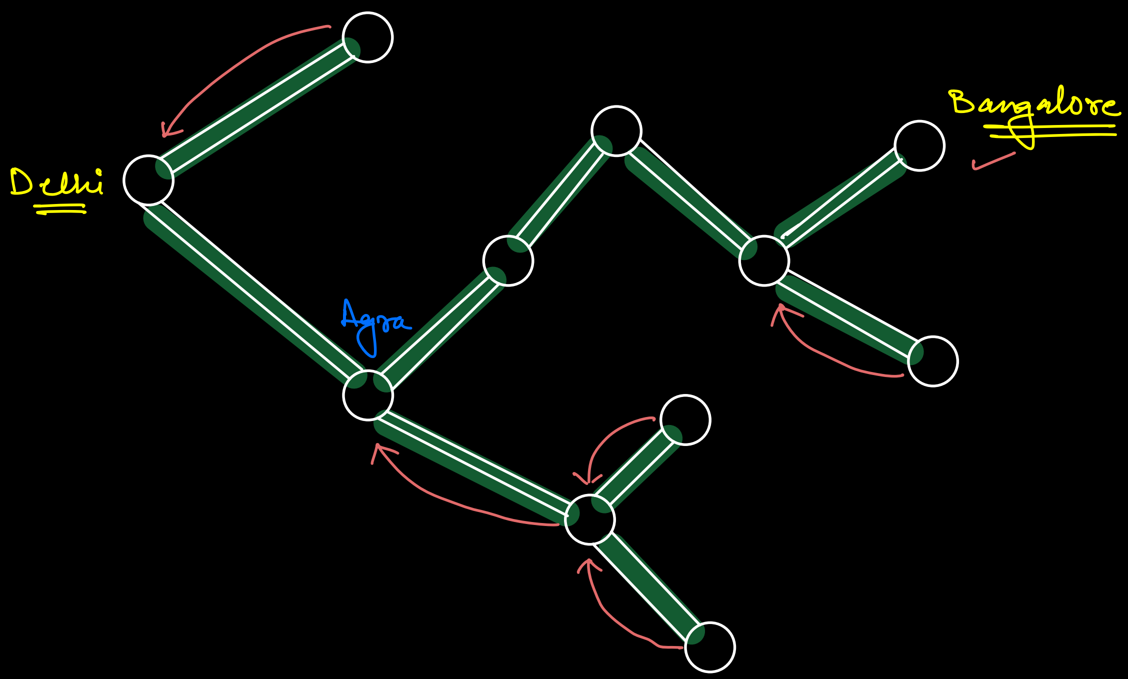


Backtracking ⇒ Try out all possibilities
(Brute force)

↓

Recursion + Backtrace

Q.1 Print all N digit numbers using {1 & 2}

N=1 ⇒ 1

2

N=2 ⇒ 1 1

1 2

2 1

2 2

N=3 (_ _ _)
0 1 2

N=3 [1 1 1]

[1 1 2]

1 2 1

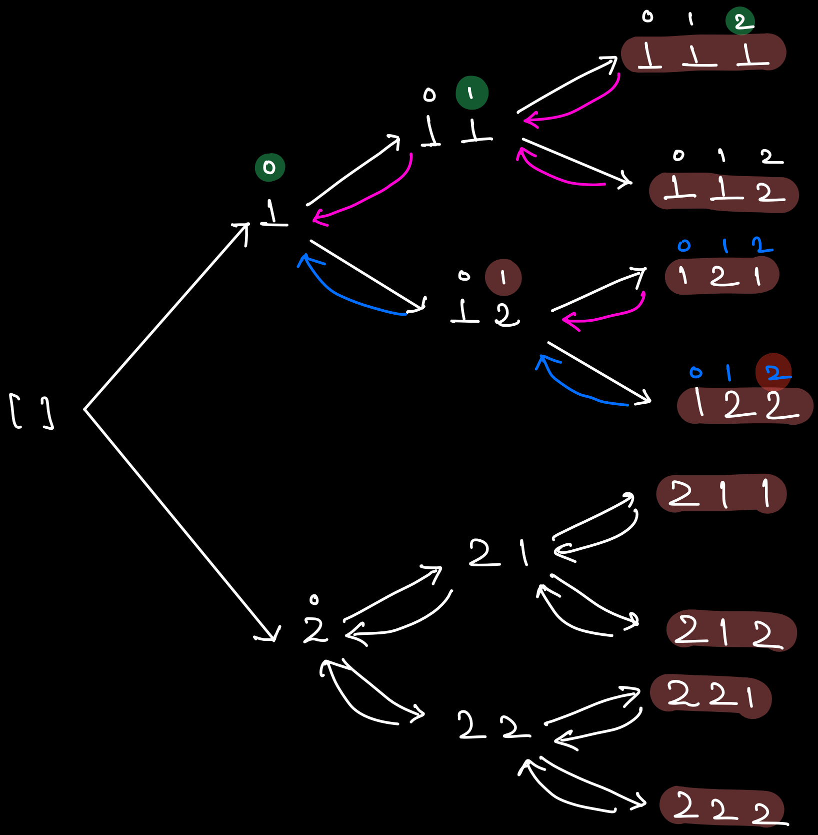
1 2 2

2 1 1

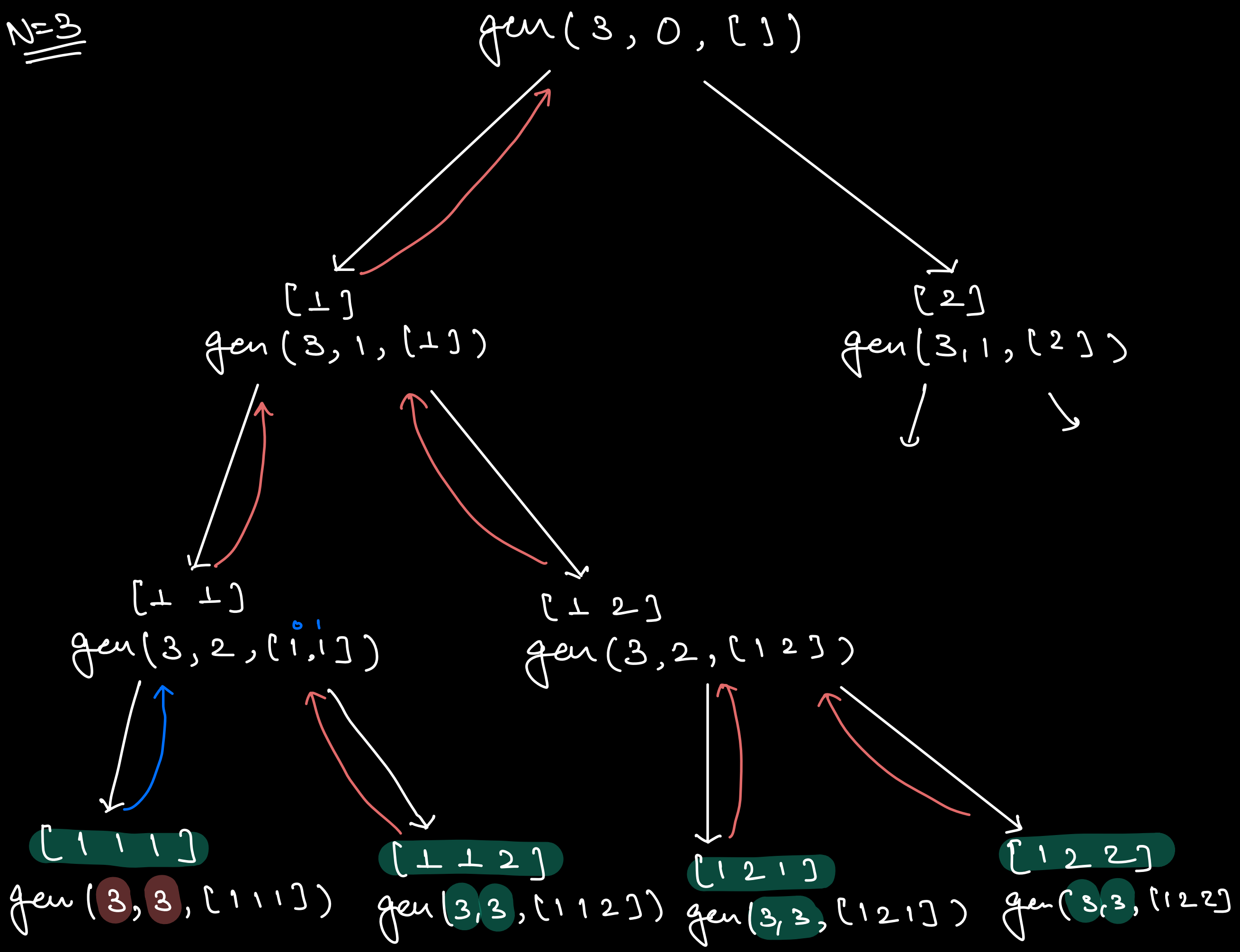
2 1 2

2 2 1

2 2 2



```
void   generateNumbers ( N, idx , currList ) {
        if ( idx == N ) {
            print (currList) ; ⇒ O(N)
            return ;
        }
        currList [idx] = 1
        generateNumbers (N, idx+1 , currList);
Backtrack [currList [idx] = 2
        generateNumbers (N, idx+1 , currList);
}
```



Quiz **TC** :

TC of a recursive fun^ :

(# of recursive fun^ calls) * TC of each fun^ call

$2^N * N$

* $\boxed{TC: O(N \cdot 2^N)}$

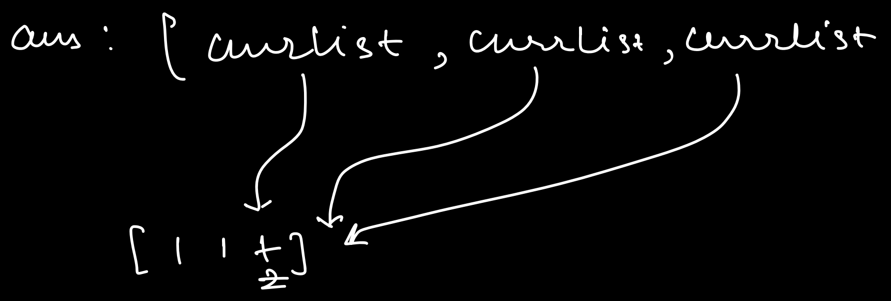
SC: $O(N)$

* Return a list of list

[[1 1 1], [1 1 2], [1 2 1], [1 2 2], [2 1 1],
[2 1 2], [2 2 1], [2 2 2]]

```
list<list<int>> ans;
```

```
void    generateNumbers ( N, idx , currList ) {
        if (idx == N) {
              ans.add (currList); ⇒ O(N)
              return;         reference.
        }
        currList [idx] = 1
        generateNumbers(N, idx+1, currList);
backtrack [currList [idx] = 2
         generateNumbers(N, idx+1, currList);
}
```



ans : [[2 2 2] [2 2 2] . . . .]

☆ HW

* Hard Copy vs Soft Copy. ⇒ Java

* Deep copy vs Shallow Copy

Currlist : [1 1 1]

temp : [1 1 1]

ans :

$$TC : O(N \cdot 2^N)$$

Q. Print all N digit numbers using {1, 2, 3, 4 4 5}.

```
void    generateNumbers ( N, idx , currList ) {
        if ( idx == N ) {
              print(currList); ⇒ O(N)
              return;
        }
        currList[idx] = 1
        generateNumbers(N, idx+1, currList);
        currList[idx] = 2
        generateNumbers(N, idx+1, currList);
        currList[idx] = 3
        generateNumbers(N, idx+1, currList);
        currList[idx] = 4
        generateNumbers(N, idx+1, currList);
        currList[idx] = 5
        generateNumbers(N, idx+1, currList);
}
   for(i=1; i<=5; i++) {
        currList[idx] = i;
        generateNumbers(N, idx+1, currList);
   }
```

TC : $O(N \cdot 5^N)$

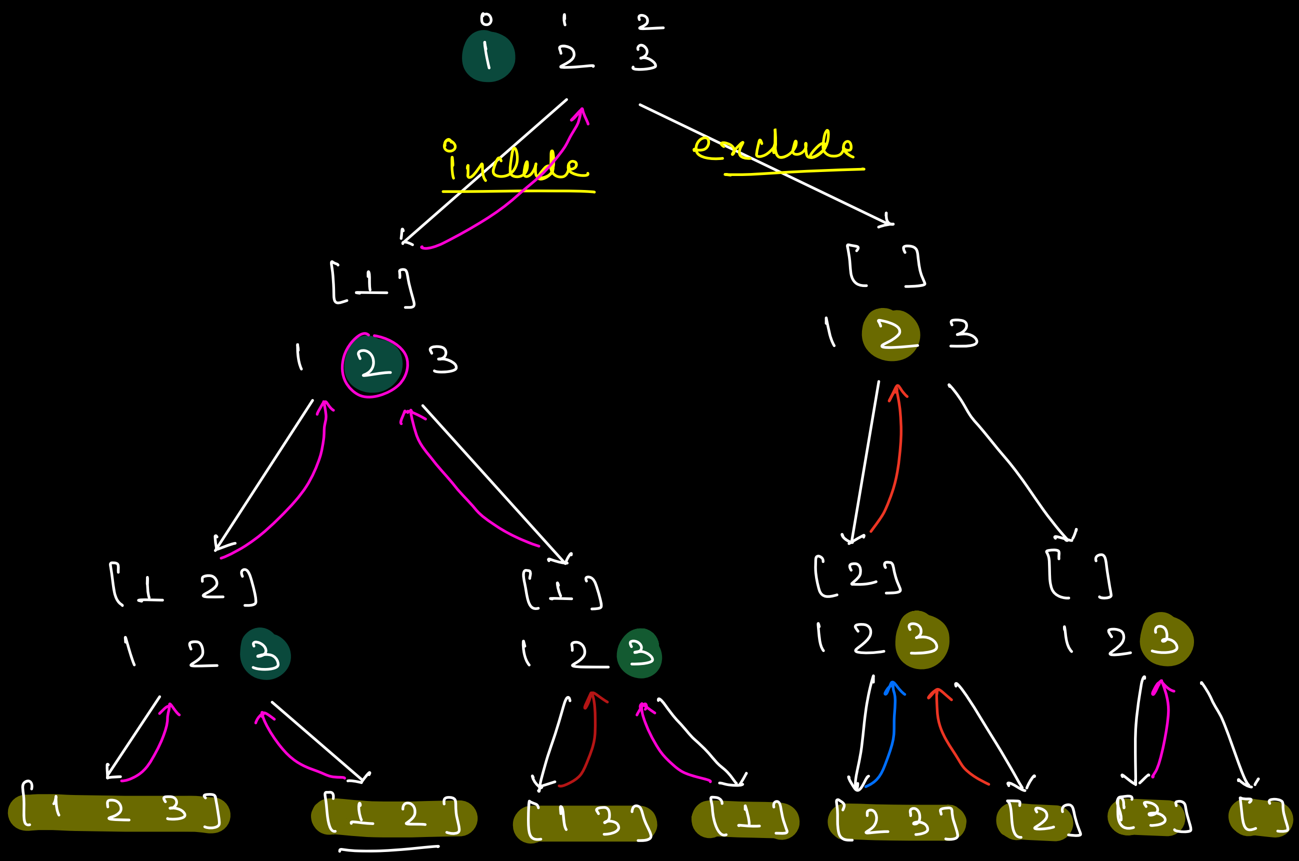
Q. Given an Array, generate all subsets of the Array.

Amazon
MS.

A : [1 2 3]

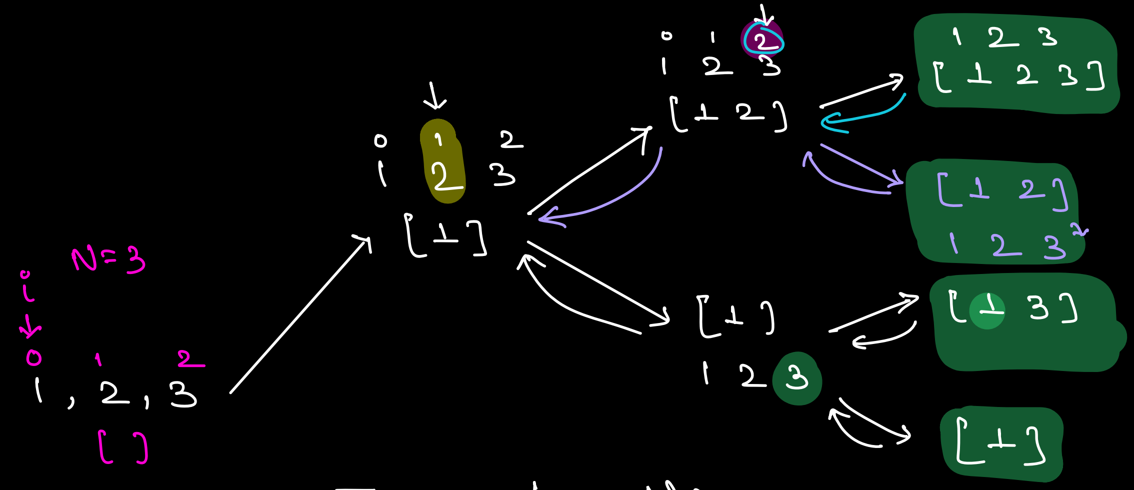
[[ ]
[1]
[2]
[3]
[1 2]
[1 3]
[2 3]
[1 2 3]]

currlist : [ ]



No. of subsets ⇒ $\underline{\underline{2^N}}$

```
                                                       [ ]
void  generateSubsets( A[] , N , idx , currList){
        if(idx == N){
            print(currList);
            return;
        }
        // Include A[idx] in the Subset
        currList.add(A[idx]);
        generateSubsets(A, N, idx+1, currList);
        // Exclude A[idx] in the Subset
        currList.pop(A[idx]);
        generateSubsets(A, N, idx+1, currList);
}
```



TC: $\underline{\underline{O(N \cdot 2^N)}}$

$2^{10} = 1024 \approx 10^3$

$2^{20} \approx 10^6$

$N \cdot 2^N \Rightarrow \underline{\underline{N=30}}$

↓

$\underline{\underline{\sim 10^9}}$ ×

$N=10^6 \Rightarrow O(N)$

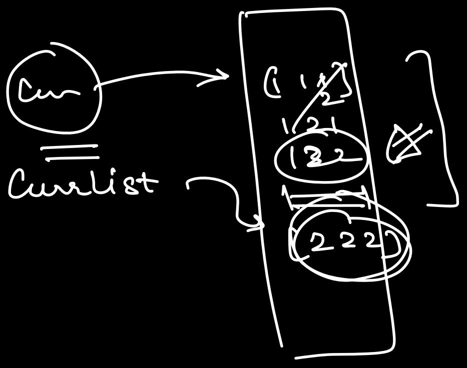
$N=10^3 \Rightarrow N^2$

$N=20 \Rightarrow$ Recursive

or

Backtracking

HW



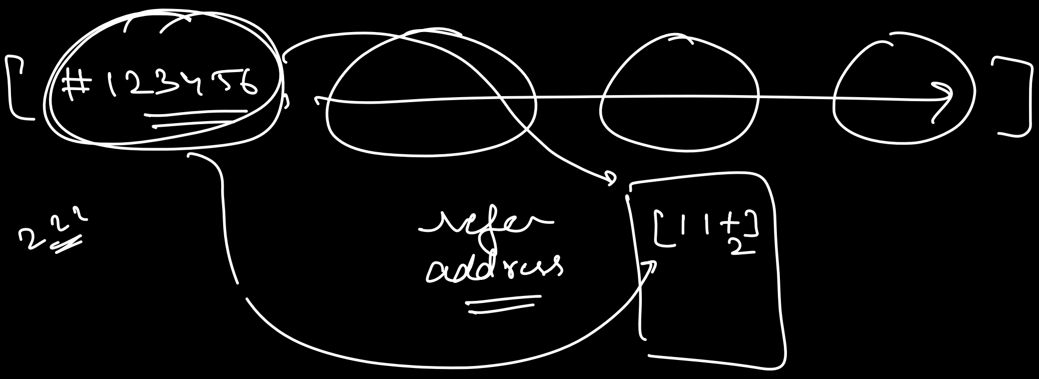
[Currlist

~~111~~

121

[[2222] [2222]

[[111] [121] [122] ...


[#123456 ,]
refer address
[1 1 ~~1~~ 2]
2^{2^2}