Session - 6

SUBSET SUM

(1) arr N : 5 12 -8 16 4 20

K = 21

Subset whose sum = 1K
{16, 5}

(1) Bit manipulation → $2^N * N$ (Recall)

(2) Do it using recursion.

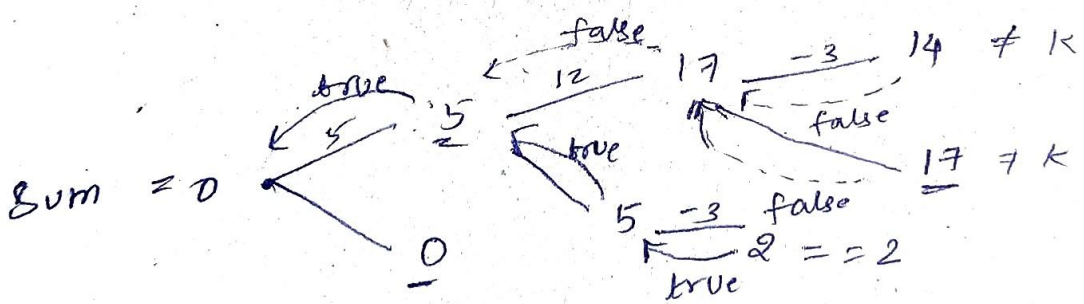
arr N : 5, 12, -3 sum = 2 → True

In an Array of size 'n', we'll have
2^n subsets.

arr N : 5 12 -3

2 chances 2 2

$2 * 2 * 2 = 2^3 \Rightarrow 2^N$



→ recursion

BACKTRACKING

(1) Figure out all the required parameters

(2) Recursion

bool subsetsum(arr, N, k, sum=0, idx=0)

k: required sum; sum: initial sum; idx: current position

{

if (idx == N)

return sum == k;

return subsetsum(arr, N, k, sum+arr[idx], idx+1)

||

subsetsum(arr, N, k, sum, idx+1);

}

T_C : $T(N) = 2T(N-1) + 1 \Rightarrow 2^N \Rightarrow O(2^N)$

(Better than previous Bit manipulation on $O(2^N * N)$

---

(2) VALID PARENTHESIS

( ) ( ) ( ) ✓

( ) (( ))) ✗ (1) Open and close brackets count should match

(( )) ( ) ✓

( )) ( ) ✗ (2) order

)) ((✗

M = 4, valid parenthesis of length N in lexicographical order

(<)

( ) ( )

(( )) ✓

N = 6,

( ) ( ) ( )

((( ))) ✓

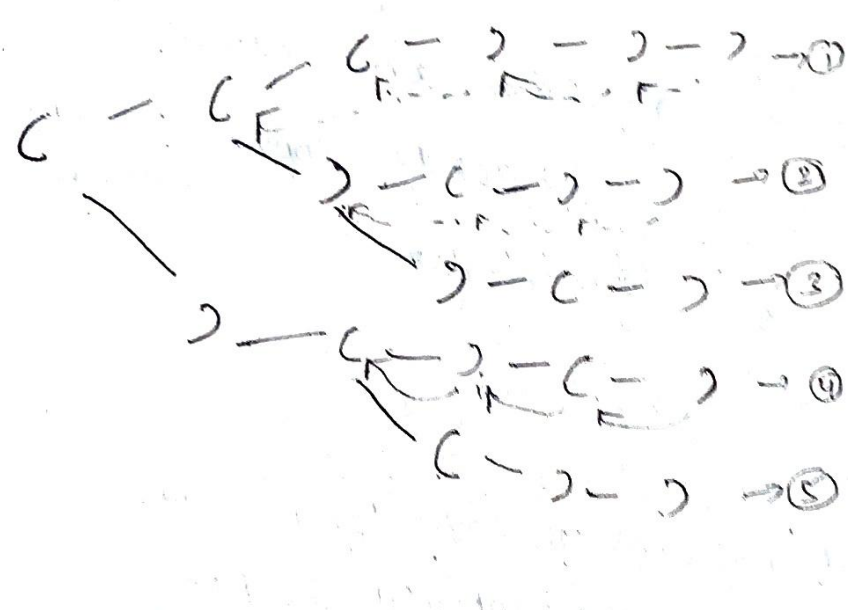
( ) (( ))

(( )) ( )

(( ) ( ))

2 ways — generate Every possibility → check → sort → 2^N

\ generate only valid one's




```
( ( ( ) ) )
( ( ) ( ) )
( ( ) ) ( )
( ) ( ) ( )
( ) ( ( ) )
```

```
void paranthesis( N, op, cl, char arr[], idx )
{
    if (idx == N) { print(arr); return; }
    if (op < N/2) {
        arr[idx] = '(';
        Paranthesis( N, op+1, cl, arr, idx+1 )
    }
    if (cl < op)
    {
        arr[idx] = ')';
        paranthesis( N, op, cl+1, arr, idx+1 )
```

Catalan number → help to figure out

T.C.

$\frac{2nCn}{n+1}$

N → 2n

$\frac{6C3}{4} = \frac{5}{=}$

TODO

---

(3)

8	9	7
3	4	1
5	2	6

→ magic square with minimum cost.

(1) rowsum = colsum = diagnol sum

(2) [1, 9]

(3) use all the digits

4	9	2
3	5	7
8	1	6

✓ magic square

8	1	6
3	5	7
4	9	2

4	9	2
8	1	6
3	5	7

cost :: Absolute difference b/w Each value

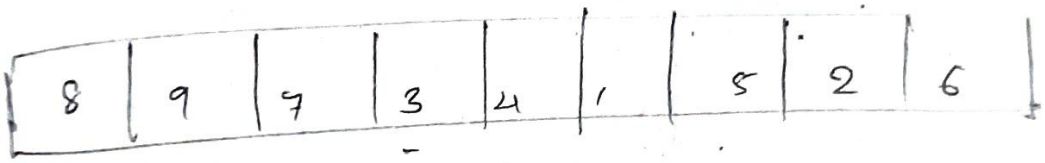
(1) → magic square generation

(2) → cost

(3) → answer → min cost

Let's make it simple,

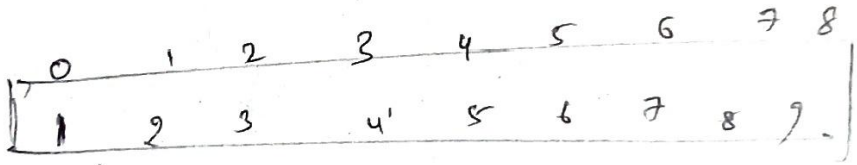
convert 2D array to 1D array



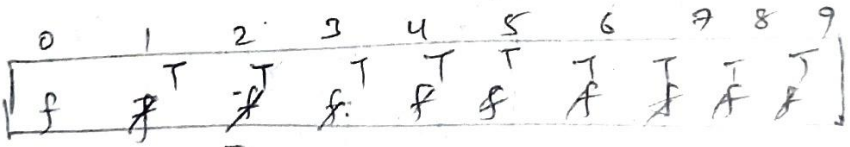
magic Array.

↳ [1,9]

magic Array



visited Array



1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 9 8

8 7 9

8 9 7

Generate Every permutation,
check whether it is
a magic square

```
int minCost = INT_MAX;
void permutations (arr, visited, magic_arr,
                   idx
{
    if (idx == 9)
    {
        if (magicSquare(magic_arr))
        {
            minCost = min(minCost, cost(arr,
                                        magic_arr))
        }
        return;
    }
    for (int i=1; i<=9; i++) {
        if(!visited[i])
        {
            magic[idx] = i;
            visited[i] = true;
            permutations(arr, vis, magicarr,
                         idx+1);
            visited[i] = false;
        }

main() {
    int arr[9];
    bool visited[10]
    int magic_arr[9]

    permutations(arr, visited, magic_arr,
                 idx)
```

Permutations * Inbuilt function.

(5)

L : { smart, mart, s, inter, view, interviews, views }

str : smartinterviews

1) T / F possible to partition str in such a way that Every partition is present in List.

2) Total no. of ways ?

3) minimum partitions (TODO)

```
int ways = 0;   partitions = INT_MAX;
bool partition( str, list, idx )
void
{
    int n = str.size();

    if ( idx == n)
        ways = ways+1; return;
        return true;

    for( int i = idx; i<N; i++)
    {   if ( str[idx:i] ∈ L ) {
            if (partition (str, list, i+1))
            {
                return true;
            }
        }
    }

    return false;
}
```