7/10/23

SRM

RECURSION

Recursion is a programming paradigm which will solve a bigger problem by solving smaller instances of it.

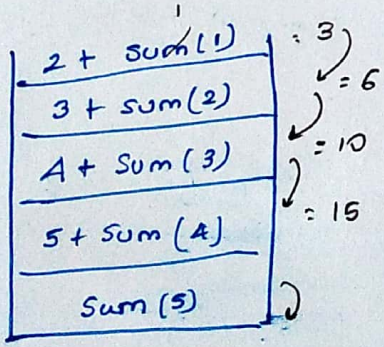
To solve a Recursive problem 3 steps

1. Assumption
2. Main logic
3. Base Condition

Order wise ↓

```
int Sum (int N)
{
    if(N==0) (or) N==1 } → Base Condition
        return 0;      return 1;
    return N + Sum (N-1); → Main logic
}
```

we will have call stack



if N==1

Main : 15

```
int fact (int N)
{
    if (N==0)
        return 1;
    else
    { return N x fact (N-1)
    }
}
```

T.C : O(N)

Recurance Relation $t(n) = t(n-1) * 1$

```
int fibo (int N)
{
    if (N==1 || N==2)
        return 1;
    return fibo(N-1) + fibo(N-2);
}
```

1 1 2 3 5 8 13

R.R: $t(N) = 1 + T(N-1) + T(N-2)$

Find the sum of N number in A.P

$a + (a+d) + (a+2d) + \ldots (a+(n-1)d)$.

approach-1

```
int apsum (int a, int d, int n)
{
    if (N==1)
        return a;
    return a + (n-1)d + apSum (a, d, n-1);
}
```

(or)

```
return a + apsum (a+d, d, n-1);
```

Time Complexity :-

for iterative Codes → we will Calculate t.c using ith iteration

for recursive Codes → Recurance Relation will be used.

also called Induction method

→ int sum (int N) → T(N)

{ if (N == 0) } → 1
return 0;

return N + sum (N-1);
↓ ↓
1 T(N-1)

}

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

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

⇒ $T(N) = T(N-1) + 1$

$= [T(N-2) + 1] + 1$ → substitute N-1

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

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

$= T(N-3) + 3$

⋮

$= T(N-K) + K$

$T(0) = 1$ → time taken to solve sum of 0 natural numbers constant time

$= T(N-N)$

$T(N-K) + K$

$= 1 + N$

$O(N+1)$

$T.C = O(N)$

$T(N-K) = T(0)$

$N - K = 0$

$N = K$

$T(0) = 1$

For fibonaaci

$T(N) = T(N-1) + T(N-2) + 1$ → Recurance Relation

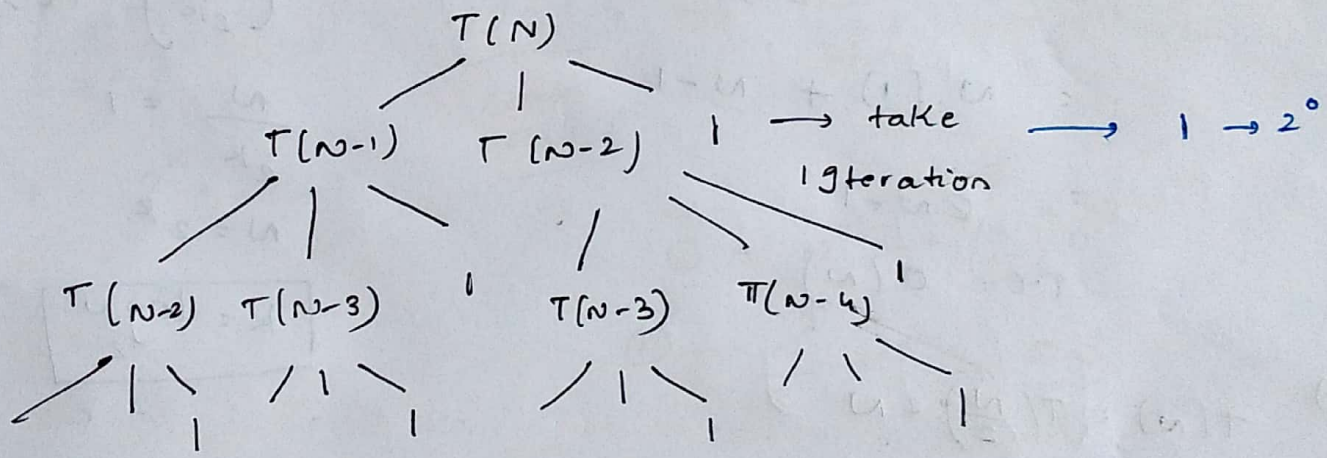
sub $(N-1)$ & $(N-2)$

$[T(N-2) + T(N-3) + 1] + [T(N-3) + T(N-4) + 1] + \cdots$

Increasing terms will be difficult

if we have more number of terms we Recurance Tree

Recurence TREE



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

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

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

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

$= 8T\left(\frac{N}{8}\right) + 7$ — ③

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

$= N(1) + N - 1$

$= 2N - 1$

T.C: $O(N)$

$T(1) = 1$

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

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

$N = 2^K$

$K = \log_2 N$

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

$= T(N/2) + N$

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

$= T(N/4) + \frac{3N}{2}$

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

$= T\left[N/8\right) + \frac{7N}{4}$

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

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

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

$= 2N - 2 + 1$

$= 2N - 1$

$= N$

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

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

$= T\left[\frac{n}{4}\right] + 2$

$= \left[T\left[\frac{n}{8}\right] + 1\right] + 2$

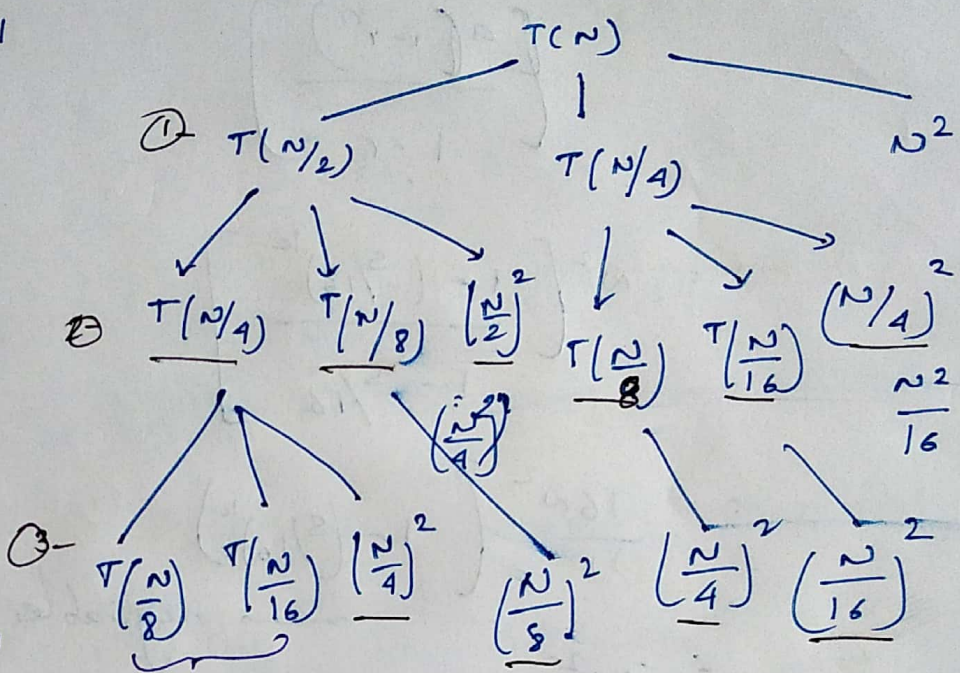
$= T\left[\frac{n}{8}\right] + 3$

$= T\left[\frac{n}{2^k}\right] + 2^{k-1} - 1$

$= 1 \cdot 2^k + 2^{-1}$

$= \log n$

5) $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + n^2$



① $n^2 - \left(\frac{5}{16}\right)^0 n^2$

② $\frac{n^2}{4} + \frac{n^2}{16} = \frac{5n^2}{16} = \left(\frac{5}{16}\right)^1 n^2$

③ $\frac{n^2}{16} + \frac{n^2}{64} + \frac{n^2}{16} + \frac{n^2}{256} = \frac{25n^2}{256}$

$\left(\frac{5}{16}\right)^2 n^2$

$\vdots$

$\left(\frac{5}{16}\right)^k n^2$

$= n^2\left[\left(\frac{5}{16}\right)^0 + \left(\frac{5}{16}\right)^1 + \cdots + \left(\frac{5}{16}\right)^k\right]$

g.p series

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

Sum of n terms in g.p

$$\left[\frac{a(1-r^n)}{1-r}\right]$$

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

$$= \frac{16n^2}{11}\left(1-(5/16)^k\right)$$

→ negligible

$$= n^2$$

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

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

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

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

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

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

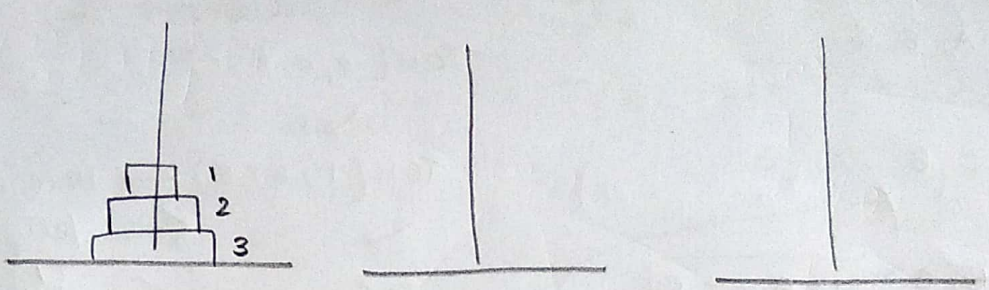
$$= n \cdot 1 + \log_2 n \cdot n$$

$$= n + n\log_2 n$$

$$= n\log_2 n$$

Towers of Hanoi

No of disks = 3



① : one disk at a time

② : Never place a bigger disk on smaller one

7 steps

1: A → C -①
2: A → B -②
1: C → B -③
3: A → C -④
1: B → A -⑤
2: B → C -⑥
1: A → C -⑦

```
Void TOH (int N, char src, char dest,
          char temp)                    → T(N)
{ //Source code
    if (N==0)                           → 1
        return.
    TOH( N-1, src, temp, dest);         → T(N-1)
    print( N: src → dest);              → 1
    TOH( N-1, temp, dest, src);         → T(N-1)
}
```

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

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

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

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

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

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

Base cond

$T(0) = 1$

$T(N-K) = T(0)$

$N-K=0$

$N=K$

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

$= 2^N \cdot O(2^N)$

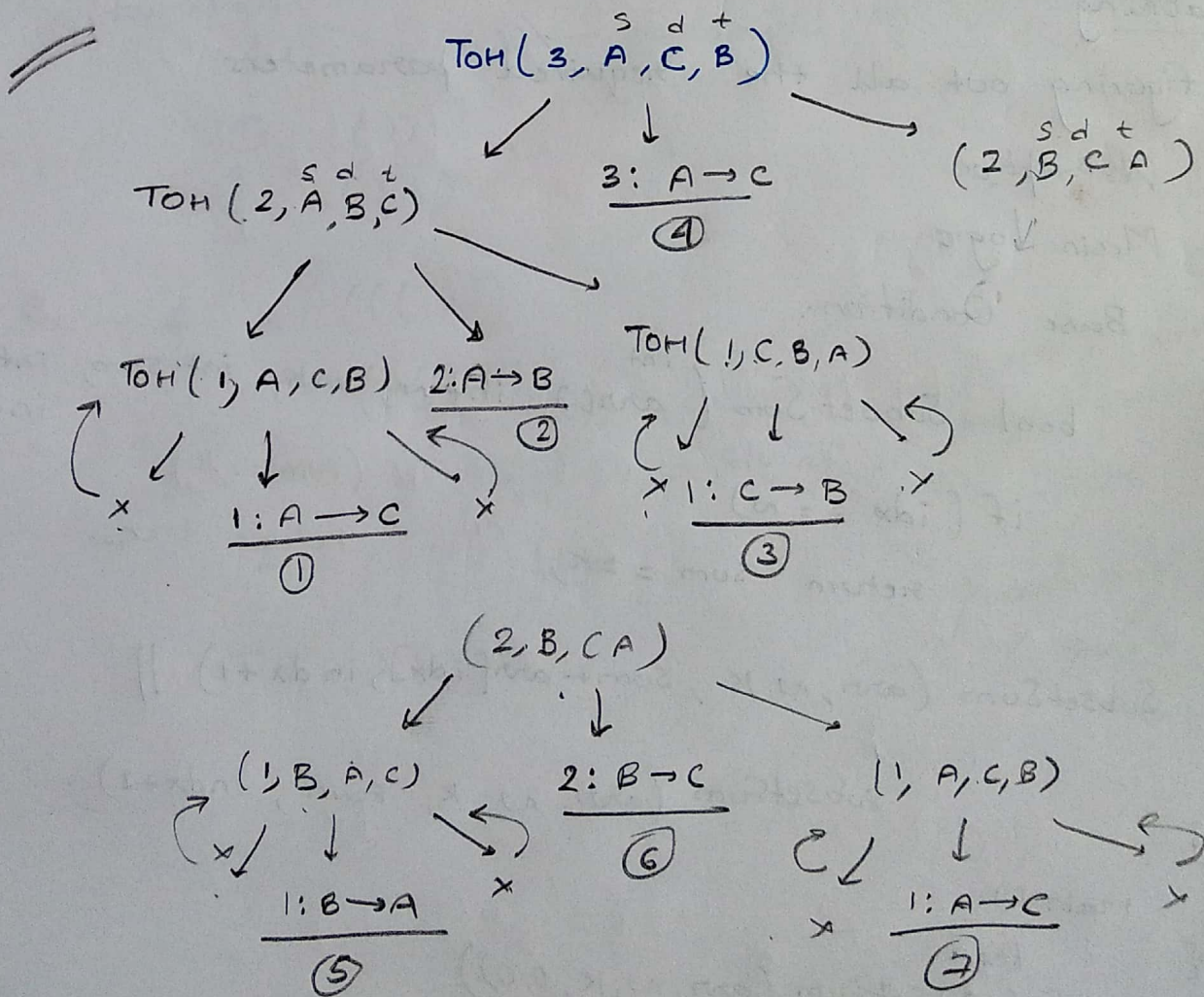
Time Complexity of Hanoi

Scanned by CamScanner

Towers of hanoi

A → source
B → temp
C → dest

```
Void TOH( int n, char src, char dest, char temp)
{
    if (n>0)
    {
        TOH( n-1, src, temp, dest);
        printf("n: src ---> dest);
        TOH( n-1, temp, dest, src);
    }
}
```



total function calls :- 15

a^N → Recursion

bit manipulation → $\log_2^N$

$a^N = a \cdot a^{N-1}$ $\quad T(N) = 1 + T(N-1)$

$O(N)$

not optimized

else

→ $a^N = a^{N/2} \cdot a^{N/2}$ → if (even) $a^{10} = a^5 \cdot a^5$

$a^N = a^{N/2} \cdot a^{N/2} \cdot a$ → if (odd) $a^{11} = a^5 \cdot a^5 \cdot a$

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

    return a * pow(a,N/2) * pow(a,N/2);
}
```

T.C :- $T(N) = 1 + 2T\left(\frac{N}{2}\right)$

$T(N) = N$

since, even though optimized we get t.c N dueto lot of function calls

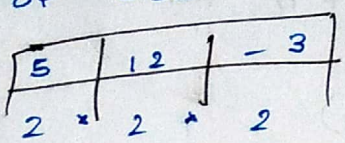
```
Pow ( int a, int N)
{  T(N) = if ( N == 0)
            return 1.
          if ( N % 2 == 0) ⟶ int x = pow (a, N/2)
          { return  x * x }

          return  x * x
```

t.c :- $T(N) = T\left(\frac{N}{2}\right) + 1$

$= \log_e N \longrightarrow$ reduced.

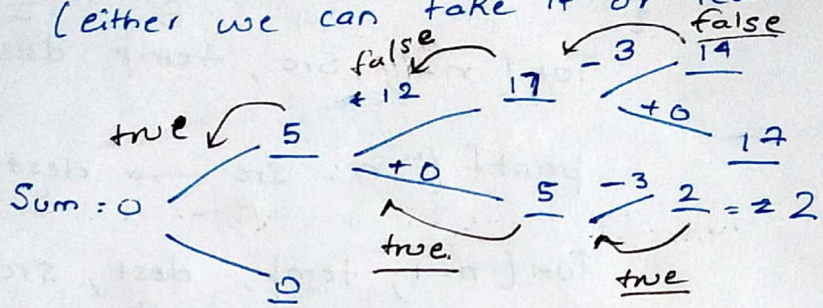
08/10/2023 रविवार Sunday

take a array of size N

arr_N : 

: $2^3 - 2^n$ possibilites

(either we can take it or leave it).



for an array of size N subsets 2^N

Backtracking

1. figuring out all the required parameters
2. Assumption
3. Main logic
4. Base Condition.

Code

```
bool SubsetSum ( int arr[], int N, int K, int sum, int idx)
{
    if ( idx == N)
        return sum == K;

    (or)  if (sum == K)
              return true.

return SubsetSum (arr, N, K , sum+arr[idx], idx+1) ||
       subsetSum (arr, N, K, sum, idx+1);
}

Main ()
{
    Read
    if ( SubsetSum (arr, N, K, 0, 0))
```

T.C = $T(N) = 2T(N-1) + 1 = 2^N$

* ⇒ ()()() → valid
()(()) → Invalid
))((→ Invalid

(())() → valid
())() → Invalid

Question

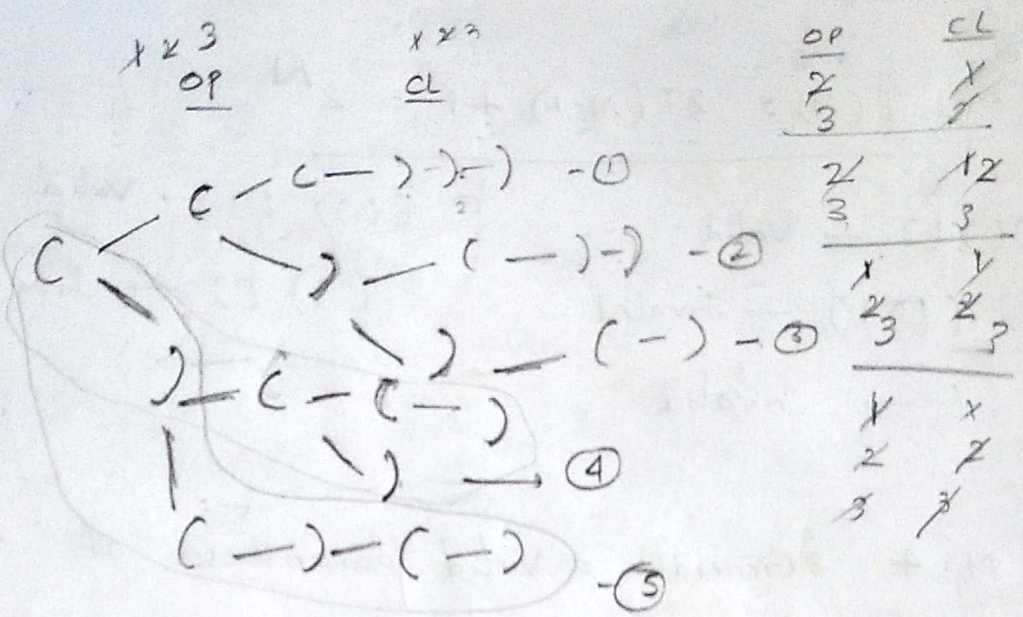
given N = 4 ⇒ Generate a valid parantheses of length N in lexiographical Order.

for N = 4 ⟨ ()() / (()) → this should be print 1st

① (())
② ()()

for N = 6 → ((()))
→ (()())
→ (())()
→ ()(())
→ ()()()

2 ways solving ⟨ generate every possibility → check → sort
generate only valid one




```
Void parantheses ( N, op, cl, char, ar[], indx)
{ // we will op until op < N/2
  // & closing will be placed until cl<op

    if (idx == N)
    { Print(arr);
      return; }

    if ( op < N/2 )
    {   arr[idx] = '(' ;
        Parantheses ( N, op+1, cl, arr, idx+1);
    }

    if( cl < op)
    {   arr[idx] = ')'
        parantheses (N, op, cl+1, arr[], idx+1);
    }
}
```

* Catalian Number :-

In the above code the t.c won't be 2^n coz it generates only valid cases i.e 5, to tell t.c we use C.

$N \rightarrow 2n \rightarrow 6$

$\frac{2n C_n}{n+1}$ ⇒ $2n = 6$, $\frac{6C_3}{4} = 5$ → i.e the no generate cases

$\frac{6!}{3!(6-3)!} = \frac{720}{6(6)}$

Question :- given a 3x3 Matrix with given elements Convert that into a Magic square with min cost.

Magic Square ⇒ row sum = col sum = diagonal sum

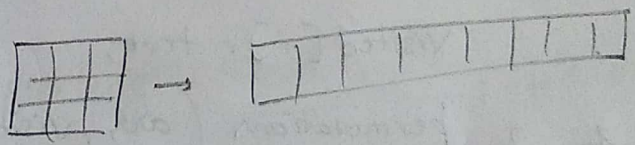
② use all 1 to 9 numbers

③ & should use only once.

① → generate a matrix → check if Magic Square or not

② → cost : $|6-8| + |3-4| + |7-9| = 5$

③ → answer

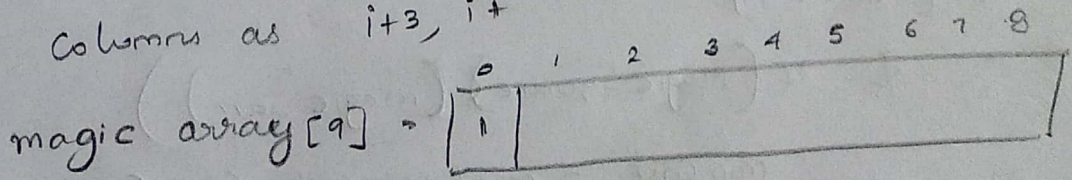


1D array

3x3 → arr[9]

rows

columns as i+3, i+



magic Array

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

→ [1,9] 1st permutation

Visited Array

0	1	2	3	4	5	6	7	8	9
T	T	T	T	T	T	T	T	T	T

2nd permutation

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

- 1st, again 9 will start.

1 2 3 4 5 6 7 9 8 — 2nd

1 2 3 4 5 6 8 7 9 -

9 8 7

at every index for loop will start every time

```
int minCost = INT_MAX;
Void permutations (arr, Visited, magic_arr, index)
{
    for (int i = 1; i <= 9; i++)
    {
        if (!visited[i])
        {
            magic[idx] = i;
            visited[i] = true;
            permutations (arr, visited, magic_arr, idx+1);
            Visited[i] = false
        }
    }
    if (idx == 9)
    {
        if (MagicSquare) (magic_arr))
            minCost = min(minCost, Cost(arr, magic_arr));
    }
    return;
```

```
Main()
{
    int arr[9];
    bool visited[10];
    int magic_arr[9];
    Permutations(arr, visited, magic_arr, 0);
    Print → min cost;
}
```

permutations → Inbuilt → next_permutation()

given a list of Words

L = { Smart, mart, s, inter, view, interviews, views }

Str: Smart interviews

1) True/False — Possible to partition str in such a way that every partition is present in the list.
2) # ways ??
3) Min Partitions = 1

```
bool Partitions( str, list, indx=0 )
{
    int n = str.length()  ⟶  if(idx==n) return true;
    for( int i = index, i<n; i++ )
    {
        if ( str[idx:i] ∈ L )   ← str.subStr(idx, i-idx+1) ∈ L
        { if(partition( str, list, i+1 ))
            { return true;
            }
    } return false;
```

→ ② Count total number of ways

```
int ways = 0
Void partition (str, list, idx, Count)
{
    int n = str.length();
    if (idx == n)
    { ways = ways + 1;
      return;
    }
  if (str[idx:i] ∈ L)
  {  { partition (str, list, i+1) }   count+1
  }
}
```