Dynamic Programming 1

- Priyansh Agarwal

Why Dynamic Programming?

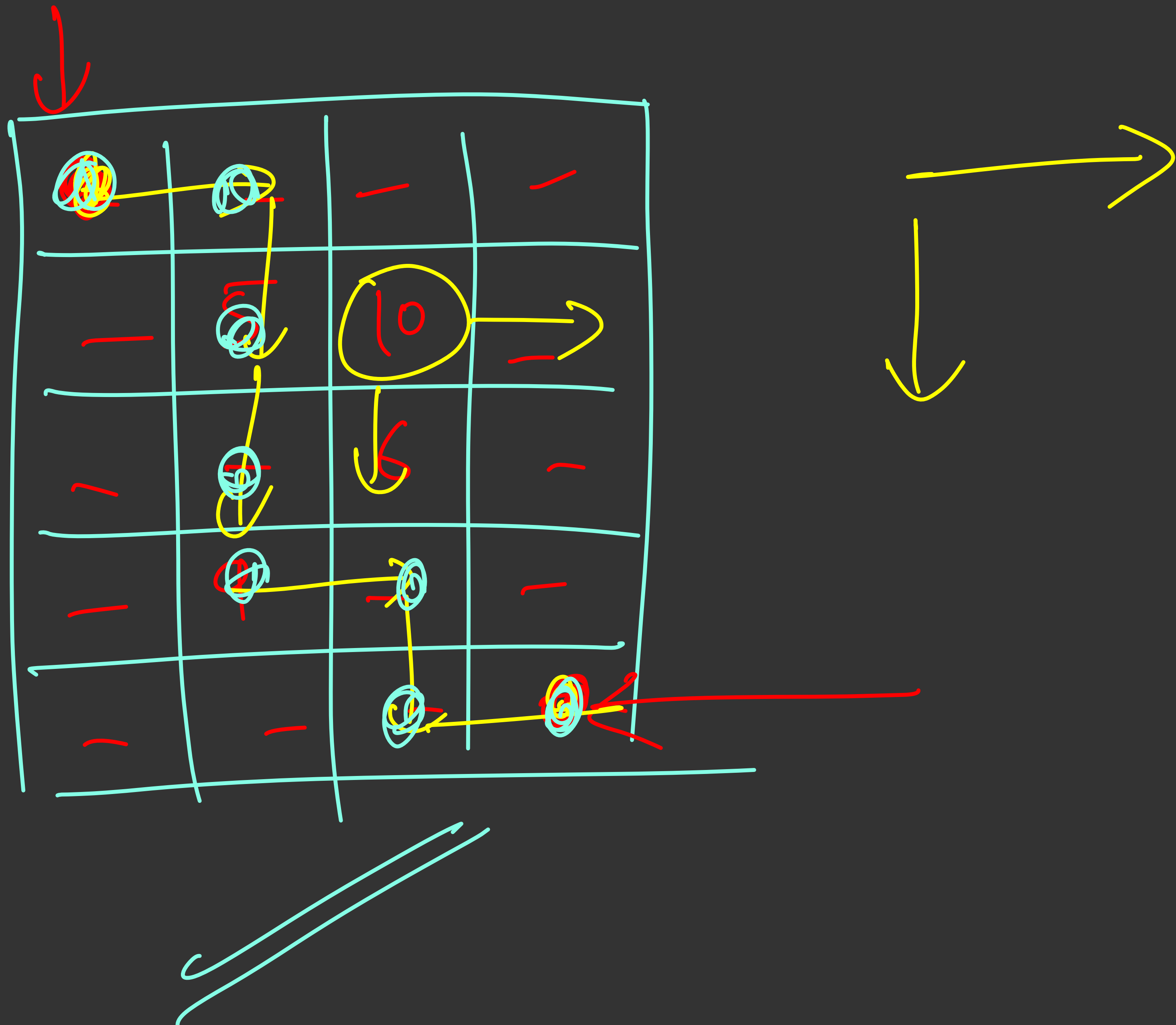
- Overlapping subproblems
- Maximize/Minimize some value
- Finding number of ways
- Covering all cases (DP vs Greedy)
- Check for possibility

$(10^5)!$

and take mod with 10^9+7

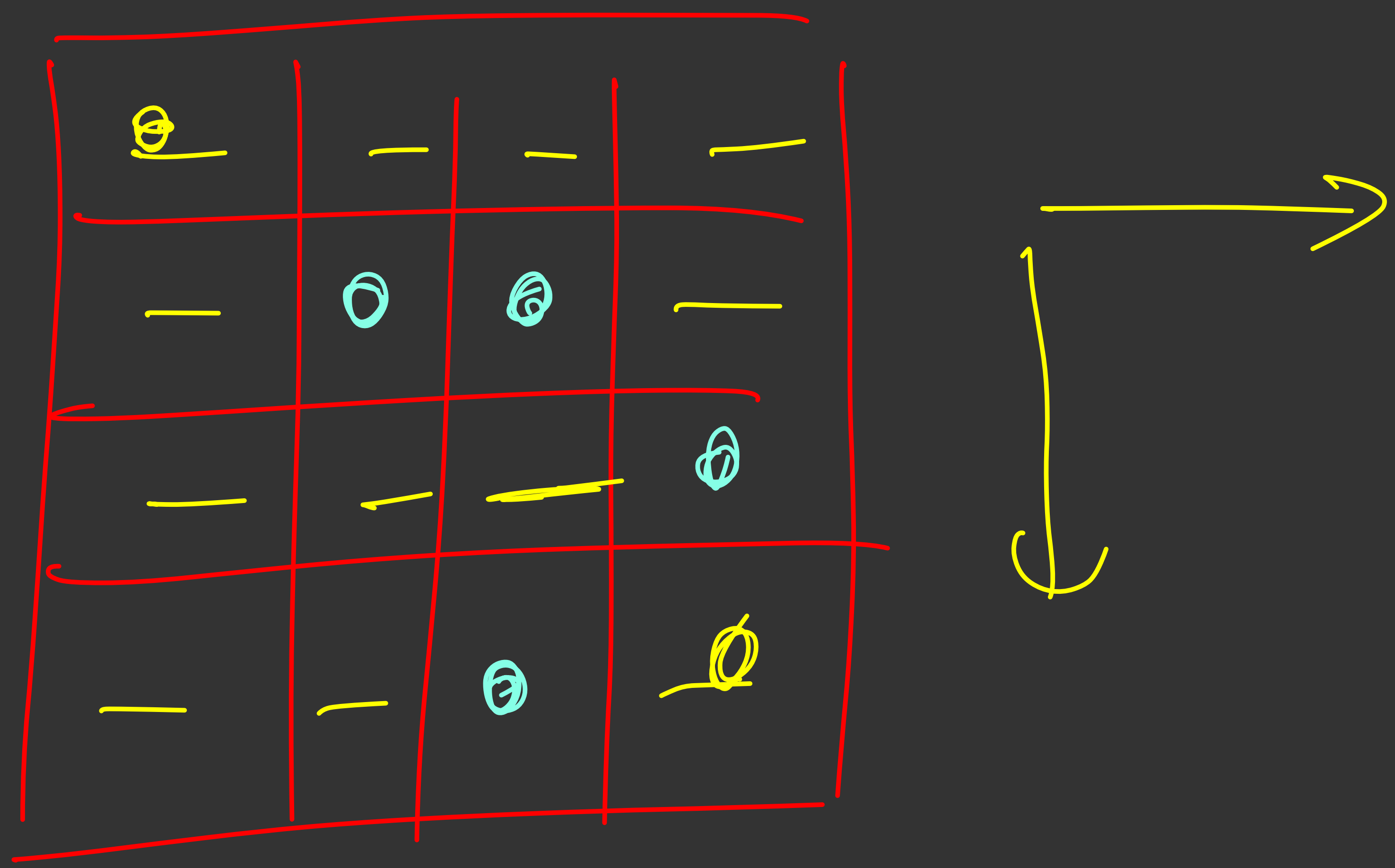
0 to 10^9+6

Optimized Brute force


10

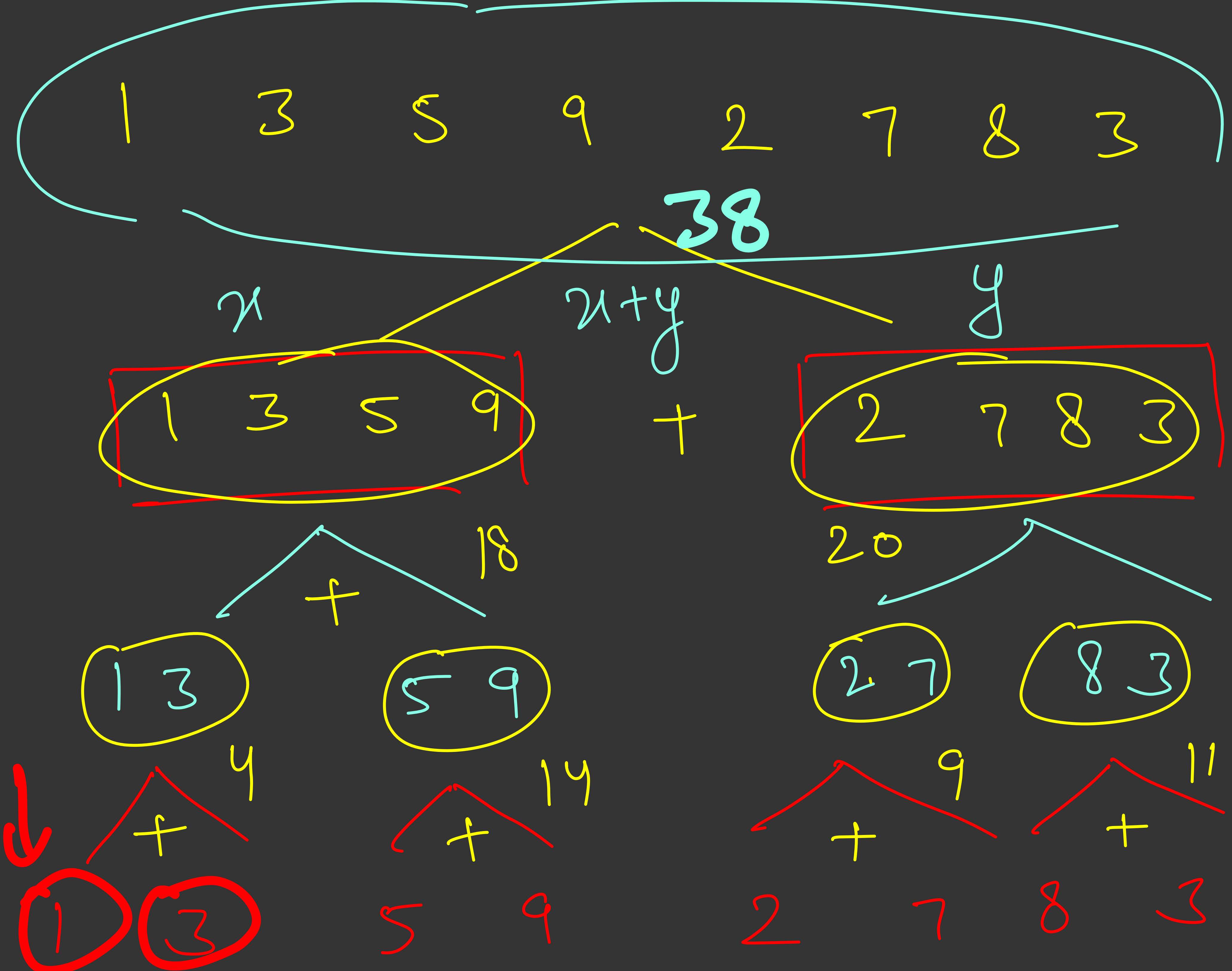
5	5	5	5
1	1	1	5
1	1	1	5
1	1	1	5
1	1	1	5

Greedy



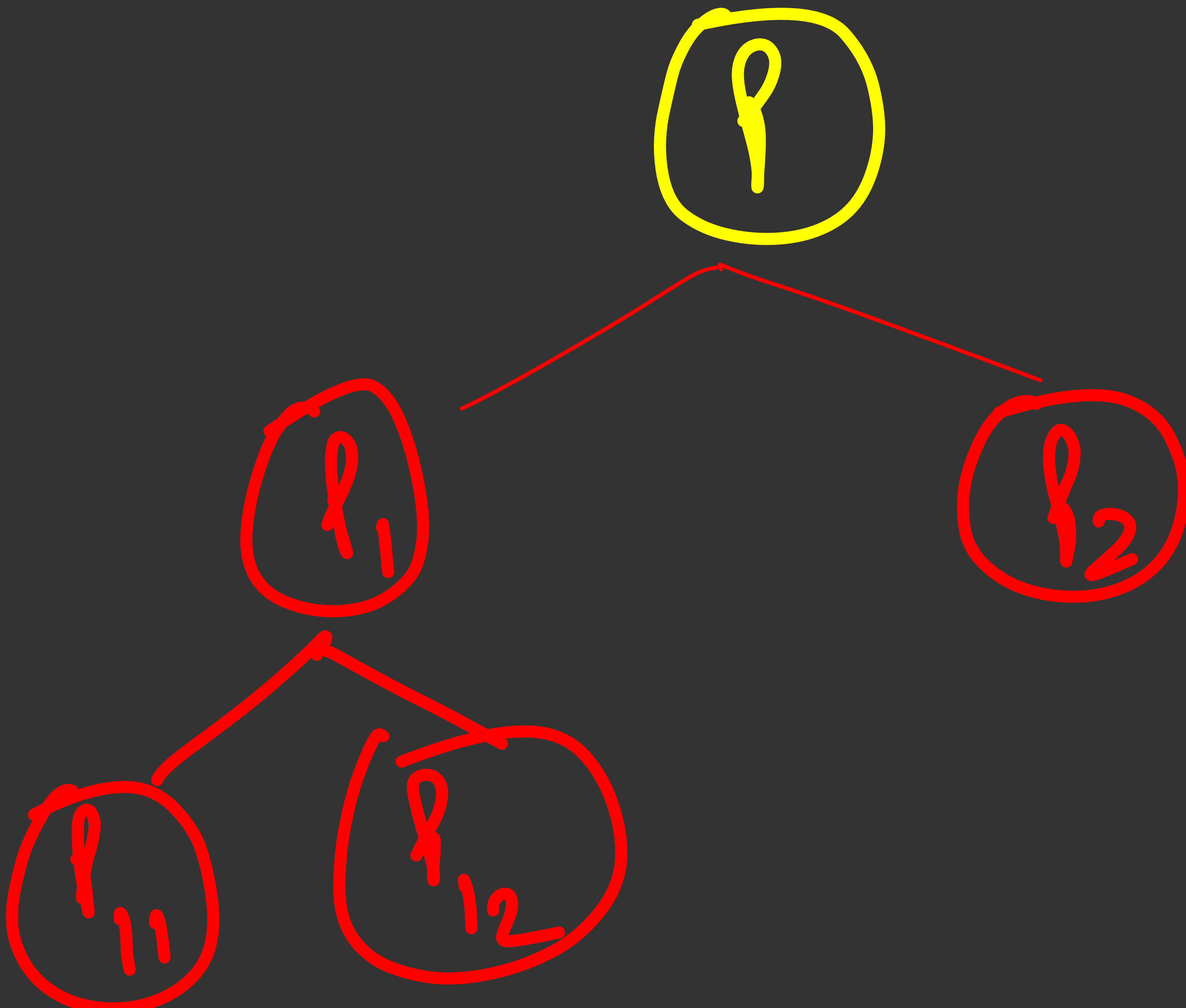
Mind set to solve DP Problems

Eg: Given an array find out the sum of all elements of array without iterating over the array



1 3 5 9 2 7 8 3
38
x
x+y
y
1 3 5 9
+
2 7 8 3
18
20
+
1 3
5 9
2 7
8 3
4
14
9
11
+
+
+
+
1 3
5 9
2 7
8 3

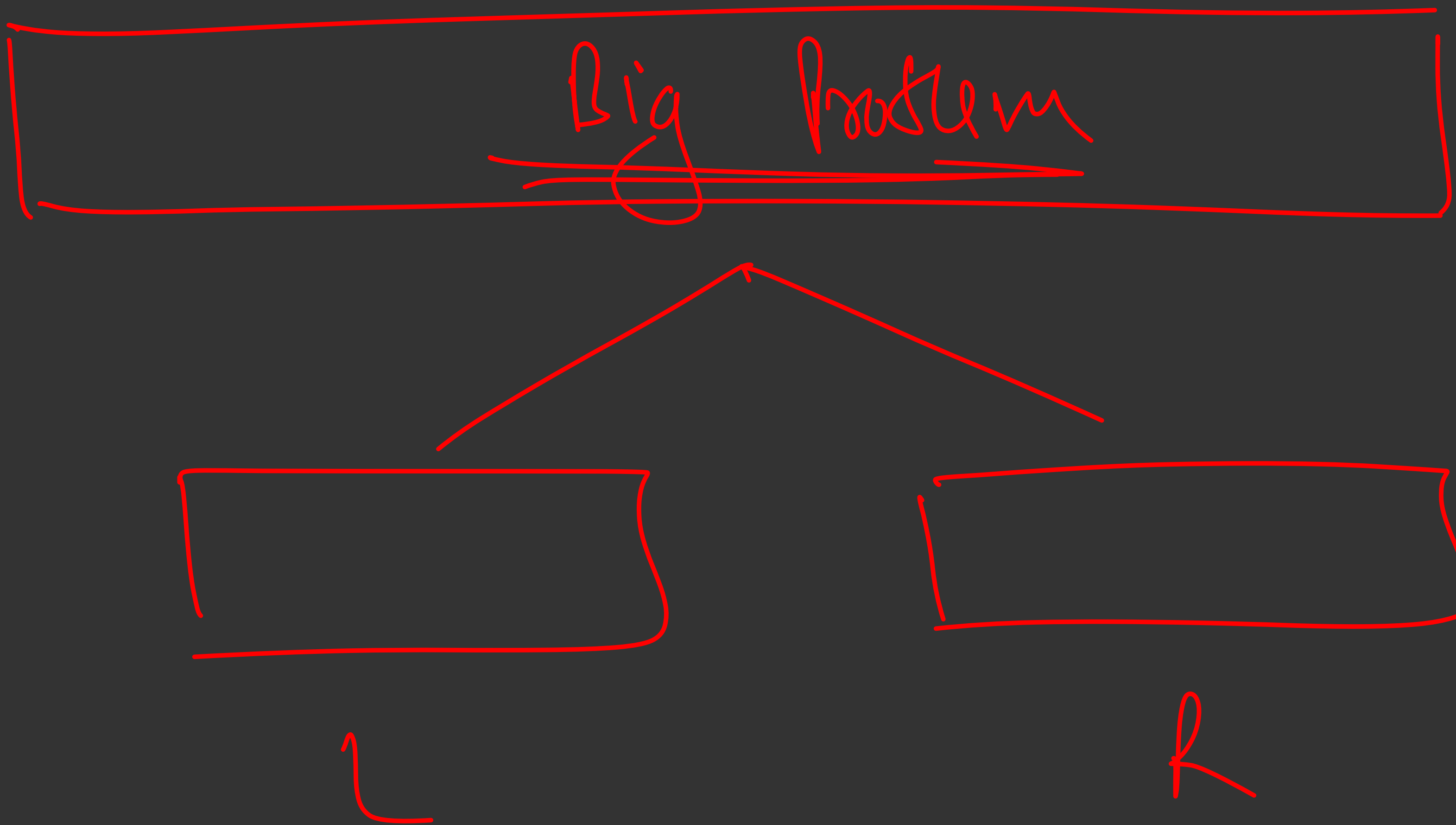
Divide & Conquer



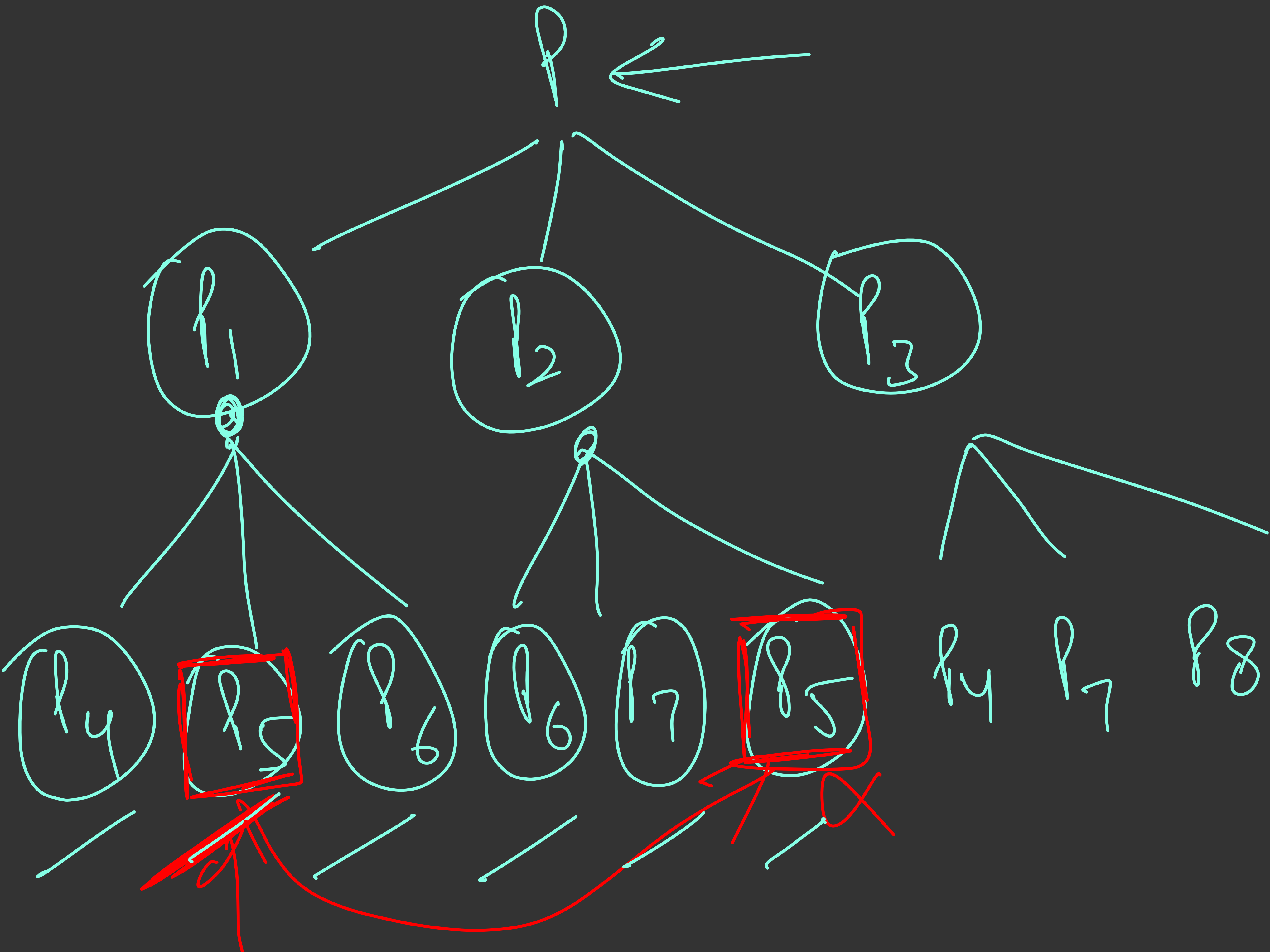
1. Divide A problem into smaller subproblems

2. Come with a relation b/w smaller subproblems to find ans for bigger subproblem

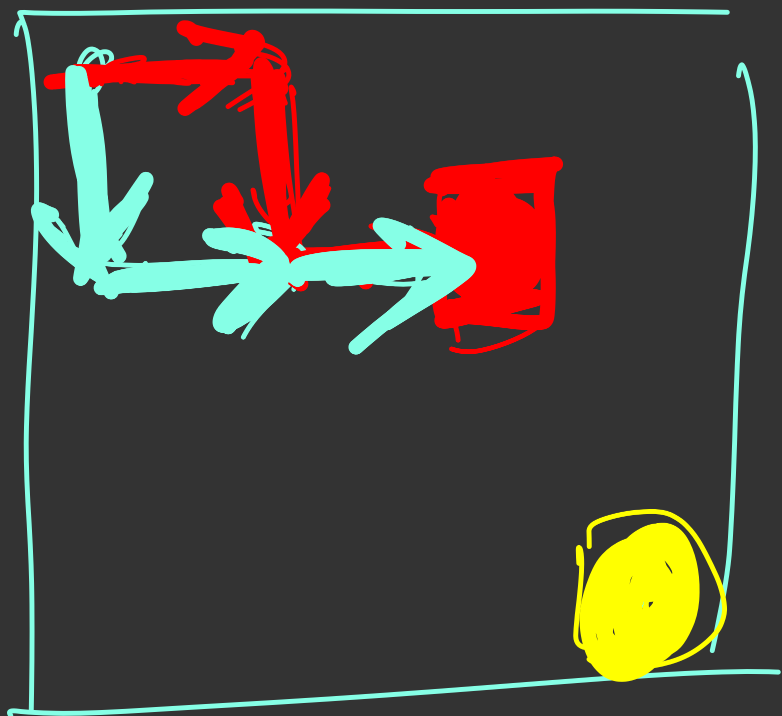
3. You define a trivial case

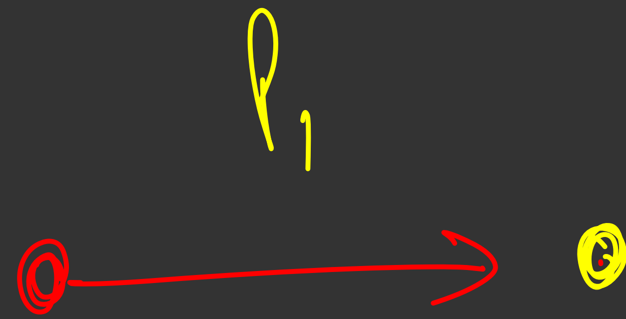


Big problem = Ans for L + Ans for R



P
P1
P2
P3
P4
P5
P6
P6
P7
P5
P4
P7
P8




P1

1. Sum of first 10 natural number

```
sum = 0
for (int i = 1 ; i <= 10 ; i++)
    sum += i
```

2. Sum of first 12 natural number

$f(x) =$ sum of first x natural no.s

$$f(x) = f(x-1) + x$$

$f(x)$

$+x$

$f(x-1)$

$x = 1$

$f(1) = 1$

$f(x)$ — $f(x-1)$ — $f(x-2)$ — $f(x-3)$

$f(5)$

$$\begin{array}{ccccccc} 15 & & 10 & & 6 & & 3 \\ f(5) & \xrightarrow{+5} & f(4) & \xrightarrow{+4} & f(3) & \xrightarrow{+3} & f(2) \end{array}$$

$$f(2) \xrightarrow{+2} \boxed{f(1)} \quad 1$$

$f(7)$

$$f(7) \xrightarrow{+7} f(6) \xrightarrow{+6} f(5)$$

$f(2) = 3$, $f(3) = 6$, $f(4) = 10$

$f(5) = 15$ $f(6) = 21$, $f(7)$

$= 28$

$$f(7) \xrightarrow{+7} f(6) \xrightarrow{+6} \boxed{f(5)}$$

$f(4) \times$

28

$$f(7) \xrightarrow{+7} f(6) \xrightarrow{+6} 15$$

21

Divide & Conquer + DP

98%

2%

Need of DP

- Let's understand this from a problem
  - Find nth fibonacci number
  - F(n) = F(n - 1) + F(n - 2)
  - F(1) = F(2) = 1

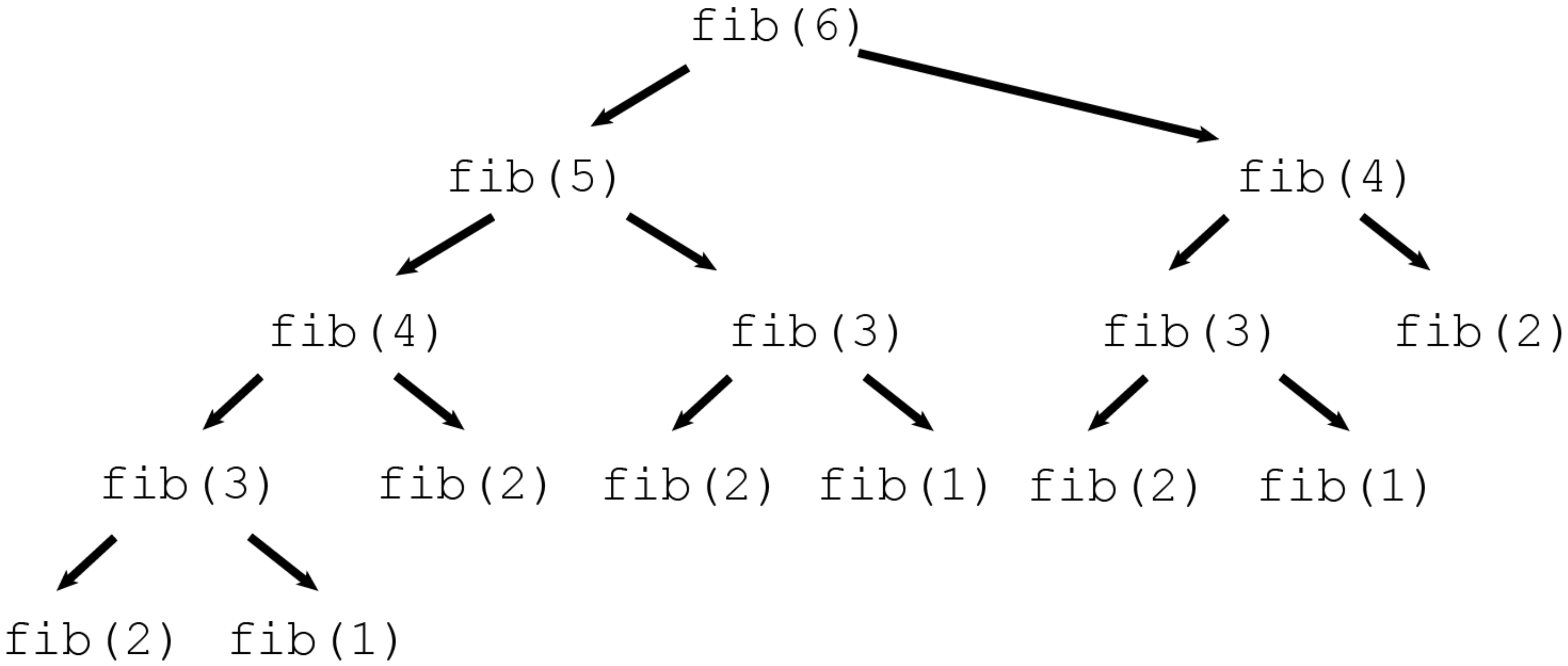
$f(x) = f(x-1) + x$

Biggest problem

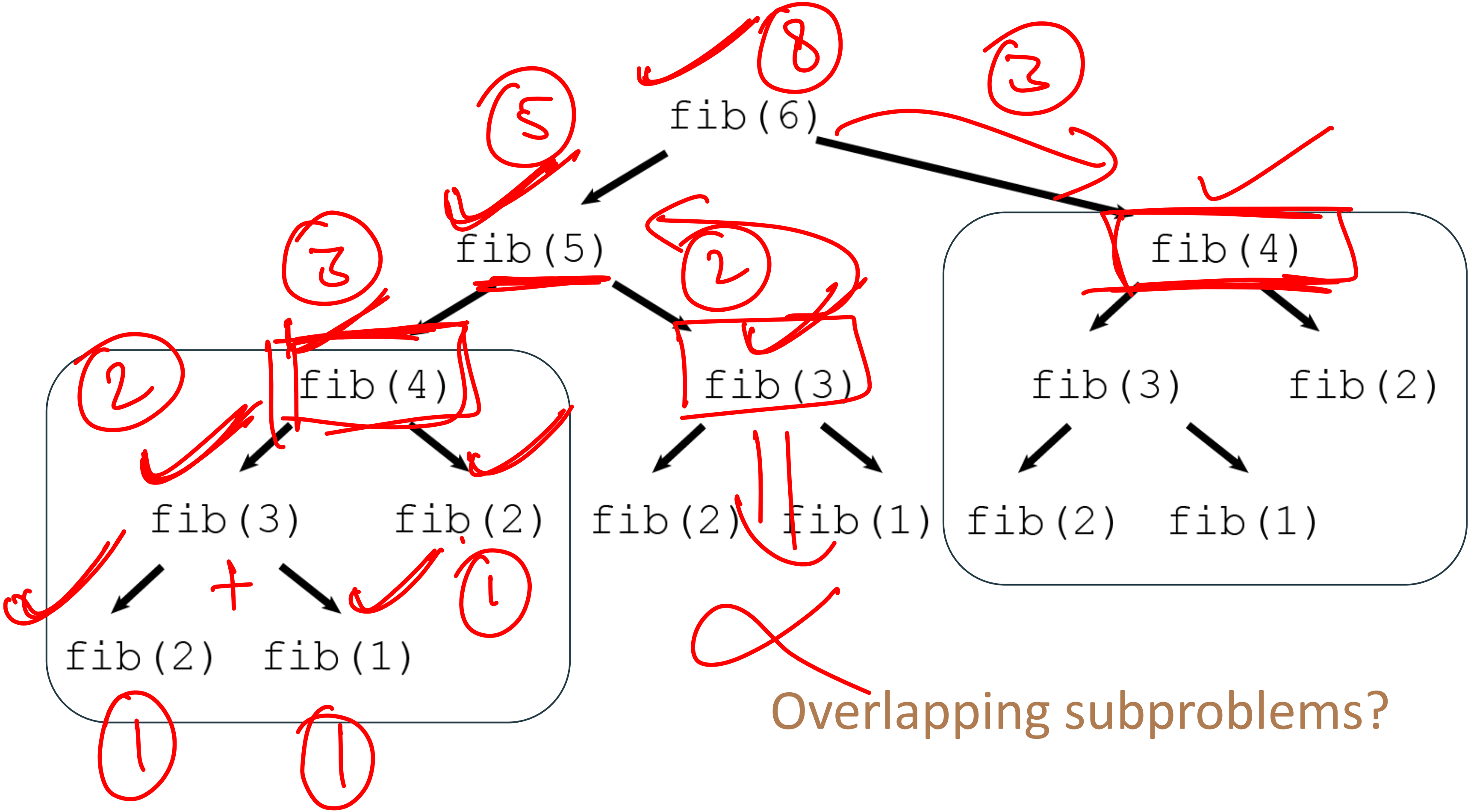
DP

Relation b/w smaller subproblems

Trivial cases



Any problem here?

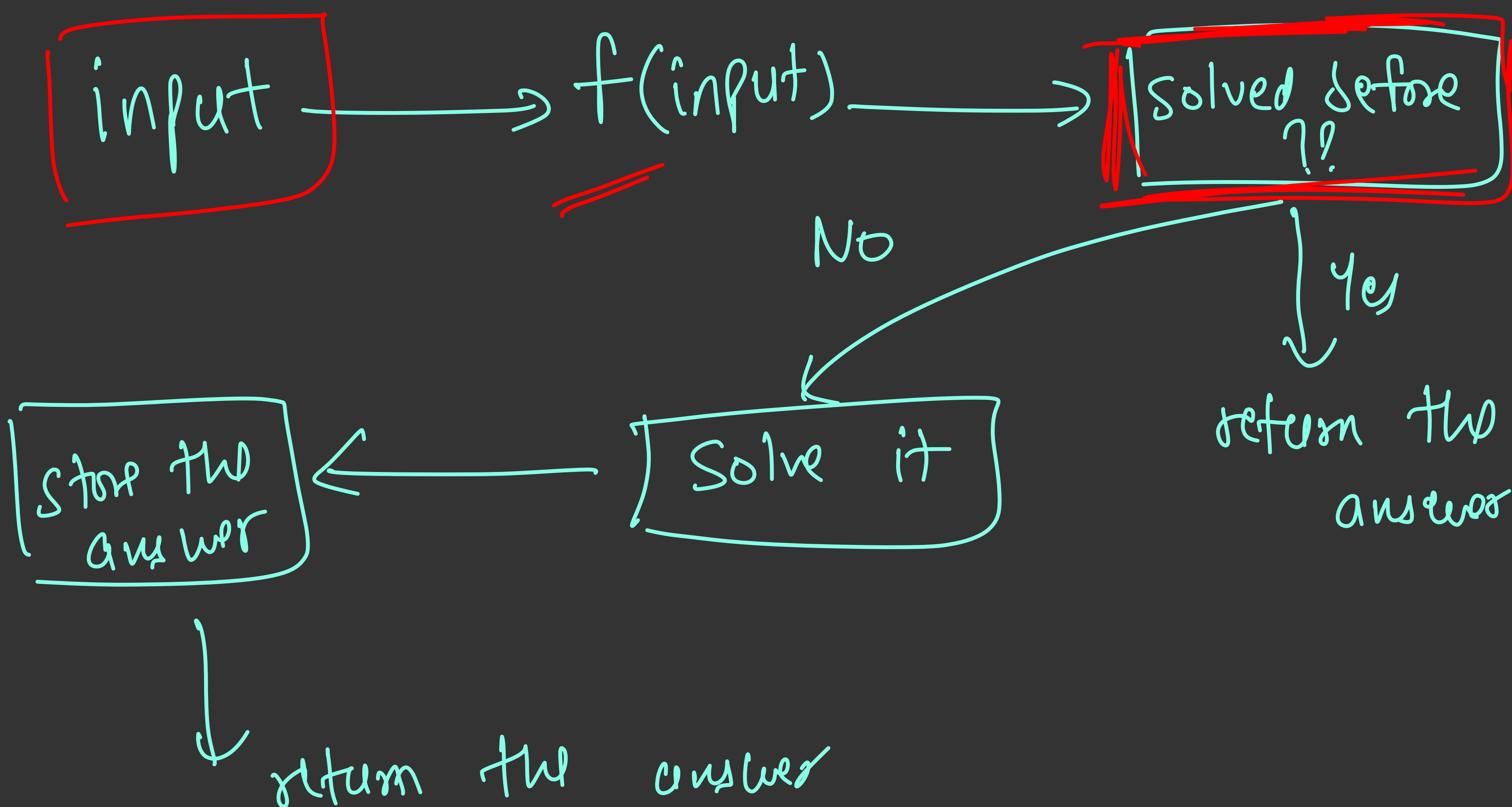


Overlapping subproblems?

Memoization

- Why calculate F(x) again and again when we can calculate it once and use it every time it is required?
  - Check if F(x) has been calculated
    - If No, calculate it and store it somewhere
    - If Yes, return the value without calculating again

Memoization



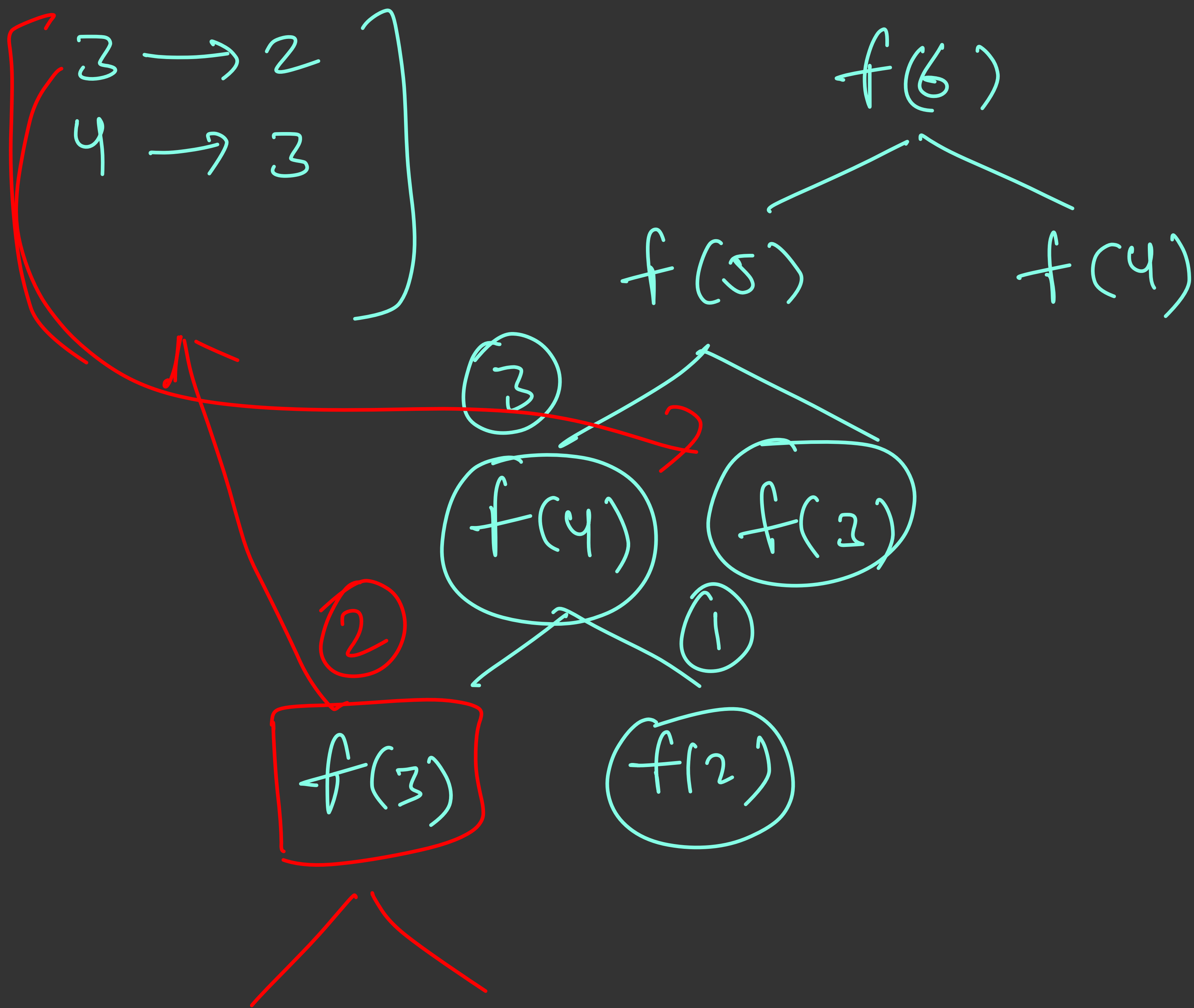
$\boxed{\text{input}} \longrightarrow$ answer if it exists

$f(\underline{6})$ $f(\underline{8})$

$f(n)$

$f(\underline{12})$

{ int_key $\longrightarrow$ value if it exists }



3 → 2
4 → 3
f(6)
f(5)
f(4)
3
f(4)
f(3)
2
1
f(3)
f(2)

Hash map $\rightarrow$ always works no matter what type of input is given

$f(8)$ f("TLE")

f([array])

{ key

}

key $\longrightarrow$ value

O(1)

1. Checking if key's answer exist
2. Update / insert the answer for a key
3. Retrieve the answer for a key

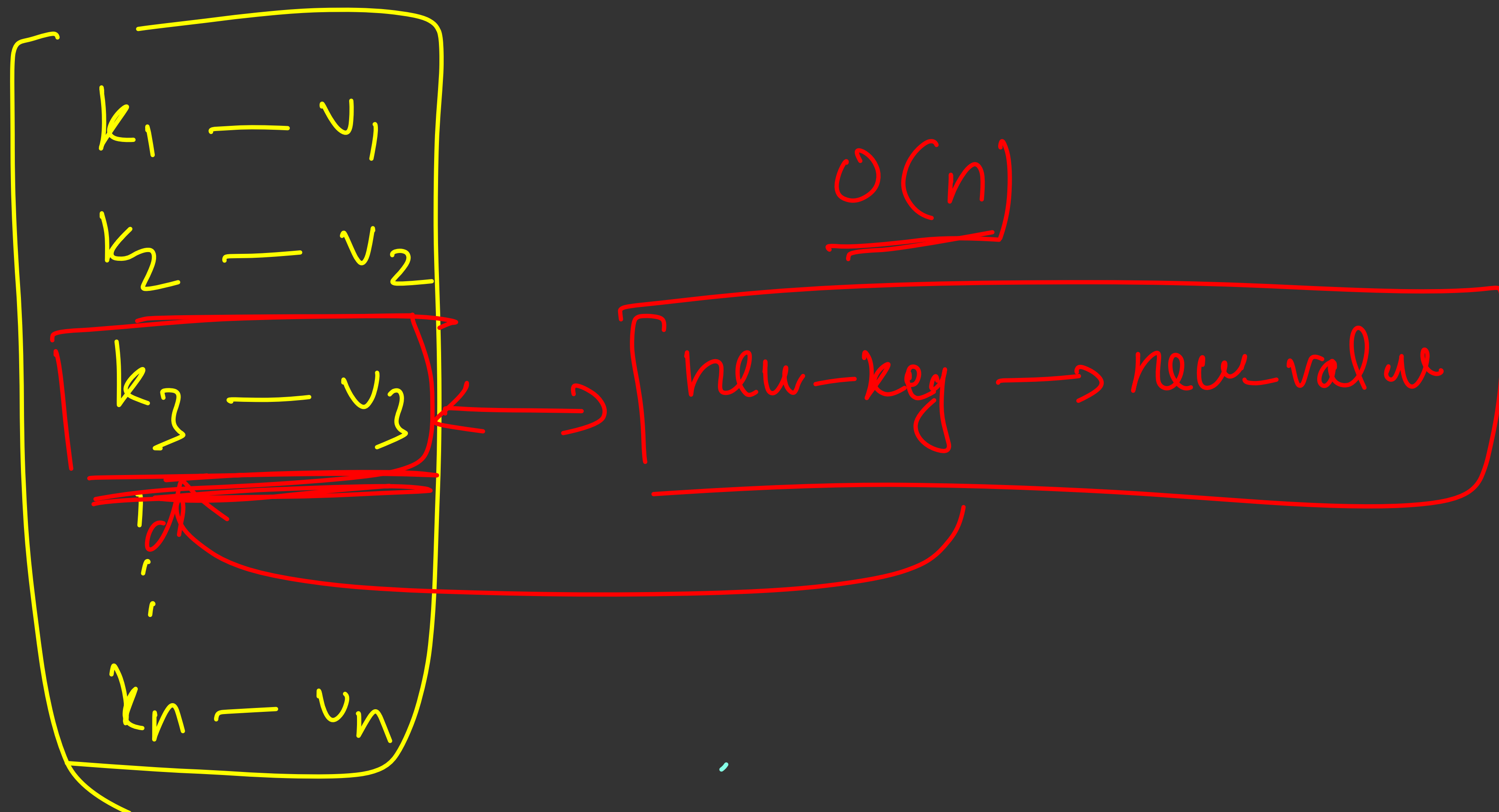
in average O(1)

in worst case O(n)

map

key $\longrightarrow$ value

"TLE" "Priyansh"



k1 — v1
k2 — v2
k3 — v3
kn — vn
O(n)
new-key → new_value

You will be given only the integer input

$$\boxed{1 \le input \le 10^5}$$

inf	inf	inf	10	inf	inf	inf	inf	inf

1 2 3 4 - - - - 10^5

input $\longrightarrow$ f(input) $\longrightarrow$ answer

$(0 \text{ to } 10^{18})$

-1	-1	-1	50	-1	-1	-1	-1
1	2	3	4	5	6	---	.. 10^5

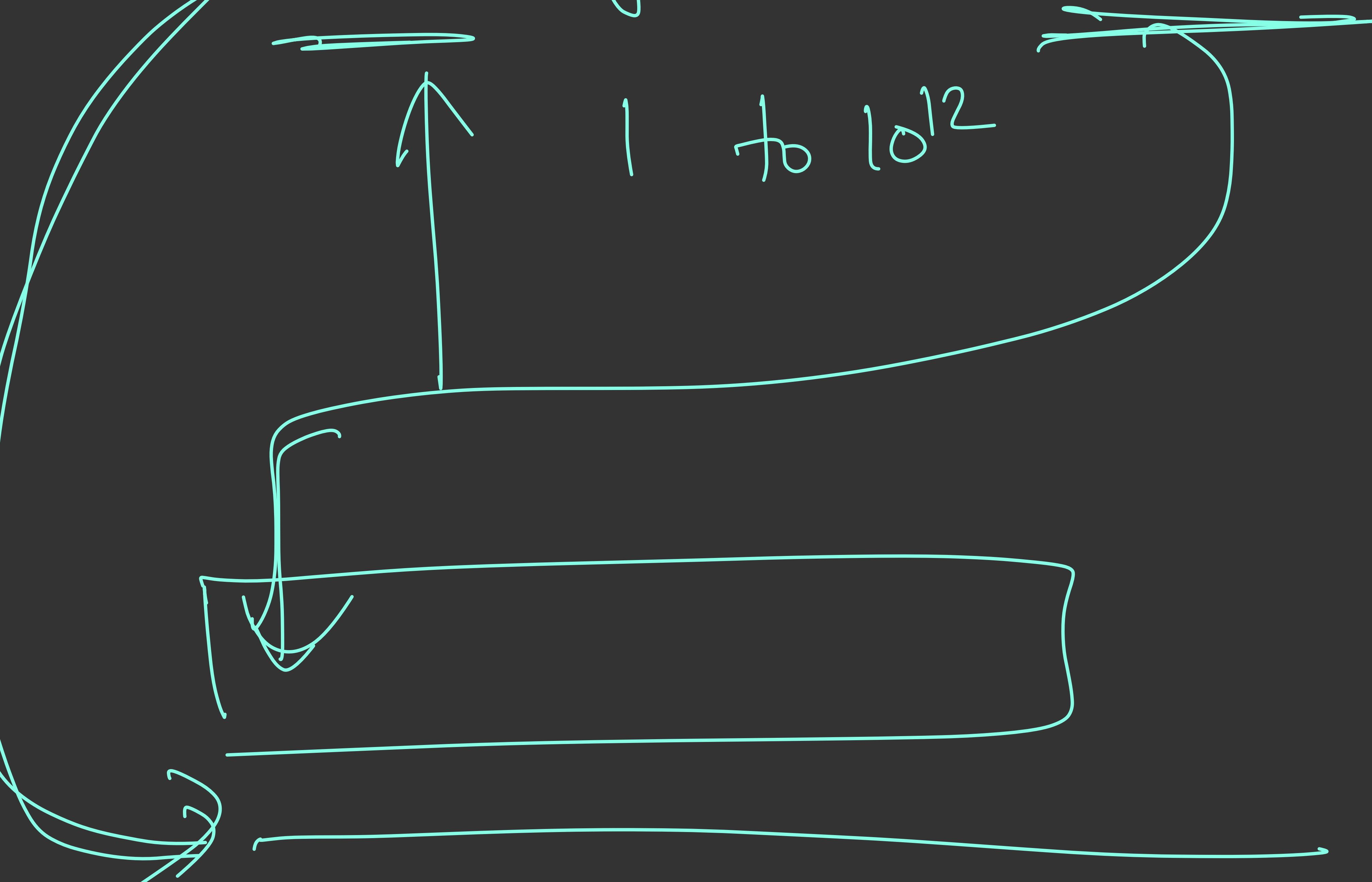
input is a string — map

input is a +ve integer from
1 to 10^{12} — map

input is from 0 to 10^5
— array

input (key) $\xrightarrow{f(\text{input})}$ value

1 to 10^{12}

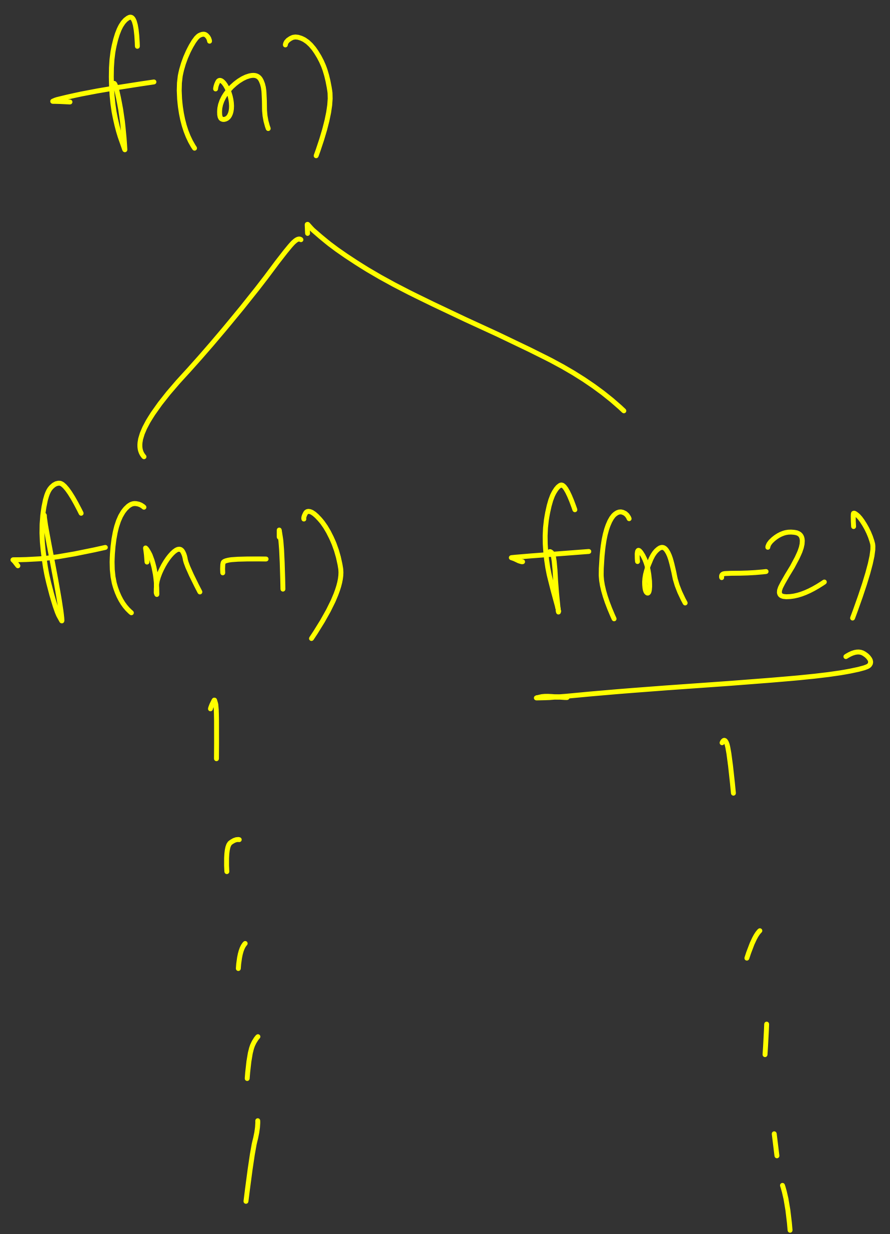


input is from -10 to 10^5

$f(x)$ $\quad arr[x] \propto arr[x+10]$

$-10 \quad -9 \quad -8 \;\ldots\ldots\; 0 \;\ldots\ldots\; 10^5$

$10^5 + 10$



$1 \leq n \leq 10^5$

$arr[10^5]$

-1

$$\text{input} \xrightarrow{f(\text{input})} \text{output}$$

$$f(1) = 1 \qquad f(2) = 1$$

---

inputs are keys & initially all

outputs are values keys have a

default value that

is different from all possible outputs

Without DP

helper (n) = f(n)

```
int functionEntered = 0;
int helper(int n){
    functionEntered++;
    if(n == 1 || n == 2){
        return 1;
    }
    return helper(n - 1) + helper(n - 2);
}
void solve(){
    int n;
    cin >> n;
    cout << helper(n) << nline;
    cout << functionEntered << nline;
}
```

functionEntered = 1664079
with n = 30

With DP

```
int functionEntered = 0;
int dp[40];
int helper(int n){
    functionEntered++;
    if(n == 1 || n == 2){
        return 1;
    }
    if(dp[n] != -1)
        return dp[n];   // does the answer exist
    return dp[n] = helper(n - 1) + helper(n - 2);
}
void solve(){
    int n;
    cin >> n;
    for(int i = 0; i <= n; i++)
        dp[i] = -1;
    cout << helper(n) << nline;
    cout << functionEntered << nline;
}
```

functionEntered = 57
with n = 30

Let's solve another problem!

Given a 2D grid (N X M) with numbers written in each cell, find the path from top left (0, 0) to bottom right (n - 1, m - 1) with minimum sum of values on the path

only go right or down at any cell

1	5	8
6	2	7
9	3	4

Naive Way

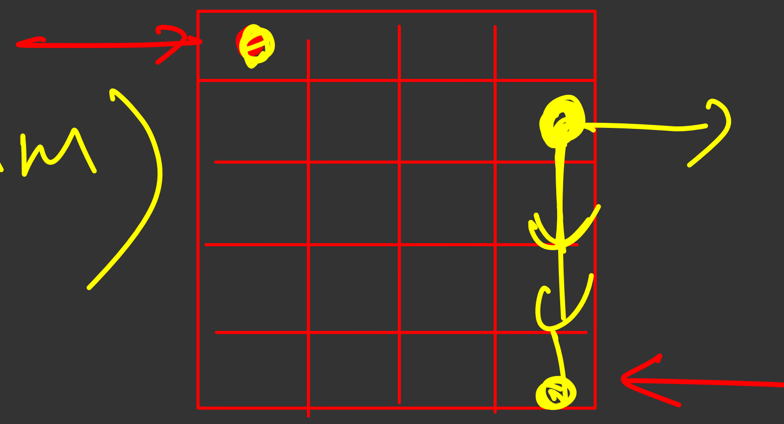
Explore all paths. Standing at (i, j) try both possibilities (i + 1, j), (i, j + 1)

Every cell has two choices

Time complexity: $O(2^{m*n})$?

Actual Time complexity: O(C(n + m - 2, m - 1))

(Big problem P)

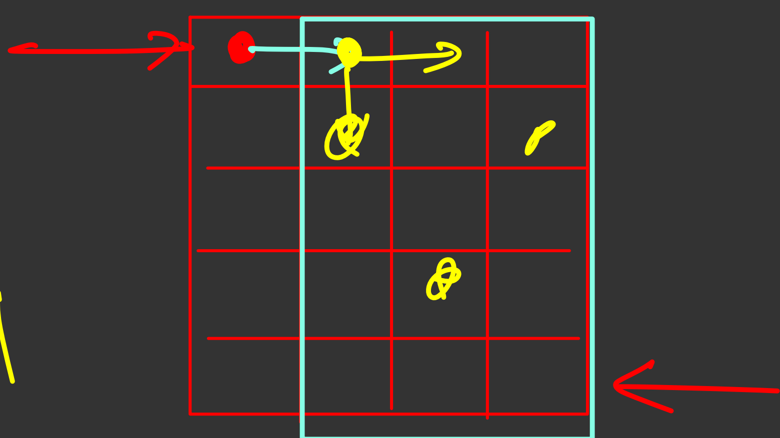


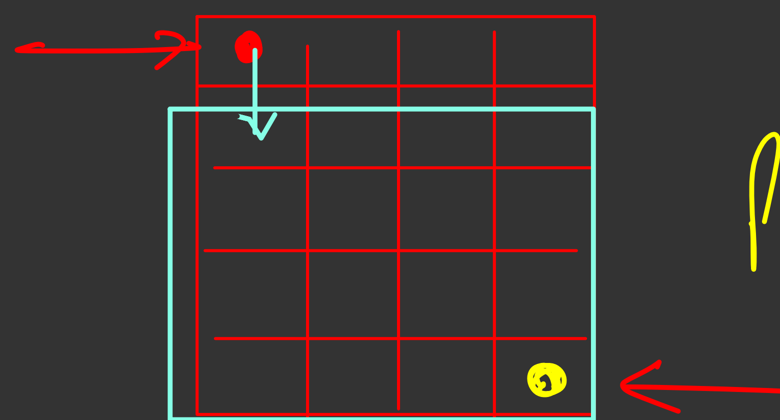
30

50 move right

move down

P1





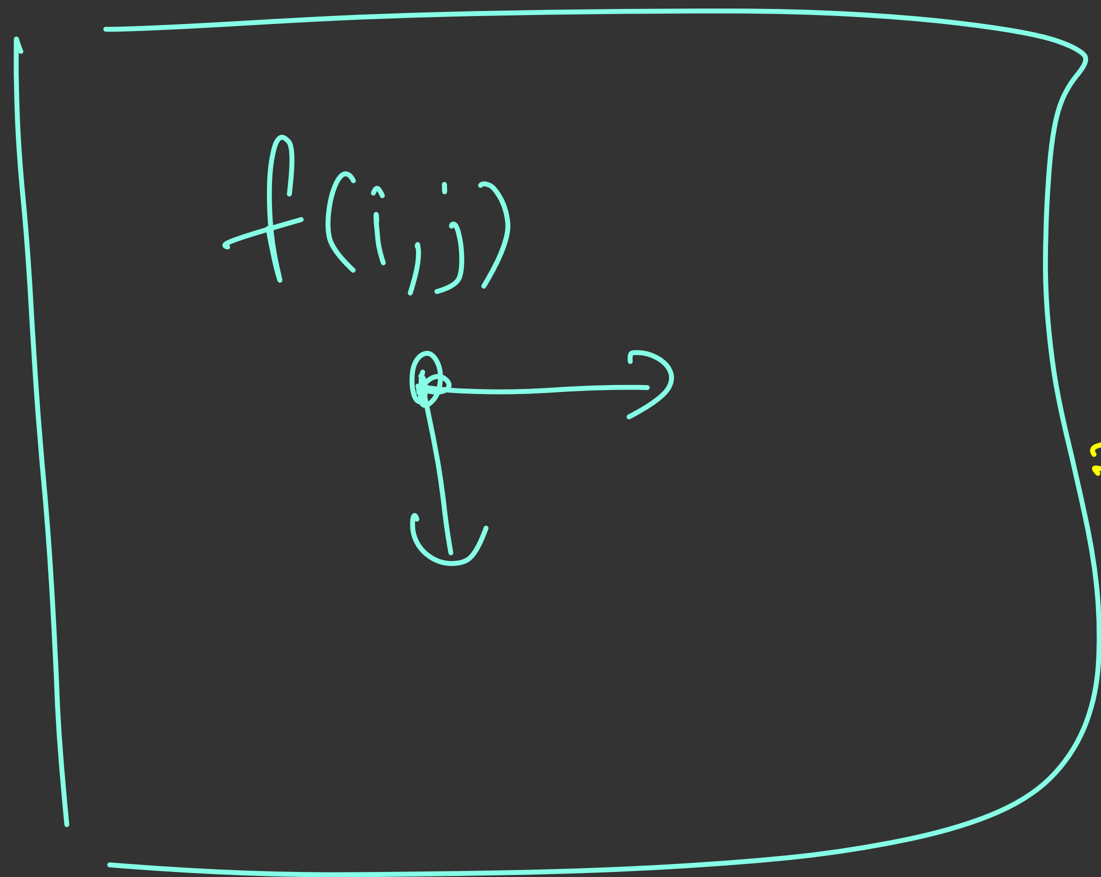
P2

$$P = \min(P_1, P_2) + cell[0][0]$$

$$ans[n-1][m-1] = cell[n-1][m-1]$$

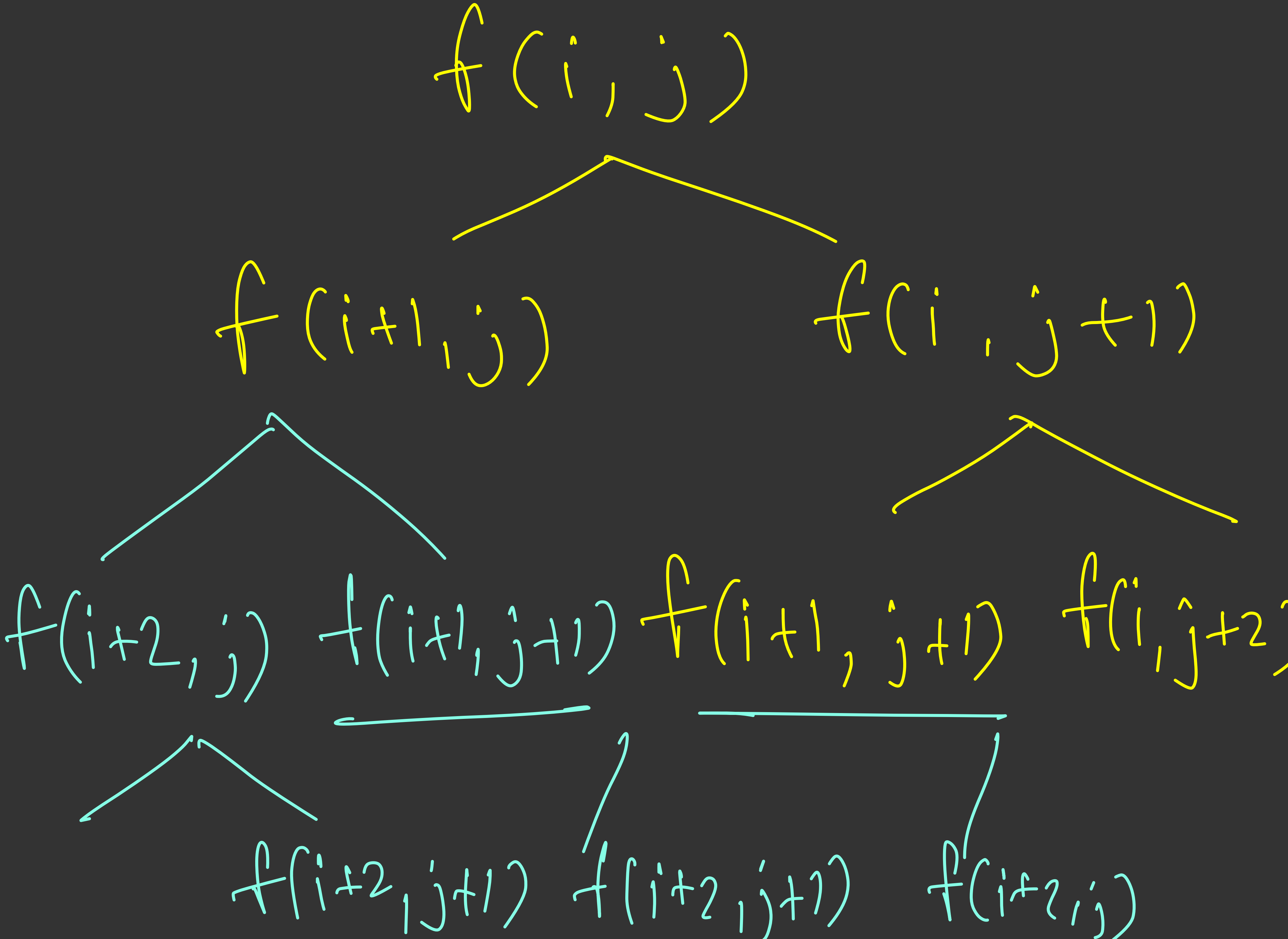
$f(0,0)$ = min sum path from $(0,0)$ to $(n-1, m-1)$

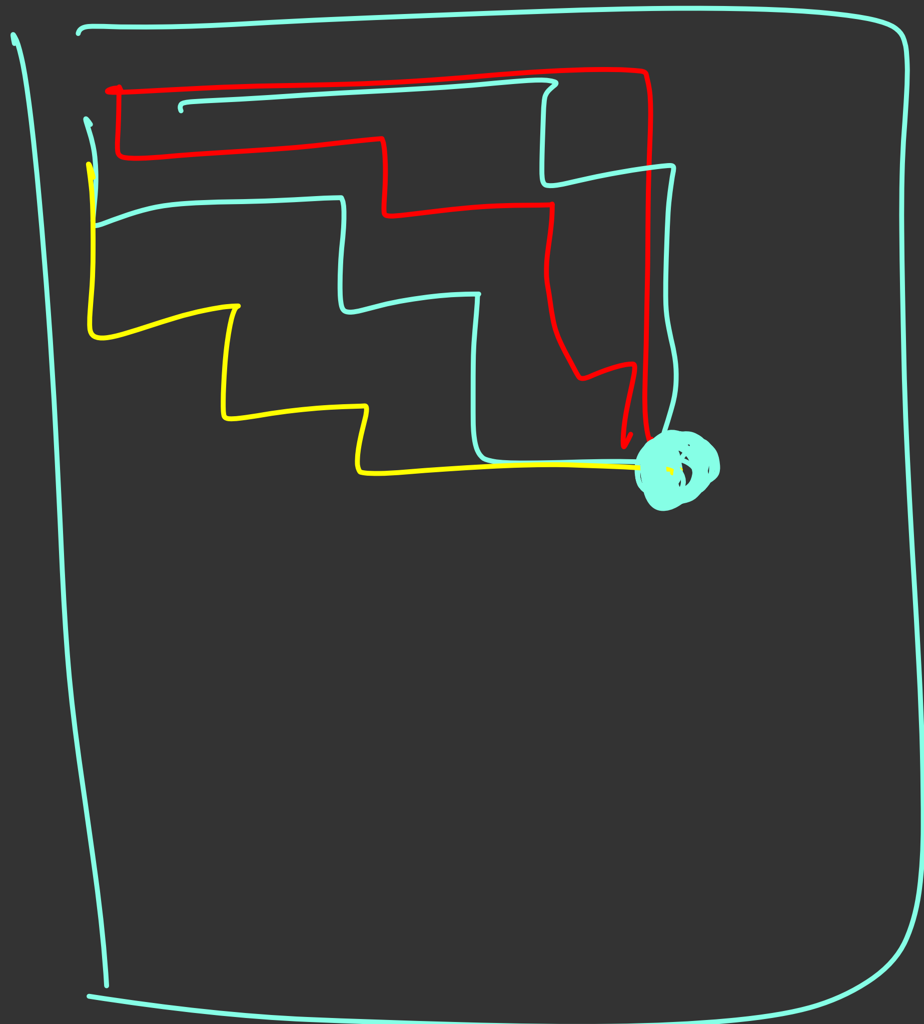
$f(i,j)$ = min sum path from (i,j) to $(n-1, m-1)$



$f(n-1, m-1) = matrix[n-1][m-1]$

$$\underline{\underline{f(i,j)}} = \min\left(f(i+1, j), f(i, j+1)\right) + matrix[i][j]$$


f(i, j)
f(i+1, j)
f(i, j+1)
f(i+2, j)
f(i+1, j+1)
f(i+1, j+1)
f(i, j+2
f(i+2, j+1)
f(i+2, j+1)
f(i+2, j)



Efficient Way

Overlapping subproblems

Memoization

Time complexity: O(n * m)

Space complexity: O(n * m)

```cpp
int grid[n][m]; // input matrix

int dp[n][m]; // every value here is -1

// subproblem: f(i, j) represents minimum sum path from (i, j) to (n - 1, m - 1)
int f(int i, int j){ //
    if(i >= n || j >= m){ // moving outside the grid // not allowed
        return INF;
    }
    if(i == n - 1 && j == m - 1) // reached the destination
        return grid[n - 1][m - 1];

    if(dp[i][j] != -1) // this state has been calculated before
        return dp[i][j];

    // state never calculated before
    dp[i][j] = grid[i][j] + min(f(i, j + 1), f(i + 1, j));
    return dp[i][j];
}

void solve(){
    cout << f(0, 0) << nline;
}
```

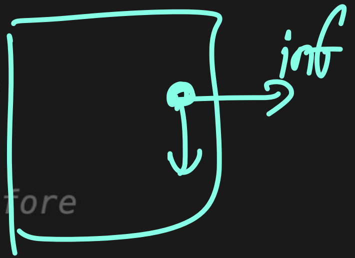
all no:s in the grid are positive

O(1)

O(1)

O(1)

O(1)



How many unique subproblems

$$n \times m$$

$f(i, j)$

$\downarrow \quad \downarrow$

$n \quad m$

$$f(i,j) \longrightarrow \min \left\{ \begin{array}{c} f(i+1, j) \\ \hline f(i, j+1) \end{array} \right\} + grid[i][j]$$

$$O(1)$$

T.C = no. of subproblems $\times$ time to find answer

$$\Rightarrow \quad O(n \cdot m) \times O(1)$$

$$\Rightarrow \quad O(n \cdot m)$$

Important Terminology

Subproblem

State: A subproblem that we want to solve. The subproblem may be complex or easy to solve but the final aim is to solve the final problem which may be defined by a relation between the smaller subproblems. Represented with some parameters.

Relation

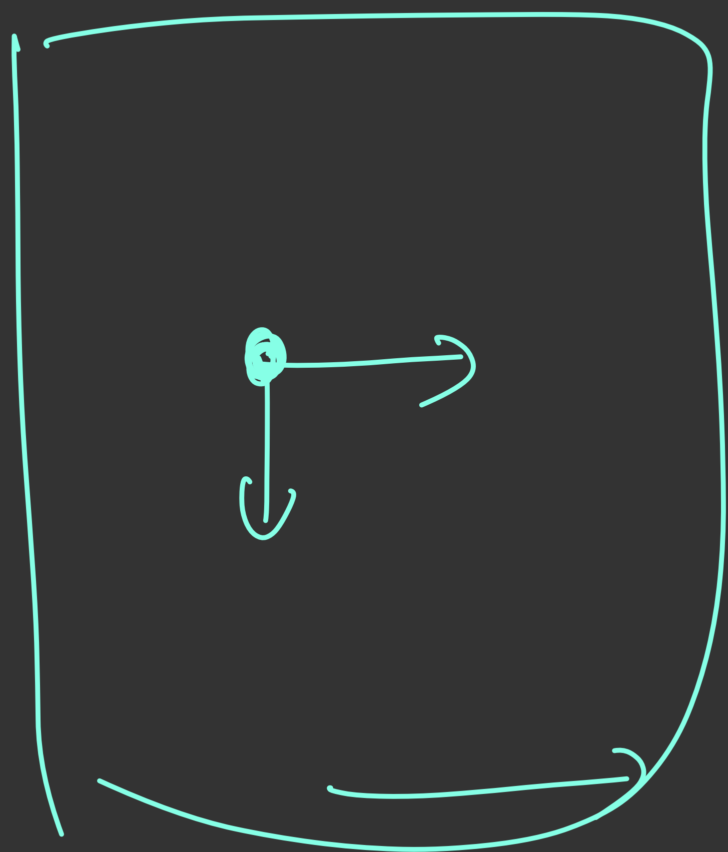
Transition: Calculating the answer for a state (subproblem) by using the answers of other smaller states (subproblems). Represented as a relation b/w states.

dp[i] , dp[i][j]

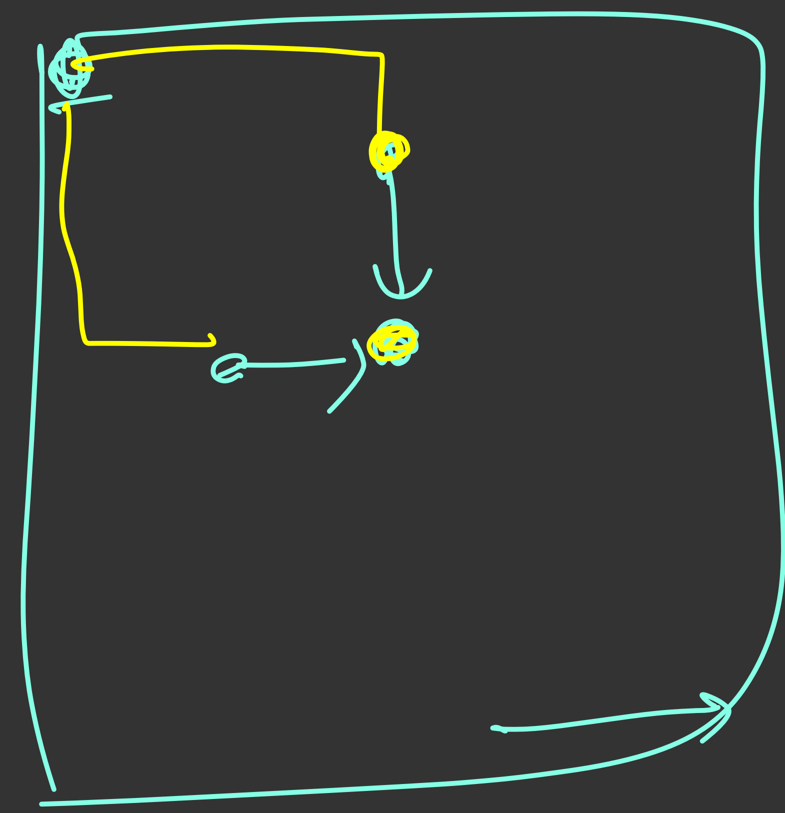
$f(i,j)$ = min sum path from (i,j) to $(n-1, m-1)$

$f(i,j)$ = min sum path from $(0,0)$ to (i,j)

$$f(i,j) = \min \begin{Bmatrix} f(i, j-1) \\ f(i-1, j) \end{Bmatrix} + \text{grid}[i][j]$$



1



2

Exercise

Fibonacci Problem:

- State
  - dp[i] or f(i) meaning i^{th} fibonacci number #
- Transition
  - dp[i] = dp[i - 1] + dp[i - 2]

- dp(1) , dp(2) = 1

Exercise

Matrix Problem:

- State
  - dp[i][j] = shortest sum path from (i, j) to (n - 1, m - 1)
- Transition
  - dp[i][j] = grid[i][j] + min(dp[i + 1][j], dp[i][j + 1])

- dp(n-1)(m-1) = grid(n-1)(m-1)

Time and Space Complexity in DP

avg | worst

Time Complexity:

Estimate: Number of States * Transition time for each state

Exact: Total transition time for all states

Space Complexity:

Number of States * Space required for each state

$$f(n) = f(n-1) + f(n-2)$$

$$O(1)$$

$$f(n) = f(n-1) + f(n-2) + f(n-3) \; \text{-------} \; f(1)$$

$$O(n)$$

$$f(n) = \frac{n}{n} \; T \cdot T$$

$$f(n-1) = \frac{n}{n-1} \; T \cdot T$$

$\vdots$

$$f(1) = \frac{n}{1} \; T \cdot T$$

$$\begin{array}{cccc} f(1) & f(2) & \text{------} & f(n) \\ \downarrow & \downarrow & & \downarrow \\ n/1 & n/2 & & n/n \end{array}$$

$$\text{Sum} = \frac{n}{1} + \frac{n}{2} \;\cdots\cdots\; n/n$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} \text{ ---- } \frac{1}{n}\right)$$

$\downarrow$

$\underline{< \log n}$

$= \quad < \quad \underline{n \log n}$