Recursion

Big Problem → Subproblems

$f(N)$

Magic Rule

1. Find out the smallest problem — Base Case
2. Assumption: Assume subproblem $f(K)$ can be solved recursively for all $K<N$
3. Express $f(N) \rightarrow f(K)$, $K<N$ — Recursive Case

Factorial

$5! = 5 * \boxed{4!}$

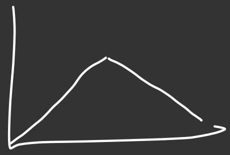
$$f(n) = n * f(n-1)$$

$$f(0) = 1$$

↑

```
f(n){
    if(n==0)
        return 1

    return n * f(n-1)
}
```



$5! \rightarrow 5 * \cancel{4!}\, 24 = 120$

↓

$4 * \cancel{3!}\, 6 = 24$

↓

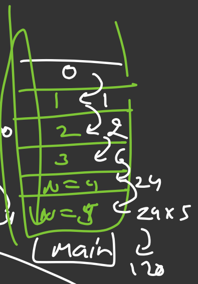
$3 \times 2! = 3 \times 2 = 6$

↓

$2 \times 1! = 2$

1

$1 \times 0!\ 1$ → Base Case



→ $O(N \times 1) = O(N)$

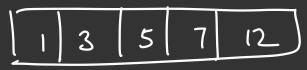
Time → Calls * work in each call

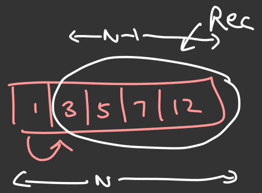
Space → Max Stack Depth * Space used in each call

$= O(N \times 1)$

$= O(N)$

Given an array, of size, check if it Sorted (inc order)





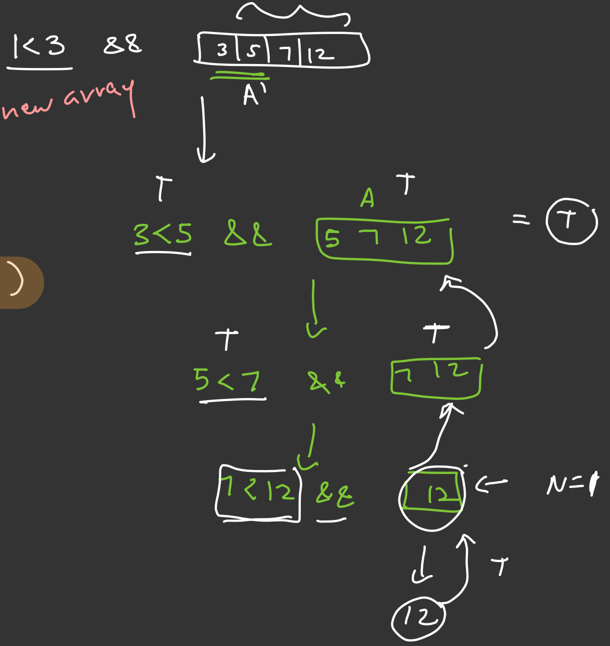
Base case

```
bool f(A, N) {
    if(N==0 or N==1) {
        return True
    }

    return A[0] < A[1] && f(A', N-1)

}
```

~~Create~~ a new array ~~O(N)~~



```
bool   f (A, N) {
        if(N==0  or  N==1) {
              return True
        }
    return  A[N-1] > a[N-2] && f(A, N-1),
}
```

Base case

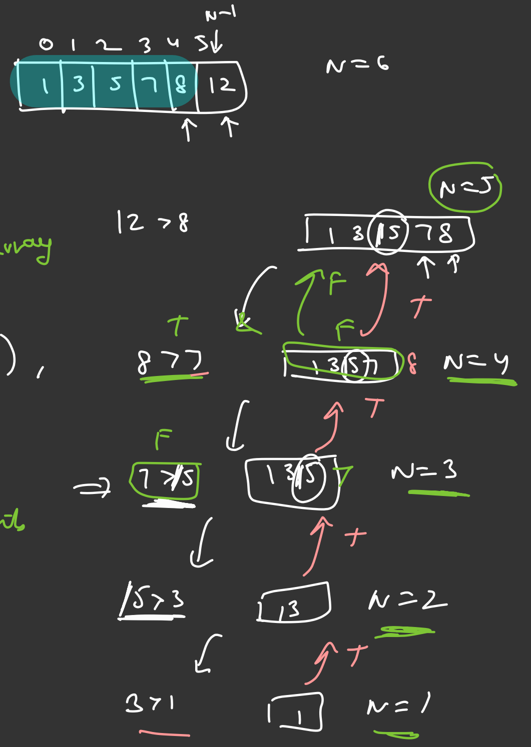
generate false value

N-1   N-2

as index to track what position to compare

same array

An array with N-1 elements

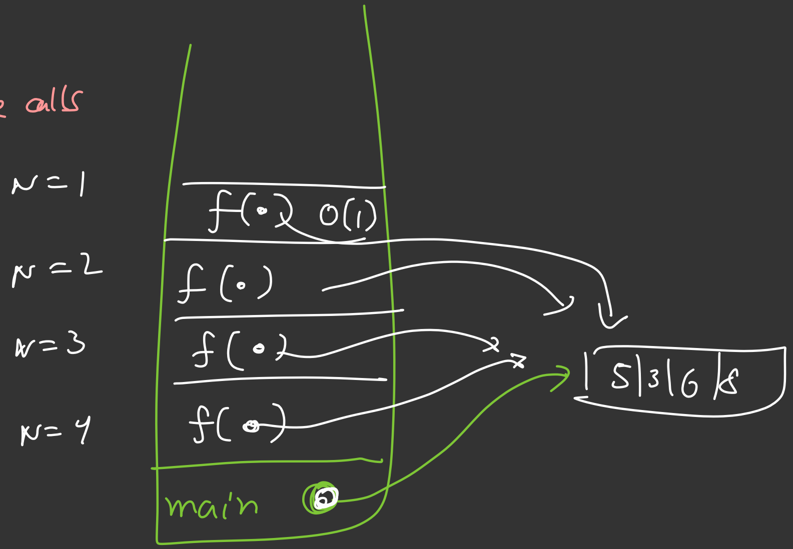


Time → O(N)

Space → Stack space due to multiple calls

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

Recursion ALWAYS come with extra space overhead



Q Given a N find sum of its digit

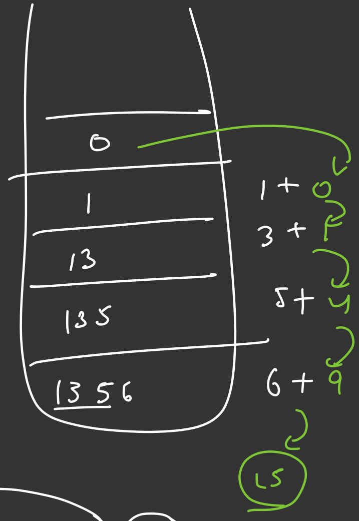
N = 13 56

Expected = 15

```
int f ( N) {
    if (N == 0)
        return 0        Base case

    return N%10 + f(N/10)
}
```

another way
if(N<10)
return N



$\boxed{1\ 3\ 5}\ \boxed{6}$

$1356 \% 10$

$= \boxed{6} + f(N/10)$

Ⓠ N friends are going for a party. each friend can go Solo or as a couple. Find out the total ways in which they can go.

N= 3 A, B, C

(AB) C

(AC) B

(BC) A

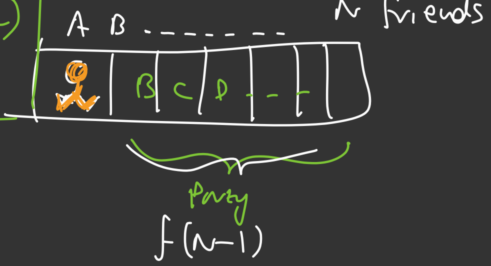
(A) (B) (C)

} 4 ways.

N= 4 A, B, C, D

solo + Couple

$$f(N) = 1 \cdot f(N-1) + \underbrace{(N-1)}_{(A \cdot gf)} \cdot f(N-2)$$

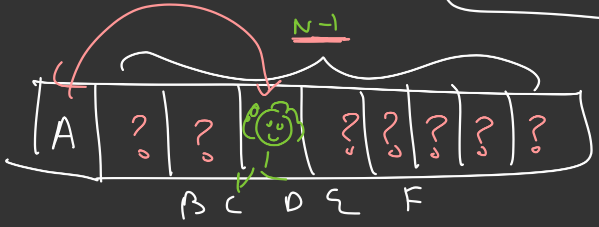


$$\boxed{\begin{aligned} f(1) &= 1 \\ f(2) &= 2 \end{aligned}}$$

Base Case

(A)

1,



Pick 1 out N-1

↳ N-1 choices

N-2 ppl

↓

f(n-2) ways

A B

(A) (B)

(A, B)

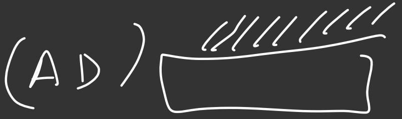
2 ways

(BC) D, E

(BD) (CE)

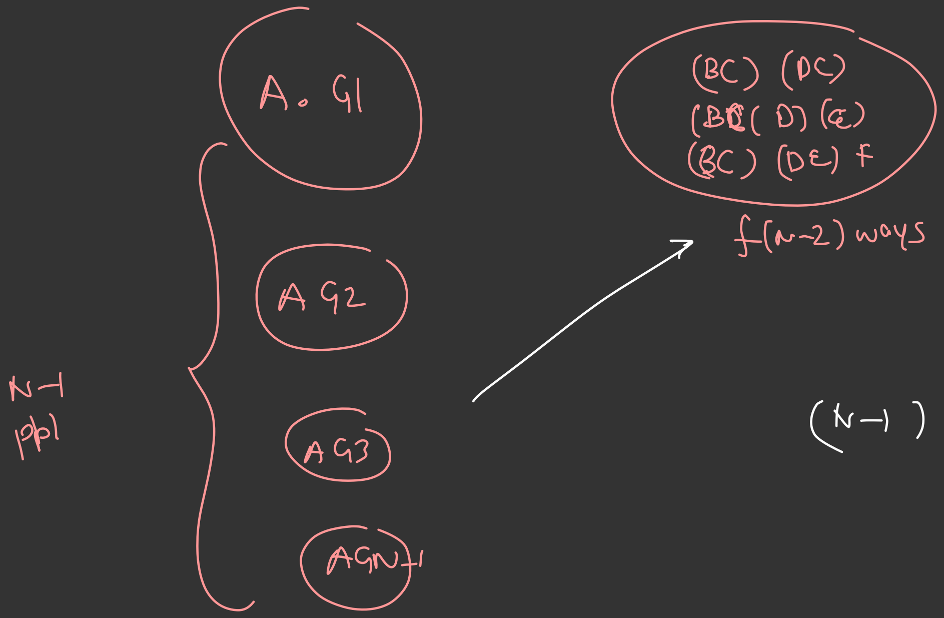
⋮

many ways



$$f(3) = f(2) + 2 \cdot f(1)$$
$$= 2 + 2 \cdot 1 = \boxed{4}$$

$$f(4) = f(3) + 3 \cdot f(2)$$
$$= 4 + 3 \cdot 2$$
$$= \boxed{10}$$



$$(N-1)\ f(N-2)$$

⇒ A, B, C, D

✓, ✓, ✓, ✓

(AB) (CD)

(AC) (BD)

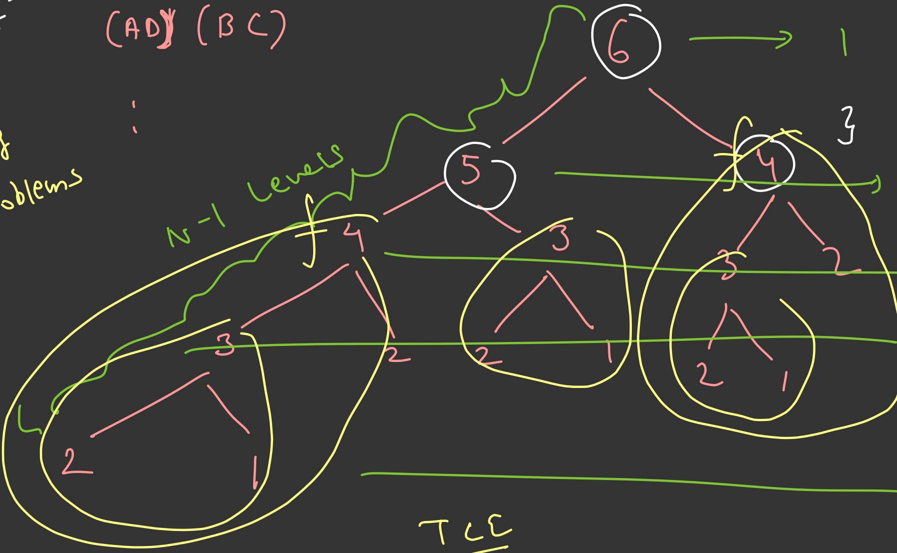
(AD) (BC)

⋮

```
int f(N) {
    if (N ≤ 2)
        return N
    return f(N-1) + (N-1) f(N-2)
}
```

Revisit

fibonacci analysis



overlapping subproblems

1

2

4

8

$1 + 2 + 4 + 8 + \ldots 2^{n-2}$

$= O(2^n)$

Time

TLE

N = 30

$2^{30} = (10^3)^3$

$= 10^9$ steps

Q Money Change

Indian currency = [1, 2, 5, 10, 20, 50, 100, 200, 500, 2000]

denom []

₹ 276 → get a Change using min Notes / coins.

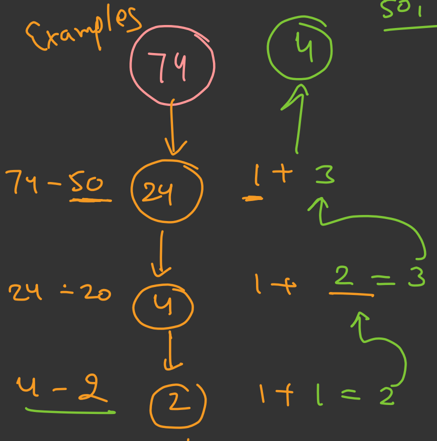
= 200 + 50 + 20 + 5 + 1 } Min

= 276   No money   5 Notes / coins

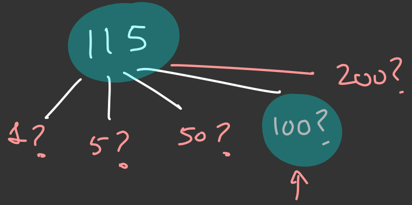
50, 20, 2, 2

```
int f ( money) {
    if (money == 0){
        return 0;
    }
}

// tue
```



15 Mins



50, 20, 2, 2

```
int note = findlargest(money)      → O(1)    1+0 = 1
    print(note);
    total_notes = 1 + f(money - note),
    return total_notes
}
```

largest
No ≤ money

115
↓ -100
1+
15
↓ -10
5
↓ -5
0

O(No ~~of~~ of term ↑ Linear Search, Binary Search → O(log Terms)

[1, 2, 5, 10, 20, 50, 100, 200, 500, 2000]

Constant size

4240 → 2000

156 → 100

224 → 200

680 → 500

Time ⇒ 10,000

4

note = 2

1 cr

$$\left(\frac{1\ cr}{2000}\right) = O\left(\frac{money}{\underline{\underline{2000}}}\right)$$

$$= O(money)$$

$$\frac{money}{note} + f(money \% \ note)$$

→ $\underline{O(money)}$

$$\left(\frac{10{,}005}{2000}\right) + f(\underbrace{money \% \ note}_{\boxed{5}})$$

$$= 5$$

10,000

↓

8000

↓

6000

↓

4000

↓

2000

↓

0

1000 0

$\underline{1000}$ 1+1+1+ - - -

1000 - 500

500 - 500

$$\frac{10{,}000}{2000} + f(0)$$

↓

5 notes

$$O\left(\frac{money}{\underline{\underline{2000}}}\right)$$

$$f(\text{money}) = \frac{\text{money}}{\text{note}} + f(\text{money} \% \text{note})$$

Every note type can be picked once

↓

$O(1)$

$$\frac{74}{50} + f(24) = 1+3 = 4$$

$$\frac{24}{20} + f(4)\ 2 = 1+2 = 3$$

$$\left[\frac{4}{2}\right] + \underset{0}{\underset{\parallel}{f(0)}} = 2$$

money

10 cr

-2000

-2000

-2000

⋮

var → money steps

const → 2000

$$\frac{10\,cr}{2000} + f(0)$$

0

$$\boxed{O(money)}$$

7

↓ 1

2

↓ 1

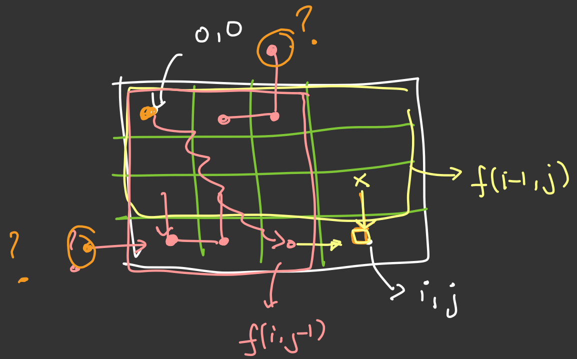
0

$⑤ + ② =$

1 + 1 = 2 steps

$$\left\{\begin{array}{l} O(money) \\ O(1) = O(\text{Notes Array}) \end{array}\right.$$

Q

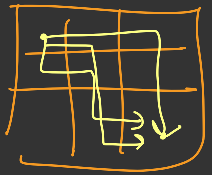


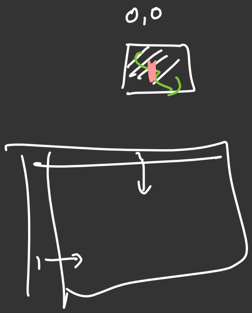
→ Right

↓ Down

at any cell

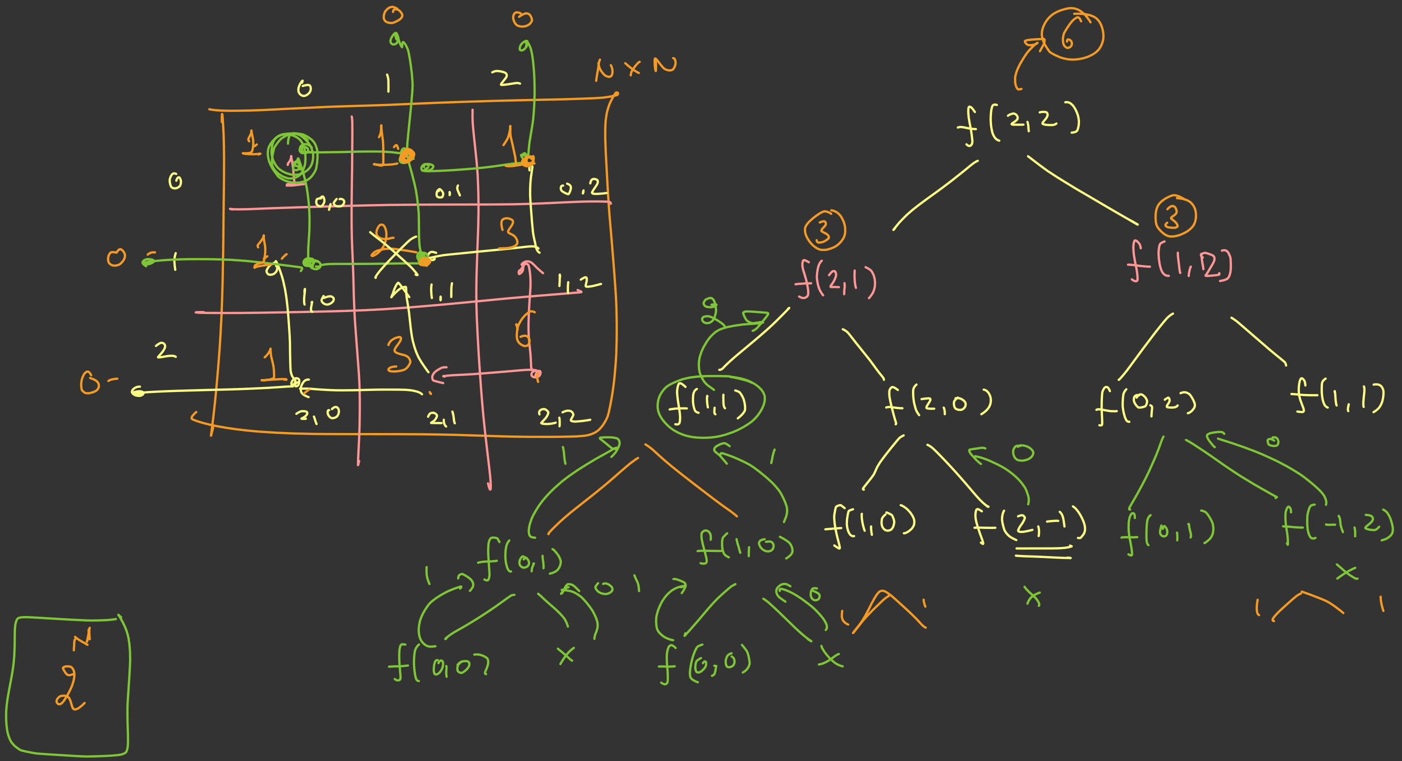
How many ways I can [illegible] cell i,j

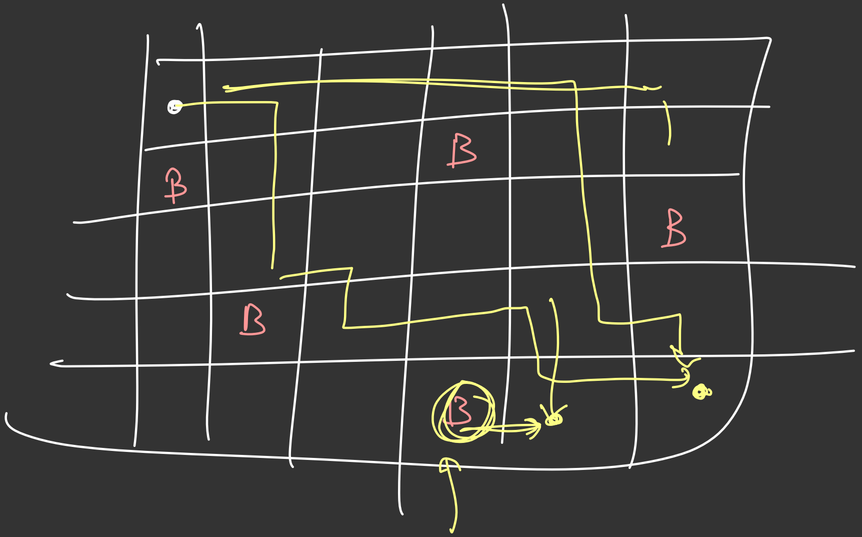




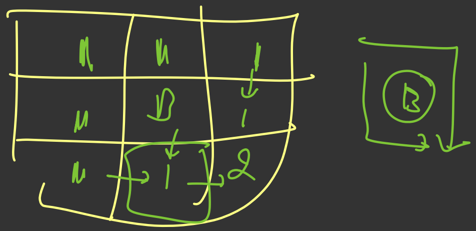
```
int f ( i , j ) {
    if (i==0 || j==0)        or cell is blocked i,j
        ~~return 1 ;~~
    if (out of Bounds) {
        return 0;
    }
    return f(i-1,j) + f(i,j-1);

}
```


N × N
0
1
2
0
1
2
0
0
0
0
1
1
1
1
3
1
3
6
0,0
0,1
0,2
1,0
1,1
1,2
2,0
2,1
2,2
6
f(2,2)
3
f(2,1)
3
f(1,2)
2
f(1,1)
f(2,0)
f(0,2)
f(1,1)
1
1
0
0
f(0,1)
f(1,0)
f(1,0)
f(2,-1)
f(0,1)
f(-1,2)
1
0
1
0
f(0,0)
x
f(0,0)
x
x
x
2^N



f(blocked) = $\underline{\underline{0}}$


u
u
1
u
B
1
u
1
2
B