T.C. of TOH

$$T(N) = 2T(N-1) + 1 \quad \text{—(1)}$$

put $N = N-1$ in eq' (1)

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

$$T(N) = 2\left(2T(N-2)+1\right) + 1$$

$$= 4T(N-2) + 2 + 1$$

put $N = N-2$ in eq' (1)

$$T(N-2) = 2T(N-3) + 1$$

$$T(N) = 4\left(2T(N-3)+1\right) + 2 + 1$$

$$= 8T(N-3) + 4 + 2 + 1$$

put $N = N-3$ in eq' (1)

$$T(N-3) = 2T(N-4) + 1$$

$$T(N) = 8\left(2T(N-4)+1\right) + 4 + 2 + 1$$

$$T(N) = 16T(N-4) + 8 + 4 + 2 + 1$$

$$= 2^4 T(N-4) + 2^3 + 2^2 + 2^1 + 2^0$$

K times

$$T(N) = 2^K T(N-K) + 2^{K-1} + 2^{K-2} + 2^{K-3} \ldots 2^0$$

$N-K = 0$

$K = N$

$$T(N) = 2^N T(N-N) + 2^{N-1} + 2^{N-2} \ldots 2^0$$

$T(N-N) \to T(1)$

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

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

$N+1$ terms

$a = 1$

$r = 2$

$$T(N) = \frac{1\left(2^{(N+1)} - 1\right)}{(2-1)} \approx 2^{N+1}$$

$$T.C. = O(2^N)$$

$$2^{N+1} = 2 \times 2^N$$

0: 1

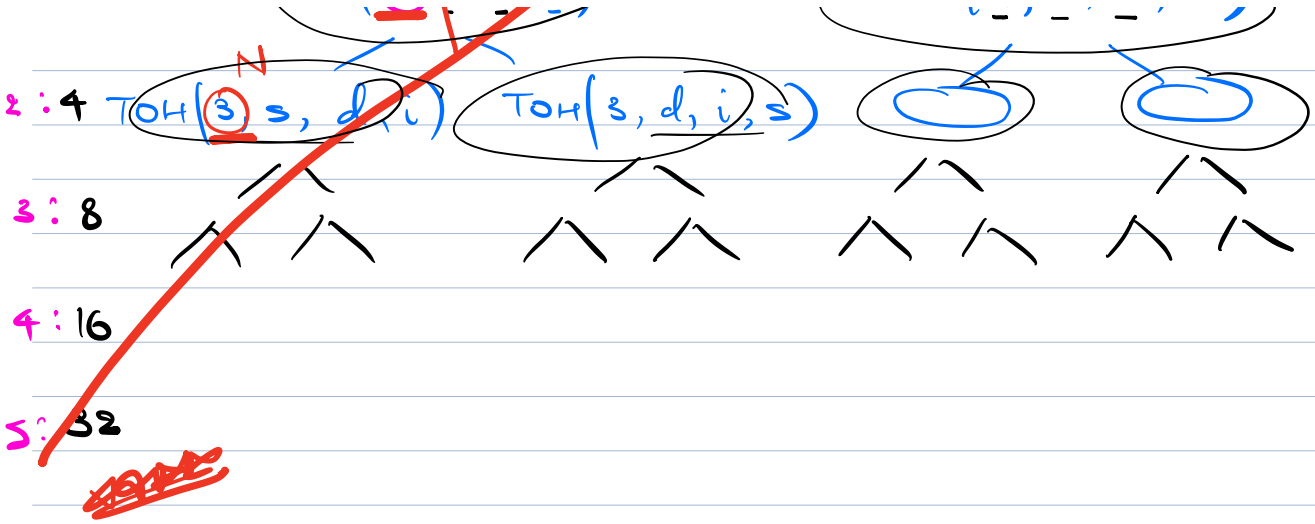
1: 2

Height $= (N)$

TOH(N, s, d, i)

TOH(N, s, i, d)

TOH(N, i, d, s)



Time Complexity of any recursive function = (No. of f^n calls) × T.C. of every recursive call.

No. of nodes in the recursive tree

$$= \left(\underbrace{1}_{2^0} + 2 + 4 + 8 \cdots\cdots \underbrace{2^{N-1}}\right) \times O(1)$$

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

$$\text{T.C.} = O(2^N)$$

```
void printTOH(N, src, dest, int){
    if (N == 1){
        print("Ring 1 : src -> dest");
```

return;

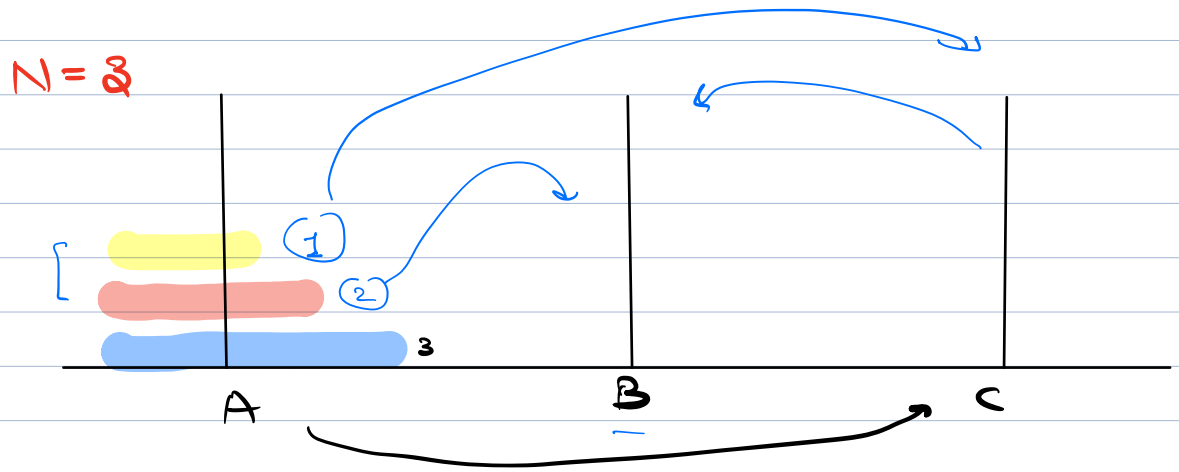
}

printTOH(N-1, src, int, dest);

print (Ring N: src → dest);

printTOH(N-1, int, dest, src);

}



printTOH(3, A, C, B)

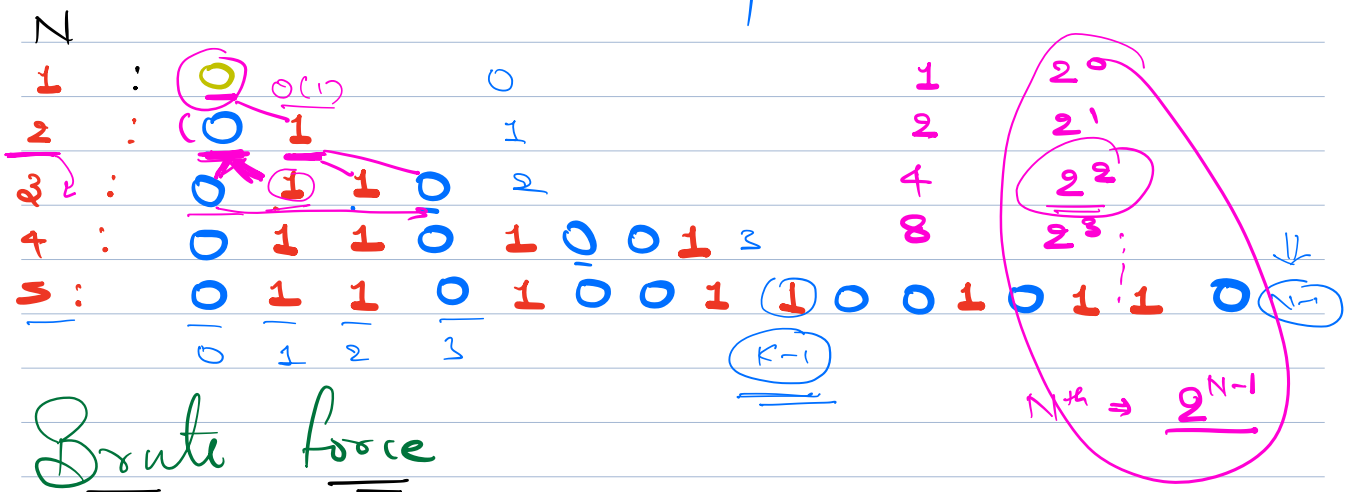
print(2, A, B, C) print printTOH(2, B, C, A)

TOH(1, A, C, B) print2: A->B TOH(1, C, B, A)

Q2 Given N & K. Return the value at the (Kth) index in (Nth) row.

(Assume that the I/P is always valid)

for the next row, change 0 to 01 & 1 to 10 in the prev row.



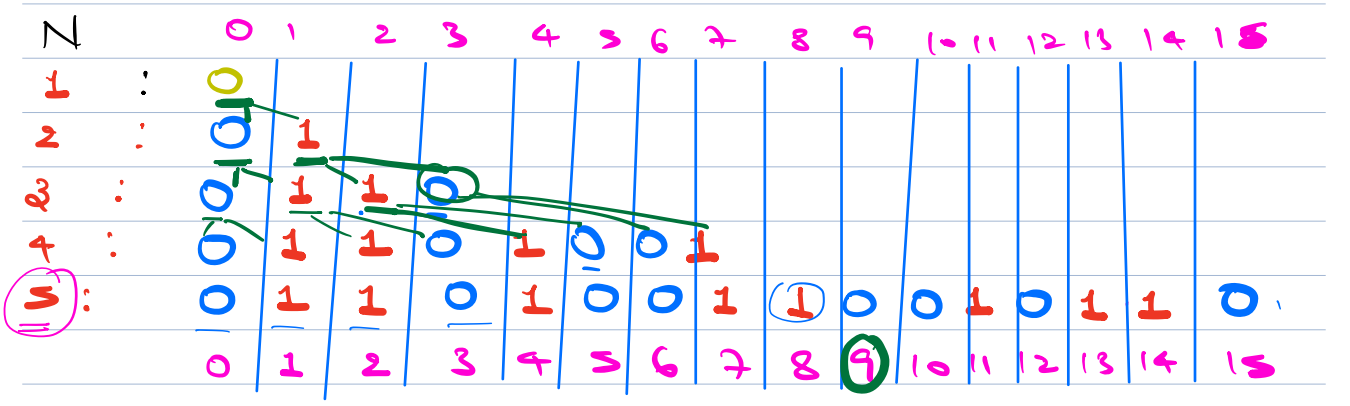
Brute force

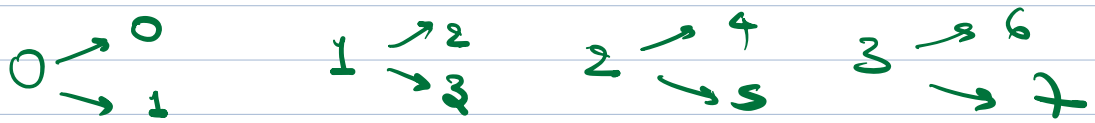
⇒ Generate N rows ⇒

⇒ Return A[N-1][K-1]

T.C. = $2^0 + 2^1 + 2^2 \ldots\ldots 2^{N-1}$

$= 2^N - 1$

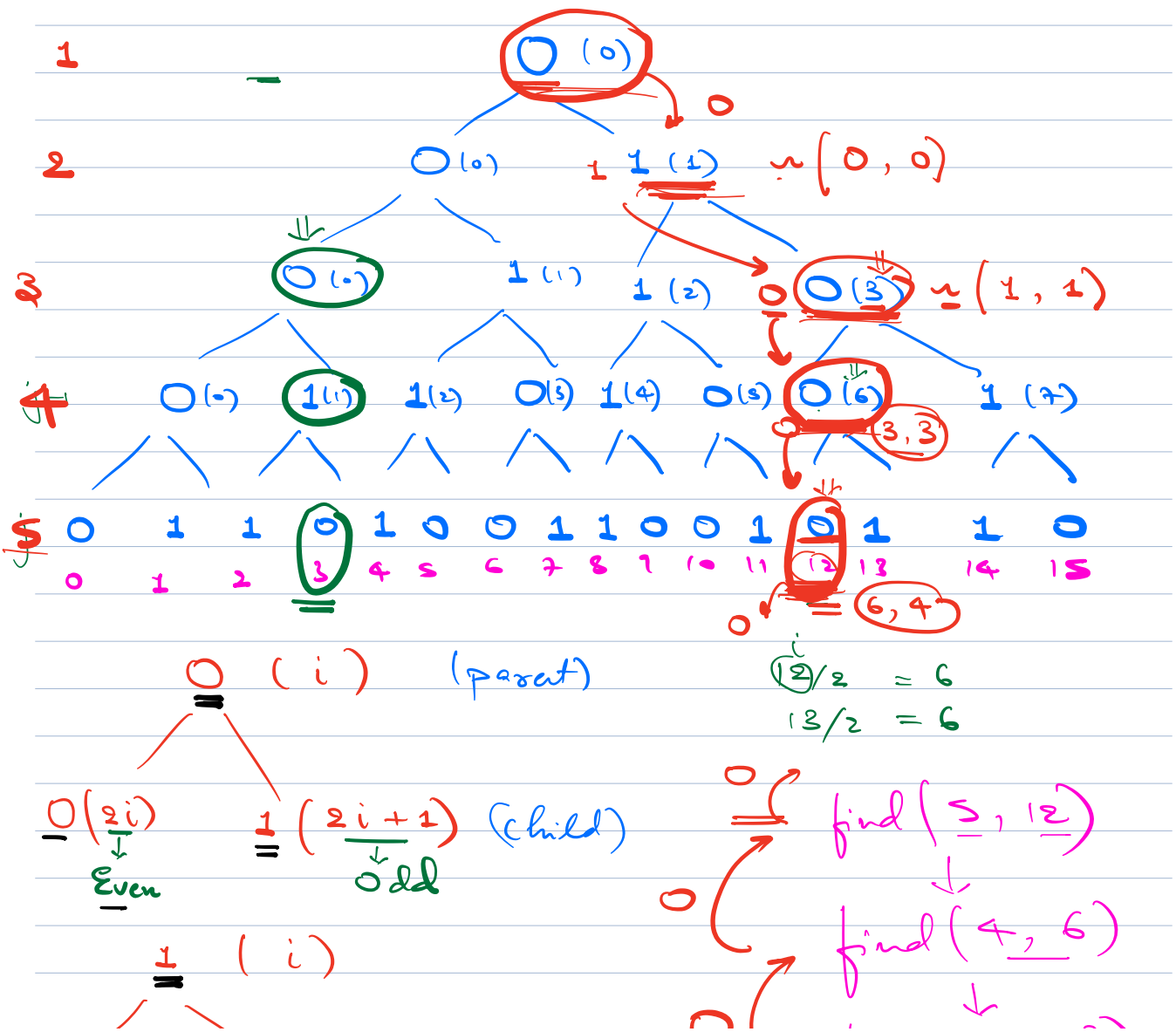




1) i^{th} index of row j generates

$2i$, $(2i+1)^{th}$ index of row $(j+1)$

2) i^{th} index of j^{th} row can be generated using $(i/2)^{th}$ element of $(j-1)^{th}$ row.



1 (2i) 0 (2i+1)

Even Odd

find (3, 3)
1 (find (2, 1)
0 (find (1, 0)

```
int find (N, K) {
    if (K == 0) { return 0; }
    int parentNode = find (N-1, K/2);
    if (K % 2 == 0) {
        return parentNode;
    }
    else {
        return (1 - parentNode);
    }
}
```

0 → 1-0 = 1
1 → 1-1 =

Any element of N^{th} row

⇓

at max N recursive calls

T.C. = $O(N)$

Gray Code

Different binary sequence where consecutive no. only differ by 1 bit

N = 3

000
001
011
010
110
111
101
100

→ 2 × 2 × 2

0 1

$2^3 = 8$

Given N, Generate all gray code no's of N bits

N = 1

0
1

N = 2

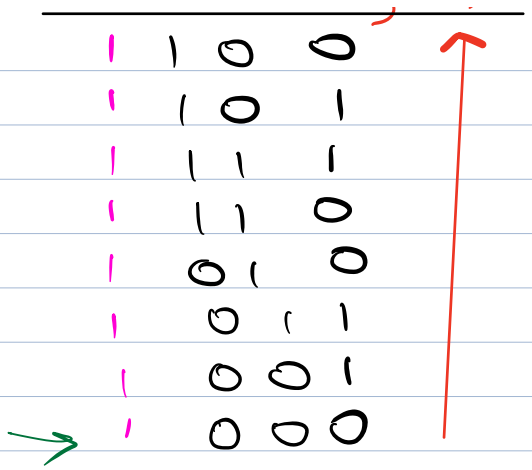
00
01
11
10

N = 3

000
001
011
010
110
111
101
100

→

0000
0001
0011
0010
0110
0111
0101
0100



not cleared though

Code

Ass: grayCode(N): return the list of possible gray codes using n bits

Main Logic: grayCode(N) uses grayCode(N-1)

Base Case: N = 1 {"0", "1"}
N = 0 {}

T(N)

```
list<string> grayCode(N) {
O(1)   if (N==0) { return {}; }
O(1)   if (N==1) { return {"0","1"}; }
O(1)   list<string> codes = grayCode(N-1);   // T(N-1)
O(1)   list<string> ans = new list<string>();
```

$O(2^{N-1})$ for (i = 0; i < codes.length(); i++) {
    ans.add('0' + codes[i]);
}

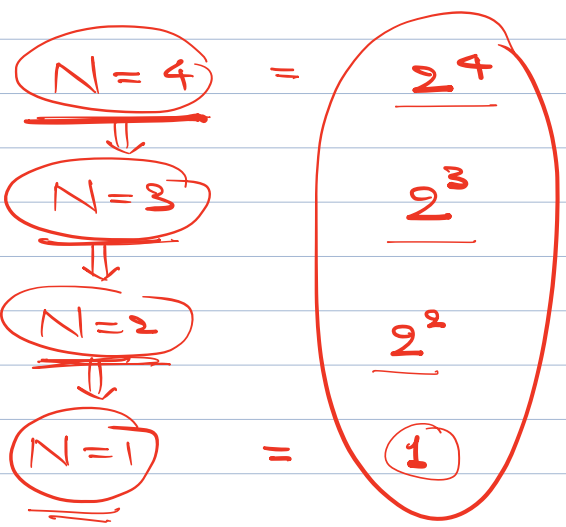
$O(2^{N-1})$ for (i = codes.length()-1; i >= 0; i--) {
    ans.add('1' + codes[i]);
}

$O(1)$ return ans;
}

T.C. of one recursive call

$= 2^{N-1} + 2^{N-1}$

T.C. $= O(2^N)$



$N = 1 + 2^2 + 2^3 + 2^4 \ldots\ldots 2^N + 2^1 - 2^1$

$$2^0 + 2^1 + 2^2 \quad \cdots \quad 2^N - 2$$

$$\Downarrow$$

$$(2^{N+1} - 1) - 2$$

$$= 2^{N+1} - 3$$

$$T.C. = O(2^{N+1})$$

$$\text{GP} = \frac{a(r^{(N+1)} - 1)}{r - 1}$$

$a = 1, \quad r = 2$

PS session on Thursday

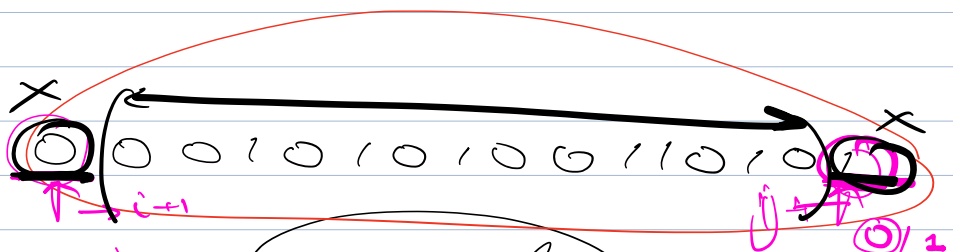
$\Downarrow$

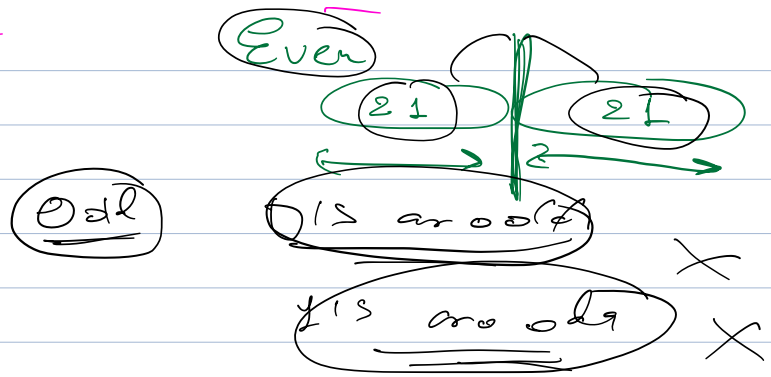
Thread

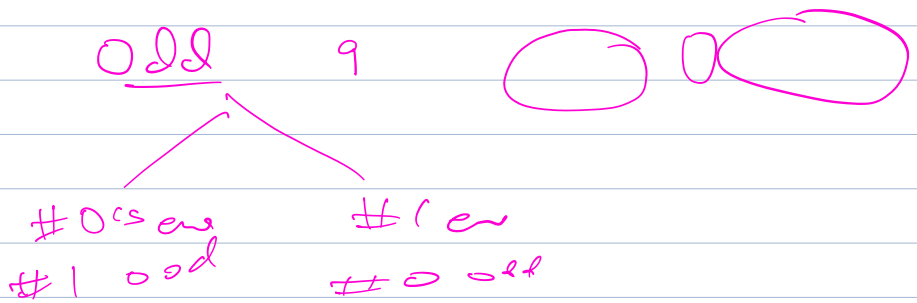
↳ Topic, problem

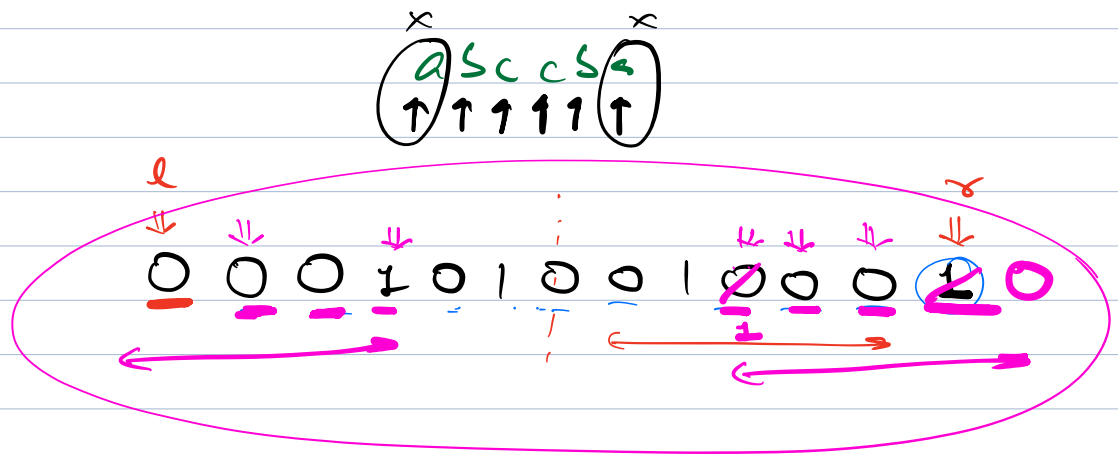
1 problem $\Rightarrow$ 1 message

Doubt









Code

$$T(N) = T(N-1) + T(N-2) + O(1)$$

$$N-1 \simeq N-2$$

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

$$2^N$$