RECURSION: breaking a big problem into smaller problem.

ex: $S(N) = 1 + 2 + 3 + \ldots$

big problem

$= S(N-1) + N$

breaking into smaller problem

1

```
int sum(int N){
  if (N==1)
    return 1;
  else
    return(N + Sum(N-1));
}
```

2

```
int fact(int N)
  ~~if (N==0)~~
    ~~return 1;~~
  if (N==1)
    return 1; // Time taken = 1
  else
    return N * fact(N-1);
```

Time taken 1

$T(N-1)$

3

```
int fib(int N)
  if (N==1 || N==2)
    return 1
  return fib(N-1) + fib(N-2) + 1
```

$T(N-1)$

$T(N-2)$

4

```
int APsum(int a, int d, int N)
  if (N==1)
    return a;
  else
    APsum(a, d, N-1);
      + a+(N-1)d
```

- Normally Time complexity = # iterations, but for Recursion we use Recurrence Relations.

for 1: $T(N) = T(N-1) + 1$

$\therefore T(N) = T(N-2) + 1 + 1$ replacing N with N-1

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

$\vdots$

$= T(N-K) + K$

compute for base condition and compute the unknown constant

$N - K = 1 \quad \therefore K = N-1$

$\therefore$ for base condition

$\therefore T(N) = T(1) + N - 1$

$= 1 + N - 1 = N \Rightarrow O(N)$

for 2:

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

$= T(N-2) + 2$

$\vdots$

$= T(N-K) + K$

base condition

$T(1) = 1, \quad N - K = 1$

$K = N-1$

$T(N) = T(N-K) + K$

$= T(1) + N - 1$

$= 1 + N - 1$

$= N$

$= O(N)$

for 4:

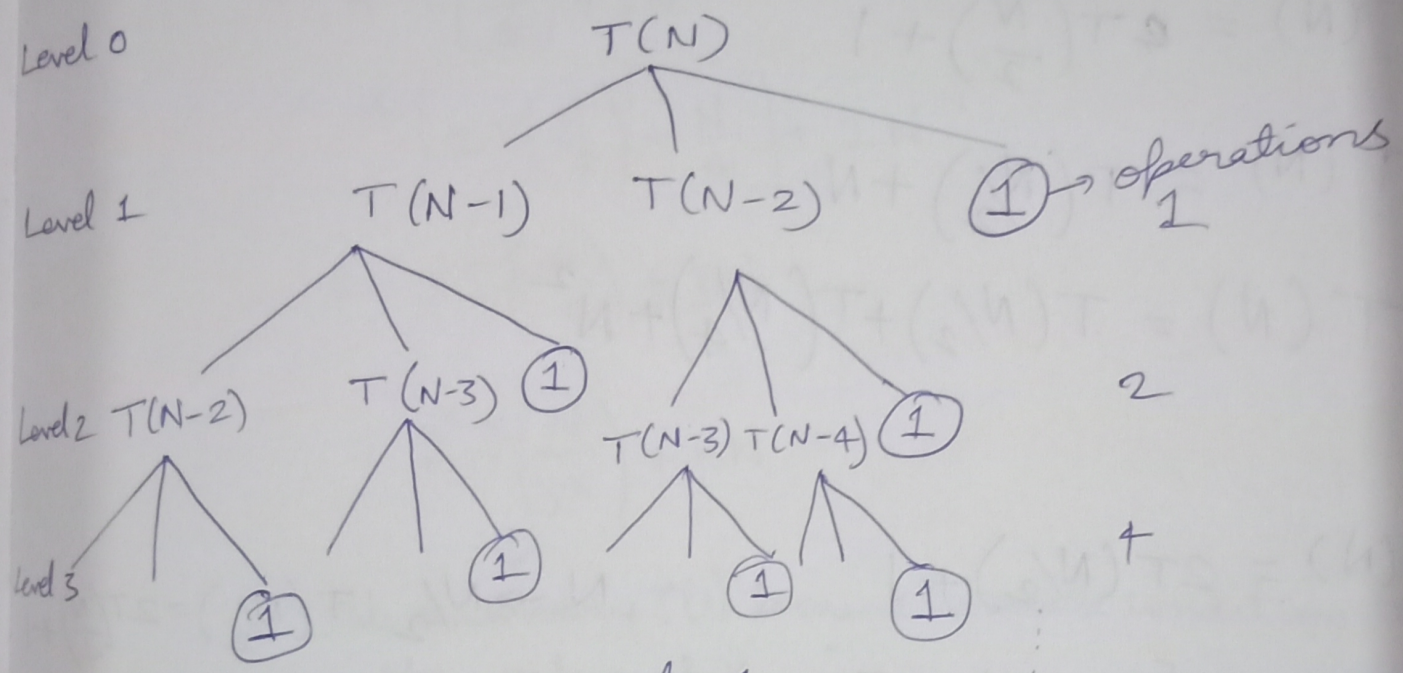
Similar to 2

for 3:

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

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

$$= \text{Recursion Tree}$$



In level 1, there is only 1 operations which is 1, because value of $T(N-1)$ and $T(N-2)$ are unknown

Level 2 : 2

Level 3 : 4

Level 4 : 8

Level 5 : 16

for level 1 $T(N-1)$

level 2 $T(N-2)$

level 3 $T(N-3)$

we should reach level N $T(0)$ until this is reached

this goes for N times

$$\therefore T(N) = \underset{2^0}{1} + \underset{2^1}{2} + \underset{2^2}{4} + \underset{2^3}{8} + \underset{2^4}{16} + \cdots \underline{2^N}$$

2^N → goes until N times

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

$$= O(2^N)$$

Sum of GP $\frac{a(r^n - 1)}{r - 1}$

$T(1) = 1$

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

2) $T(N) = 2T\left(\frac{N}{2}\right) + N$

3) $T(N) = 2T\left(\frac{N}{2}\right) + 1$

4) $T(N) = T(N/2) + N$

5) $T(N) = T(N/2) + T(N/4) + N^2$

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

$N \to N/2 : T(N/2) = 2T\left(\frac{N}{4}\right)$

$N \to N/4 : T\left(\frac{N}{4}\right) = 2T\left(\frac{N}{8}\right)$

$$= 2\left[2T\left(\frac{N}{4}\right) + 1\right] + 1$$

$$= 4T\left(\frac{N}{4}\right) + 3$$

$$= 4\left[2T\left(\frac{N}{8}\right) + 1\right] + 3$$

$$= 8T\left(\frac{N}{8}\right) + 7$$

$\vdots$

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

This should become $T(1)$ because it is base case

$\therefore \frac{N}{2^K} = 1$

$2^K = N$

$K = \log_2 N$

Substituting K value

$$= 2^{\log_2 N} T(1) + 2^{\log_2 N} - 1$$

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

) $T(N) = 2T(N/2) + N$

$N \to N/2 : T\left(\frac{N}{2}\right) = 2T\left(\frac{N}{4}\right) + \frac{N}{2}$

$N \to N/4 : T\left(\frac{N}{4}\right) = 2T\left(\frac{N}{8}\right) + \frac{N}{4}$

$$= 2\left[2T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N$$

$$= 4T\left(\frac{N}{4}\right) + 2N$$

$$= 4\left[2T\left(\frac{N}{8}\right) + \frac{N}{4}\right] + 2N$$

$$= 8T\left(\frac{N}{8}\right) + 3N$$

$$\vdots$$

$$= 2^K T\left(\frac{N}{2^K}\right) + KN$$

$\downarrow$

This should become $T(1)$ because it is base condition to know the value of K

$$\therefore \frac{N}{2^K} = 1$$

$$K = \log_2 N$$

substituting K value

$$T(N) = 2^{\log_2 N} T\left(\cancel{N}/N\right) + (\log_2 N) \times N$$

$$= NT(1) + N\log_2 N$$

$$\therefore T(N) = \cancel{N} + N\log_2 N \Rightarrow O(N\log_2 N)$$

$N \to \frac{N}{2} : T\left(\frac{N}{2}\right) = T\left(\frac{N}{4}\right) + 1$

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

$$= T\left(\frac{N}{4}\right) + 1 + 1$$

$$= T\left(\frac{N}{8}\right) + 1 + 1 + 1$$

$$\vdots$$

$$= T\left(\frac{N}{2^K}\right) + \cancel{K}$$

$T(1) \downarrow \therefore K = \log_2 N$

$T(N) = T(1) + \log_2 N$

$\Rightarrow O(\log_2 N)$

4) $T(N) = T\left(\frac{N}{2}\right) + N$

$$= T\left(\frac{N}{4}\right) + \frac{N}{2} + N = T\left(\frac{N}{4}\right) + \frac{3N}{2} \quad \frac{(4-1)}{2}N$$

$$= T\left(\frac{N}{8}\right) + \frac{3N}{2} + \frac{N}{4} = T\left(\frac{N}{8}\right) + \frac{7N}{4} \quad \frac{(8-1)}{2^2}N$$

$$\vdots$$

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

$$\downarrow$$

$$T(1)$$

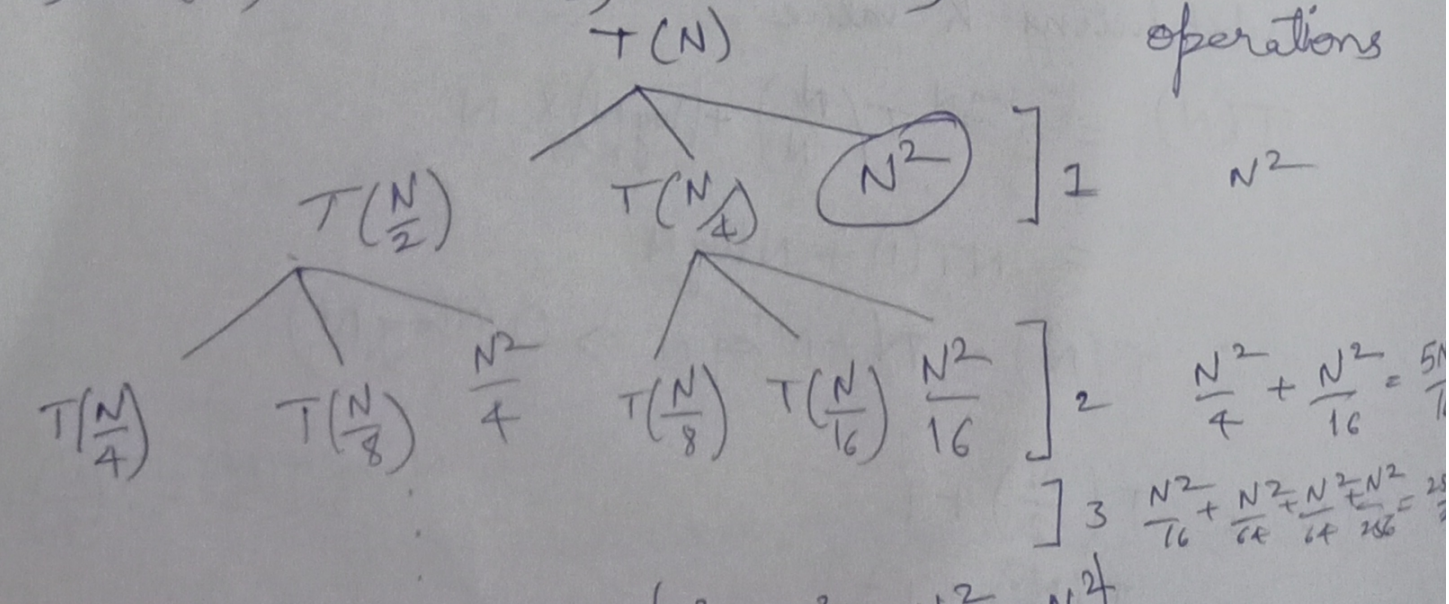
$\therefore k = \log_2 N$, substituting k value

$$\therefore T(N) = T(1) + \frac{2^{\log_2 N} - 1}{2^{\log_2 N - 1}} \times N$$

$$= 1 + \frac{N-1}{\frac{N}{2}} \times N$$

$$= 1 + 2N - 2 = 2N - 1 \Rightarrow O(N)$$

5) $T(N) = T(N/2) + T(N/4) + N^2$



$$\text{Total} = N^2 + \left(\frac{N^2}{4} + \frac{N^2}{16}\right) + \left(\frac{N^2}{16} + \frac{N^2}{64} + \frac{N^2}{64} + \frac{N^2}{256}\right) + \ldots$$

$$= \left(\frac{5}{6}\right)^0 N^2 + \left(\frac{5}{16}\right) N^2 + \left(\frac{5}{16}\right)^2 N^2 + \ldots \quad = N^2\left(\left(\frac{5}{6}\right)^0 + \left(\frac{5}{6}\right)^1 + \left(\frac{5}{6}\right)^2 \ldots\right)$$

$$= N^2 \left[\frac{1 \cdot \left(1 - \left(\frac{5}{16}\right)^n\right)}{1 - 5/16}\right] = N^2 \left[\frac{1 - \left(5/16\right)^n}{11/16}\right]$$

$$= \frac{16N^2}{11}\left(1-\left(\frac{5}{16}\right)^n\right) \quad \text{if } n=1: \overset{(5/16)^n}{0.3}$$

$n=2: 0.09$

$n=3: 0.027$

$\vdots$

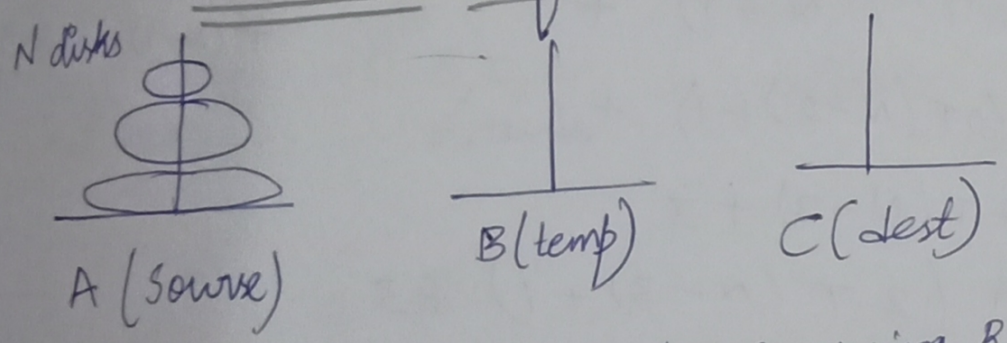
$\therefore$ as n increases $\left(\frac{5}{16}\right)^n$ approaches 0

$$= \frac{16N^2}{11}(1-0)$$

$$\therefore T(N) = \frac{16N^2}{11} \Rightarrow O(N^2)$$

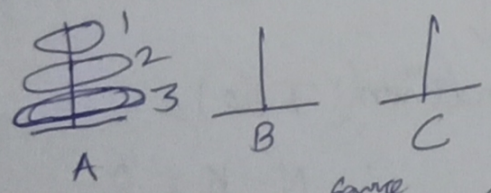
→ Tower of Hanoi

(Given N, print the steps taken to move from A to C)



move from A to C using B with constraints that ① only one disk at a time and ② no bigger disk on smaller disk

if $N=3$



1: A → C

2: A → B

1: C → B

3: A → C

1: B → A

2: B → C

1: A → C

N : A → C, B (source, dest, temp) ~~to~~ where should move N^{th} disk to C, to achieve this N-1 disks should be in B.

$\therefore$ N-1 : A → B, C

print N^{th}: A → C

after moving N-1 disks to B, we can move N^{th} disk to C.

MAIN: TOH(3, A, C, B)

```
void TOH(int N, char src, char dest, char tmp)
{
    If(N==0) return; } ① 1
    TOH(N-1, src, tmp, dest); } ② T(N-1)
System.out.println(N+" : "+src+"→"dest); } ③ 1
    TOH(N-1, tmp, dest, src); } ④ T(N-1)
}
```

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

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

$N \to N-1$

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

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

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

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

$$= 8T(N-3) + 7$$

$$\vdots$$

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

$\downarrow$

base condition $T(0) = 1$

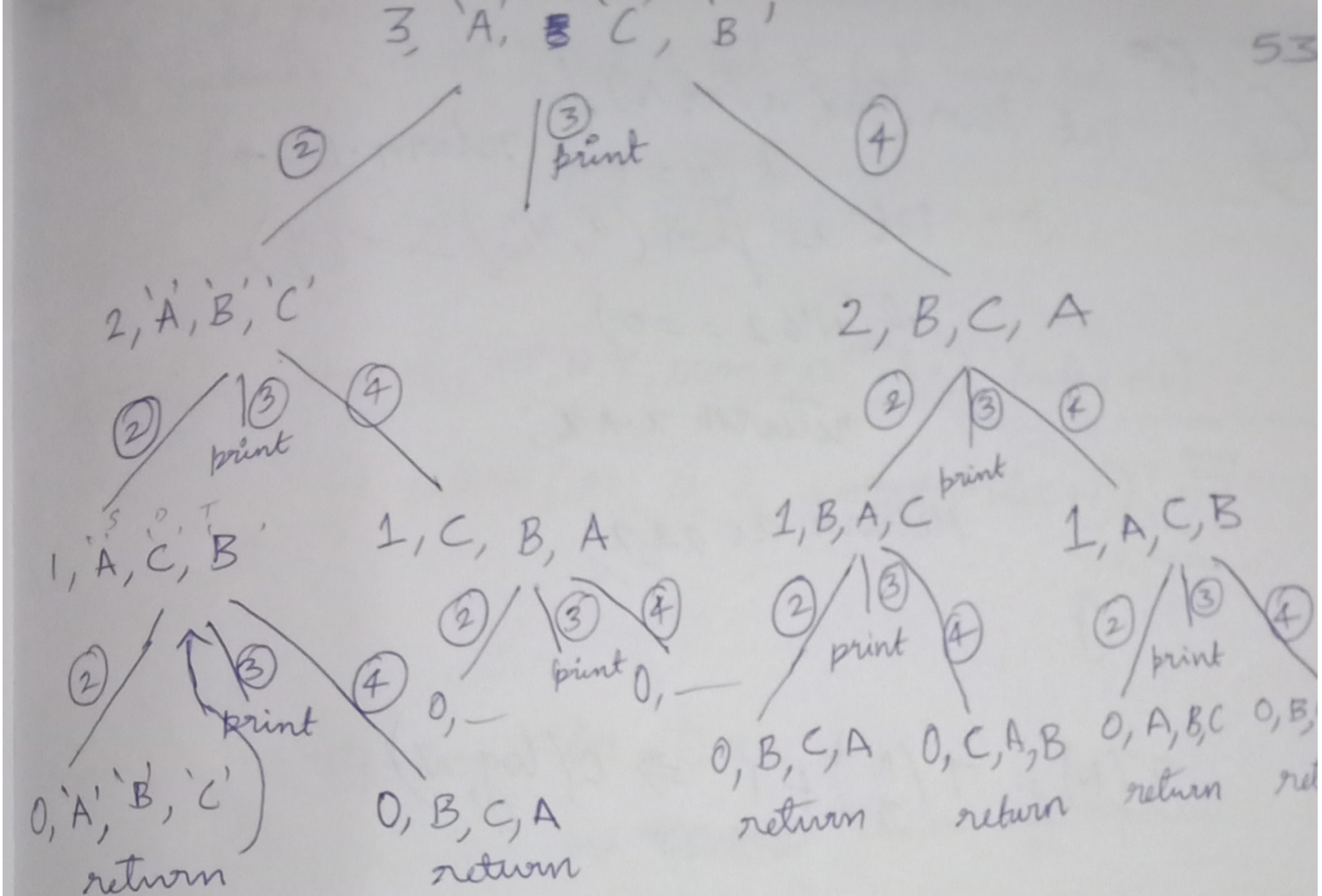
$\therefore N - K = 0$

$K = N$

$$\therefore T(N) = 2^N T(0) + 2^N - 1$$

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

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



$\longrightarrow$ $\underline{\underline{a^N}} = a \cdot a^{N-1}$

A:

```
int pwr(int a, int N)
{
    if (N==0) return 1;
    return a*pwr(a, N-1);
}
```

$T(N) = T(N-1)+1$
$\Rightarrow O(N)$

$\Rightarrow O(N)$

B:

```
int pwr(int a, int N) {
    if(N==0) return 1;
    if(N%2==0)
        return pwr(a, N/2) * pwr(a, N/2);
    return a* pwr(a, N/2) * pwr(a, N/2);
```

$a^N = a^{N/2} \cdot a^{N/2}$

$a^{12} = a^6 \cdot a^6$

$a^{13} = a \cdot a^6 \cdot a^6$

$T(N) = T(\frac{N}{2}) + T(\frac{N}{2}) + 1$
$= 2T(\frac{N}{2}) + 1$
$= O(N)$

C:

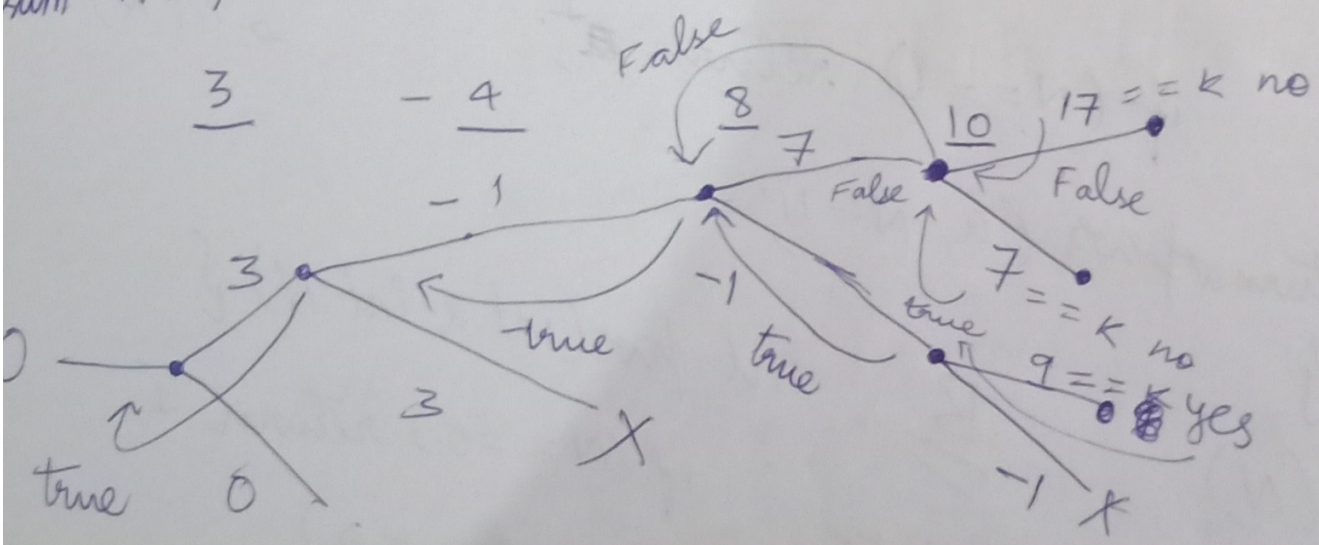
```
int pow(int a, int N) {
    if (N==0) return 1; → 1
    int x = pow(a, N/2); → T(N/2)
    if (N%2==0)
        return x*x;
    return a*x*x;
}
```

$$T(N) = T\left(\frac{N}{2}\right) + 1 \Rightarrow O(\log_2 N)$$

→ Subset Sum:

	1	2	3
ar	3	-4	8	10
N	2 *	2 *	2 * 2
	take don't take	take don't take	

sum K = 9



MAIN: if (subsets (ar, N, K, 0, 0) = = T) print Yes
else print NO

```
bool subsets (int ar[ ], int N, int K, int sum, int index)
{
    if (index = = N) return sum = = K;

return  subsets(ar, N, K, sum + ar[index], index + 1)
                                            ↳ index increments once ∴ T(N-1)
        || subsets( ar, N, K, sum, index + 1)
                                            ↳ T(N-1)
}
```

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

similar to Tower of Hanoi

$\therefore T(N) = O(2^N)$

→

N = 4

1) ( ) ( )
2) (( ))

if '(' < ')'
∴ after (( ))
arranging ( ) ( )

N = 6

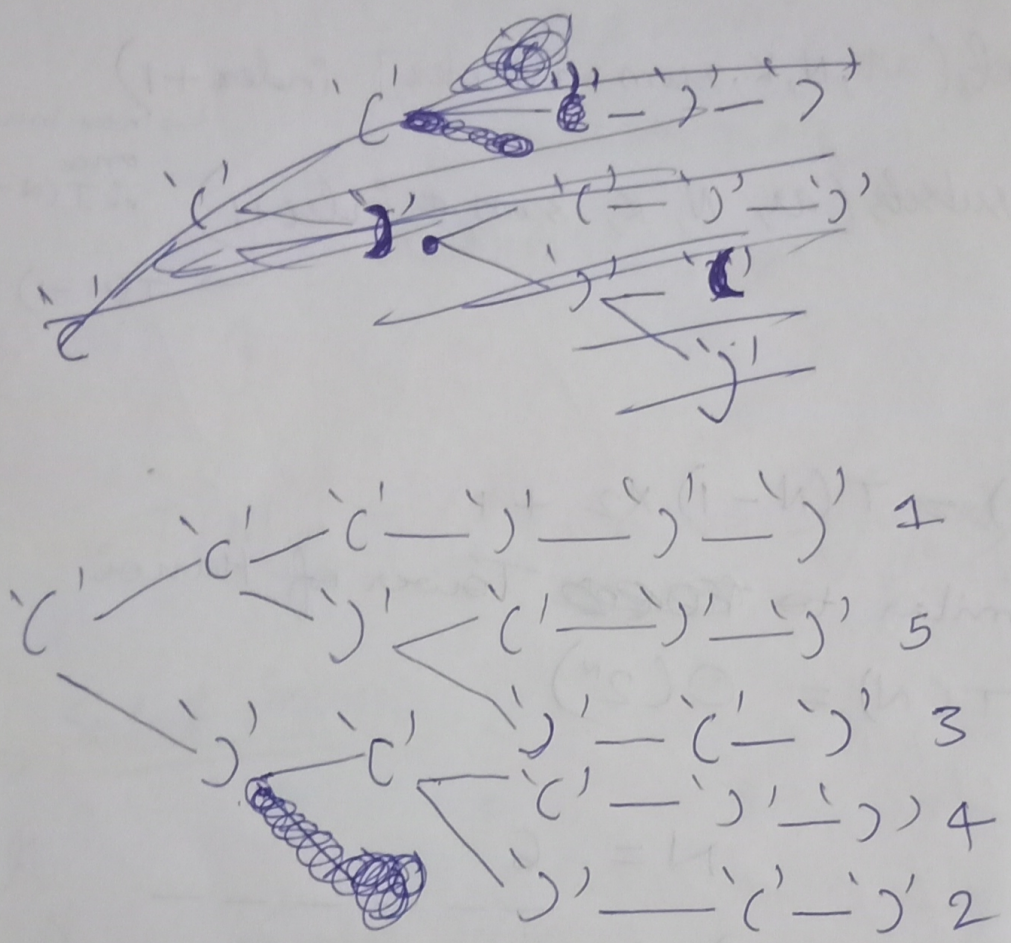
1) ((( )))
2) ( ) ( ) ( )
3) (( )) ( )
4) ( ) (( ))
5) (( ) ( ))

order: 1, 5, 3, 4, 2

Given N paranthesis, print paranthesis in lexicographic order.

N = 6

0	1	2	3	4	5


'('
')'
'('
')'
'('
')'

→ Mat(i)(j) ∈ [1, 9] unique, sum of all rows, columns and diagonals should be same

Magic Squares

M1

4 (2)	9 (1)	2 (3)
3 (2)	5 (2)	7 (0)
8 (4)	1 (8)	6 (0)

cost = 22

M2

8	3	4
1	5	9
6	7	2

cost = 24

I/p:

2	8	5
1	3	7
4	9	6

convert into magic square

o/p: min cost magic square

Solution approach:

A: → Convert into 1-D array

0	1	2	3	4	5	6	7	8

Generate all permutations and find array with magic square

Inbuilt library for generating next permutation Google it

```
while (—) {
    ar = next per
    if (M.S(ar) == true)
    {
        ans = min (ans, cost (M.S(ar)))
    }
}
```

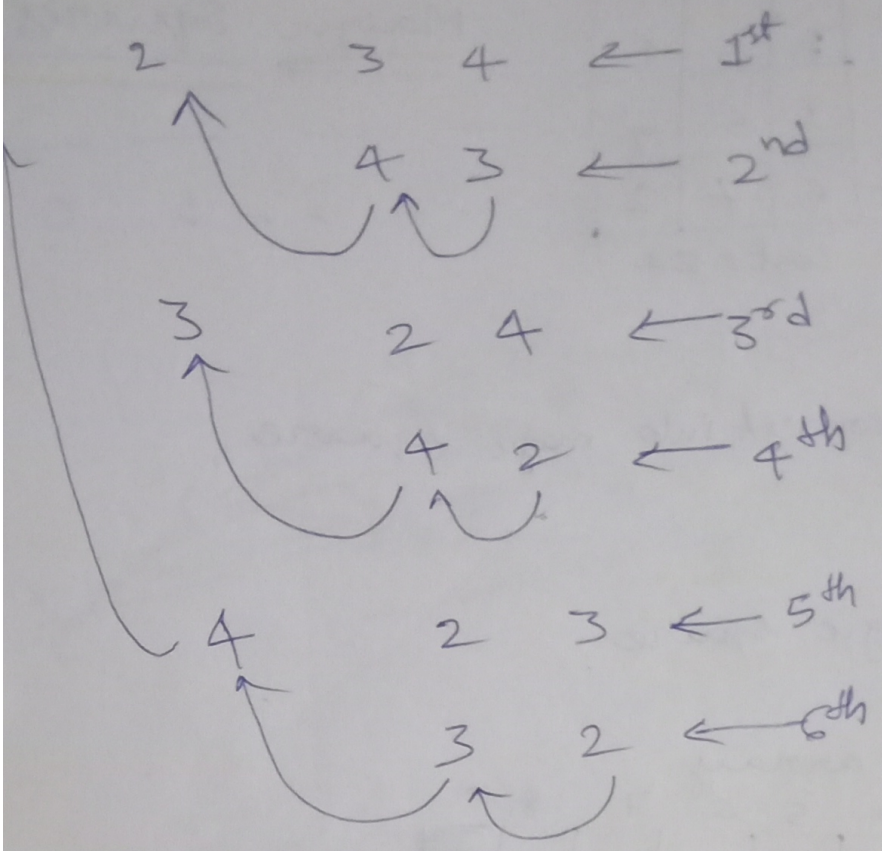
• Finding Permutations program

ar:

0	1	2	3

array size = 4

∴ total permutations = 4! = 24



2 ⟶ 6

3 ⟶ 6

4 ⟶ 6

24 possibilities

```
bool taken[10] = {F};
int ans =    ;

void solve (int ar[ ], int idx, bool taken[ ]) {
    if (idx == 9) { if (M.S(ar) == True) ans = min(ans, cost(I/P, ar));
                    return;
    for(int i=1; i<= 9; i++){
        if (taken[i] == false){
            taken[i] = T;
            ar[idx] = i;
            solve(ar, idx+1, taken);
            taken[i] = F;
}  }  }
```