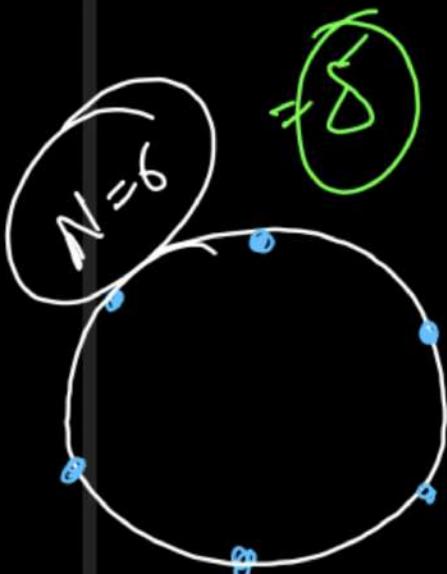


②



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

$N=0$: (1) C_0
 $N=2$: (1) C_1 , (2) C_2
 $N=4$: (2) C_2 , (3) C_3
 $N=6$: (5) C_3 , (11) C_4
 $N=8$: (14) C_4



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

Dyk words



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

$N=2$

XYXY

$N=3$

xyxy xy

XXYY

xyxy yy xy

②

xy xnyy

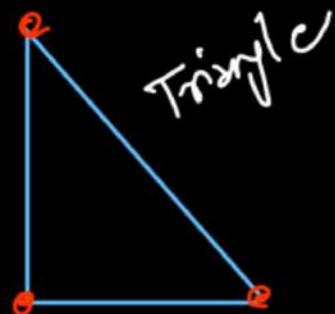
$x = 'C'$

$y = ')'$

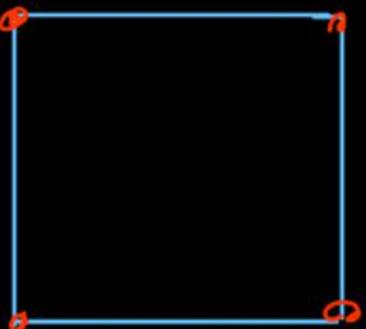
↓
count of balanced
parenthesis strings

xyxnyyy ⑤
xyxy y

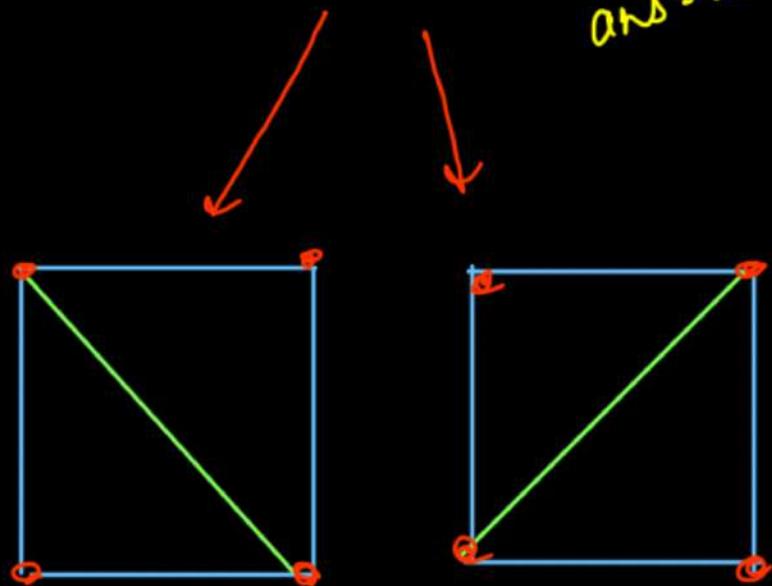
⑧ Count ways of Triangulation



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



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



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

$$N=2 \Rightarrow ① \rightarrow C_0$$

$$N=3 \Rightarrow ① \rightarrow C_1$$

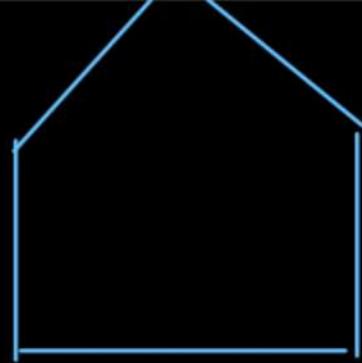
$$N=4 \Rightarrow ② \rightarrow C_2$$

$$N=5 \Rightarrow ⑤ \rightarrow C_3$$

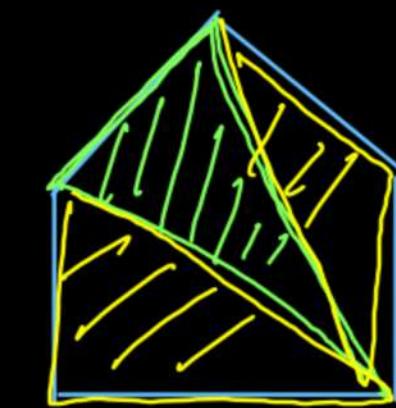
$$N=6 \Rightarrow ⑯ \rightarrow C_4$$

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

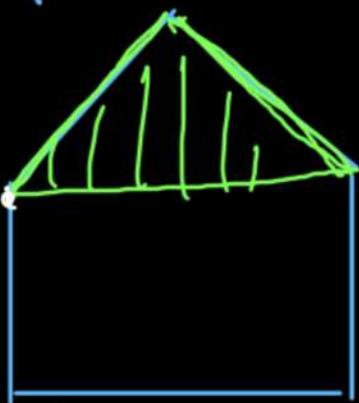
$\text{polygon}(n) = \sum \text{left}(i) * \text{light}(j)$



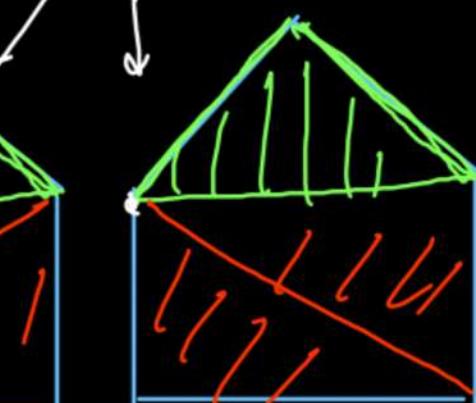
1



①



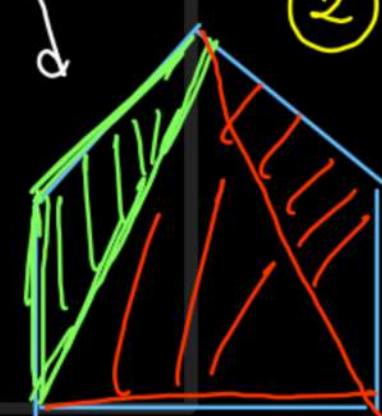
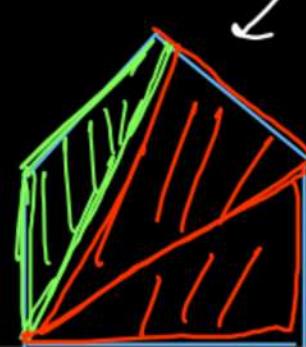
②



2



①

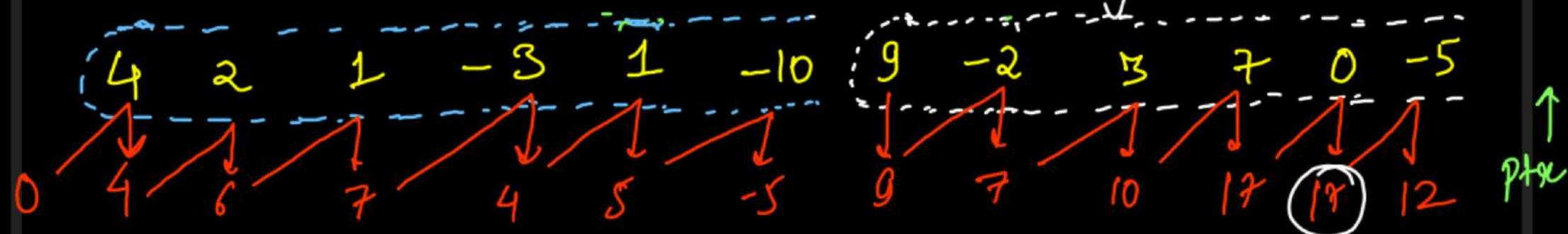


②

Kadane's Algorithm

- Maximum sum Subarray → Kadane (DP) → D & C
- Maximum sum Circular Subarray → Prefix & 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

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

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

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

$$=5-10=-8 \quad 9-2=$$

$$7+3=10+7$$

$$=18+0=18$$

$$12-5=12$$

Kadane's Algorithm

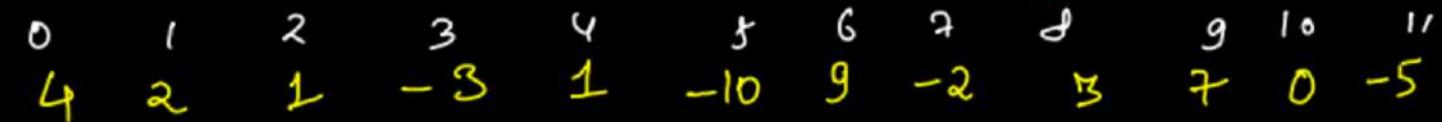
```
public int maxSubArray(int[] nums) {  
    int currSum = 0, maxSum = Integer.MIN_VALUE;  
  
    for(int i=0; i<nums.length; i++){  
  
        if(nums[i] + currSum >= nums[i])  
            currSum += nums[i];  
        else  
            currSum = nums[i];  
  
        maxSum = Math.max(maxSum, currSum);  
    }  
  
    return maxSum;  
}
```

Time $\rightarrow O(N)$

Space $\rightarrow O(1)$

Divide & Conquer

$$\{17, 12, 12\}$$



↑
left

۱۷۸

↑
mid

↑
night

{x, x, s}

$$\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 1 & -3 & 1 & -10 \\ \downarrow & & \uparrow & & & \uparrow \\ x & & m & & & y \end{array}$$

6	7	8	9	10	11
9	-2	3	7	0	-5
↓		↑ m			↑ s

$$\{1^2, 1^2, 1^2\}$$

$$\{x, y\} \leftarrow$$

A diagram consisting of two numbers, 2 and 1, arranged horizontally. An arrow points from the number 2 towards the number 1.

$$\downarrow \{1, -2, -9\}$$

$$\{10, 10, 10\}$$

$$\begin{array}{c} \downarrow \\ \{7, 7, 2\} \\ 7 \quad 0 \quad -5 \end{array}$$

2

$$\{2, 2, 2\}$$

$$\{-3, -3, -3\}$$

(9,95)

27 {2,7} 50,0,0,0

{5, 5, 8}

Return Type

→ manSum Subarray

→ manSum begin

→ manSum suffix

→ totalSum

manSum

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

PrefixSum

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

$$\text{leftTotal} + \text{rightPrefix}$$

SuffixSum

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

$$\text{rightTotal} + \text{leftSuffix}$$

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

```
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 $\rightarrow O(\log n)$

$\hookrightarrow R.C.S^*$

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

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

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

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

$$\downarrow$$

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

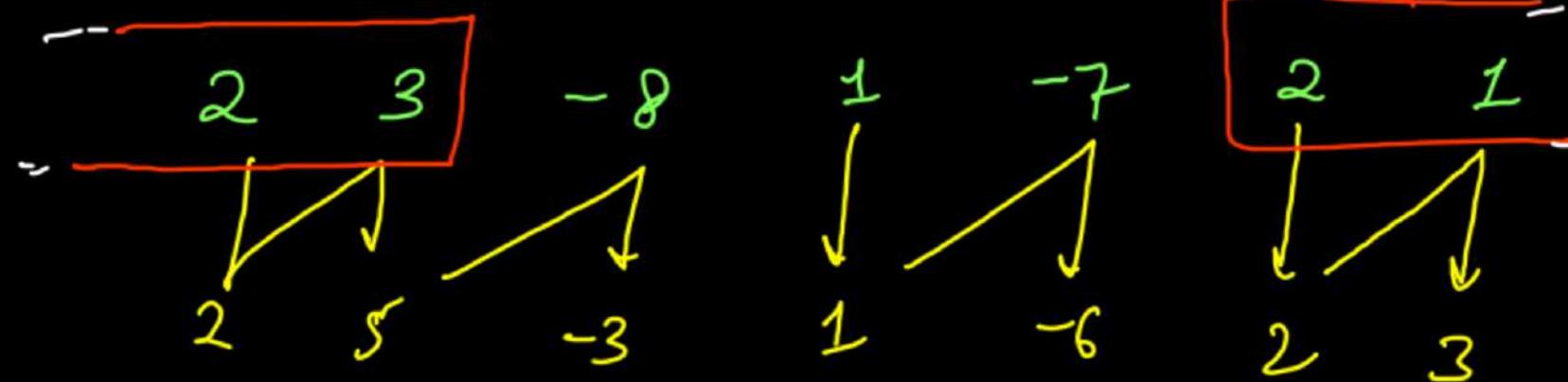
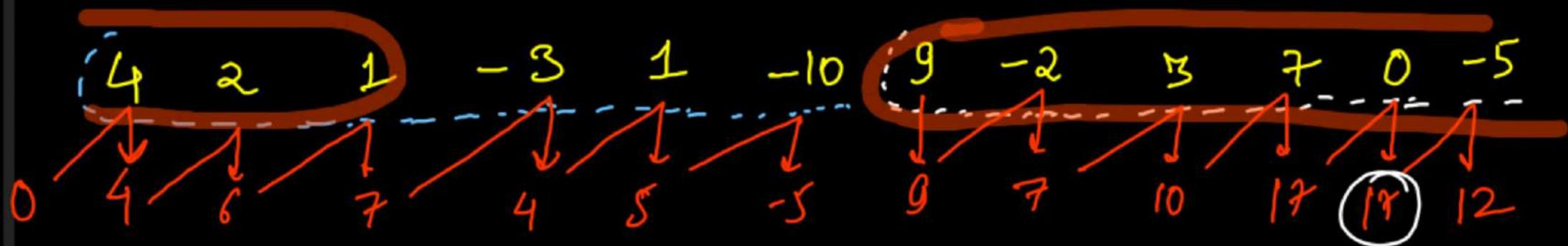
$$T(n) = n + (2 + 2^2 + 2^3 + \dots + 2^{\log_2 n})$$

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

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

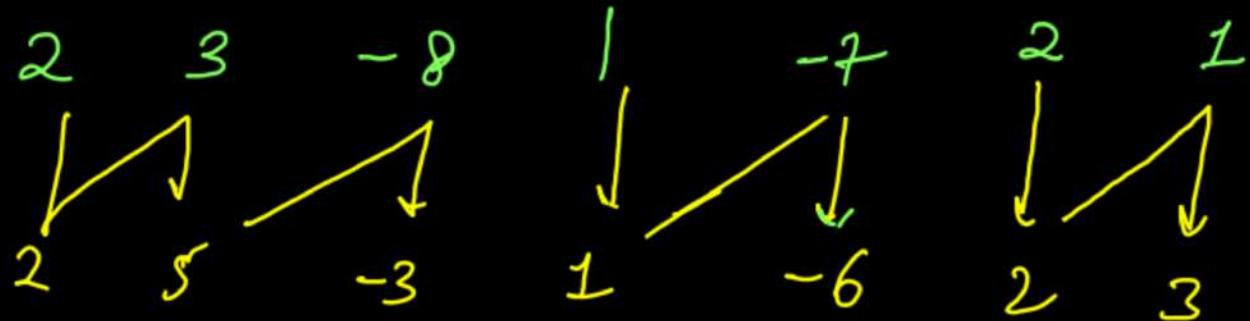
$$= \frac{3n-1}{2} \Rightarrow O(n)$$

Max sum Circular subarray

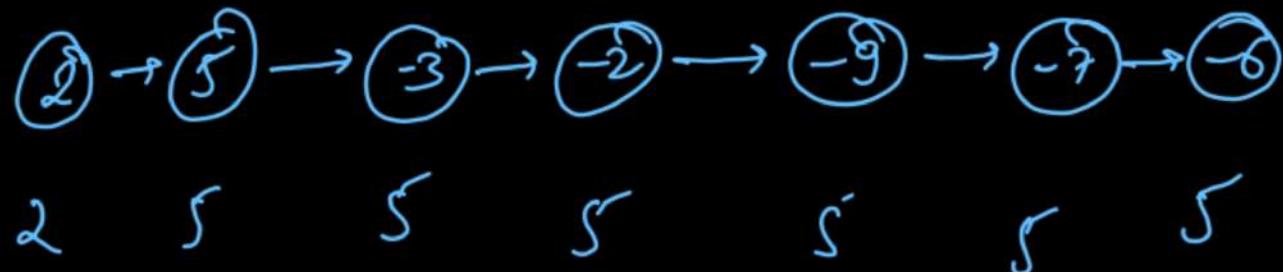


$$2+3+2+1 = 8 \text{ max sum}$$

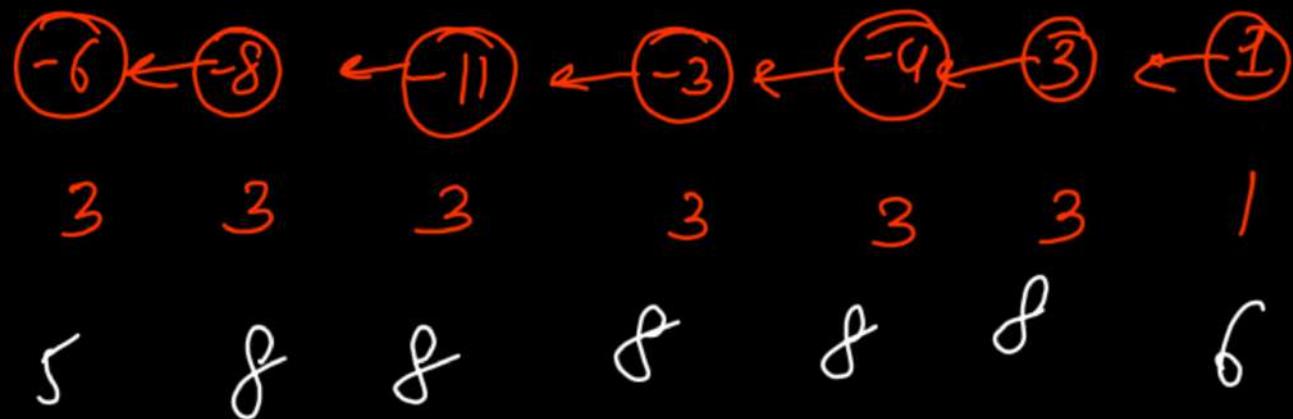
1D Kadane



Prefix



Suffix



prefix (i)

max sum prefix
ending at index
 $\leq i$

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

```
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 $\rightarrow O(n)$

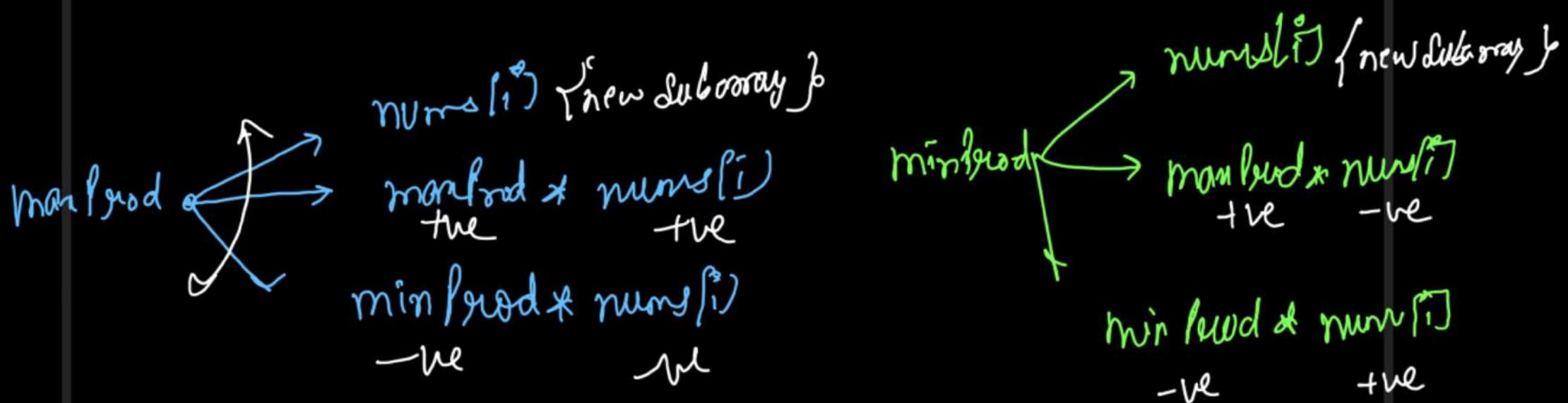
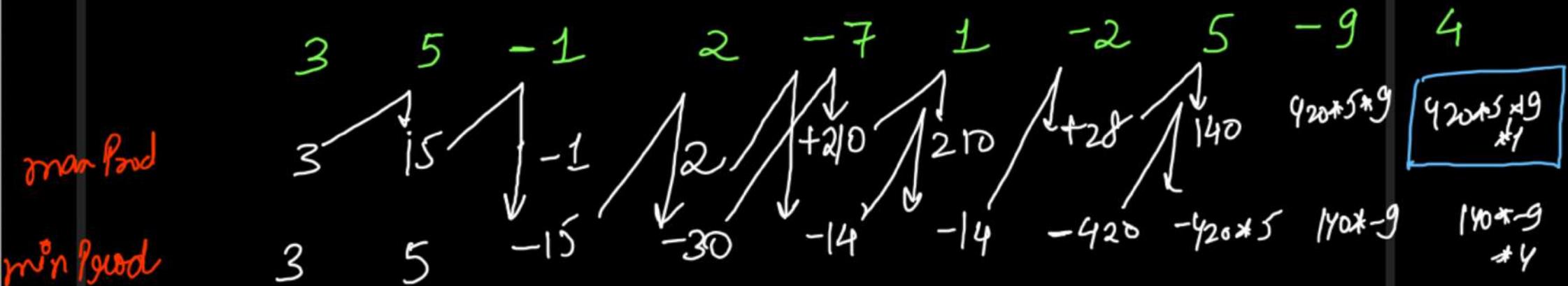
Space $\rightarrow O(1)$

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

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

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

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.

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

$k=1 \Rightarrow$ Apply Kadane's

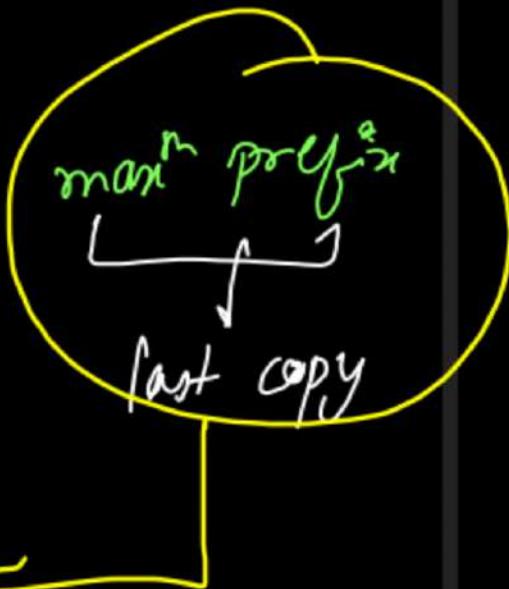
~~KT?~~

If $\text{sum} > 0$

\Rightarrow

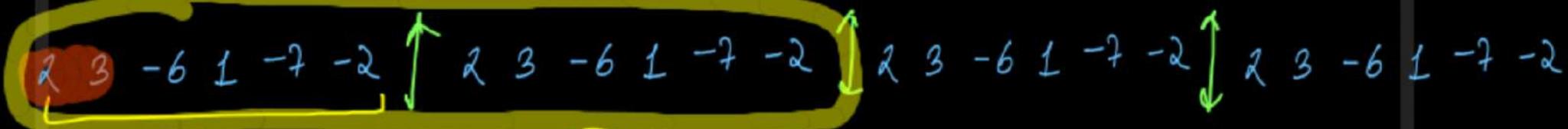


$$\left. + \underbrace{(K-2) * \text{sum}}_{\text{all middle copies}} \right\} \overset{\text{total}}{+}$$



\Rightarrow Kadane on 2 copies

eg5

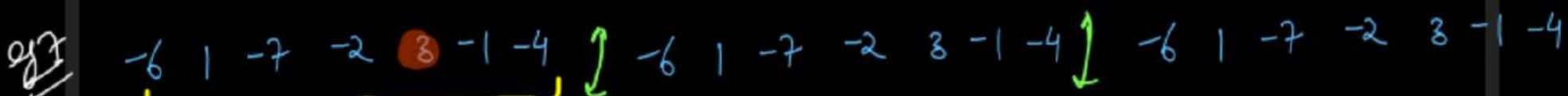


$$\text{sum} = 2 + 3 - 6 + 1 - 7 - 2 = (-9)$$

eg6

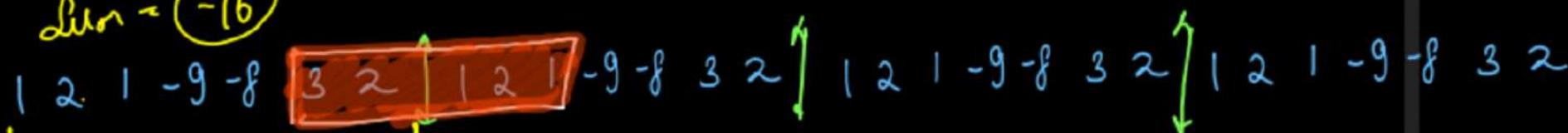


$$\text{sum} = -6 + 1 - 7 - 2 + 3 + 2 = (-9)$$



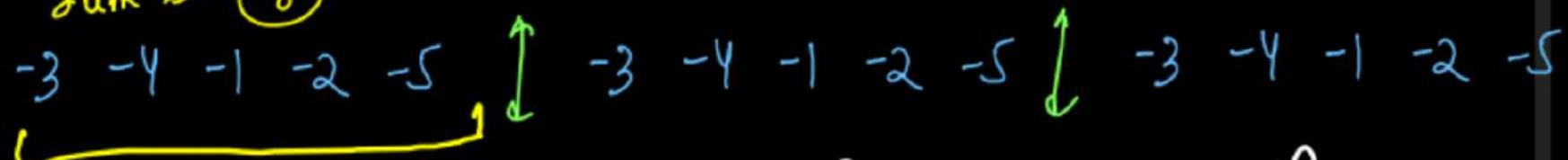
$$\text{sum} = (-16)$$

eg7



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

eg8



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

$\hookrightarrow \text{number} = 0 \{ \text{empty subarray} \}$

$\frac{\text{sum} < 0}{k=7/2}$

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

2	3	1
-4	1	0
3	1	-4
0	0	0

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

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

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

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

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

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

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

$(-1, 2, -4)$

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

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

$(3, 1, -4)$

DP with Game Theory / Minimax Algo

→ Optimal Game Strategy

→ Predict Winner (486)

→ Stone Game (877)

→ GFG → maxsum

→ Wine Selling Problem

→ Egg Dropping puzzle

N floors, 1 egg

N floors, 2 eggs

N floors, K eggs

N floors, ∞ eggs

∞ floors, ∞ eggs

#

2 players
play optimally
turn by turn

Predict the winner

↑(30), 38

5, 30, 10, 8, 15

A turn (max)

②, 40

5, 30, 10, 8, 15

(30) 38

B turn (min)

our player → ①

opponent → ②

first turn → ①

②, 40

5, 30, 10, 8, 15

④, 25

5, 30, 10, 8, 15

③, 15

5, 30, 10, 8, 15

③, 38

5, 30, 10, 8, 15

A turn (max)

5, 30, 10, 8, 15

30, 38

5, 30, 10, 8, 15

23, 45

5, 30, 10, 8, 15

28, 40

5, 30, 10, 8, 15

43, 25

5, 30, 10, 8, 15

33, 45

5, 30, 10, 8, 15

53, 15

5, 30, 10, 8, 15

33, 35

5, 30, 10, 8, 15

30, 38

B turn (min)

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



07x

~~left = 0, right = n - 1~~

game(left, right)

You

arr[left] + game(left + 1, right)

OPP

arr[right] + game(left, right - 1)

OPP

arr[left + 1]

min + game(left + 2, right)

arr[right]

+ game(left + 1, right - 1)

arr[left + 1] +

game(left + 1, right - 1)

arr[right - 1]

+

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;

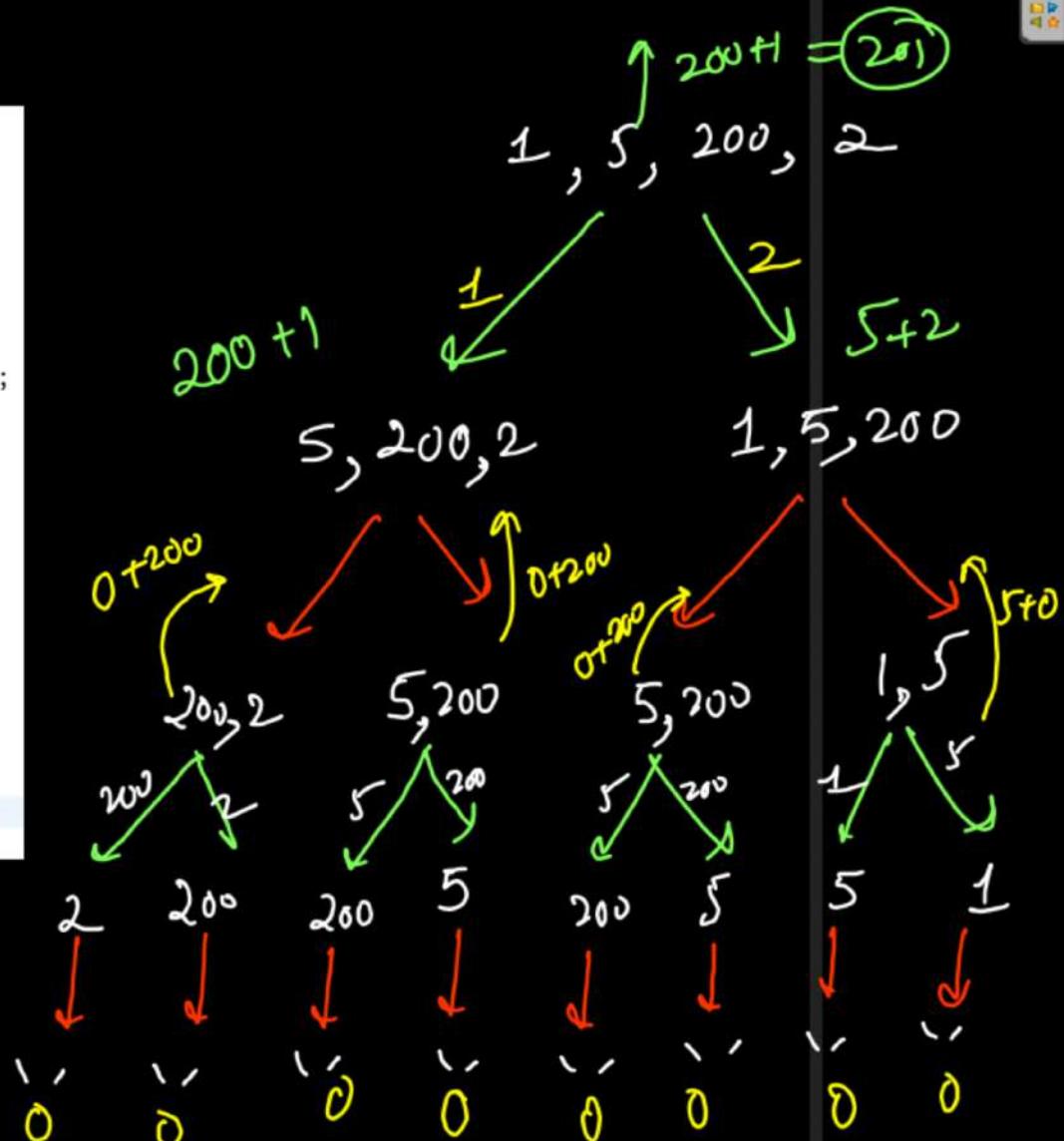
    if(ascore >= bscore) return true;
    return false;
}

```

Recursion

Time $\rightarrow O(3^n)$

Space $\rightarrow O(n)$ R.C.S.



~~Top down~~

```
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 $\rightarrow O(n^2)$

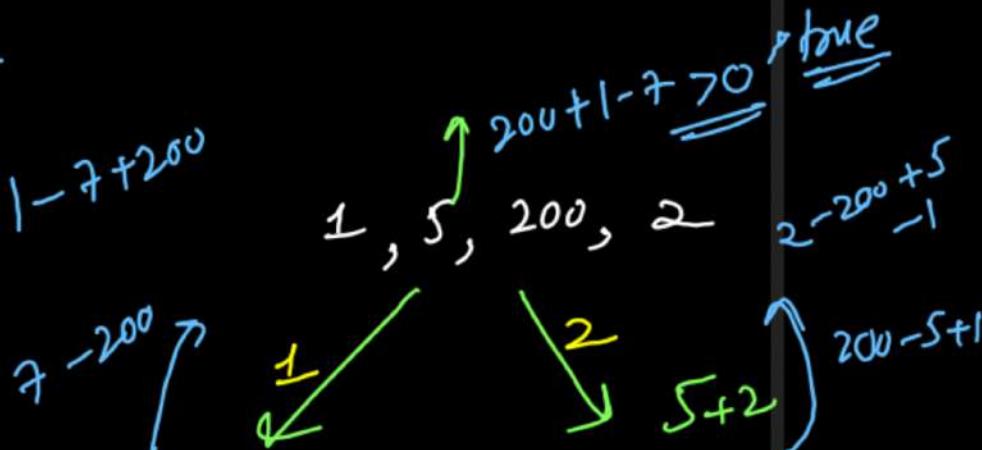
2D DP

~~Approach 2~~

$A \rightarrow B$ return difference of score

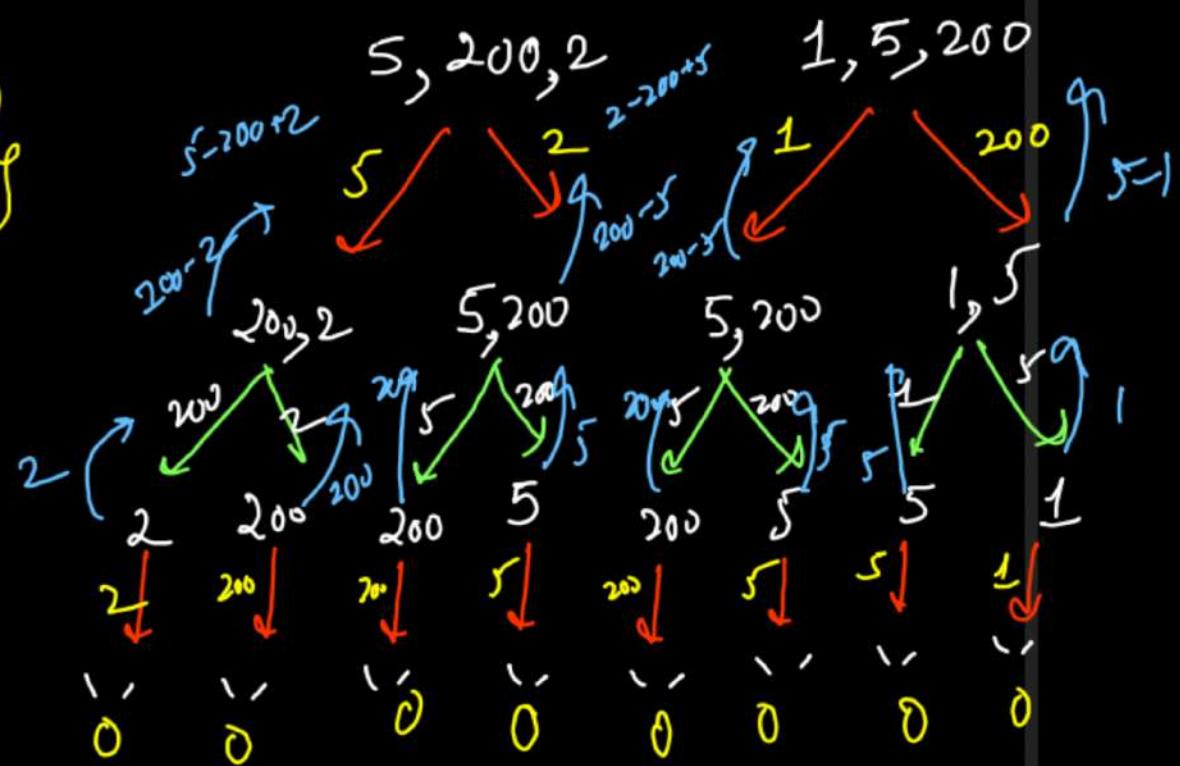
$A \geq B \rightarrow A - B \geq 0 \Rightarrow \text{true}$

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



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

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



~~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
True

Optimal Game Theory-II {Stone Game}

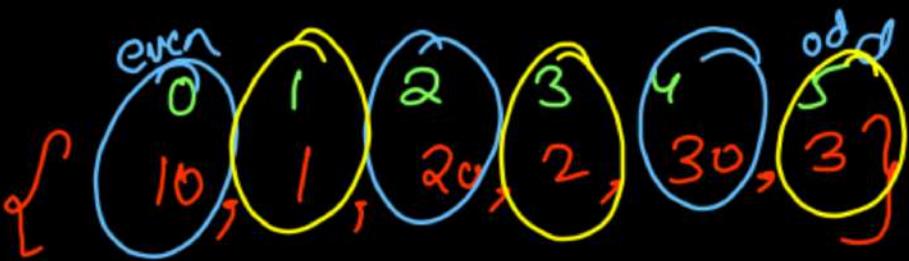
[877]

Extra Constraints

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

Answer : → Always  will be answer

eg1



(A) player

even sum > odd sum

even indices → odd indices

eg2



(A) player

odd sum > even sum

Greedy soln
return true;

Wine Selling Problem

main profit(i^{th} , l , r)
 ↓
 $n(1)*j + P(i+1, l+1, r)$
 ↓
 $n(2)*j + P(i+1, l, r-1)$
 ↓
 $n(1)*j + P(i+1, l, r-1)$
 ↓
 $n(1)*j + P(i+1, l, r-1)$
 ↓
 main

pr { 2, 4, 6, 2, 5 }
 1st 3rd 5th 4th 2nd

$$2*1 + 5*2 + 4*3 + 2*4 + 6*3 = 64$$

$$\begin{aligned} \text{Profit} &= 2*1 + 5*2 + 4*3 + 2*4 + 6*3 \\ &= 2 + 10 + 12 + 8 + 30 \end{aligned}$$

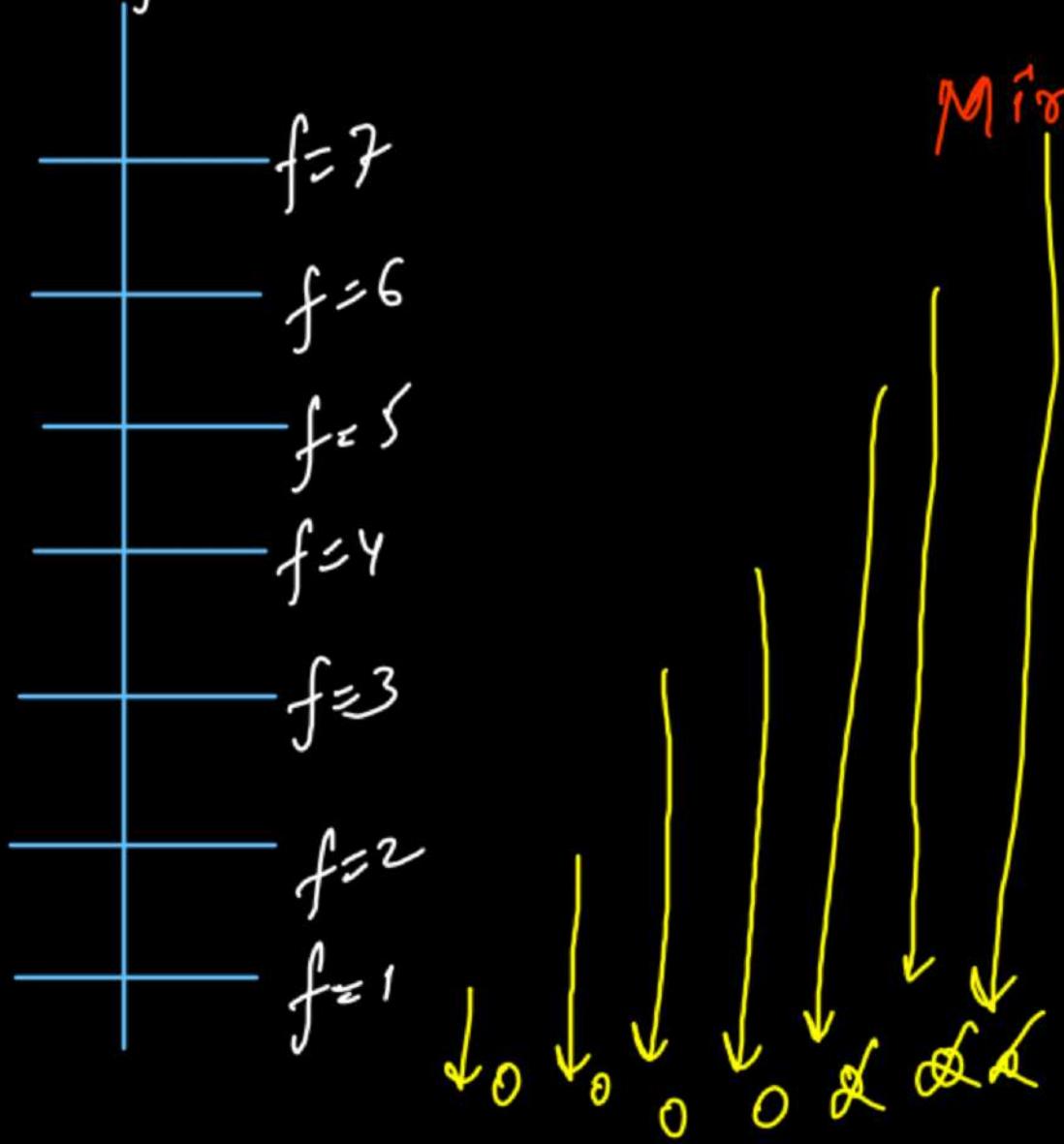
$$= 62$$

Code yourself

Egg Dropping Puzzle

- ① f floors, 1 egg \rightarrow Trivial Case (Linear Search)
 $\downarrow f$ floors
- ② f floors, 2 eggs \rightarrow Greedy algo (Mathematical formula)
- ③ f floors, 12 eggs \rightarrow Dynamic programming
- ④ f floors, ∞ eggs \rightarrow Binary Search
- ⑤ ∞ floors, ∞ eggs \rightarrow Binary Search on
Infinite Sorted Array

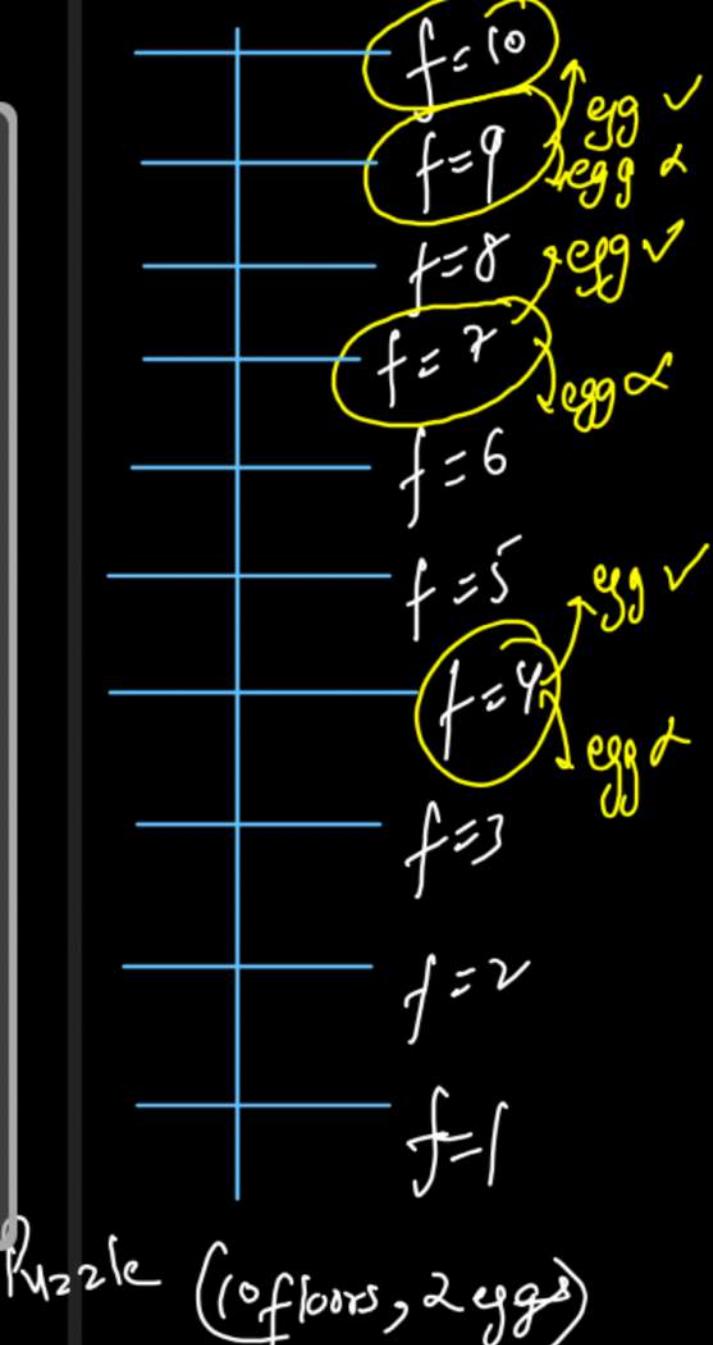
7 floors



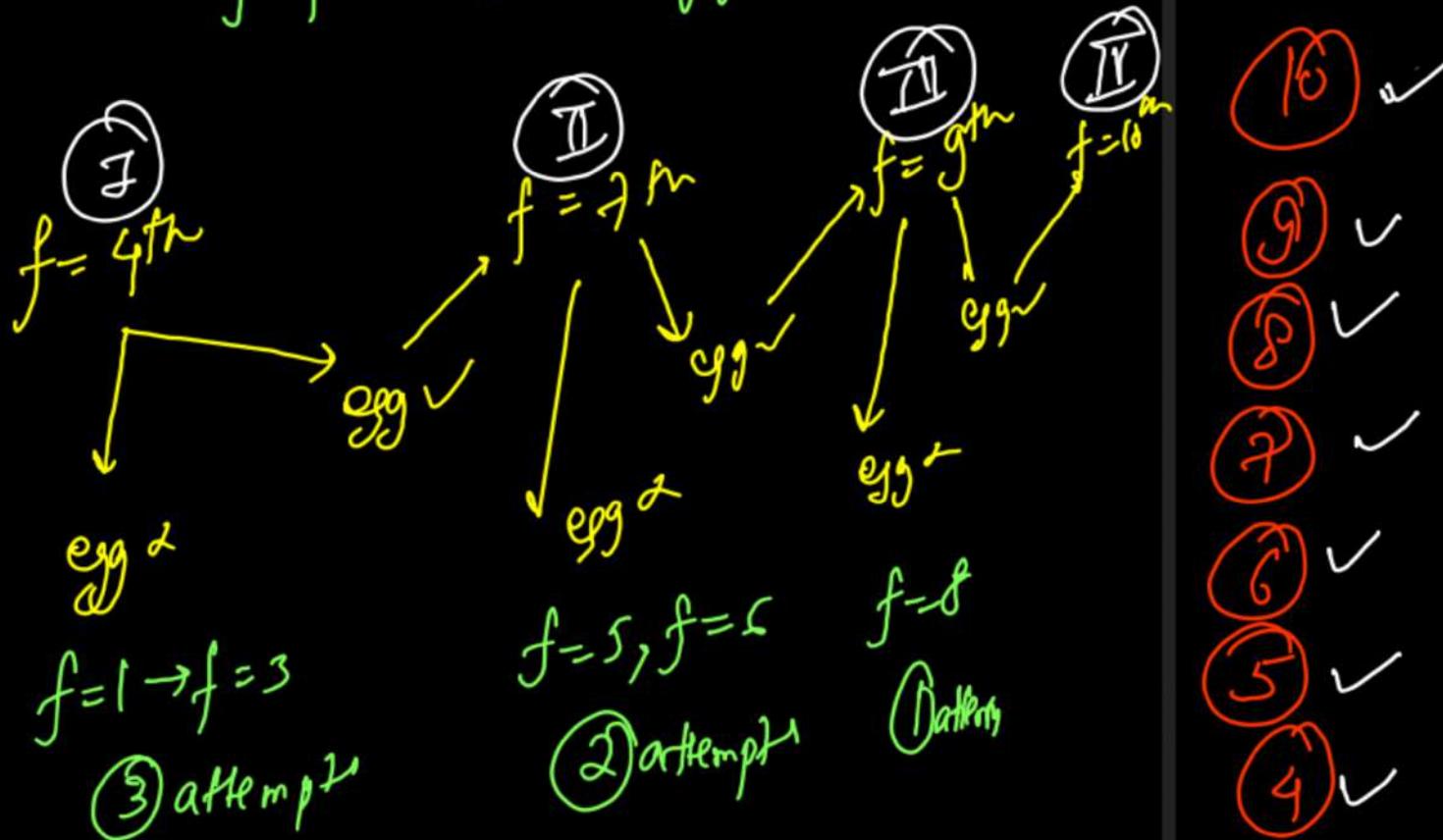
Guaranteed

- ① $tf = 1^m \Rightarrow 1 \text{ attempt}$
- ② $tf = 4^m \Rightarrow 4 \text{ attempts}$
- ③ $tf = 7^m \Rightarrow 7 \text{ attempts}$
- ④ $tf = 8^m \Rightarrow 7 \text{ attempts}$

1 egg \Rightarrow linear search
(f) floors



f floors, 2 eggs



$$\begin{aligned}
 & n + (n-1) + (n-2) + (n-3) + \dots \\
 & = f \text{ floors}
 \end{aligned}$$

Ruiz 21e (10 floors, 2 eggs)

$$x + (x-1) + (x-2) + \dots - 1 = f$$

$$\frac{x(x+1)}{2} = f$$

```
double ans = (-1.0 + Math.sqrt(1.0 + 8.0 * n)) / 2.0;  
if(ans == (int)ans) return (int)ans;  
return (int)ans + 1;
```

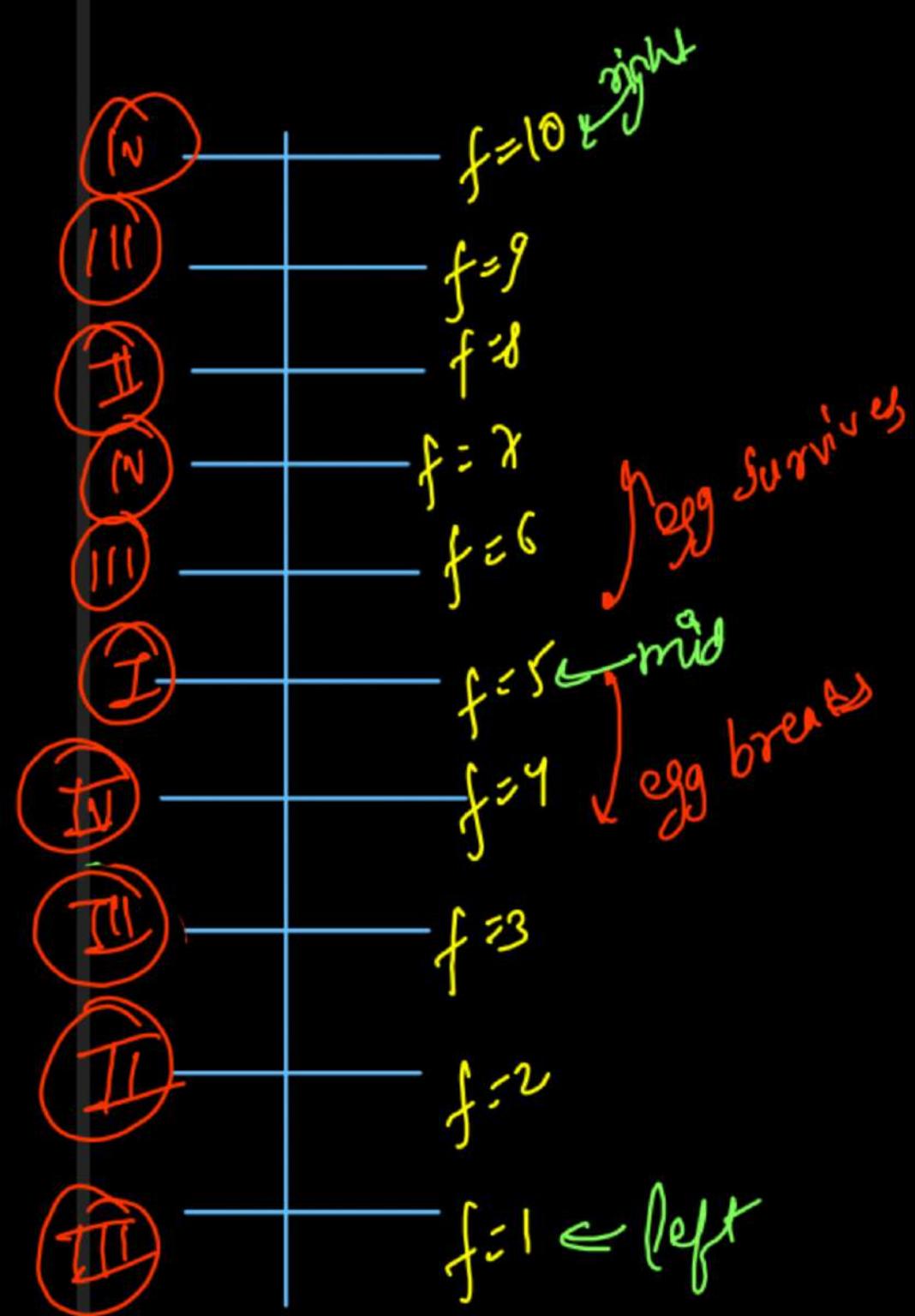
$$x^2 + x = 2f$$

$$\begin{aligned}f &= 10 \\n &= 40 \\&\text{ceil}(n) &= 41 \\&\text{ceil}(f) &= 5 \\n &= 4.2 \times f \\n &= (4+1) \times 5 \\&\text{ceil}(n) &= 5\end{aligned}$$

$$x^2 + x - 2f = 0$$

$$x = \frac{-1.0 + \sqrt{1 + 8f}}{2.0}$$

$\text{ceil}(x)$
↳ Answer

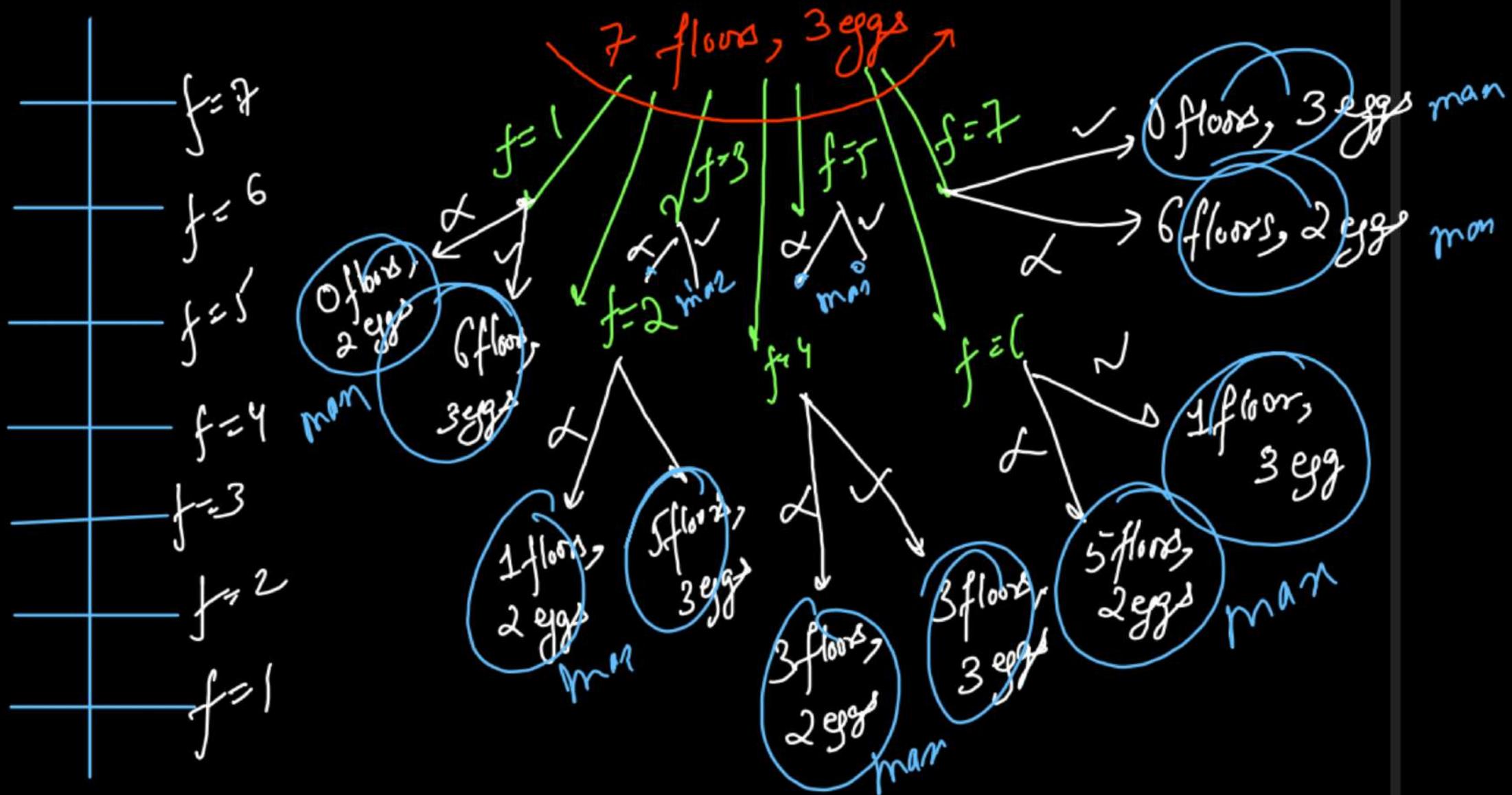


f floors, ~~∞ eggs~~
 Infinite Supply



f floors, k eggs

3 eggs



11:43

helper(f floors, k eggs)

if

$\sqrt{2f}$

3)

f^{th} f

at
wood car

~~in floor~~

egg survived

egg breaks

helper($i-1$, $K-1$)

$(i-1)$ floors
below with 1
less egg

your
attempt

min

f floors

i

many

$i=0$

edges

$\overline{f-i}$, K)

$(f-i)$ floors

above

with same no
of eggs

```

static int eggDrop(int floors, int eggs, int[][] dp){
    if(floors == 0) return 0;
    if(eggs == 1) return floors;
    if(dp[floors][eggs] != -1) return dp[floors][eggs];

    int min = floors;
    for(int i=1; i<=floors; i++){
        int eggBreak = eggDrop(i - 1, eggs - 1, dp);
        int eggSurvive = eggDrop(floors - i, eggs, dp);

        min = Math.min(min, 1 + Math.max(eggBreak, eggSurvive));
    }

    return dp[floors][eggs] = min;
}

```

Space

$\rightarrow O(Floors * Eggs)$

2D DP

Time

$O(Floors * Eggs * Floors)$

Cells

for loop (work)

$$= O(n^3)$$

```

static int eggDrop(int eggs, int floors)
{
    int[][] dp = new int[floors + 1][eggs + 1];
    for(int i=0; i<=floors; i++){
        for(int j=0; j<=eggs; j++){
            dp[i][j] = -1;
        }
    }

    return eggDrop(floors, eggs, dp);
}

```

Matrix Chain Multiplication

Partition DP

Partition will not
be in the form
of subset but
in the form of
subarray

- Palindrome Partitioning
- optimal BST
- Burst Balloons
- Scramble String
- Boolean Parenthesization
- Rectangle Cutting
- Rod Cutting
- Word Break
- Text Justification

Matrix Chain Multiplication

Matrix multiplication

$$A_{n_1 \times n_2} * B_{n_2 \times n_3}$$

$C_1 = A_1 * B_1$
 ↳ matrix multiplication

$$\text{Total cost} = \boxed{n_1 * C_1 * n_2}$$

$$C_{11}^{\text{ops}} = a_{11} * b_{11} + a_{12} * b_{21} \\ + a_{13} * b_{31} + a_{14} * b_{41}$$

$$C_{12}^{\text{ops}} = a_{11} * b_{21} + a_{12} * b_{22} \\ + a_{13} * b_{32} + a_{14} * b_{42}$$

$$C_{21}^{\text{ops}} = 2^{\text{nd}} \text{ row} * 1^{\text{st}} \text{ col}$$

$$C_{22}^{\text{ops}} = 2^{\text{nd}} \text{ row} * 2^{\text{nd}} \text{ col}$$

$$C_{31}^{\text{ops}} = 3^{\text{rd}} \text{ row} * 1^{\text{st}} \text{ col}$$

$$C_{32}^{\text{ops}} = 3^{\text{rd}} \text{ row} * 2^{\text{nd}} \text{ col}$$

$$\text{Total cost} = 3 * 4 * 2 \\ = 24 \text{ ops}$$

A 3*4

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

B 4*2

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \\ C_{31} & C_{32} \end{bmatrix}_{3*2} \\ C(\text{res})$$

Matrix Chain Multiplication

$$A_{20 \times 50} \quad B_{50 \times 10} \quad C_{10 \times 40}$$

way 1

$$(A_{20 \times 50} * B_{50 \times 10}) * C_{10 \times 40}$$

\downarrow

$$AB_{20 \times 10} * C_{10 \times 40}$$

$\cancel{20 \times 50 \times 10} + \cancel{20 \times 10 \times 40}$

$$\Rightarrow RC_{20 \times 40}$$

$$20 \times 50 \times 10 + 20 \times 10 \times 40$$

$$= 10000 + 8000 = \boxed{18000}$$

way 2

$$A_{20 \times 50} * (B_{50 \times 10} * C_{10 \times 40})$$

\downarrow

$$A_{20 \times 50} * BC_{50 \times 40}$$

$\cancel{50 \times 10 \times 40} \rightarrow \text{Res.}$

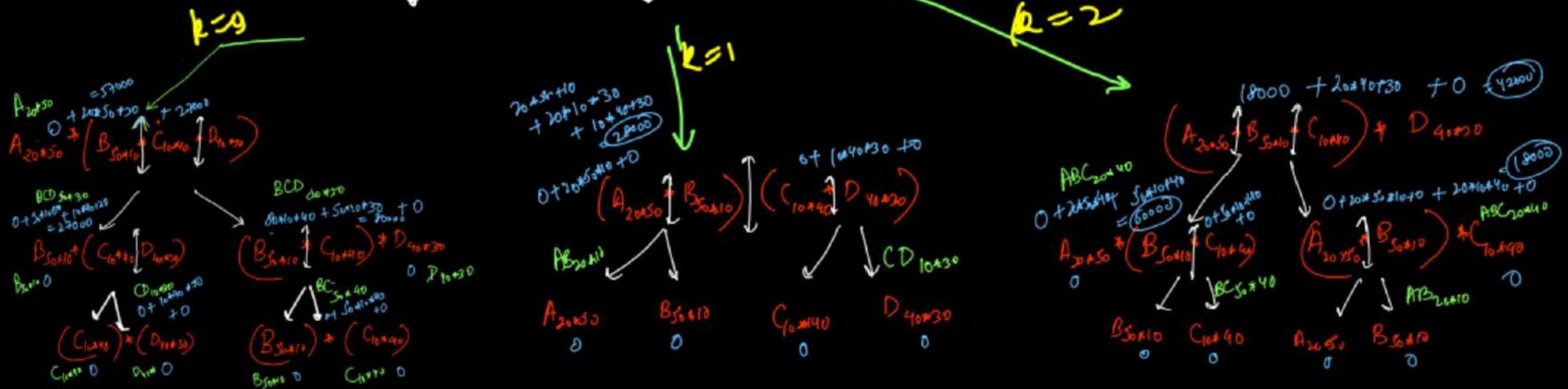
$$50 \times 10 \times 40 + 20 \times 50 \times 40$$

$$= 20000 + 40000 = 60000$$

Matrix Chain Multiplication	
$A_{20 \times 50}$	$B_{50 \times 10}$
$C_{10 \times 40}$	
$\cancel{20 \times 50 \times 10} + \cancel{20 \times 10 \times 40}$	
$\Rightarrow RC_{20 \times 40}$	

Input: $\{20, 50, 10, 40, 30\}$

$$\min(57000 \text{ vs } 28000 \text{ vs } 42000) = 28000$$



DP table \rightarrow 2D DP \rightarrow (left, right)

for($\text{int } k = 0; k < n; k++$)

$\{ \begin{matrix} 0 \\ 20 \\ 1 \\ 50 \\ 10 \\ 40 \\ 30 \end{matrix}, \dots \}_{l=0}^{l=4}$

$\{ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \end{matrix} \}_{k=0}^{k=4}$

$\gamma = 3$, $n = 5$

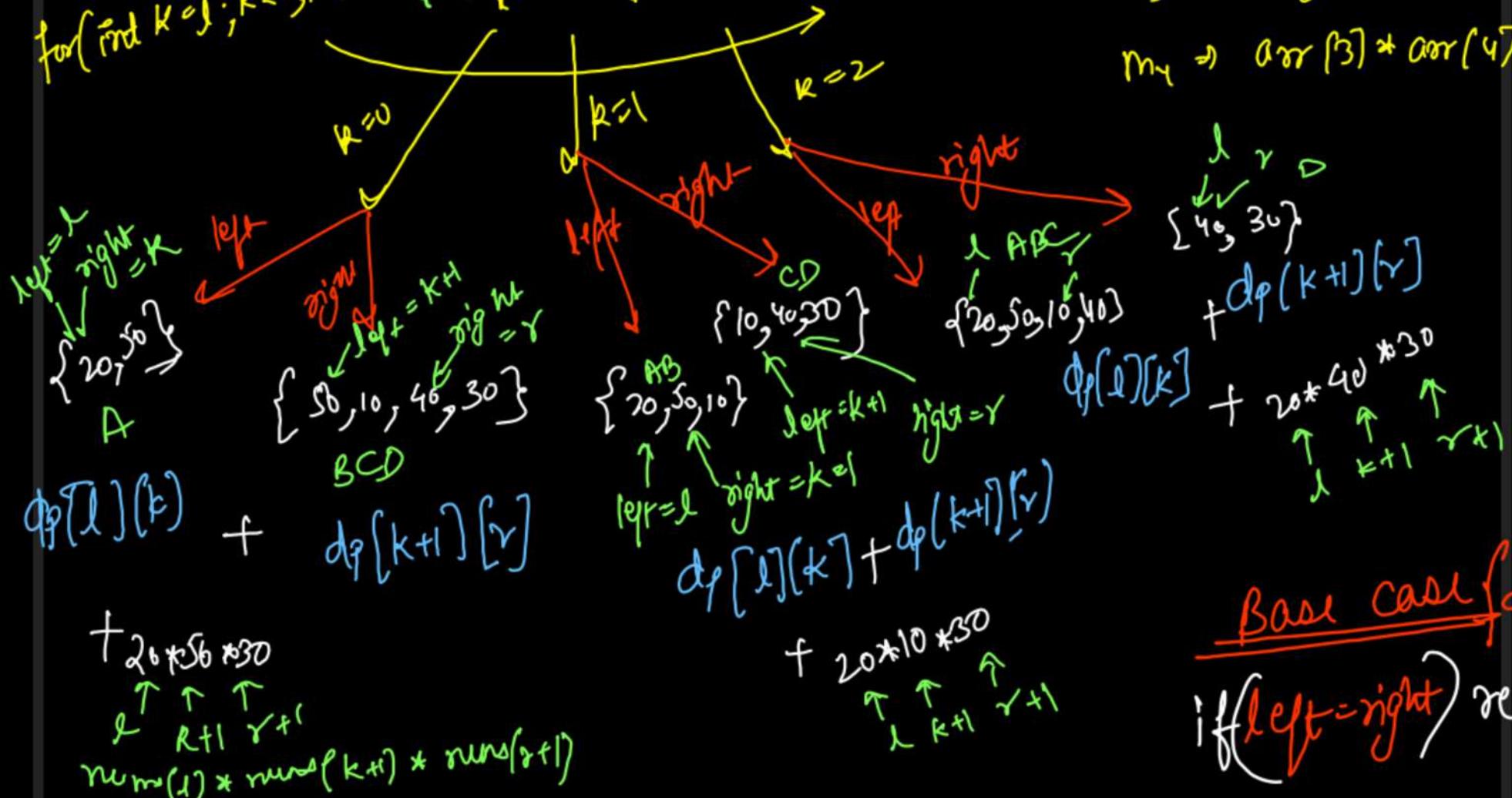
$$m_1 \Rightarrow \text{arr}(0) * \text{arr}(1)$$

$$m_2 \Rightarrow \text{arr}(1) * \text{arr}(2)$$

$$m_3 \Rightarrow \text{arr}(2) * \text{arr}(3)$$

$$m_4 \Rightarrow \text{arr}(3) * \text{arr}(4)$$

$$\begin{aligned} \text{matrix} \\ = nr \\ = 4 \end{aligned}$$



```

static int helper(int l, int r, int[] nums){
    if(l == r) return 0; // Single Matrix -> Multiplication Cost = 0

    int minCost = Integer.MAX_VALUE;
    // Creating partitions
    for(int k=l; k<r; k++){
        // Left Matrices -> Multiplication Min Cost
        int left = helper(l, k, nums);
        // Right Matrices -> Multiplication Min Cost
        int right = helper(k + 1, r, nums);
        // Left * Right Multiplication Cost
        int cost = left + (nums[l] * nums[k + 1] * nums[r + 1]) + right;
        minCost = Math.min(minCost, cost);
    }

    return minCost;
}

static int matrixMultiplication(int N, int[] nums) {
    return helper(0, N - 2, nums);
}

```

Recursion

Time]
 $O(\text{Exponential})$

Space]
 $O(N)$

R.C.S.

```

static int helper(int l, int r, int[] nums, int[][] dp){
    if(l == r) return 0; // Single Matrix -> Multiplication Cost = 0
    if(dp[l][r] != -1) return dp[l][r];

    int minCost = Integer.MAX_VALUE;
    // Creating partitions
    for(int k=l; k<r; k++){
        // Left Matrices -> Multiplication Min Cost
        int left = helper(l, k, nums, dp);
        // Right Matrices -> Multiplication Min Cost
        int right = helper(k + 1, r, nums, dp);
        // Left * Right Multiplication Cost
        int cost = left + (nums[l] * nums[k + 1] * nums[r + 1]) + right;
        minCost = Math.min(minCost, cost);
    }

    return dp[l][r] = minCost;
}

```

```

static int matrixMultiplication(int N, int nums[]){
    int[][] dp = new int[N + 1][N + 1];
    for(int i=0; i<=N; i++)
        for(int j=0; j<=N; j++)
            dp[i][j] = -1;

    return helper(0, N - 2, nums, dp);
}

```

Memoization

* Time $\rightarrow O(n^{\frac{n}{\downarrow}})$
 cells in
 2D DP
 for loop
 Partition

$$\Rightarrow O(n^3)$$

* Space $\rightarrow O(n^2)$ 2D DP
 $\rightarrow O(n)$ R.C.S.



Matrix Chain Multiplication

Tabulation

Bottom-up

smallest (A, B, C, D)



biggest ($A * B * C * D$)



diagonal by diagonal

gap
(col-row = 3)

gap
(col-row = 2)

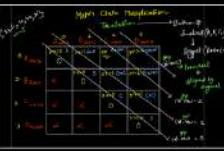
gap
(col-row = 1)

gap
(col-row = 0)

$\{ A_{20 \times 50}, B_{50 \times 10}, C_{10 \times 40}, D_{40 \times 30} \}$
m = 5

		0	1	2	3
0	$0 \times 0 = 0$	$1 \times 1 = 1$	$2 \times 2 = 0$	$3 \times 3 = 3$	
1	α	0	$2 \times 1 = 1$	$3 \times 2 = 2$	
2	α	α	0	$3 \times 1 = 1$	
3	α	α	α	0	

$A_{20 \times 50}$	$B_{50 \times 10}$	$C_{10 \times 40}$	$D_{40 \times 30}$	M_n
\textcircled{O}	$O + 20 \times 50 \times 10 + O$	$\min\left(\begin{array}{l} A + BC + AC \\ AB + C + B^T \end{array}\right)$ <u>$A + BC + AC$</u>	$O + 40 \times 30 + O$ <u>$A + BC + CD$</u>	$A + BCD + A \times BCD$ $AB + CD + AB \times CD$ $ABC + D + ABC \times D$
$\cancel{\textcircled{O}}$	\textcircled{O}	$O + 50 \times 10 \times 40 + O$ $= 20000$	$\min\left(\begin{array}{l} B + CD + B^T D \\ BC + D + C^T D \end{array}\right)$	
$\cancel{\textcircled{O}}$	$\cancel{\textcircled{O}}$	\textcircled{O}	$O + 10 \times 40 \times 30 + O$	$B \times C \times D$
$\cancel{\textcircled{O}}$	$\cancel{\textcircled{O}}$	$\cancel{\textcircled{O}}$	\textcircled{O}	D



$dp(l)(r) = \text{Min cost of multiplying matrices from } l \text{ to } r.$

```
int[][] dp = new int[N + 1][N + 1];  
  
for(int gap=1; gap<N-1; gap++){  
    for(int l=0, r=gap; r<N-1; l++, r++){  
  
        int minCost = Integer.MAX_VALUE;  
        // Creating partitions  
        for(int k=l; k<r; k++){  
            // Left Matrices -> Multiplication Min Cost  
            int left = dp[l][k];  
            // Right Matrices -> Multiplication Min Cost  
            int right = dp[k + 1][r];  
            // Left * Right Multiplication Cost  
            int cost = left + (nums[l] * nums[k + 1] * nums[r + 1]) + right;  
            minCost = Math.min(minCost, cost);  
        }  
  
        dp[l][r] = minCost;  
    }  
  
    return dp[0][N - 2]; ← top right cell is the answer
```

Tabulation

gap strategy

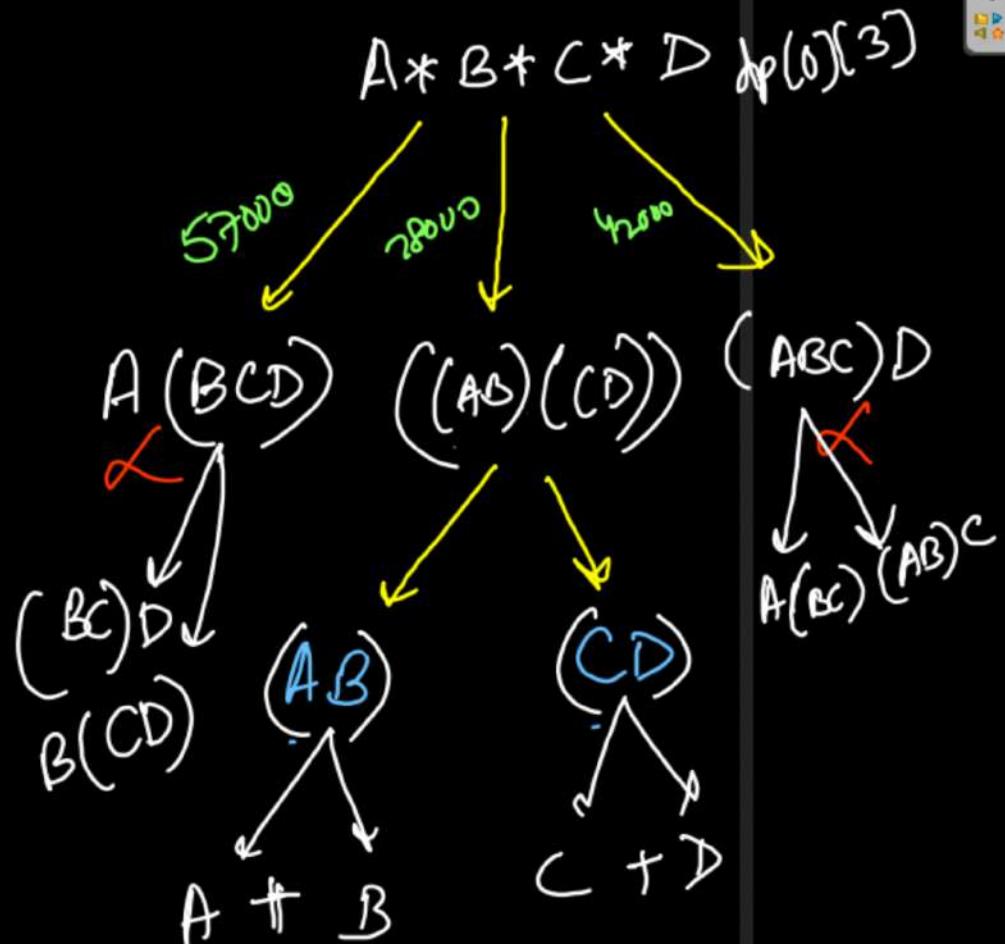
⇒ Diagonal by diagonal

Time $\Rightarrow O(N^3)$

Space $\Rightarrow O(N^2)$

2D DP

	$A_{20 \times 50}$	$B_{50 \times 10}$	$C_{10 \times 40}$	$D_{40 \times 30}$
0	O	A	$A * B$	$A * B * C$
1	$A_{20 \times 50}$	O	10000	18000
2	$B_{50 \times 10}$	O	20000	$B * C$
3	$C_{10 \times 40}$	O	27000	$B * C * D$
4	$D_{40 \times 30}$	O	12000	$C * D$
5	O	O	D	



```

static String backtrack(int l, int r, int[] nums, int[][] dp){
    if(l == r) return ("" + (char)('A' + l));
    for(int k=l; k<r; k++){
        int cost = dp[l][k] + (nums[l] * nums[k + 1] * nums[r + 1]) + dp[k + 1][r];
        if(cost == dp[l][r]){
            return '(' + backtrack(l, k, nums, dp) + backtrack(k + 1, r, nums, dp) + ')';
        }
    }
    return "";
}

static String matrixChainOrder(int[] nums, int N){
    int[][] dp = new int[N][N];

    for(int gap=1; gap<N-1; gap++){
        for(int l=0, r=gap; r<N-1; l++, r++){

            int minCost = Integer.MAX_VALUE;
            for(int k=l; k<r; k++){
                int cost = dp[l][k] + (nums[l] * nums[k + 1] * nums[r + 1]) + dp[k + 1][r];
                minCost = Math.min(minCost, cost);
            }

            dp[l][r] = minCost;
        }
    }

    return backtrack(0, N-2, nums, dp);
}

```



$0 \rightarrow 'A'$ ($'A' + 0 = 'A'$)
 $1 \rightarrow 'B'$ ($'A' + 1 = 'B'$)
 $2 \rightarrow 'C'$ ($'A' + 2 = 'C'$)
 $3 \rightarrow 'D'$ ($'A' + 3 = 'D'$)



Burst Balloons

You are given `n` balloons, indexed from `0` to `n - 1`. Each balloon is painted with a number on it represented by an array `nums`. You are asked to burst all the balloons.

If you burst the i^{th} balloon, you will get `nums[i - 1] * nums[i] * nums[i + 1]` coins. If $i - 1$ or $i + 1$ goes out of bounds of the array, then treat it as if there is a balloon with a `1` painted on it.

Return the maximum coins you can collect by bursting the balloons wisely.

{ 2, 2, 2, 8 }

$$6 \times 3 \times 1 \quad 3$$

$$1 \times 1 \times 5 \quad 1$$

$$1 \times 5 \times 8 \quad 5$$

$$1 \times 8 \times 1 \quad 8$$

$$\underline{56}$$

$$3 \times 1 \times 5 \quad 1$$

$$5 \times 8 \times 1 \quad 8$$

$$3 \times 5 \times 1 \quad 5$$

$$\underline{1 \times 3 \times 1 \quad 3}$$
$$\underline{73}$$

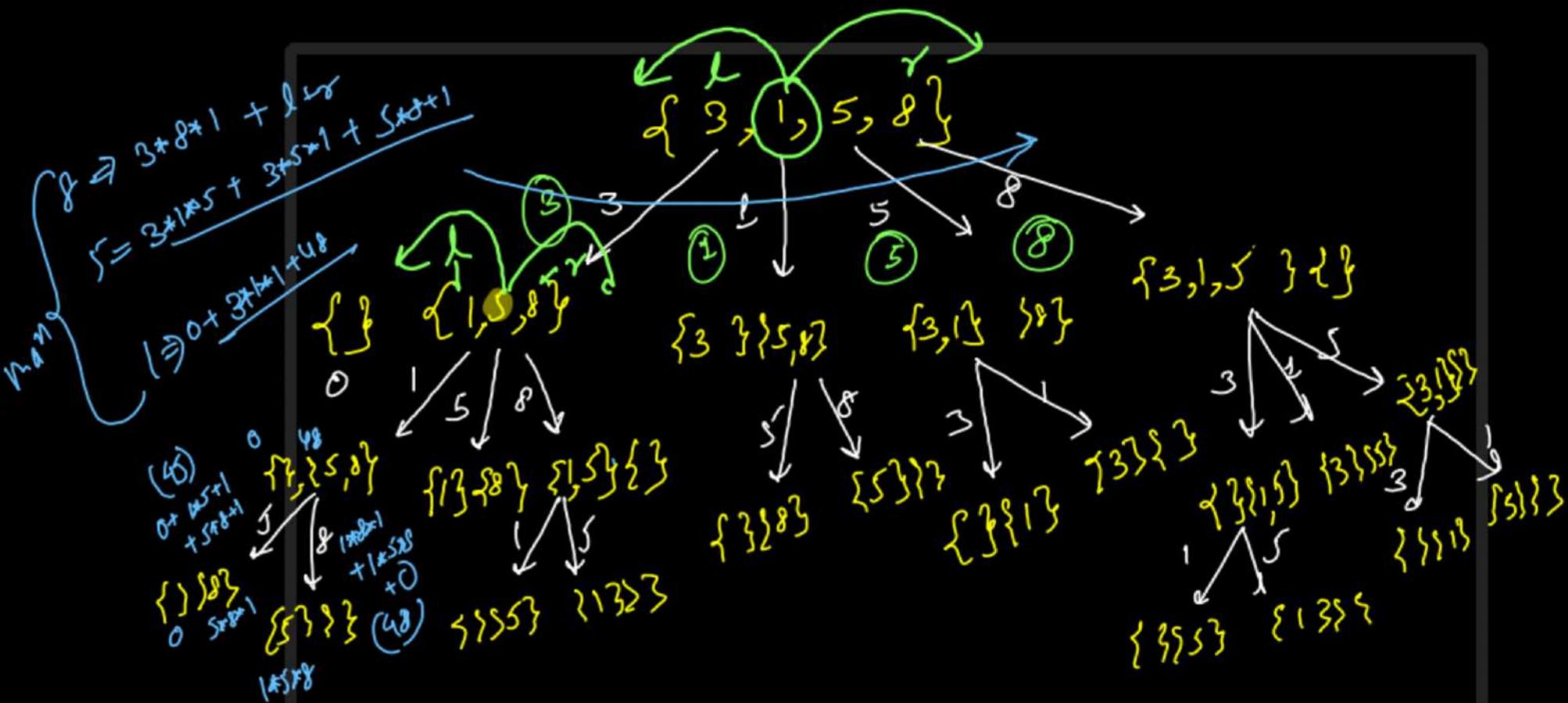
$$3 \times 1 \times 5 \quad 1$$

$$3 \times 5 \times 8 \quad 5$$

$$1 \times 3 \times 8 \quad 3$$

$$1 \times 8 \times 1 \quad 8$$

$$\underline{167}$$



$dp[l][r] \rightarrow \text{Max}^m \text{ score to burst balloons from } l \text{ to } r$

partition \Rightarrow last balloon to be burst $\rightarrow k \in l, l+1, l+2, \dots, r$

$dp[l][k] \rightarrow \text{Max}^m \text{ score to burst balloons in left range}$

$dp[k+1][r] \rightarrow \text{Max}^m \text{ score to burst balloons in right range}$

cost to burst $\rightarrow \text{num}[k-1] + \text{num}(k) * \text{num}(r+1)$
from balloon

```

public int helper(int l, int r, int[] nums, int[][] dp){
    if(l > r) return 0; // No Balloons -> 0 Score
    if(dp[l][r] != -1) return dp[l][r];

    int maxCost = 0;
    for(int k=l; k<=r; k++){
        int left = helper(l, k - 1, nums, dp);
        int right = helper(k + 1, r, nums, dp);
        int cost = ((l == 0) ? 1 : nums[l - 1]) * nums[k] * ((r == nums.length - 1) ? 1 : nums[r + 1]);
        maxCost = Math.max(maxCost, left + cost + right);
    }

    return dp[l][r] = maxCost;
}

```

```

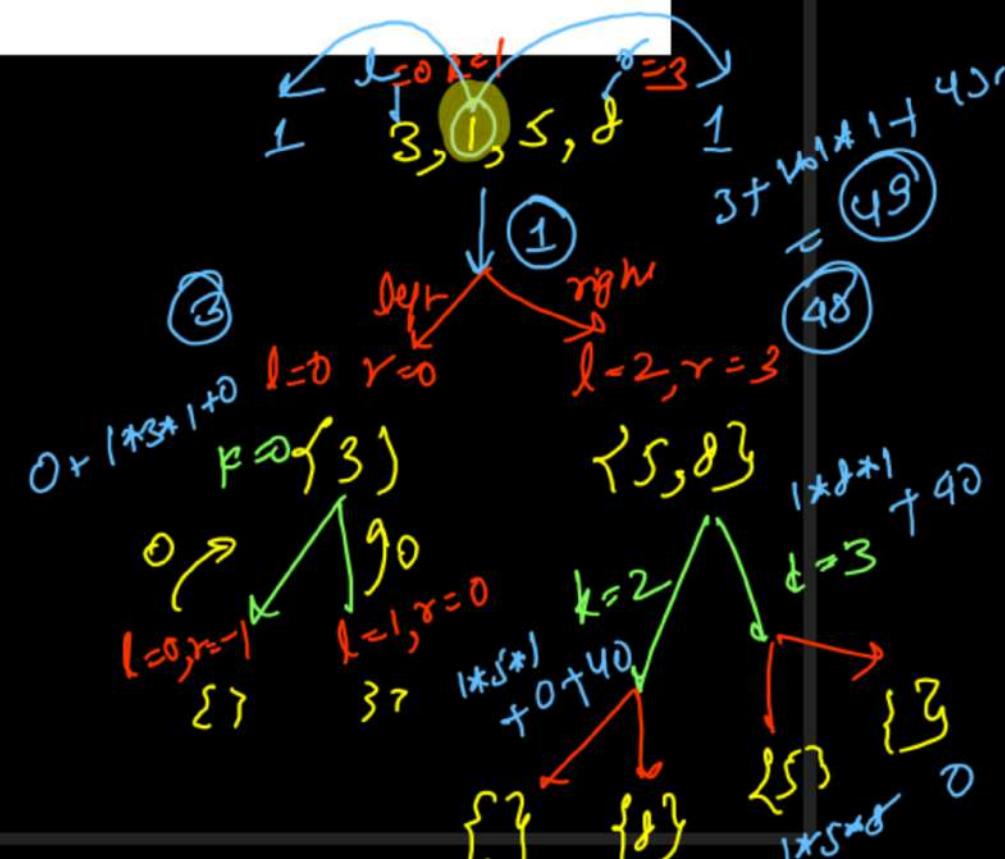
public int maxCoins(int[] nums) {
    int[][] dp = new int[nums.length + 1][nums.length + 1];
    for(int i=0; i<=nums.length; i++)
        for(int j=0; j<=nums.length; j++)
            dp[i][j] = -1;

    return helper(0, nums.length - 1, nums, dp);
}

```

Time $\rightarrow O(n^3)$

Space $\rightarrow O(n^2)$



Order of five classes of Schedule?

- ① DP → Hard Category → MCM → Thursday 1st
- ② Hashmap & Heap → level 1 {without construction}
3-4 classes
- ③ Graphs → level 1 + level 2 {Scratch → Advanced}
- ④ Hashmap & Heap → Construction + level 2 (Array & String)
6-7 classes
- ⑤ Bit Manipulation & Number Theory → Competitive
7-8 classes

131

Palindrome Partitioning

madam

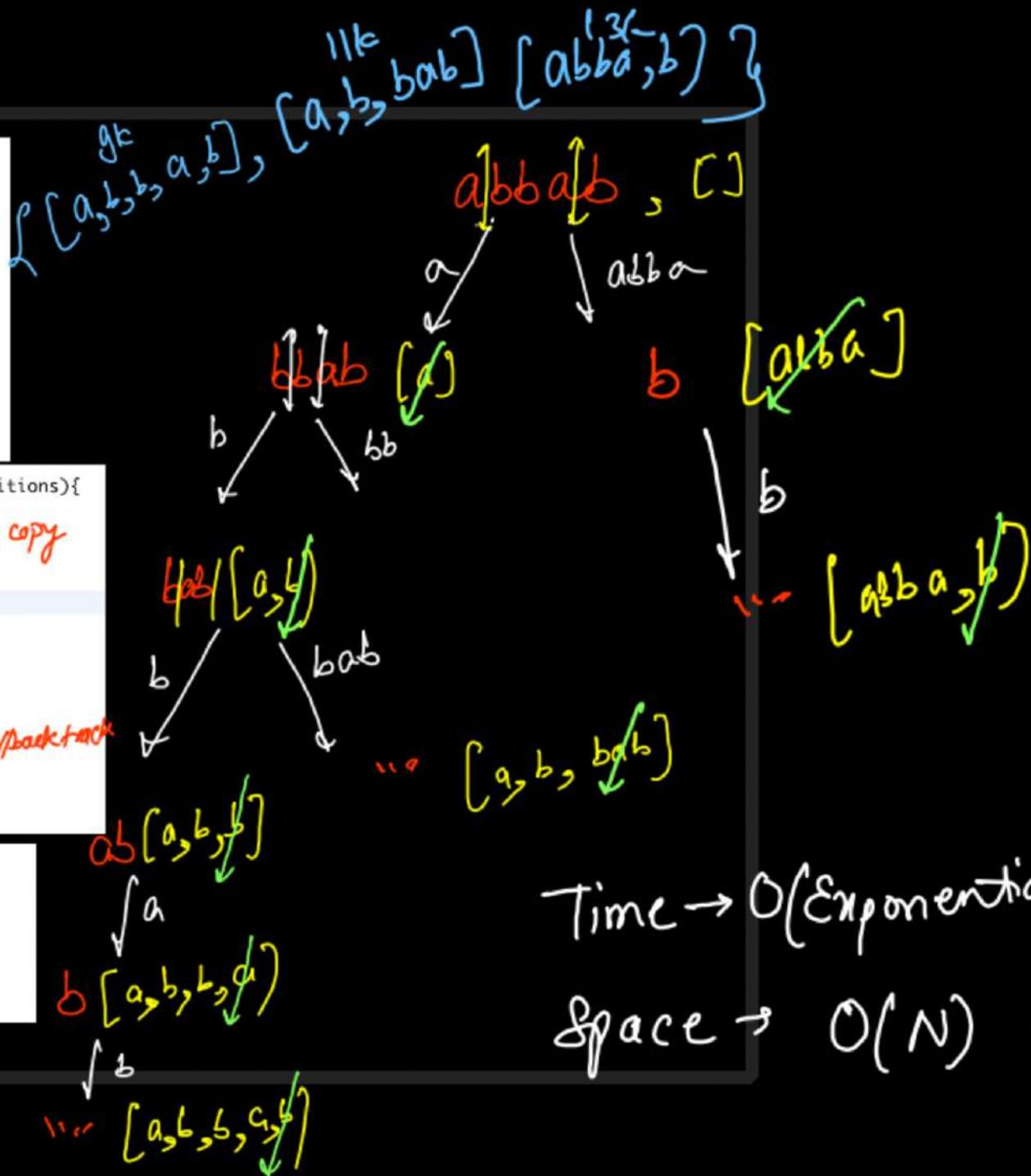


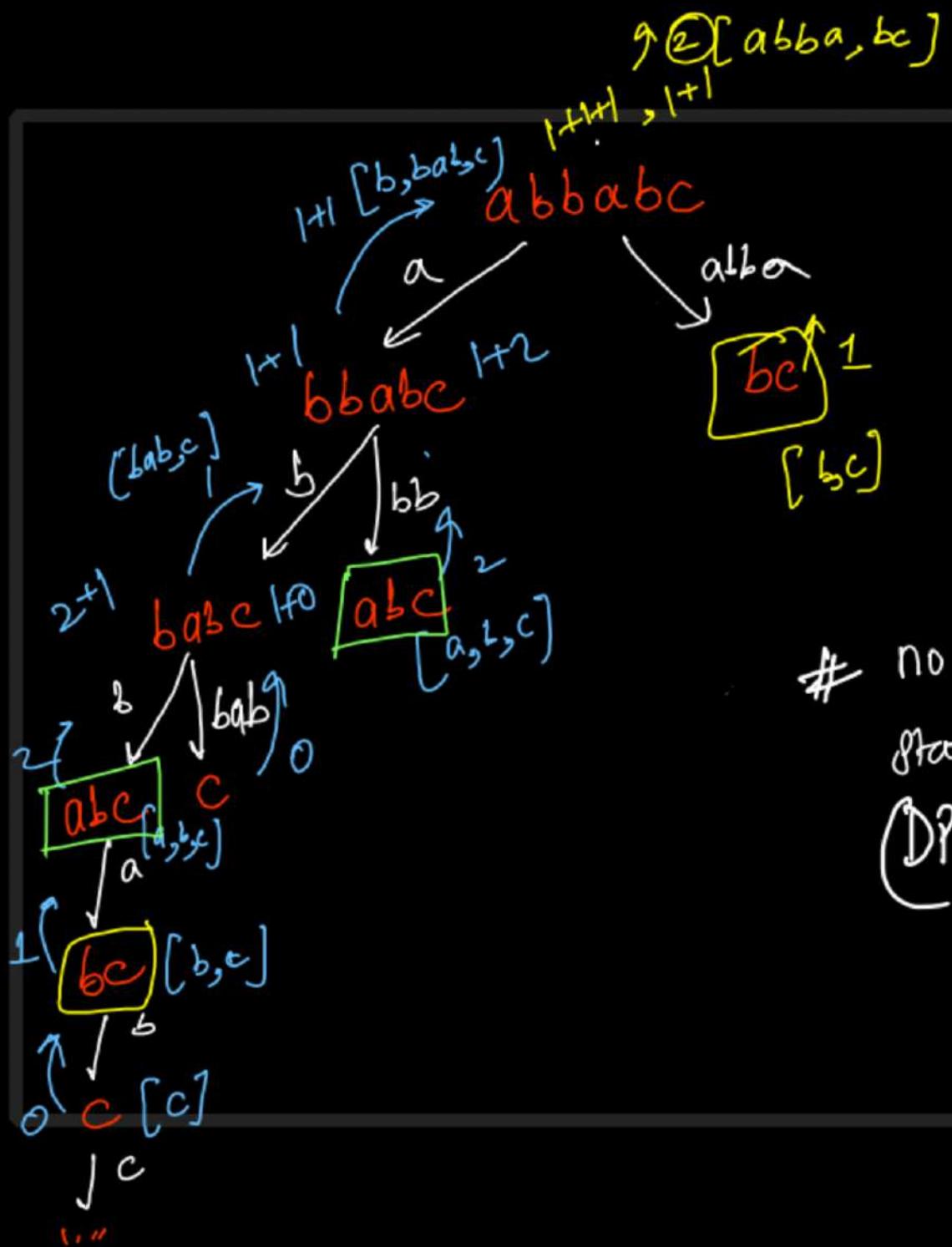
```
List<List<String>> res;
```

```
public boolean isPalindrome(String s){  
    int l = 0, r = s.length() - 1;  
    while(l <= r){  
        if(s.charAt(l) != s.charAt(r))  
            return false;  
        l++; r--;  
    }  
    return true;  
}
```

```
public void helper(String s, int i, List<String> partitions){  
    if(i == s.length())  
        res.add(new ArrayList<>(partitions)); //deep copy  
        return;  
  
    for(int j=i; j<s.length(); j++){  
        String left = s.substring(i, j + 1);  
        if(isPalindrome(left) == true){  
            partitions.add(left);  
            helper(s, j + 1, partitions);  
            partitions.remove(partitions.size() - 1); //backtrack  
        }  
    }  
}
```

```
public List<List<String>> partition(String s) {  
    res = new ArrayList<>();  
    List<String> partitions = new ArrayList<>();  
    helper(s, 0, partitions);  
    return res;  
}
```





$\#$ no of variable
 states \Rightarrow y index
 (DP 8 states)

15a
 min cuts
 is palindromic
 if position

```
public boolean isPalindrome(String s){  
    public int helper(String s, int i){  
        if(i == s.length()) return 0;  
        if(isPalindrome(s.substring(i))) return 0;  
  
        int minCuts = s.length() - i + 1;  
        for(int j=i; j<s.length(); j++){  
            String left = s.substring(i, j + 1);  
            if(isPalindrome(left) == true){  
                int cuts = helper(s, j + 1);  
                minCuts = Math.min(minCuts, cuts + 1);  
            }  
        }  
  
        return minCuts;  
    }  
  
    public int minCut(String s) {  
        return helper(s, 0);  
    }  
}
```

Recursive (T(E))
Time → Exponential
Space → $O(N)$ R.C.S.

```

public boolean isPalindrome(String s){}

public int helper(String s, int i, int[] dp){
    if(i == s.length()) return 0;
    if(dp[i] != -1) return dp[i];
    if(isPalindrome(s.substring(i))) return dp[i] = 0;

    int minCuts = Integer.MAX_VALUE;
    for(int j=i; j<s.length(); j++){
        String left = s.substring(i, j + 1);
        if(isPalindrome(left) == true){
            int cuts = helper(s, j + 1, dp);
            minCuts = Math.min(minCuts, cuts + 1);
        }
    }
    return dp[i] = minCuts;
}

public int minCut(String s) {
    int[] dp = new int[s.length() + 1];
    Arrays.fill(dp, -1);
    return helper(s, 0, dp);
}

```

Time

$$O(N * N) = O(N^2)$$

DP state for loop (partition)

Space

1D DP $\Rightarrow O(N)$

RCS $\Rightarrow O(N)$

$\wedge \rightarrow \text{AND}$

$| \rightarrow \text{OR}$

$\wedge \rightarrow \text{XOR}$

Operand
true false

Boolean Parenthesization

$$\begin{array}{c} T \wedge T | F \\ \swarrow \quad \searrow \\ (T \wedge T) | F \\ T | F = T \end{array}$$

$$\begin{array}{c} T \wedge (T | F) \\ T \wedge T = T \end{array}$$

$$\begin{array}{c} T \wedge F | T \\ \swarrow \quad \searrow \\ (T \wedge F) | T \\ F | T = T \end{array}$$

$$\begin{array}{c} T \wedge (F | T) \\ T \wedge T = T \end{array}$$

$$\begin{array}{c} T \wedge F | T \\ \swarrow \quad \searrow \\ (T \wedge F) | T \\ T | T = T \end{array}$$

$$\begin{array}{c} T \wedge (F | T) \\ T \wedge T = F \end{array}$$

$$\begin{array}{c} F | T \wedge F \\ \swarrow \quad \searrow \\ (F | T) \wedge F \\ T \wedge F = F \end{array}$$

$$\begin{array}{c} F | (T \wedge F) \\ F | F = F \end{array}$$

	γT	γF
δT	T	F
δF	F	F

	γT	γF
δT	T	T
δF	T	F

	γT	γF
δT	F	T
δF	T	F

$$\text{ans}(\delta, \gamma) T$$

$$= \delta T * \gamma T + \delta F * \gamma T + \delta T * \gamma F$$

$$\text{ans}(\delta, \gamma) T$$

$$= \delta T * \gamma F + \delta F * \gamma T$$

$$\text{ans}(\delta, \gamma) F$$

$$= \delta F * \gamma F$$

$$\text{ans}(\delta, \gamma) F$$

$$= \delta T * \gamma T + \delta F * \gamma F$$

$$\begin{aligned} \text{ans}(\delta, \gamma) F \\ = \delta T * \gamma F + \delta F * \gamma T \\ + \delta T * \gamma F \end{aligned}$$

$$0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \leftarrow (4,1)$$

T | T & F \wedge T

$$|*0 + |*1, |*1 + |*0 = (1, 1)$$

$$1 \neq 2 + 0 \times 2 + 1 \times 0,$$

$$0 \neq 0$$

$$\in (2, 0)$$

$$(T) \mid (T \wedge F)$$

$$\begin{array}{c} \text{, } \partial \neq 1^1 \\ \downarrow \quad \downarrow \\ \left(\begin{matrix} 1, 0 \\ T \vdash T \end{matrix} \right) \& \left(\begin{matrix} 1, 0 \\ F \vdash T \end{matrix} \right) \end{array}$$

$$(T | T \& F) \wedge T$$

$$1^{\star 0} > 0^{\star 1} + 0^{\star 0} \\ + 1^{\star 1} = (0, 1)$$

1,0
T & (FNT)

$$\downarrow \quad 0 \neq 0 + 1 \neq 1, \quad 0 \neq 1 \\ (T^{\text{RF}}) \wedge T = 1, 0$$

$$1 \neq 0 + 1 \neq 1 + 0 \neq 0$$

$$\begin{array}{c} \text{E}(1,0) \\ \text{T} | (\text{TE}) \\ 1,0 \quad 0,1 \end{array}$$

$$(T|T) \& F$$

partitions \rightarrow odd indices (operators)

```
static int[] eval(int[] l, int[] r, char operator){
    long[] curr = {0, 0};

    if(operator == '&'){
        curr[0] = l[0] * r[0];
        curr[1] = l[0] * r[1] + l[1] * r[0] + l[1] * r[1];
    } else if(operator == '|'){
        curr[0] = l[0] * r[0] + l[0] * r[1] + l[1] * r[0];
        curr[1] = l[1] * r[1];
    } else if(operator == '^'){
        curr[0] = l[0] * r[1] + l[1] * r[0];
        curr[1] = l[0] * r[0] + l[1] * r[1];
    }

    curr[0] = curr[0] % 1003;
    curr[1] = curr[1] % 1003;

    return new int[]{(int)curr[0], (int)curr[1]};
}
```

```

static int[] helper(int l, int r, String str, int[][][] dp){
    if(l == r){
        char operand = str.charAt(l);
        if(operand == 'T') return new int[]{1, 0};
        return new int[]{0, 1};
    }

    if(dp[l][r][0] != -1) return dp[l][r];

    int[] res = {0, 0};

    // Create Partitions over Operators
    for(int k=l+1; k<r; k+=2){

        int[] left = helper(l, k - 1, str, dp);
        int[] right = helper(k + 1, r, str, dp);
        int[] curr = eval(left, right, str.charAt(k));
        res[0] = (res[0] + curr[0]) % 1003;
        res[1] = (res[1] + curr[1]) % 1003;
    }

    return dp[l][r] = res;
}

```

```
static int countWays(int N, String str){  
    int[][][] dp = new int[N + 1][N + 1][2];  
    for(int i=0; i<=N; i++){  
        for(int j=0; j<=N; j++){  
            dp[i][j][0] = -1;  
            dp[i][j][1] = -1;  
        }  
    }  
  
    return helper(0, N - 1, str, dp)[0];  
}
```

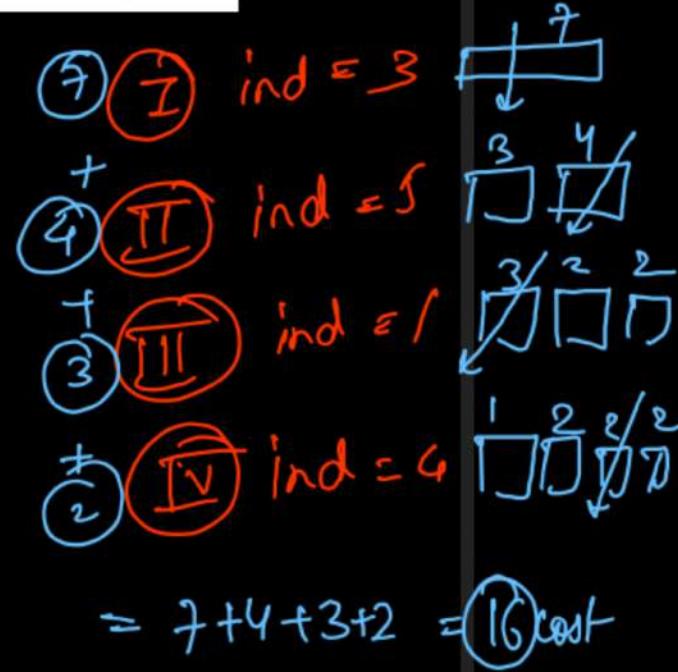
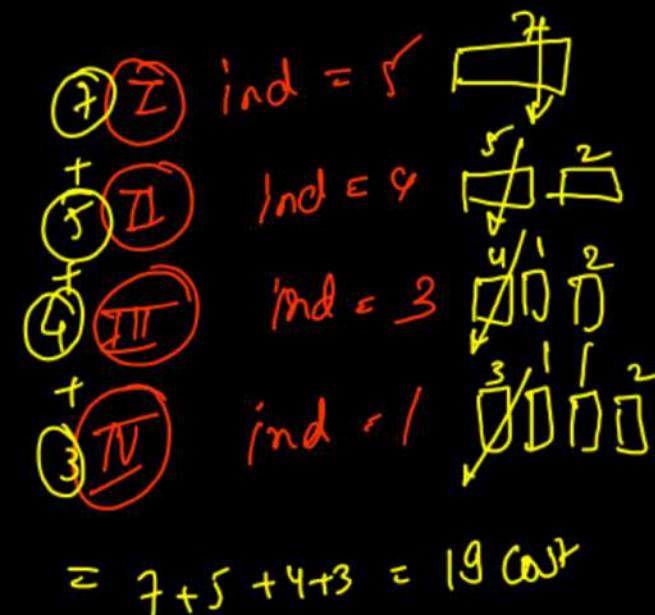
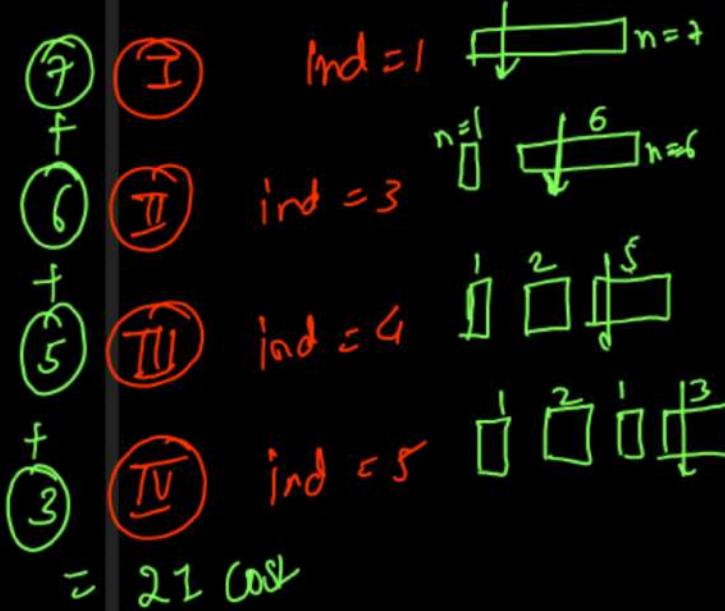
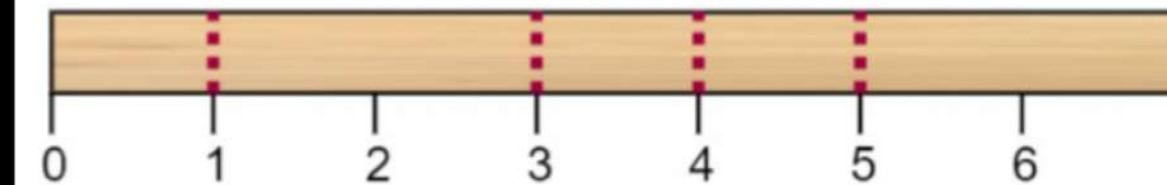
Time $\Theta(n^2 * n) = \Theta(n^3)$
 DP for loop
 # Space $\Theta(n^2)$ 2D DP
 $\Theta(n)$ R.C.S.

Minimum Cost To Cut Stick

→ Partition DP

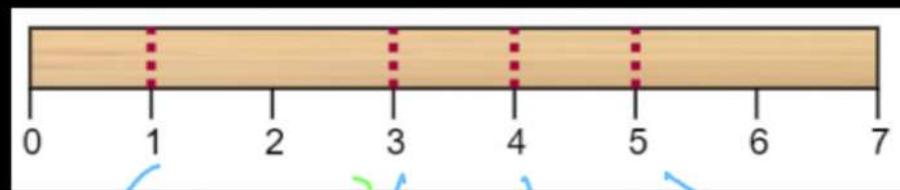
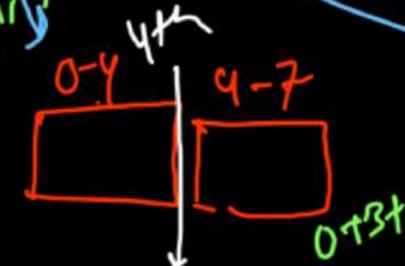
$$n=7 \\ \{1, 3, 4, 5\}$$

cuts = [3, 5, 1, 4] (Optimal Ordering)

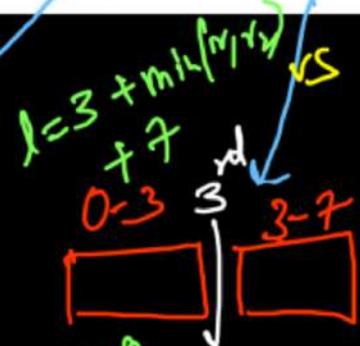
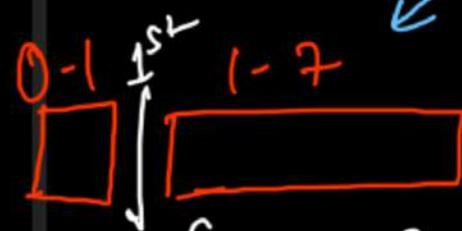


$$\{1, 3, 4, 5\}, n=7$$

$$l = \min(l_1, l_2) + 0 + 7$$



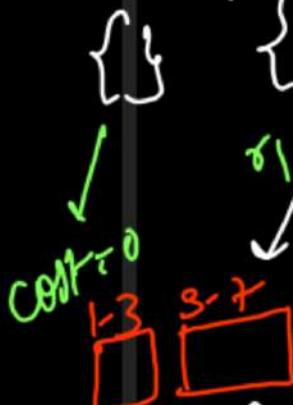
$$l = 0 + \min(l_1, l_2) + 7$$



$$0 + 3 + 0$$



$$l=0 \quad r=0$$

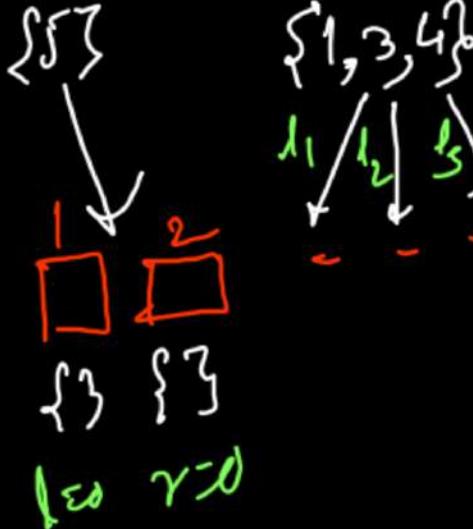


$$\{3\} \quad \{4, 7\}$$



$$\{3\} \quad \{5\}$$

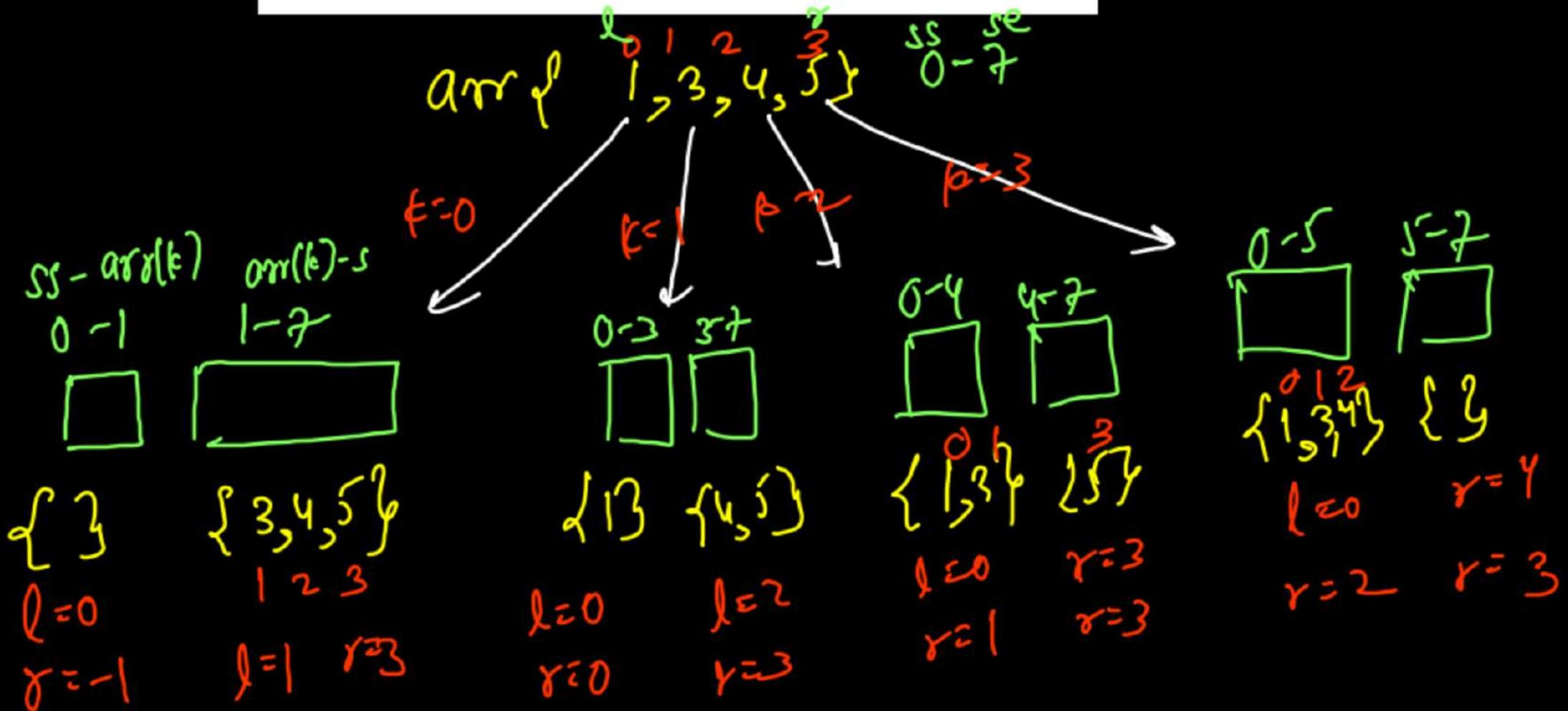
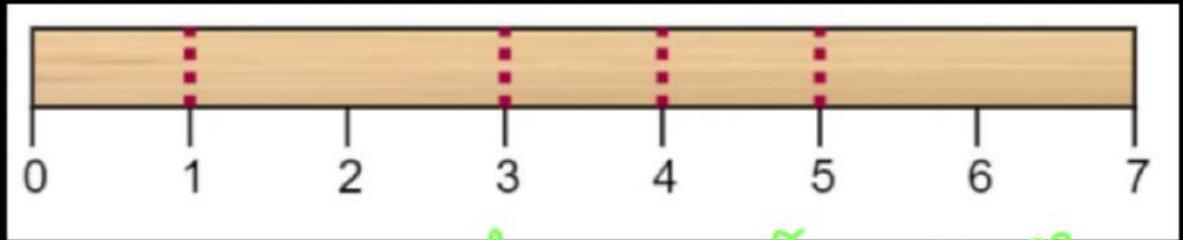
$$l=0 \quad r=0$$



$$l=0 \quad r=0$$

$$\{1, 3, 4\} \quad \{5\}$$

$$\text{cost} = 0$$



```
public int minCost(int ss, int se, int[] arr, int l, int r){  
    if(l > r) return 0; // No Partitions  
  
    int minCost = Integer.MAX_VALUE;  
    // Partitions  
    for(int k=l; k<=r; k++){  
        int left = minCost(ss, arr[k], arr, l, k - 1);  
        int right = minCost(arr[k], se, arr, k + 1, r);  
        int cost = left + (se - ss) + right;  
        minCost = Math.min(minCost, cost);  
    }  
  
    return minCost;  
}  
  
public int minCost(int n, int[] cuts) {  
    Arrays.sort(cuts);  
    return minCost(0, n, cuts, 0, cuts.length - 1);  
}
```

Time → Exponential
Space → O(Cuts)
R.C.S.

```

public int minCost(int ss, int se, int[] arr, int l, int r, int[][] dp){
    if(l > r) return 0; // No Partitions
    if(dp[l][r] != -1) return dp[l][r];

    int minCost = Integer.MAX_VALUE;
    // Partitions
    for(int k=l; k<=r; k++){
        int left = minCost(ss, arr[k], arr, l, k - 1, dp);
        int right = minCost(arr[k], se, arr, k + 1, r, dp);
        int cost = left + (se - ss) + right;
        minCost = Math.min(minCost, cost);
    }

    return dp[l][r] = minCost;
}

public int minCost(int n, int[] cuts) {
    Arrays.sort(cuts);

    int[][] dp = new int[cuts.length + 1][cuts.length + 1];
    for(int i=0; i<dp.length; i++)
        for(int j=0; j<dp[0].length; j++)
            dp[i][j] = -1;

    return minCost(0, n, cuts, 0, cuts.length - 1, dp);
}

```

~~Time~~
 $\rightarrow O(cuts^2 * cuts)$
 $= O(cuts^3)$

~~Space~~
 $\rightarrow O(cuts^2)$ ~~2D DP~~
 $\rightarrow O(cuts)$ ~~RxC~~

Word Break

- ① Word Break - I LC 139
- ② Word Break - II LC 140
- ③ Text Justification → Greedy LC 68
- ④ Text Justification → DP
 {Word Wrap} QFG
- ⑤ Scramble String

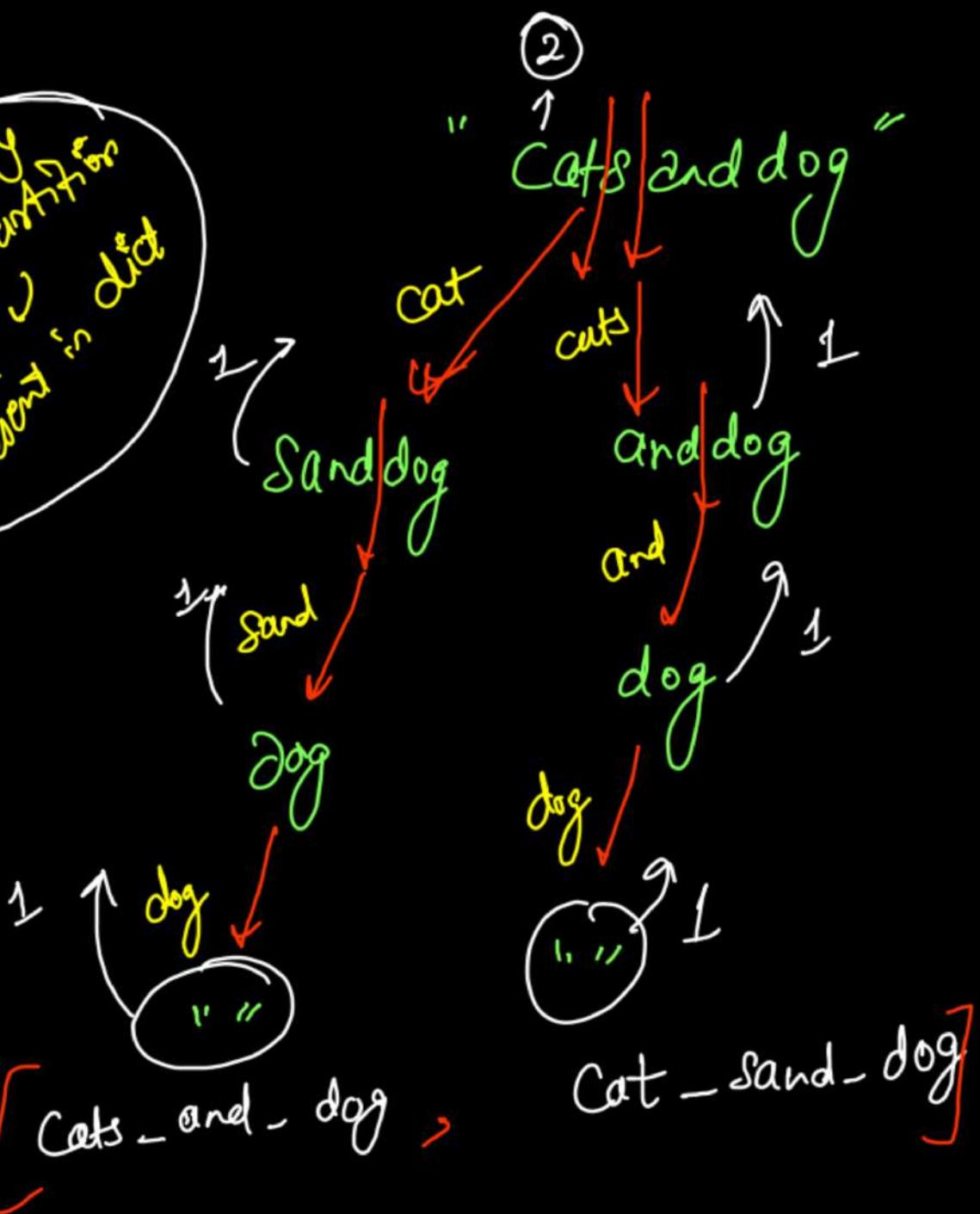
Word Break - I

"apple/pen/apple" True {apple, pen}

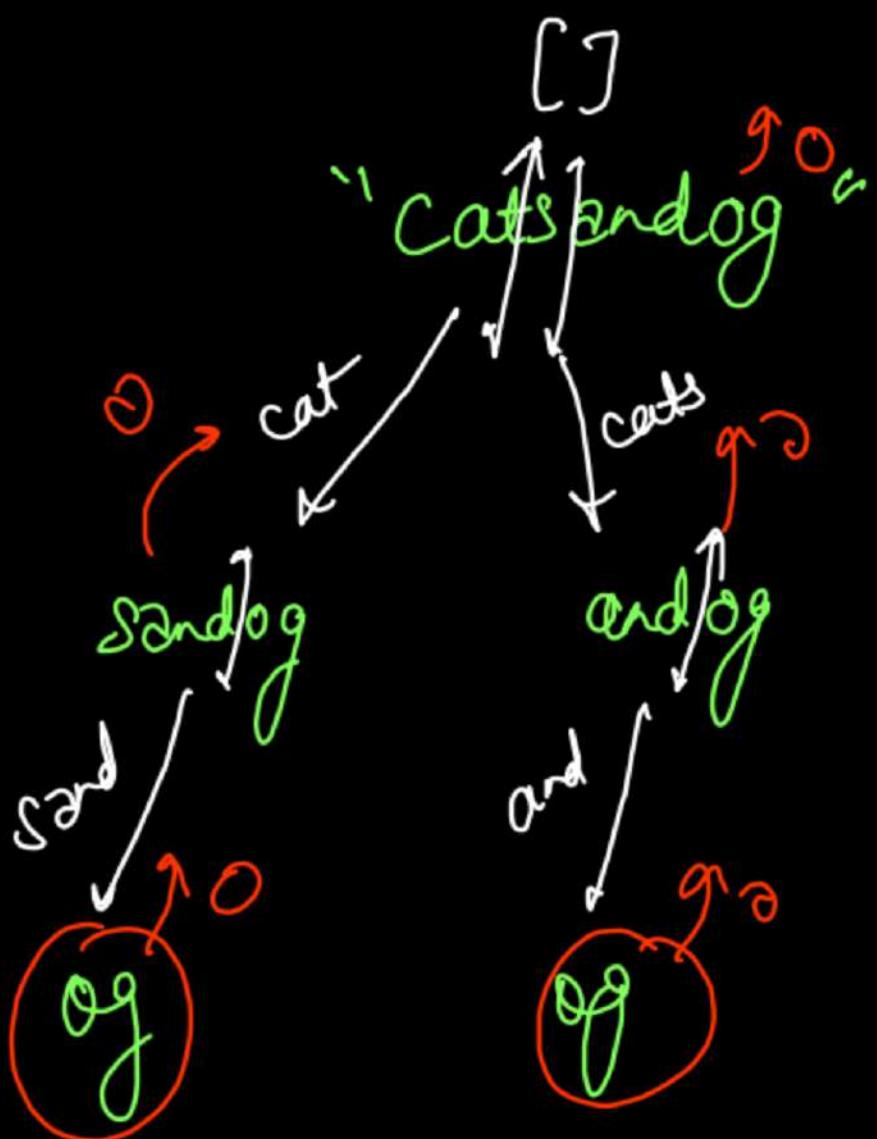
"apple" True

"cats and dog" False {cats, dog, sand, cat}

Cheeky
 Upstart
) did
 Back in
 is present in



{cats, dog, cat, and, sand}
 ↑ ↑ ↑ ↑ ↑
 → LS $\rightarrow O(D) \propto L$
 → BS $\rightarrow O(\log_2 D) \propto L$
 ↳ HashSet (unordered_set)
 ↳ O(1)
 → trie $\rightarrow O(L)$



$\{ \text{cats, dog, cat, and, sand} \}$
 ↑ ↑ ↑ ↑ ↑
 $\rightarrow \text{LS} \rightarrow O(1)$ $\rightarrow \text{BS} \rightarrow O(\log_2 D)$
 $\rightarrow \text{Hashset (unordered_set)}$
 $\hookrightarrow O(1)$
 - fric

"aaaaa[aab]"

{ a, aa, aaa, aaaa

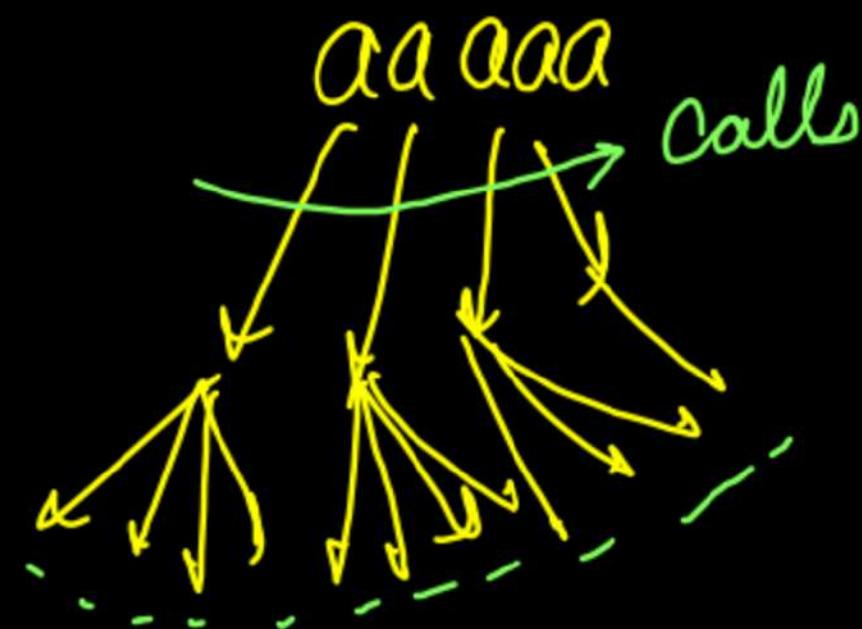
a, a, a, a → false

aa, aa → false

a, aaa → false

a, aa, a → false

aaa, a → false



Exponential
⇒ $(N)^N$

```

HashSet<String> dict;

public int wordBreak(int i, String s, int[] dp){
    if(i == s.length()) return 1;
    if(dp[i] != -1) return dp[i];

    for(int j=i; j<s.length(); j++){
        String prefix = s.substring(i, j + 1);
        if(dict.contains(prefix) == true && wordBreak(j + 1, s, dp) == 1)
            return dp[i] = 1;
    }

    return dp[i] = 0;
}

public boolean wordBreak(String s, List<String> wordDict) {
    dict = new HashSet<>();
    int[] dp = new int[s.length() + 1];
    Arrays.fill(dp, -1);

    for(String str: wordDict) dict.add(str);

    return (wordBreak(0, s, dp) == 1) ? true : false;
}

```

Time

$\hookrightarrow O(N \times N) = O(N^2)$

Space

1D DP $\rightarrow O(N)$

R.C.S $\rightarrow O(N)$

Word Break II

```
HashSet<String> dict;
List<String> res;

public void wordBreak(int i, String s, String curr){
    if(i == s.length()) {
        res.add(curr.substring(0, curr.length() - 1));
        return;
    }

    for(int j=i; j<s.length(); j++){
        String prefix = s.substring(i, j + 1);
        if(dict.contains(prefix) == true)
            wordBreak(j + 1, s, curr + prefix + " ");
    }
}

public List<String> wordBreak(String s, List<String> wordDict) {
    dict = new HashSet<>();
    for(String str: wordDict) dict.add(str);

    res = new ArrayList<>();
    wordBreak(0, s, "");
    return res;
}
```

Time
Exponential

Space
① $O(D)$ HashSet
② $O(N)$ R.C.S

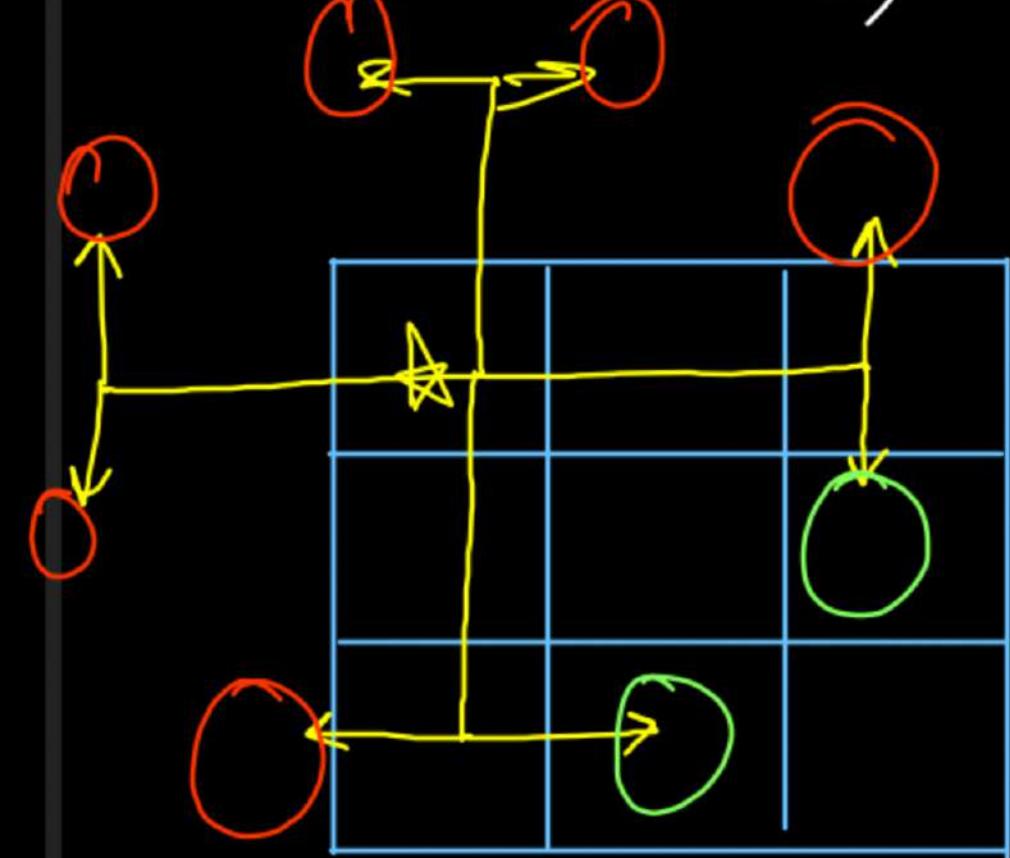
~~HC 688~~ Knight's Probability in chess

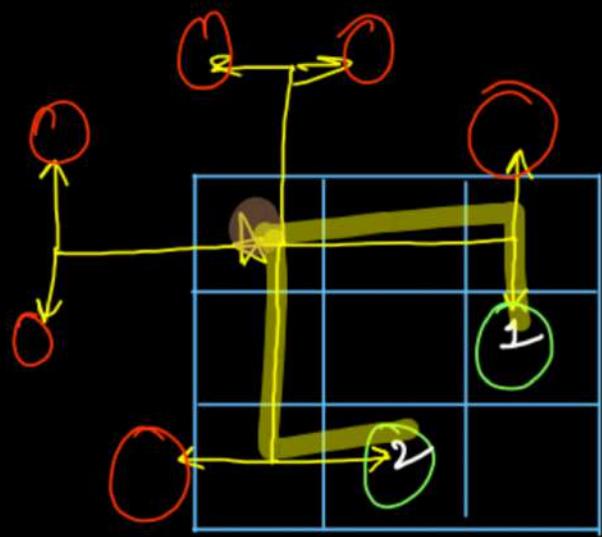
Starting from (r, c) in $n \times n$ board

after k moves

knights probability in chess

$k=2$ moves



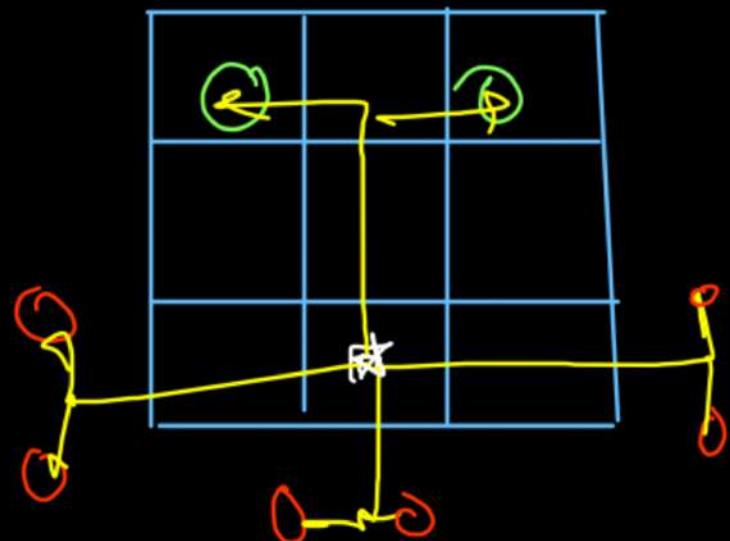
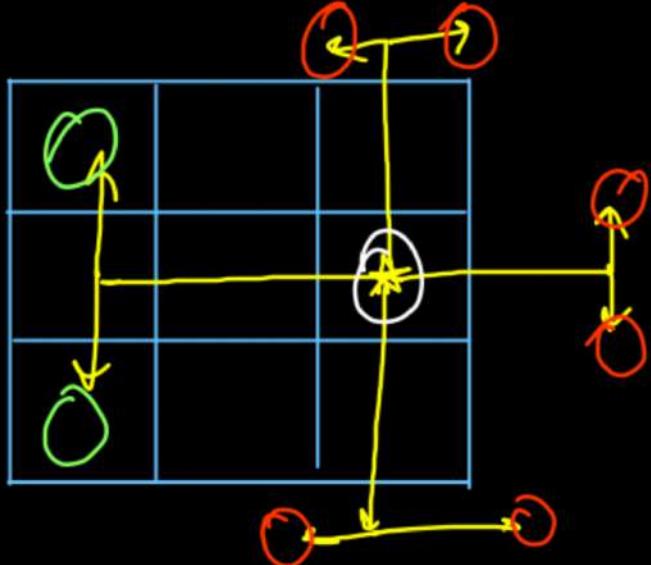


$$\begin{aligned}
 & \frac{1}{8} * \frac{2}{8} + \frac{1}{8} * \frac{2}{8} \\
 & \cancel{\frac{2}{8}} * \left(\frac{2}{8} + \frac{2}{8} \right) \\
 & \frac{2}{8} * \left(\frac{2}{8} + \frac{2}{8} \right)
 \end{aligned}$$

$$= \frac{1}{4} * \frac{1}{4} = \frac{1}{16}$$

$$= 0.0625$$

$\frac{2}{8}$



$\frac{2}{8}$

```

int[][] moves = {{+1, +2}, {+1, -2}, {-1, +2}, {-1, -2},
                 {+2, +1}, {+2, -1}, {-2, +1}, {-2, -1}};

public double helper(int n, int k, int r, int c, double[][][] dp){
    if(k == 0) return 1.0;
    if(dp[r][c][k] != -1.0) return dp[r][c][k];

    double prob = 0.0;
    for(int m=0; m<8; m++){
        int nr = r + moves[m][0];
        int nc = c + moves[m][1];

        if(nr >= 0 && nr < n && nc >= 0 && nc < n){
            prob = prob + ((1.0 / 8.0) * helper(n, k - 1, nr, nc, dp));
        }
    }

    return dp[r][c][k] = prob;
}

```

Time $\Theta(n \times n \times k \times 8)$

```

public double knightProbability(int n, int k, int r, int c) {
    double[][][] dp = new double[n + 1][n + 1][k + 1];
    for(int i=0; i<=n; i++){
        for(int j=0; j<=n; j++){
            for(int l=0; l<=k; l++){
                dp[i][j][l] = -1.0;
            }
        }
    }

    return helper(n, k, r, c, dp);
}

```

Space $O(n \times n \times k)$
 3D DP

Dynamic Programming

Lecture 40

① → Word Wrap / Text Justification

Greedy (LC 68)
DP (GFG)

② → Scramble String (LC 87)

I (LC 118)

II (LC 119)

Binomial coeff ($nC_r \cdot m$) {QFG}

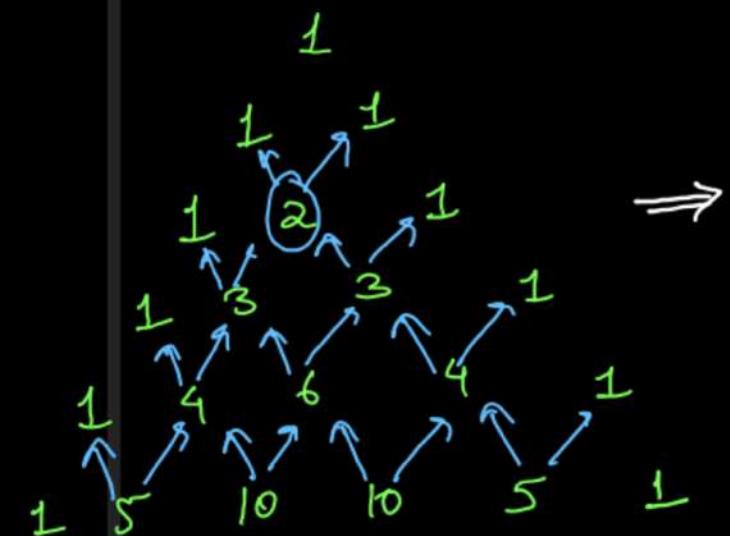
Water overflow {QFG}

④ → Maximal Square {LC 221}

Advanced Dynamic Programming

- Digit DP
- DP with Bitmasking → Travelling Salesman
- Sum over Subsets DP
- DP with Geometry → Convex hull
- DP with Game Theory

Pascal Triangle



$\text{mat}[i][j] = {}^i C_j$	1	1 1	1 2 1	1 3 3 1	1 4 6 4 1	1 5 10 10 5 1
	1	1 1	1 2 1	1 3 3 1	1 4 6 4 1	1 5 10 10 5 1
		2C ₀	2C ₁	2C ₂	2C ₃	2C ₄
		3C ₀	3C ₁	3C ₂	3C ₃	3C ₄
		4C ₀	4C ₁	4C ₂	4C ₃	4C ₄
		5C ₀	5C ₁	5C ₂	5C ₃	5C ₄

${}^0 C_0$	${}^1 C_0$	${}^1 C_1$	
${}^2 C_0$	${}^2 C_1$	${}^2 C_2$	
${}^3 C_0$	${}^3 C_1$	${}^3 C_2$	${}^3 C_3$
${}^4 C_0$	${}^4 C_1$	${}^4 C_2$	${}^4 C_3$
${}^5 C_0$	${}^5 C_1$	${}^5 C_2$	${}^5 C_3$

~~Recurrence Relation #~~ $nC_k = {}^{n-1}C_k + {}^{n-1}C_{k-1}$

$nC_k = nC_{n-k}$

Base case
 $nC_0 = nC_n = 1$

LeetCode 118

```

public List<List<Integer>> generate(int rows) {
    List<List<Integer>> pascal = new ArrayList<>();
    pascal.add(new ArrayList<>());
    pascal.get(0).add(1);

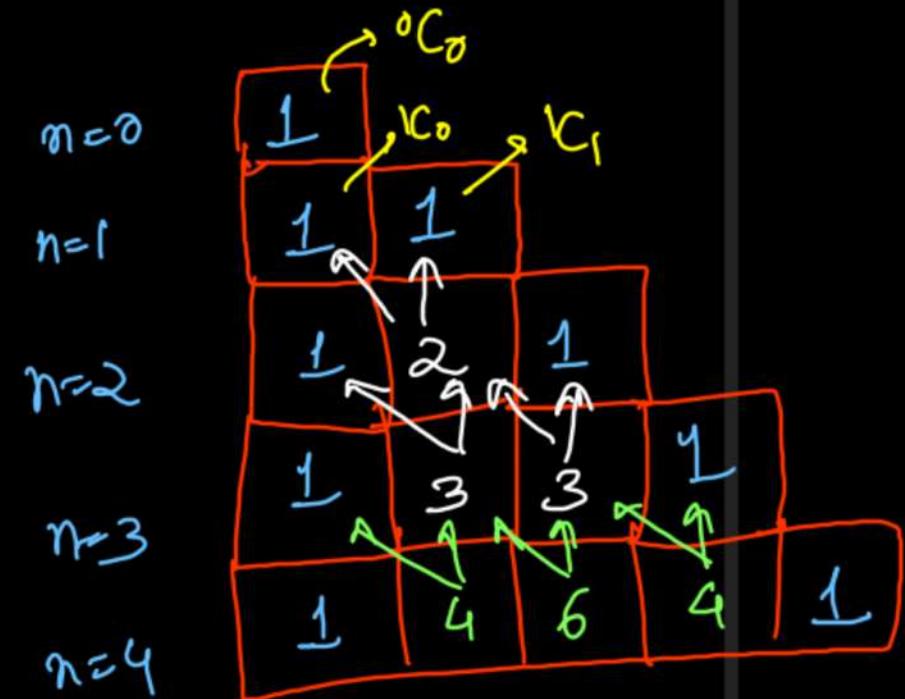
    for(int n=1; n<rows; n++){
        List<Integer> row = new ArrayList<>();
        row.add(1); // nC0

        for(int k=1; k<n; k++){
            // nCk = n-1Ck-1 + n-1Ck
            int nCk = pascal.get(n - 1).get(k) + pascal.get(n - 1).get(k - 1);
            row.add(nCk);
        }

        row.add(1); // nCn
        pascal.add(row);
    }

    return pascal;
}

```



Time : $O(n*k)$, Space $\rightarrow O(m*k)$ $2D DP$

$$\frac{nC_k \% \text{mod}}{=}$$

$nC_0 = 1$

$nC_r = nC_{r-1}$

$nC_0 = 0$

```
if(cols > rows) return 0;
if(rows == 0 || rows == cols || cols == 0) return 1;
```

```
List<Integer> prev = new ArrayList<>();
prev.add(1);

for(int n=1; n<=rows; n++){
    List<Integer> curr = new ArrayList<>();
    curr.add(1); // nC0

    for(int k=1; k<n; k++){
        // nCk = n-1Ck-1 + n-1Ck
        int nCk = (prev.get(k) + prev.get(k - 1)) % 1000000007;
        curr.add(nCk);
    }

    curr.add(1); // nCn
    prev = curr;
}

return prev.get(cols);
```

Time $\rightarrow O(n \times k)$

Space $\rightarrow O(n)$

1D DP

leetCode (19)

```
public List<Integer>getRow(int rows) {  
    List<Integer> prev = new ArrayList<>();  
    prev.add(1);  
  
    for(int n=1; n<=rows; n++){  
        List<Integer> curr = new ArrayList<>();  
        curr.add(1); // nC0  
  
        for(int k=1; k<n; k++){  
            // nCk = n-1Ck-1 + n-1Ck  
            int nCk = prev.get(k) + prev.get(k - 1);  
            curr.add(nCk);  
        }  
  
        curr.add(1); // nCn  
        prev = curr;  
    }  
  
    return prev;  
}
```

{ 1 }

{ 1, 1 }

{ 1, 2, 1 }

prev → { 1, 3, 3, 1 }

curr →

Time → $O(n \cdot k)$
Space → $O(n)$ DDP

Get the Last Row

```
public List<Integer> getRow(int rows) {  
    List<List<Integer>> pascal = new ArrayList<>();  
    pascal.add(new ArrayList<>());  
    pascal.get(0).add(1);  
  
    for(int n=1; n<=rows; n++){  
        List<Integer> row = new ArrayList<>();  
        row.add(1); // nC0  
  
        for(int k=1; k<n; k++){  
            // nCk = n-1Ck-1 + n-1Ck  
            int nCk = pascal.get(n - 1).get(k) + pascal.get(n - 1).get(k - 1);  
            row.add(nCk);  
        }  
  
        row.add(1); // nCn  
        pascal.add(row);  
    }  
  
    return pascal.get(rows);  
}
```

Time $\rightarrow O(n \times k)$

Space $\rightarrow O(n \times k)$

2D DP

Water overflow

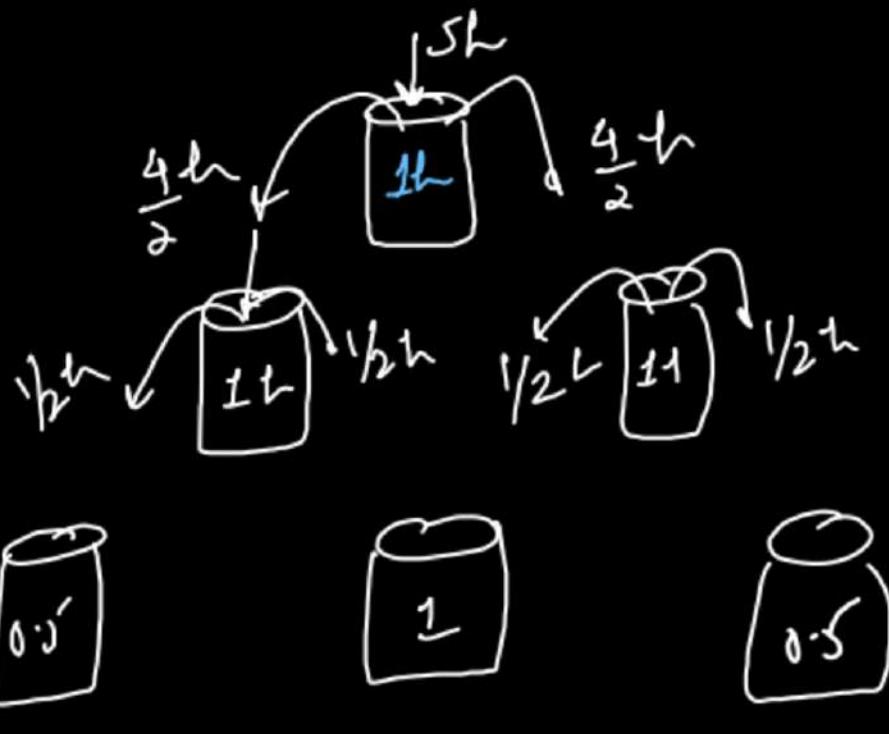
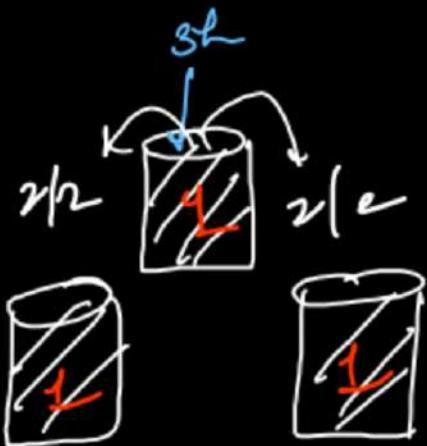
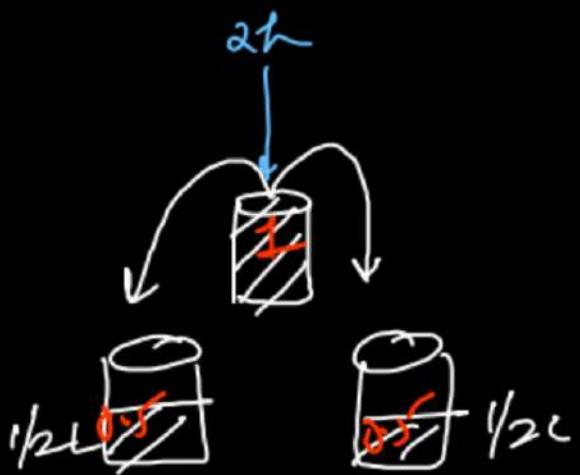


Diagram illustrating a 5x5 grid container with various volume calculations:

6h				
1h		2.5h	1h	1h
2.5h		1h	0.75h	0.75h
1h		1h	0.75h	0.75h
0.75h	0.75h	1h	0.25h	0.25h
0	0.25h	0.25h	0.25h	0.25h
0	0	0	0	0
0	0	0	0	0

volume > 1

↳ our container $\rightarrow 1$

↳ $(\text{volume} - 1)/2 \rightarrow \text{down}$

↳ $(\text{volume} - 1)/2 \rightarrow \text{downright}$

volume ≤ 1

↳ our container

↳ volume

```

static double waterOverflow(int K, int R, int C) {
    double[][] pascal = new double[501][501];
    pascal[0][0] = K;

    for(int i=0; i<R; i++){
        for(int j=0; j<=i; j++){ → overflow
            if(pascal[i][j] > 1.0){
                pascal[i + 1][j] += (pascal[i][j] - 1.0) / 2.0;
                pascal[i + 1][j + 1] += (pascal[i][j] - 1.0) / 2.0;
                pascal[i][j] = 1.0;
            }
        }
    }

    return pascal[R - 1][C - 1];
}

```

$T_{\text{Time}} \rightarrow O(R * C)$

$S_{\text{Space}} \rightarrow O(R * C)$

Word Wrap

hc

This is an example of Tent Justification

Greedy

T H I S I S A N
E X A M P L E O F
T E C H T
J U S T I F I C A T I O N

max width = 16

$\{ \underline{P} \underline{I} \underline{E} \quad \underline{I} \underline{S} \quad \underline{A} \underline{N} \quad \underline{N} \underline{U} \underline{M} \underline{B} \underline{E} \underline{R} \}$

$R=6$

$$1^2 + 4^2 = \textcircled{17}$$

$$3^2 + 2^2 = \textcircled{13}$$

<u>P</u>	<u>I</u>	<u>E</u>	<u>I</u>	<u>S</u>
<u>A</u>	<u>N</u>			
<u>N</u>	<u>U</u>	<u>M</u>	<u>B</u>	<u>E</u>
			<u>R</u>	

<u>P</u>	<u>I</u>	<u>E</u>			
<u>I</u>	<u>S</u>		<u>A</u>	<u>N</u>	
<u>N</u>	<u>U</u>	<u>M</u>	<u>B</u>	<u>E</u>	<u>R</u>

Greedy \rightarrow In each line, fit
as many words
as possible

Greedy will fail if
cost is $s_1^2 + s_2^2 + s_3^2$
 \hookrightarrow DP

P I E --- rem = 3

$$P + 4^2 + 1^2 \\ \neq 19$$

(same)

IS
(new)

$$3^2 + 1^2 + 1^2 \\ IS$$

$$4^2 + 4^2 + 3^2$$

P I E - I S
rem = 0

AN(same)

↓ AN(new)

$$4^2 + 8^2$$

P I E ---

I S ---

rem = 4

P + P
AN(same)

P I E ---

I S - A N ---

P I E ---

I S - A N ---

rem = 4

apple
(same)

apple
(new)

apple
(same)

apple(dif)

P I E - I S

A N
APPL E

P I E ---

I S - A N ---
APPL E

P I E ---
I S ---
A N ---
APPL E

```

public int helper(int idx, int rem, int len, int[] nums, int[][] dp){
    if(idx == nums.length) return 0;
    if(dp[idx][rem] != -1) return dp[idx][rem];

    int newRem = (rem == len) ? rem - nums[idx] : rem - nums[idx] - 1;
    int sameLine = (nums[idx] < rem) ? helper(idx + 1, newRem, len, nums, dp)
                                      : Integer.MAX_VALUE;

    int diffLine = (rem * rem) + helper(idx + 1, len - nums[idx], len, nums, dp);
    return dp[idx][rem] = Math.min(sameLine, diffLine);
}

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

    return helper(0, k, k, nums, dp);
}

```

Time $\rightarrow O(n*k)$

Space $\rightarrow O(n*k)$

2D DP

① DP → Category
Helper class works