What is Recursion?

How to write recursive code?

Function call Tracing

4 problems

TC and SC

Recursion → Function calling itself

```
void add (     ) {
    ___
    ___
    ___
    add (     )
}
```

Bigger Doll → smaller Doll
similar Doll
End Doll

Recursion → solving larger problems using smaller similar subproblem

$$sum(N) = 1 + 2 + 3 + \ldots + (N-1) + N$$

$$sum(N) = sum(N-1) + N$$

$$sum(5) = sum(4) + 5$$

↓

$1 + 2 + 3 + 4$

How to write a recursive code?

3 Logical steps:

1. Assumption : Decide what your fn does

2. Main Logic :
   Recursive relation to code /
   breaking big problem into smaller subproblem

3. Base condition
   End condition where recursion stops /
   smallest prob for which you already know answer

Q. Recursive code to calculate sum of N natural nos.

N = 4 ans = 10

```
// Given N, it will return sum of first N natural nos.
int sum (int N) {
    If (N==0) return 0
    If (N==1) return 1
    return sum(N-1) + N
}
```

Fn call Tracing

seq of fn calls that are made when a program is executed

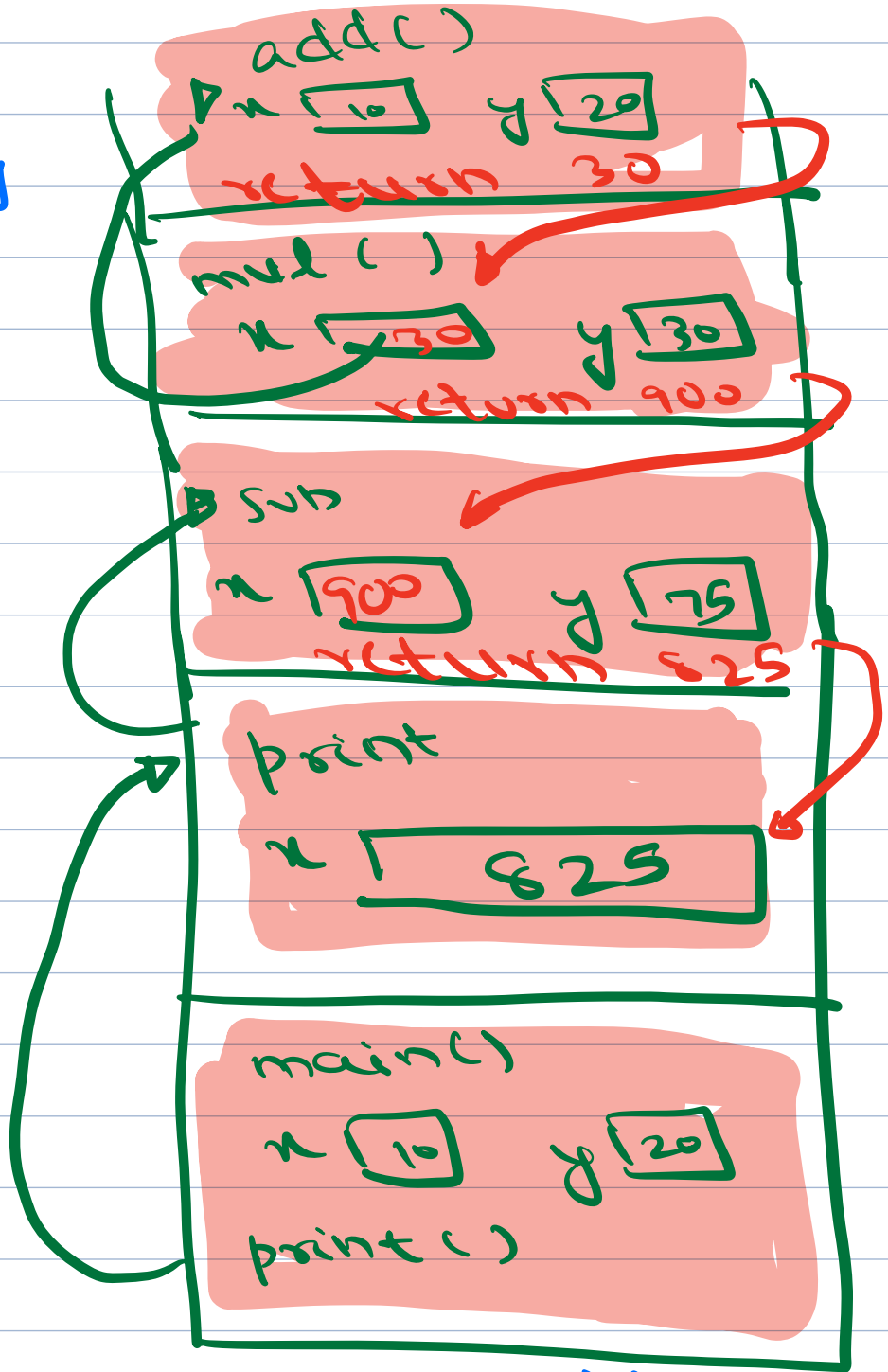
```
fn add(x,y) {
    return x+y
}

fn sub(x,y) {
    return x-y
}

fn mul(x,y) {
    return x*y
}

fn print(x) {
    cout << x
}

fn main() {
    x=10, y=20
    print(sub(mul(add(x,y),30),75))
}
```



NOTE: $1! = 1$

$0! = 1$

1. Given a +ve integer N, find factorial of N

N = 3 ans = 6 $3! = 3.2.1$

N = 4 ans = 24 $4! = 4.3.2.1$

N = 5 ans = 120

$fact(N) = 1 * 2 * 3 * 4 * \ldots (N-1) * N$

$fact(N) = fact(N-1) \times N$

$4! = 1 \times 2 \times 3 \times 4$

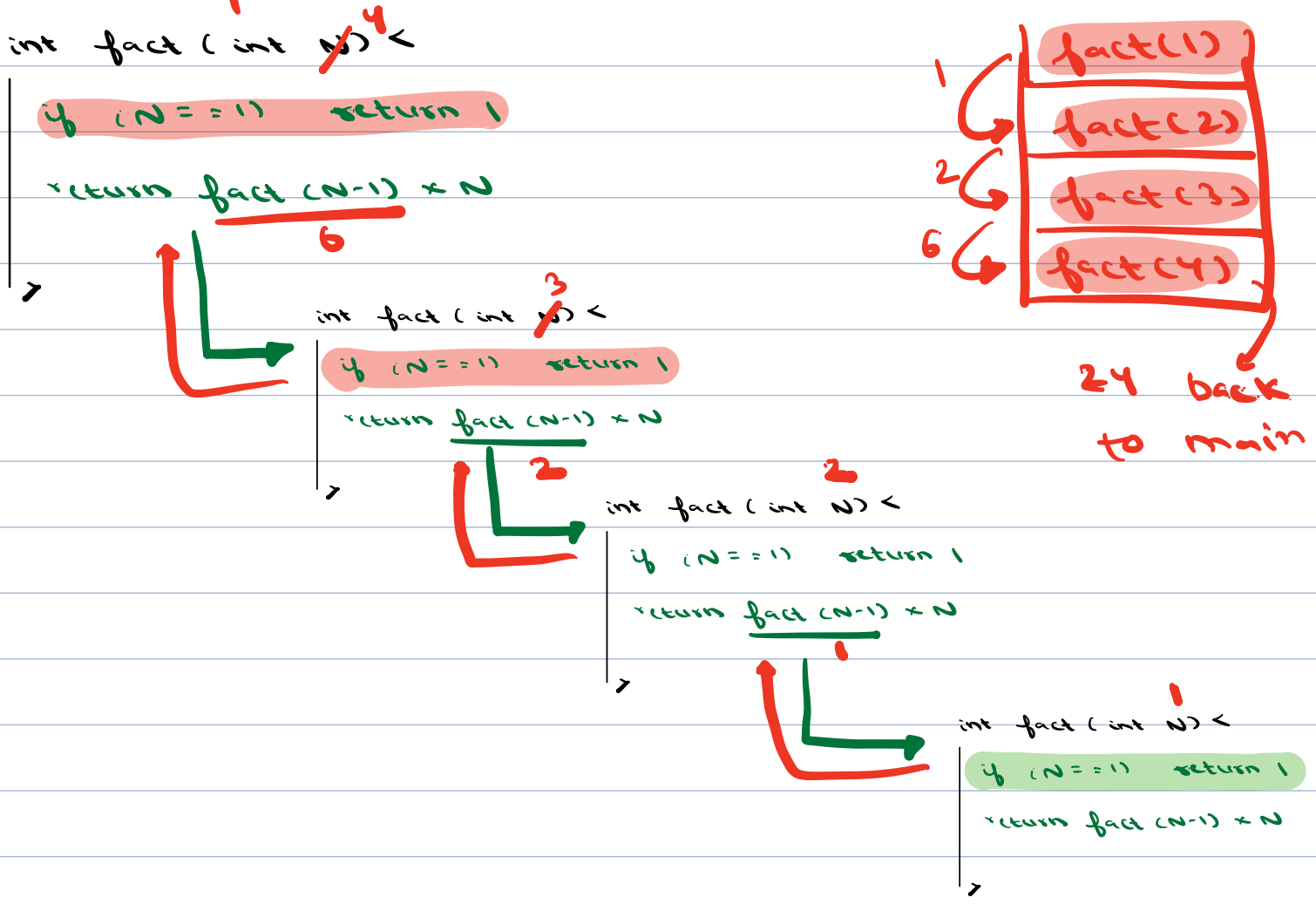
$4! = 3! \times 4$

```
① // Given N, it will return N!
  int fact (int N) {
③   if (N == 1) return 1
②   return fact(N-1) * N
  }
```

$N! = (N-1)! \times N$

This code fails for N = 0

24



fact(0)

↓

fact(-1)

↓

fact(-2)

↓

fact(-3)

↓

⋮

Infinite recursion

↓

MLE (Memory Limit Exceeded)

Fn call stack becomes full

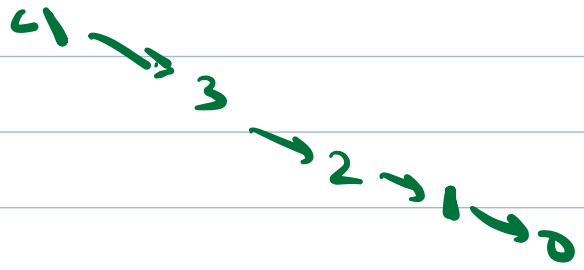
Memory is exhausted before time

① // Given N, it will return N!

```
int fact ( int N) {
    if (N == 0)   return 1
    return fact (N-1) * N
}
```

③

②




```
int main() {
    // I/P → n
    if (N ≥ 0)
    print ( fact (n))
    else
    print ("Invalid")
}
```

2. Given N, print all numbers from 1 → N in increasing order.

Inc (5) = 1 2 3 4 5

Inc (7) = 1 2 3 4 5 6 7

Inc (7) = Inc (6) + 7

Inc (N) = 1 2 3 4 5 ... (N-1) N

Inc (N) = { Inc (N-1)
print (N)

① // Given N, fn will print all nos. 1→N

void inc (int N) {

③ if (N == 0) return

inc (N-1) // print 1 .... N-1

② print (N)

}

Base case : N == 0

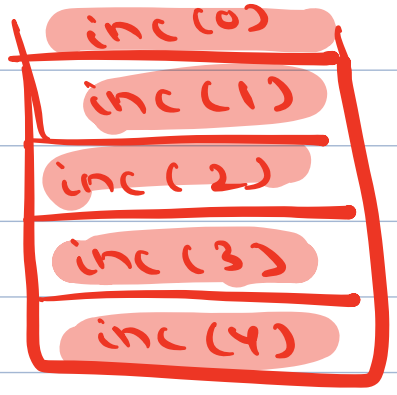
: if (N == 1) {

print (1)
return

}

```
void inc(int N) {
    if (N==0) return
    inc(N-1)
    print(N)
}
```

4




```
void inc(int N) {
    if (N==0) return
    inc(N-1)
    print(N)
}
```

3

```
void inc(int N) {
    if (N==0) return
    inc(N-1)
    print(N)
}
```

2

O/P : 1 2 3 4

```
void inc(int N) {
    if (N==0) return
    inc(N-1)
    print(N)
}
```

1

```
void inc(int N) {
    if (N==0) return
    inc(N-1)
    print(N)
}
```

0

```
void inc(int N) {
    if (N==0) return
    inc(N-1)
    print(N)
}
```

3. Given N, print all numbers from 1 → N in decreasing order.

dec (5) = 5 4 3 2 1

dec (5) = print (5)
dec (4)

dec (N) = N N-1 N-2 . . . . . 2 1

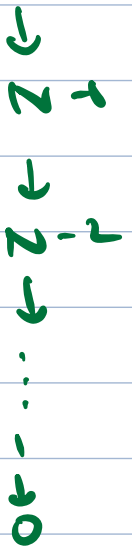
$$dec(N) = \begin{cases} print(N) \\ dec(N-1) \end{cases}$$

```
// Given N, fn will print all nos. N→1
void dec(int N) {
    if (N==0) return
    print(N)
    dec(N-1)
}
```

HW : Function call stack for N=5

Time Complexity = No. of function calls x Time per function call

fact(N)



(N+1) x 1

Fact →
Inc →
Dec →
Sum →

TC : O(N)

Space Complexity = Maximum stack space / recursion tree depth + Space per function call

amt. of memory required by a recursive algo as it executes

Fact →
Inc →
Dec →
Sum →

SC : O(N)

4. Fibonacci Series

The Fibonacci Series is a series of numbers in which each number is sum of two preceding nos.

N	0	1	2	3	4	5	6	7	8
no.	0	1	1	2	3	5	8	13	21 ....

Given a +ve integer N, write a function to return N^{th} fibonacci number.

N = 6 ans = 8

N = 7 ans = 13

Fib (N) = Fib (N-1) + Fib (N-2)

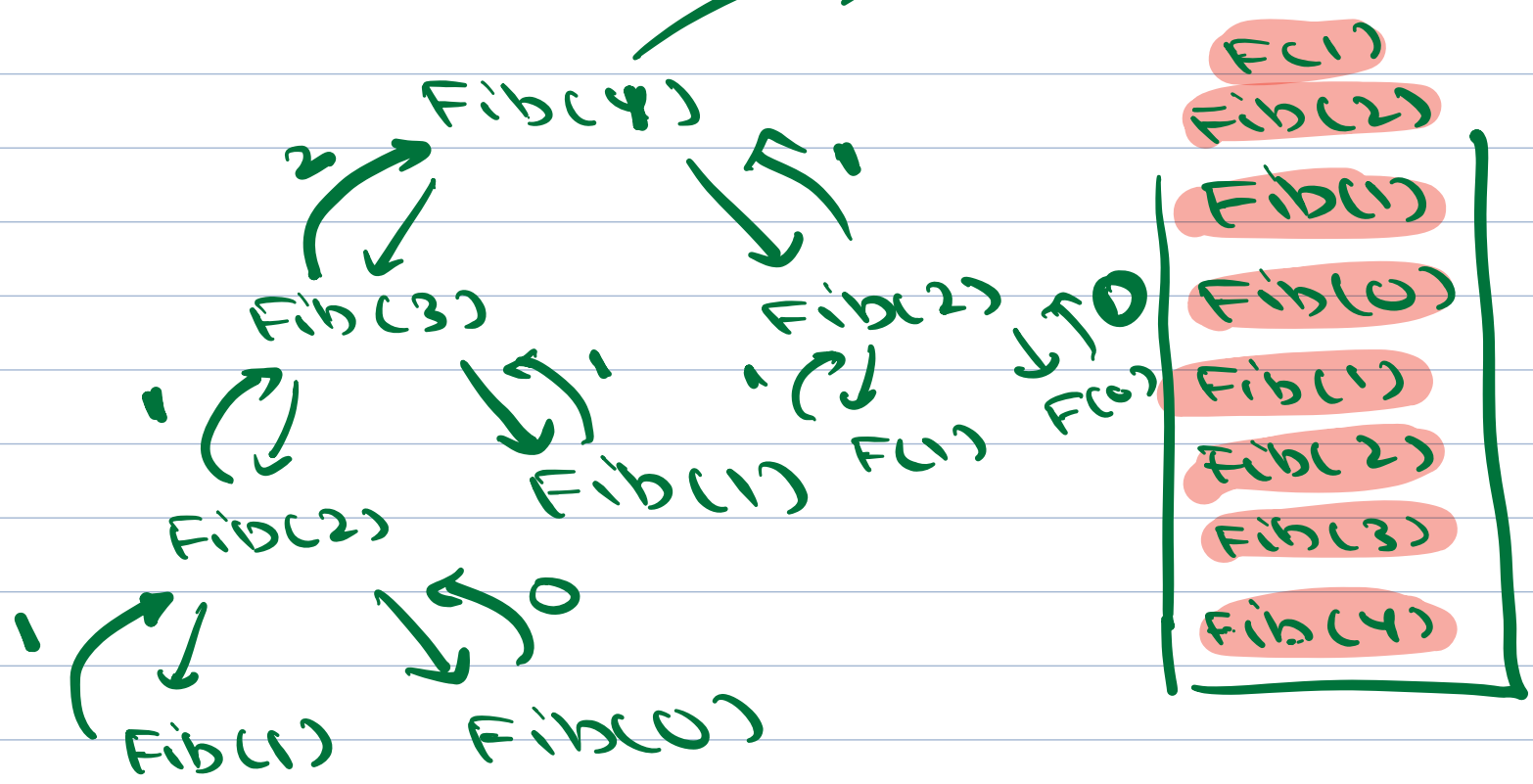
```
// Given N, return Nth term in series.
int fib (int N) {
    If (N == 0)   return 0
    If (N == 1)   return 1
    If (N <= 1)   return N

    return fib (N-1) + fib (N-2)
}
```

Fib(1) = Fib (0) + Fib (-1) ✗

Fib(0) = Fib (-1) + Fib (-2) ✗

→ 3

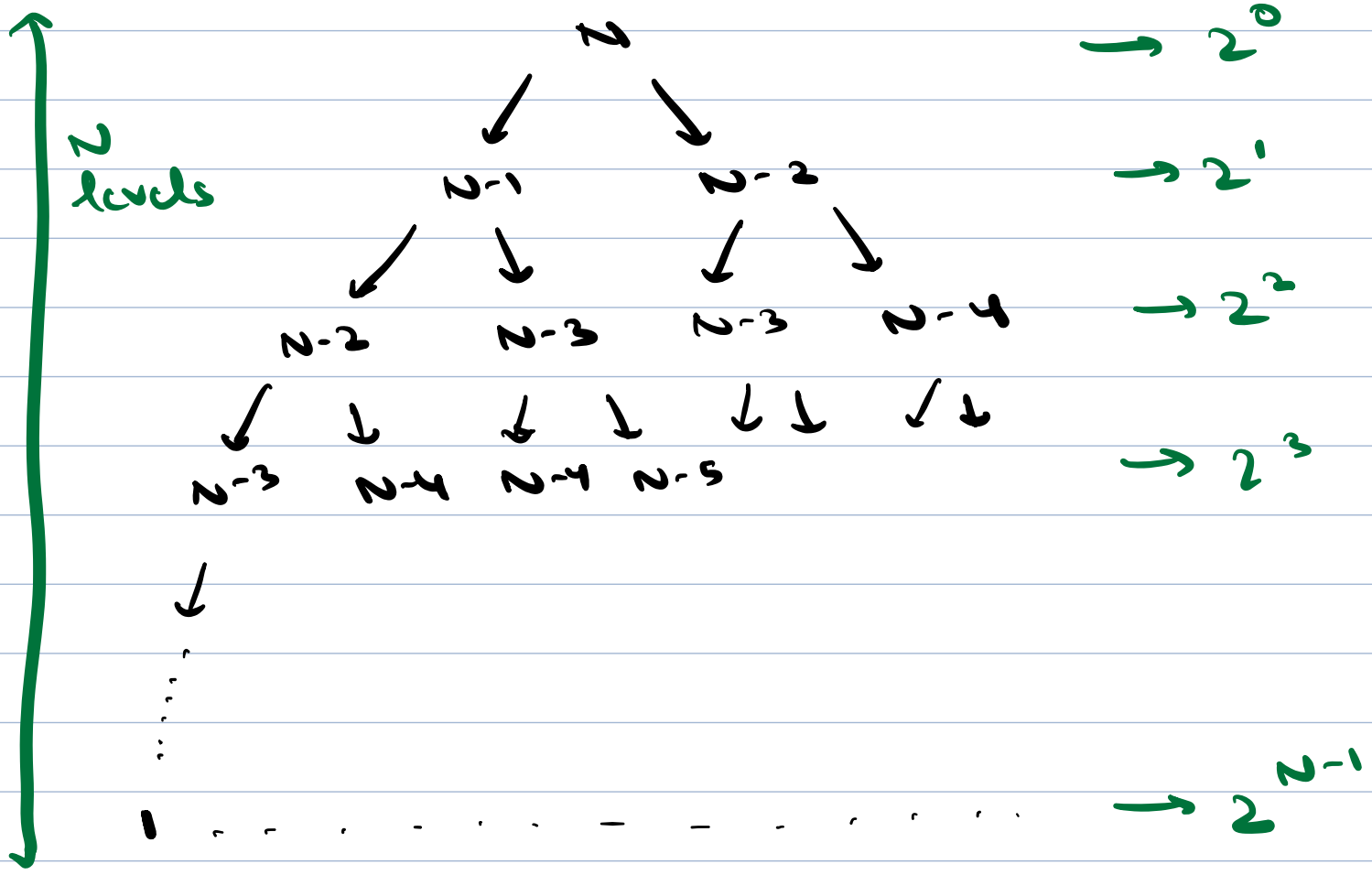


stack
size : 4

① If I go back to parent / up in the tree → stack size ↓

② If I go down in tree / called some fn → stack size ↑

SC : O(N) → fn call stack



Total fn calls = $2^0 + 2^1 + 2^2 + \ldots + 2^{N-1}$

$= \frac{1(2^N - 1)}{2-1} = 2^N - 1$

$S = \frac{a(r^N - 1)}{r-1}$

$a = 1$, $r = 2$

$N = N$

TC: Total fn calls x time of each call

$= (2^N - 1) \times 1$

TC : $O(2^N)$

SC : $O(N)$

Sum of XOR of all pairs

1, 2, 3

001, 010, 011

```
  1   001        1   001
^ 3   011      ^ 2   010
---   ---      ---   ---
  2    10        3    11
```

ans = 2 + 3 + 1 = 6

```
  2   010
^ 3   011
---   ---
  1   001
```


```
for (i = 0; i < n ; i++) {
    for ( j= i+1 ; j<n ; j++) {
        sum = sum + (arr[i] ^ arr[j])
    }
}
```

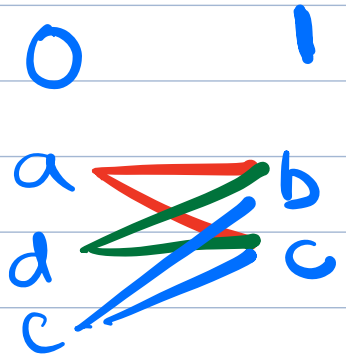
$O(N^2)$

$N = 10^5$

31 . . . . . 1 0

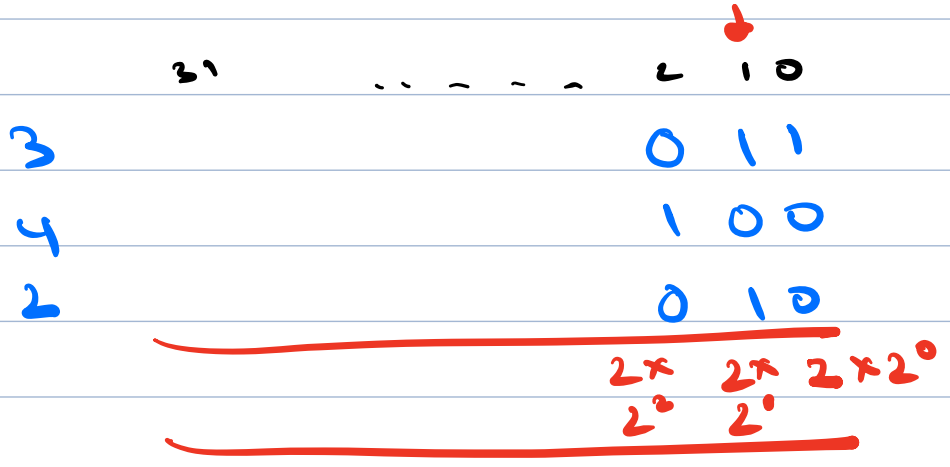
a 0
b 1
c 1
d 0
e 0

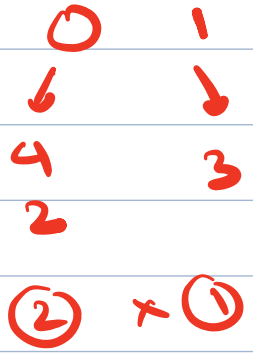
Contribution of ith bit = In how many xor pairs will the ith bit be set?



(3) x (2) (6)

Contri of 0^{th} bit = $6 \times 2^0 = 6$





ans = 2 + 4 + 8 = 14

sum = 0

```
for (pos = 0; pos < 31; pos++) {
    int cntSet = 0
    for (i = 0; i < N; i++) {
        if (checkBit(arr[i], pos) == true)
            cntSet++
    }
    pairs = (cntSet) * (N - cntSet)
    sum = sum + (pairs * (1 << pos))
}
```

2^{pos}

TC : O(32 N)

32×10^5

SC : O(1)

---

int N = 10^9

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

$\frac{10^9 \times (10^9+1)}{2}$

N = 5

Subarrays $= \frac{5 \times 6}{2} = 15$

① $(a+b) \% m = (a \% m + b \% m) \% m$

$0 \to m-1$ $\quad$ $0 \to m-1$ $\quad$ $0 \to m-1$

$(7+5) \% 8$

$\downarrow$

$12 \% 8$

4

$7 \% 8 + 5 \% 8$

$\downarrow \quad \downarrow$

$7 + 5$

$(12) \% 8$

4

$(a+m) \% m = (a \% m + \cancel{m \% m}^{\,0}) \% m$

$(a \% m) \% m$

$= a \% m$

$5 \% 8$

$= 5$

$(-5) \% 8 = (-5+8) \% 8$

$= 3 \% 8$

$= 3$

```
bar(x,y) {
    if (y==0) return 0
    return   x + bar(x,y-1)
}
```

$bar(x,y) = x + bar(x,y-1)$ ✓

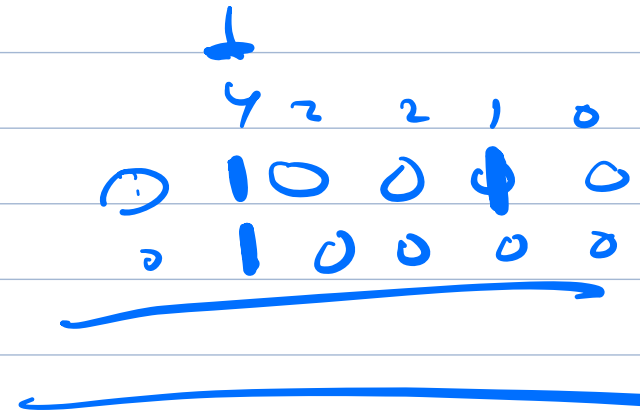
$= x + x + bar(x,y-2)$

$bar(x,y) = 2x + bar(x,y-2)$ ✓

$bar(x,y) = 2x + x + bar(x,y-3)$

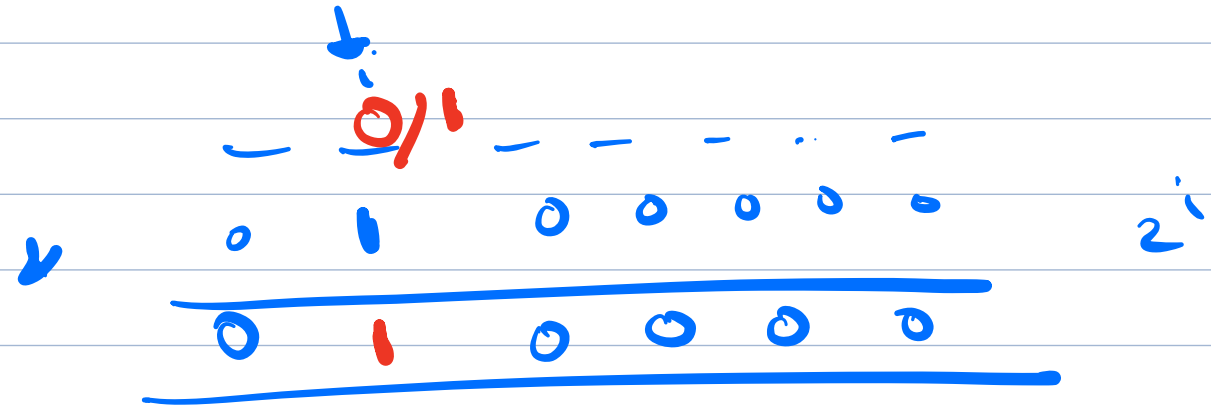
$= 3x + bar(x,y-3)$ ✓

$bar(x,y) =$



$1 << 4$

2^4



If num has ith bit unset

num & (1<<i) == 0

If num has ith bit set

num & (1<<i) == 1<<i