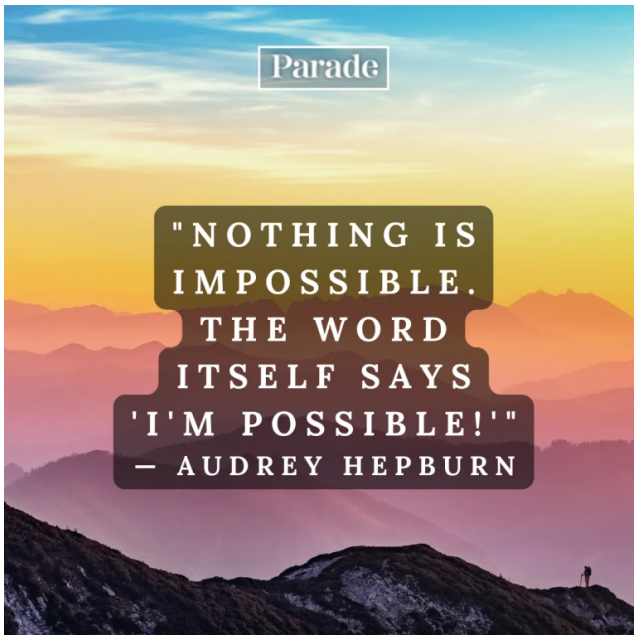


Good Evening

* Content →
01. Rat in a maze
02 Permutations - I & II
03. Subset

Recursion : Solving Problems using subproblem

Backtracking : An algorithmic technique by (Brute force) which we can try out all possibilities using Recursion.

✓	✓	✓
Box 1	Box 2	Box 3

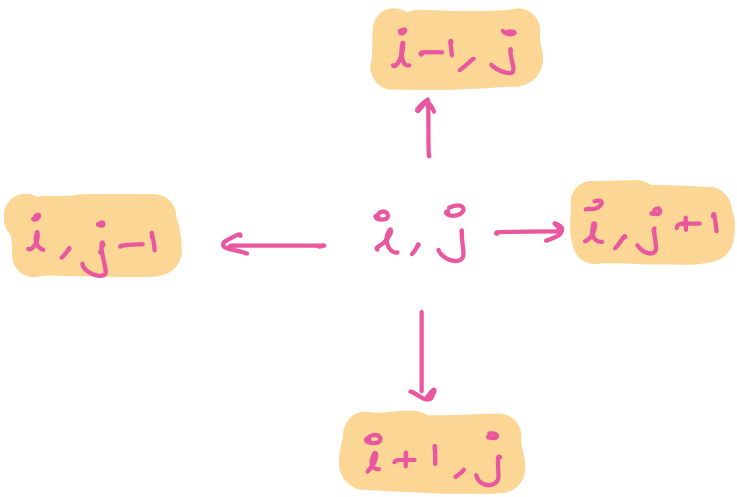
Rat in a Maze

Check if it is possible to go from top-left to bottom-right cell in a maze with blocked cell.

	0	1	2	3	4	5	6
0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0
2	0	1	0	0	1	0	0
3	0	0	1	0	1	0	1
4	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0

arr[i][j]=0 Empty

arr[i][j]=1 blocked



code

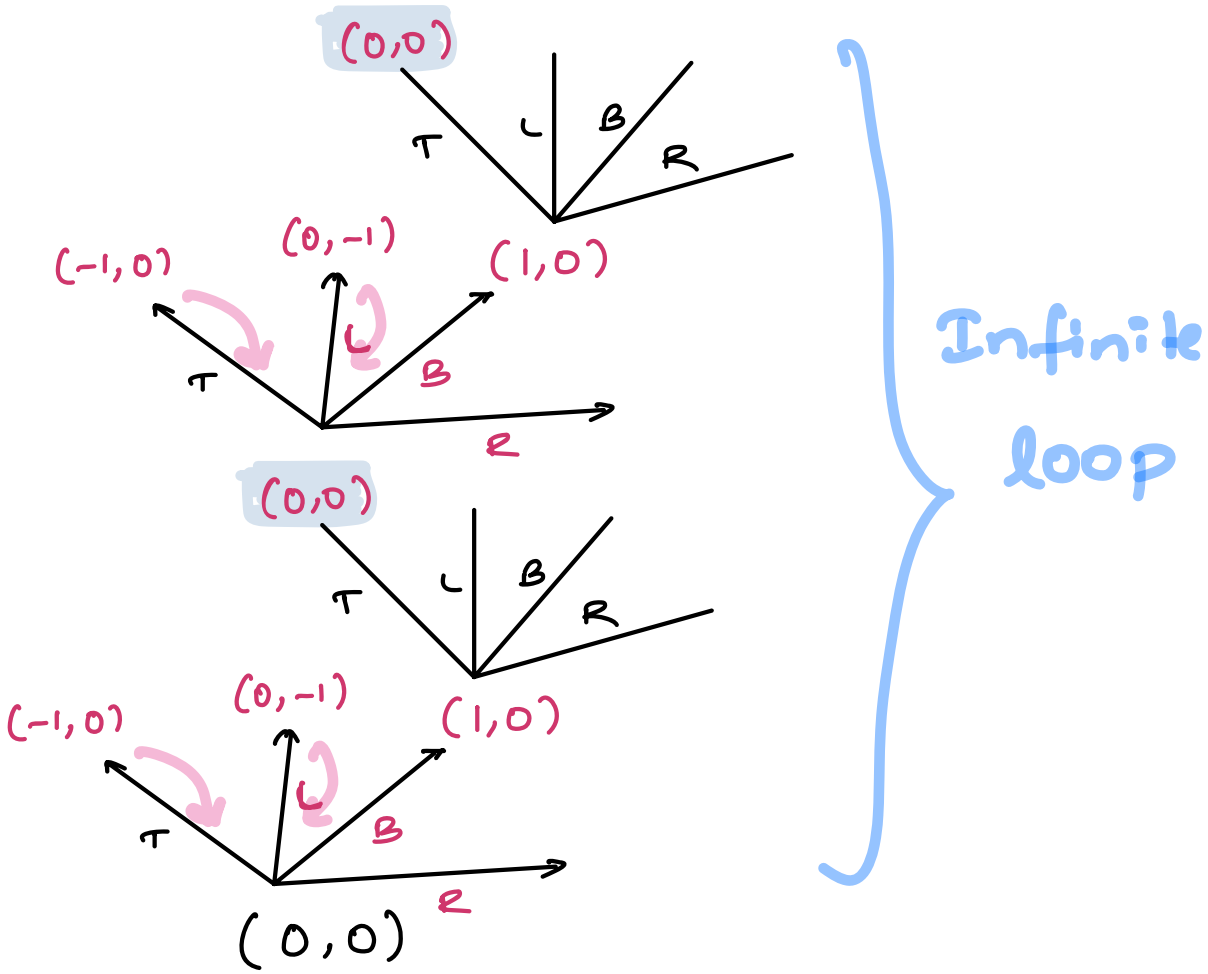
```
                                    0   0
boolean check ( arr[][], i, j )
{
    if ( i == n-1 && j == m-1 ) return true;

    if ( i<0 || j<0 || i≥N || j≥M ||
    arr[i][j] == 1 || arr[i][j] == 2 ) return false;

    arr[i][j] = 2 ;     // arr[i][j] is visited

return ( check ( ar, i-1, j ) || check(ar, i, j-1)

|| check(ar, i+1, j ) || check(ar, i, j+1) );
}
```

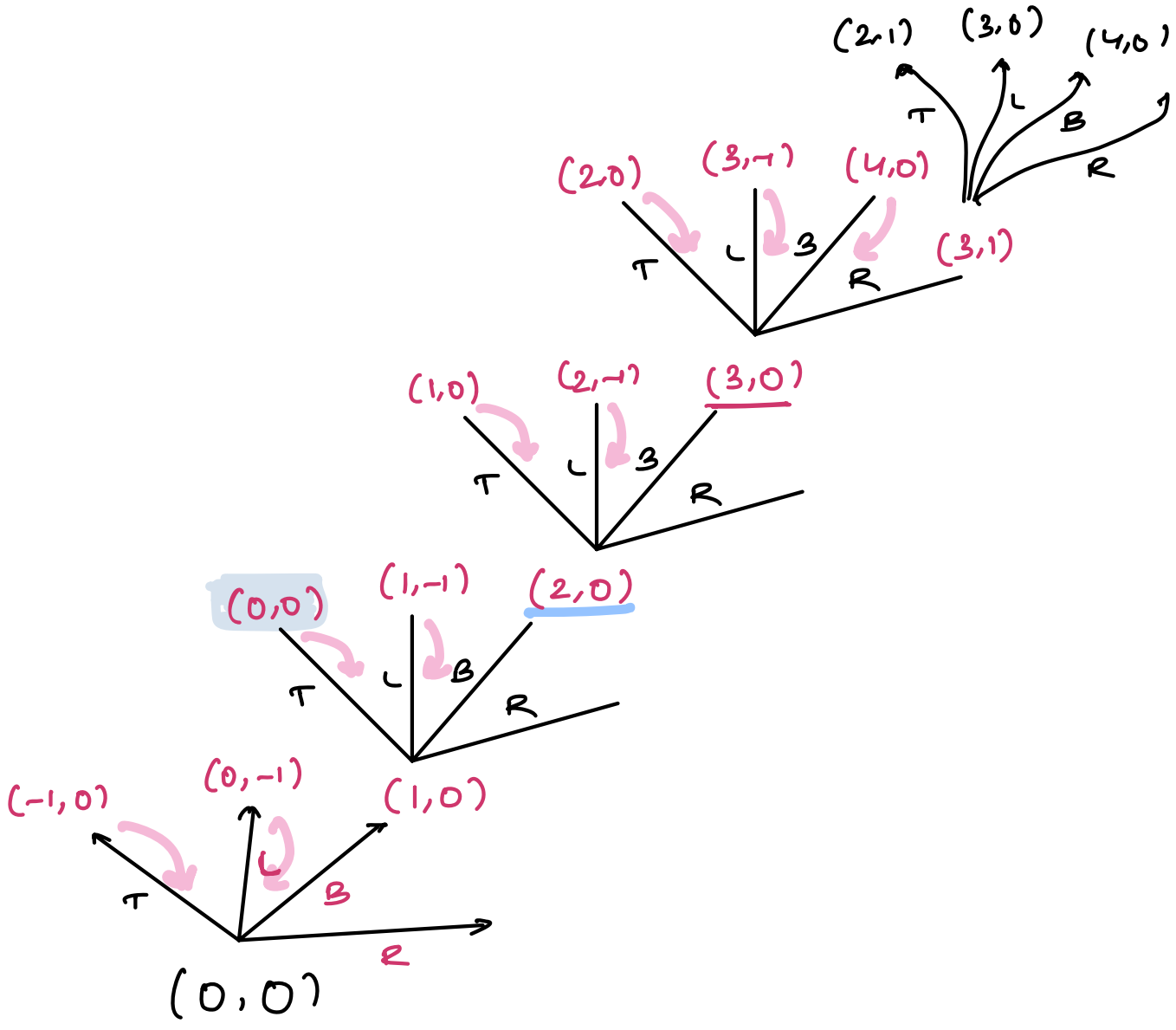


	0	1	2	3	4	5	6
0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0
2	0	1	0	0	1	0	0
3	0	0	1	0	1	0	1
4	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0

boolean vis[N][M]
- true (cell is visited)
- false (cell is unvisited)

arr[N][M]
- 0 empty cell
- 1 blocked cell
- 2 visited cell

	0	1	2	3	4	5	6
0	2	0	0	1	0	0	0
1	2	1	0	1	0	1	0
2	2	1	0	0	1	0	0
3	2	2	1	0	1	0	1
4	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0



* Pseudocode

```
boolean check ( int [][] ar, int i, int j )
{
    if ( i == n-1 && j == m-1 ) return true;

    ar[i][j] = 2;

    dx = [-1, 0, 1, 0]
    dy = [0, -1, 0, 1]

    for ( k=0; k<4; k++ )
    {
        ni = i + dx[k]
        nj = j + dy[k]

        if ( ni >= 0 && nj >= 0 && ni < n && nj < m
             && ar[ni][nj] == 0 )
        {
            boolean ans = check ( ar, ni, nj )

            if ( ans == true ) return true;
        }
    }
    return false;
}
```

TC : O(n*m)

SC : O(n*m)

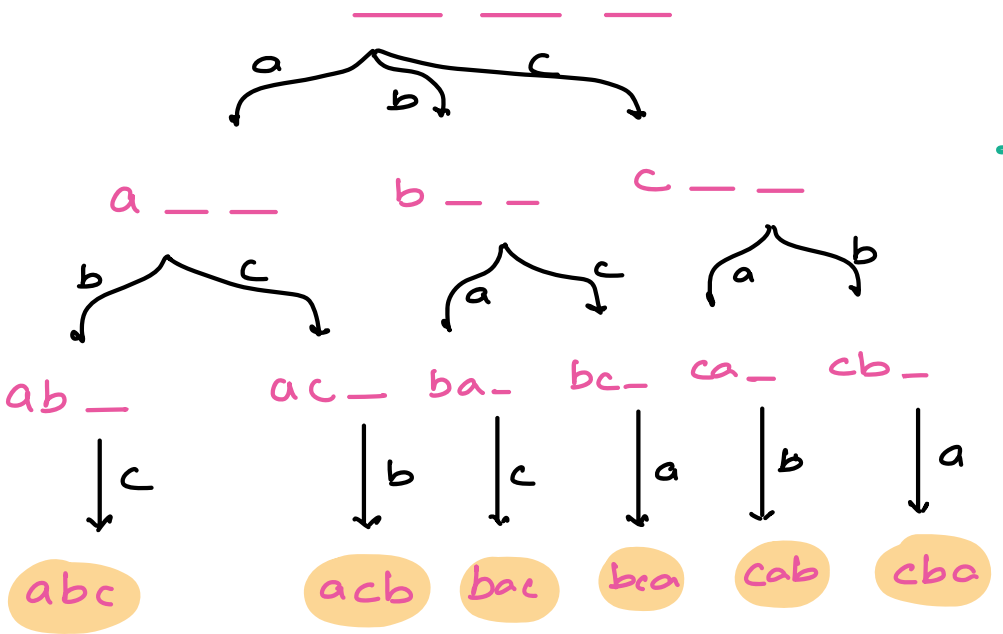
* Permutations — 1

Given a character array with distinct elements. Print all permutations of it without modifying it.

A = { a, b, c } O/P → { abc bac cab acb bca cba }

$3 * 2 * 1 = 3! = 6$

$N * (N-1) * (N-2) \ldots 1 = N!$



{ keep a track of visited / used char }

```
void permutations(ar, idx, ans[N], visited[N])
    if (idx == ar.length) { print ans[], return; }

    for (i=0; i<n; i++)
        if (vis[i] == false){
            vis[i] = true;
            ans[idx] = ar[i] ✓
            permutation(ar, idx+1, ans, vis);
            vis[i] = false;
        }
```

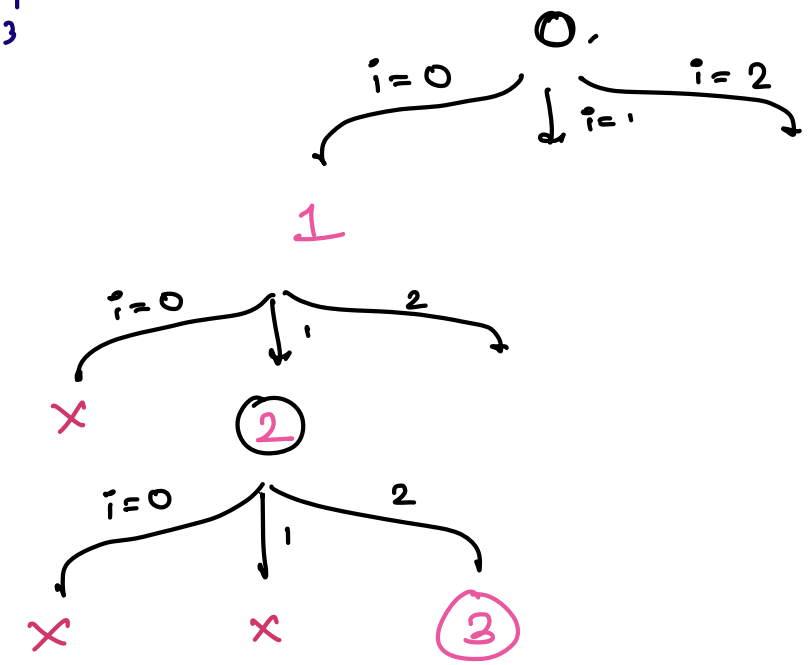
TC = O(n! * n)

SC = O(n)

arr = {a, b, c}

ans = {a, b c}

vis = {T T T}



abc

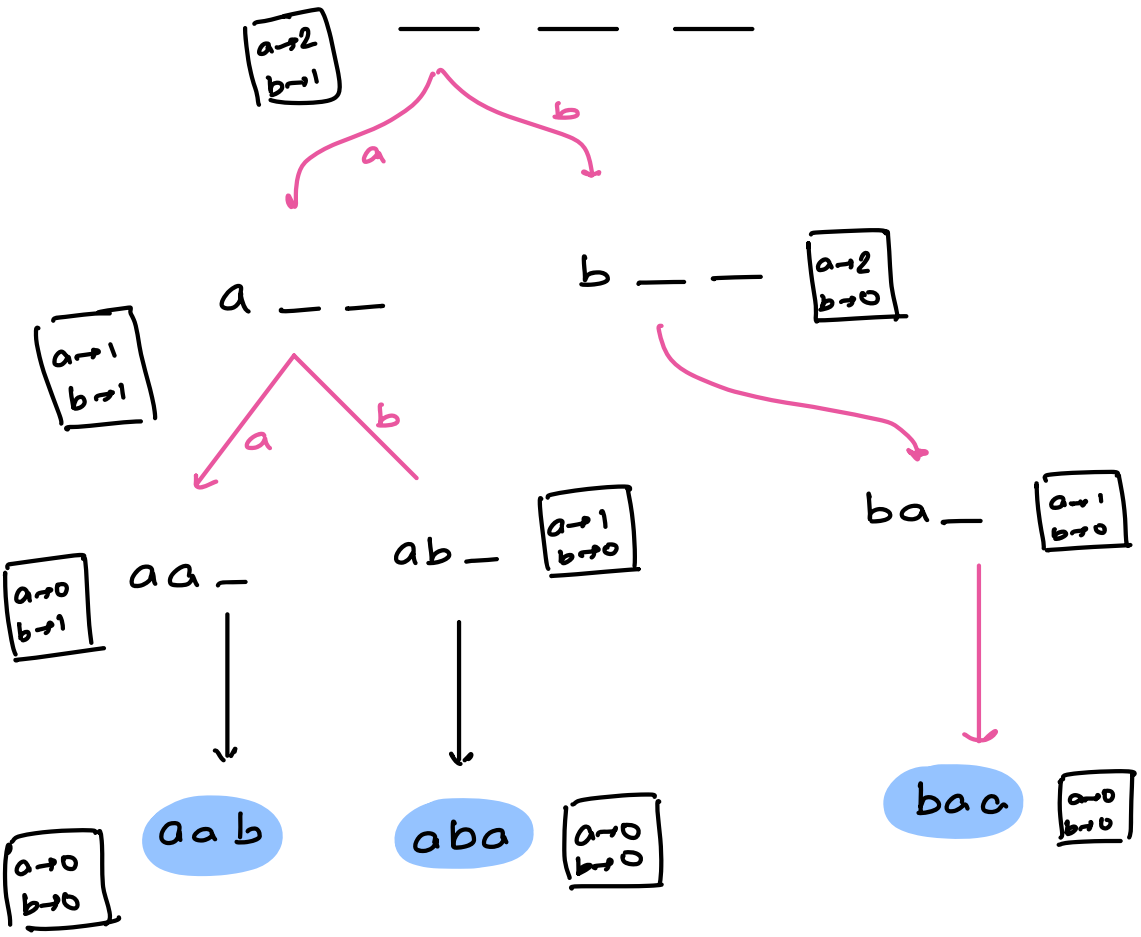
10:16 PM → 10:26 pm

Permutations 2

Print all permutations of given char array

str: [a, b, a] ⟶ O/p { aba, aab, baa }

MISSISSIPI ⇒ $\frac{N!}{f(a)!\ f(b)!\ f(c)!\ \cdots\ f(z)!}$



```
void permute ( far[26], N, ans[N], idx )
    if (idx == N) { print(ans), return; }
    for (i=0; i<26; i++)
        if (freq[i] > 0)
            freq[i] -= 1;
            ans[idx] = (char)(i+'a');
            permute (farr, N, ans, idx+1);
            freq[i] += 1;
        }
    }
}
```

TC = O(N!)

SC = O(N)

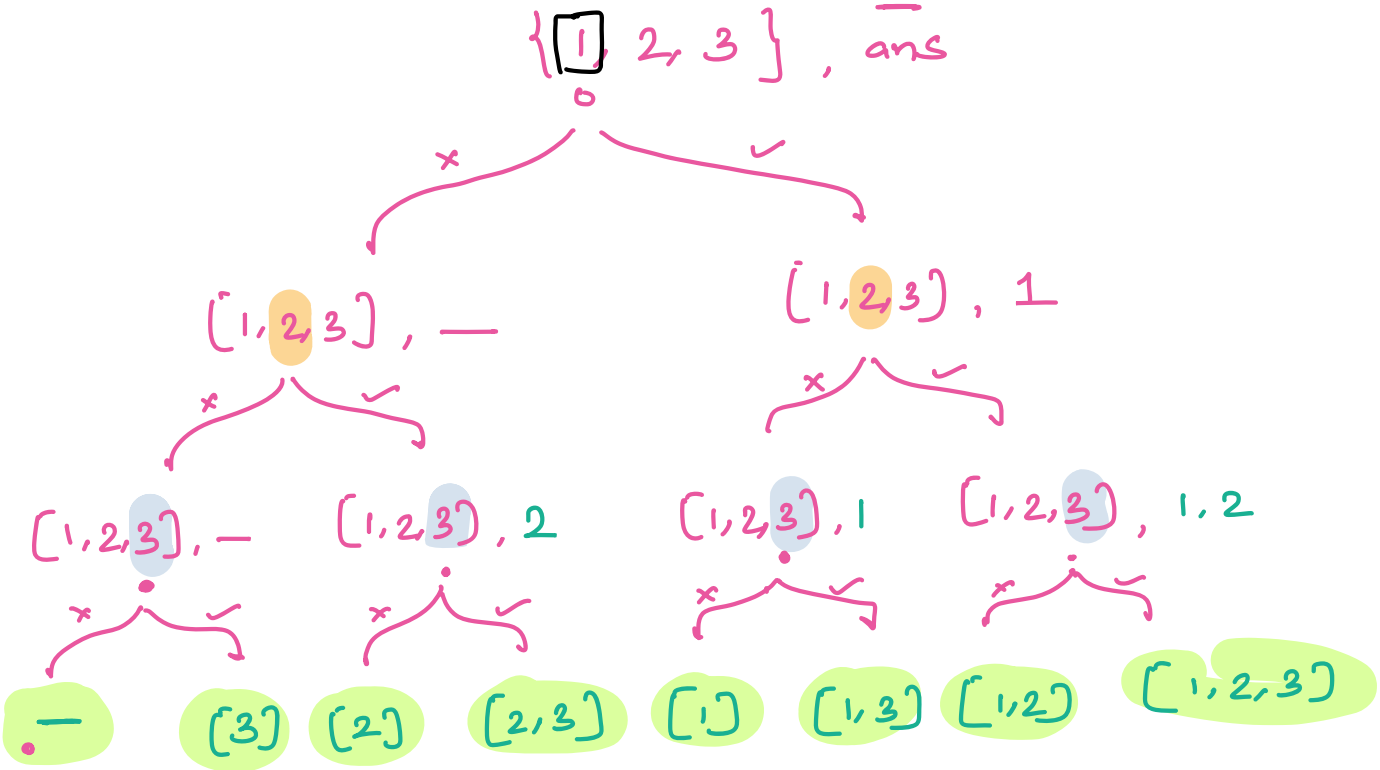
* Subsets

Given an array with distinct integers. Print all subsets using recursion

arr[] = {1, 2, 3}

{ [], [1], [2], [3], [1,2] [2,3] [1,2,3] [1,3] }

subsets = 2^n



```
subsets ( arr [] , idx , AL <I> ans )
    if ( idx == arr.length ) { print (ans) return; }

    subsets (arr, idx+1 , ans);   // Not Include

    ans.insert (arr[idx]);
    subsets (arr, idx+1 , ans);   // Include
    ans.remove (ans.size()-1);
}
```

TC = $O(2^n * n)$

SC = $O(n)$