Backtracking : Try all possibilities

↓

Brute force

Q1 Print all N digit numbers using {1, 2}.

↓

only these digits can be used.

Ex1 N = 1

↳ [1, 2]

Ex2 N = 2

↳ [11, 21, 12, 22]

Ex3 N = 3

↳ [111, 211, 121, 221
112, 212, 122, 222]

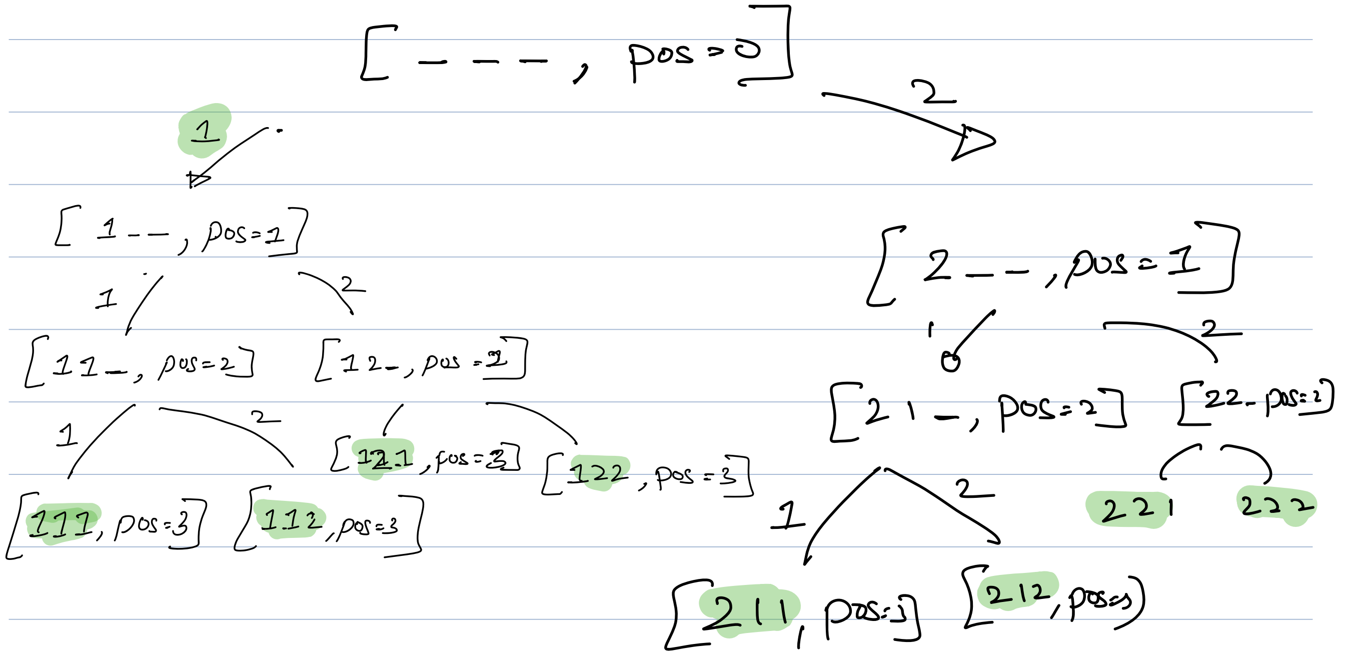
N=3

_ _ _

↓ ↓ ↓

2 × 2 × 2 ⇒ 8 possibilities

{1, 2}


[_ _ _, pos=0]
1
2
[1 _ _, pos=1]
[2 _ _, pos=1]
1
2
1
2
[11_, pos=2]
[12_, pos=2]
[21_, pos=2]
[22_ pos=2]
1
2
1
2
[111, pos=3]
[112, pos=3]
[121, pos=3]
[122, pos=3]
[211, pos=3]
[212, pos=3)
221
222

Pseudo Code !

● passed by reference
↑

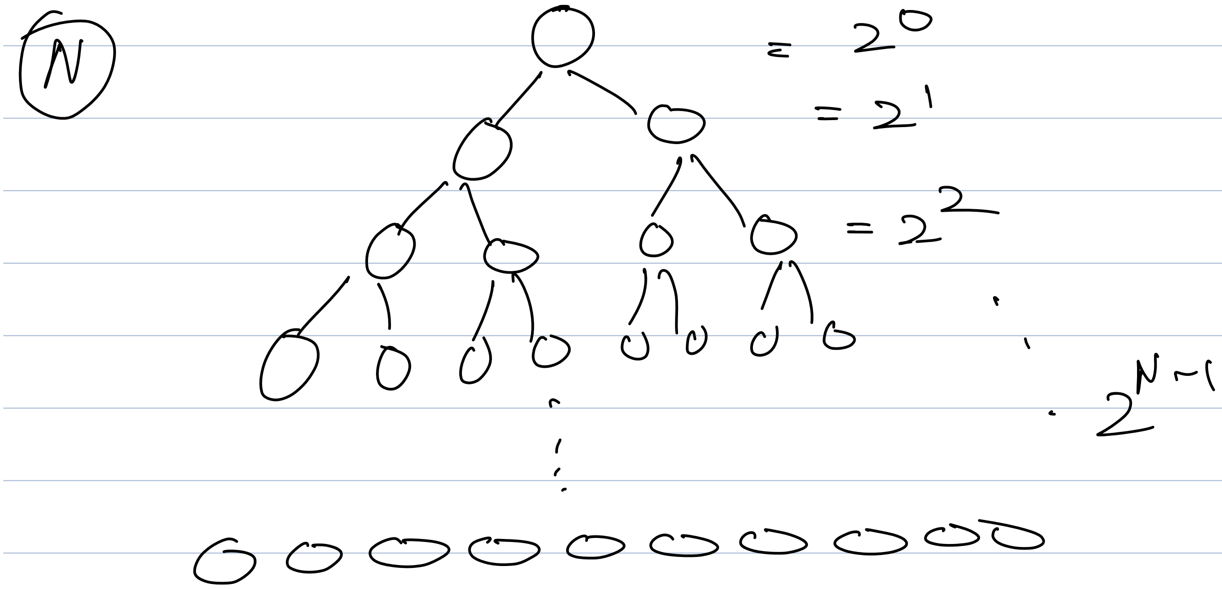
```
void generate ( int pos, int N, arr[] ) {
    if (pos == N) {                    // Base case.
        print arr;
        return;
    }
    arr[pos] = 1                       // Set
    generate ( pos+1, N, arr);

                                       // Unset

    arr[pos] = 2                       // Set
    generate (pos+1, N, arr);
}
```

# TC for a Base : O(N)

# TC for a non base case ⇒ Recursive
+
O(1)



Non base calls ⇒ $2^0 + 2^1 + 2^2 \dots 2^{n-1}$

⇒ $2^n - 1$

Time taken ⇒ 2^n.

Total Base Calls ⇒ 2^n

Time taken ⇒ $n \times 2^n$

TC : $(n \times 2^n)$

SC : $O(n) + O(n) \Rightarrow O(n)$

Stack space array.

Q1 Print all N digit numbers using {1, 2, 3, 4, 5}

↓

only these digits can be used.

```
for (int i = 1 ; i ≤ 5 ; i++) {
    arr [pos] = i;
    generate (                    );
}
```

Q3 Given N array elements. Count no of subsets which have sum = K.

Ex1 arr : [10, 2, 7, 6, 1, 5]

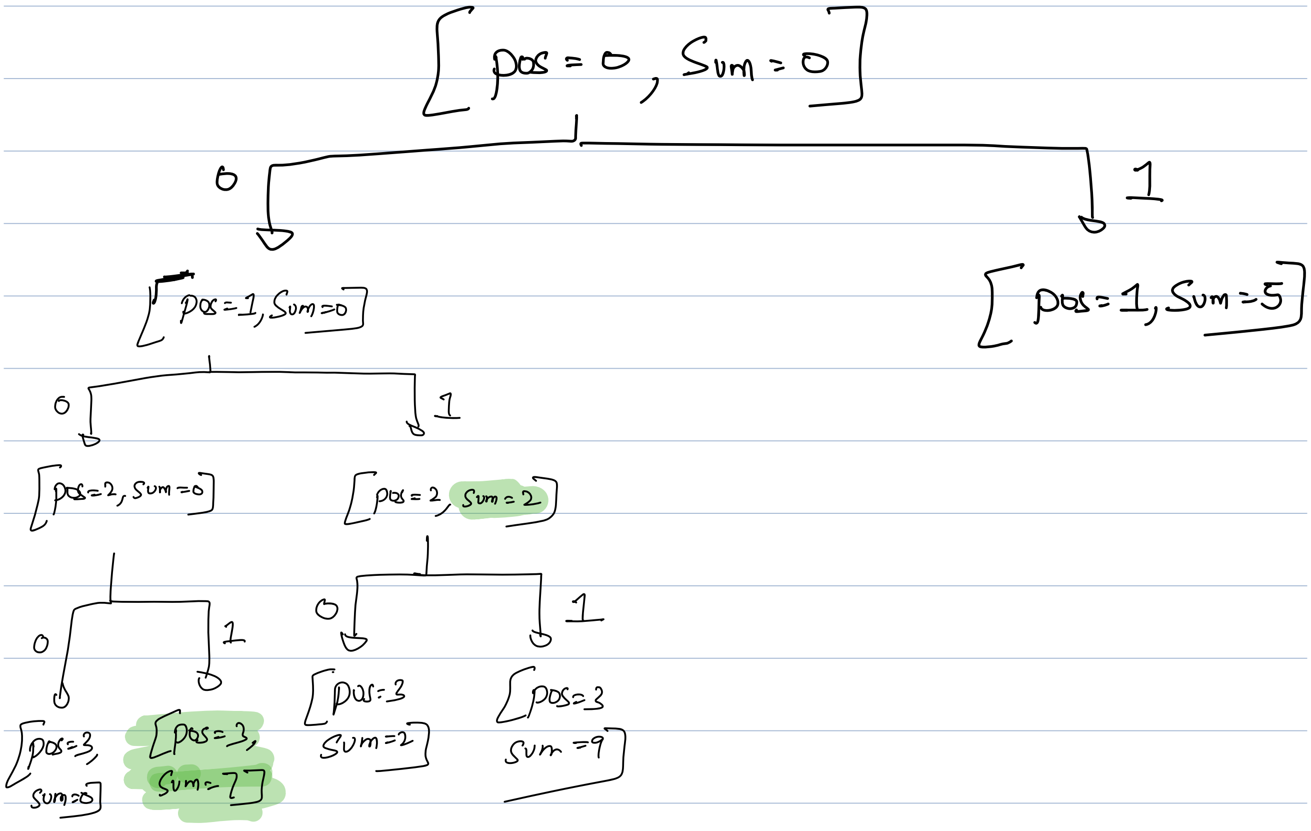
K ⇒ 8 ⇒ [2, 6]

[7, 1]

[2, 1, 5]

ans = 3

arr = [5, 2, 7]   K = 7   ⇒ [5 2 7]   ans=2



TC : 2^N

SC : O(N)

Q3 Print all subsets !

↓

Ordering is not important

Ex: arr : [2,3] = [ ]

[2]

[3]

[2,3] = [3,2]

```
void generate (int arr[], int N, int pos,
                                 list<int> l)
     if (N == pos) {
           print l;
          return;
     }

     l.push_back (arr[pos]);
     generate (arr, N, pos+1, l);
    l.pop_back()
  generate (arr, N, pos+1, l);
```

}

TC : $O(n \times 2^n)$

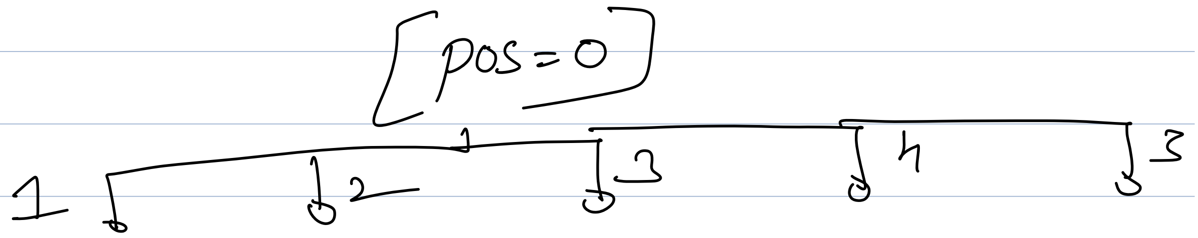
SC : $O(n)$

Qn Print all permutations ⇒ Distinct Digits.

arr[] = [1, 2, 3, 4, 5] = 5!

_ _ _ _ _

$5 \times 4 \times 3 \times 2 \times 1 = 5!$



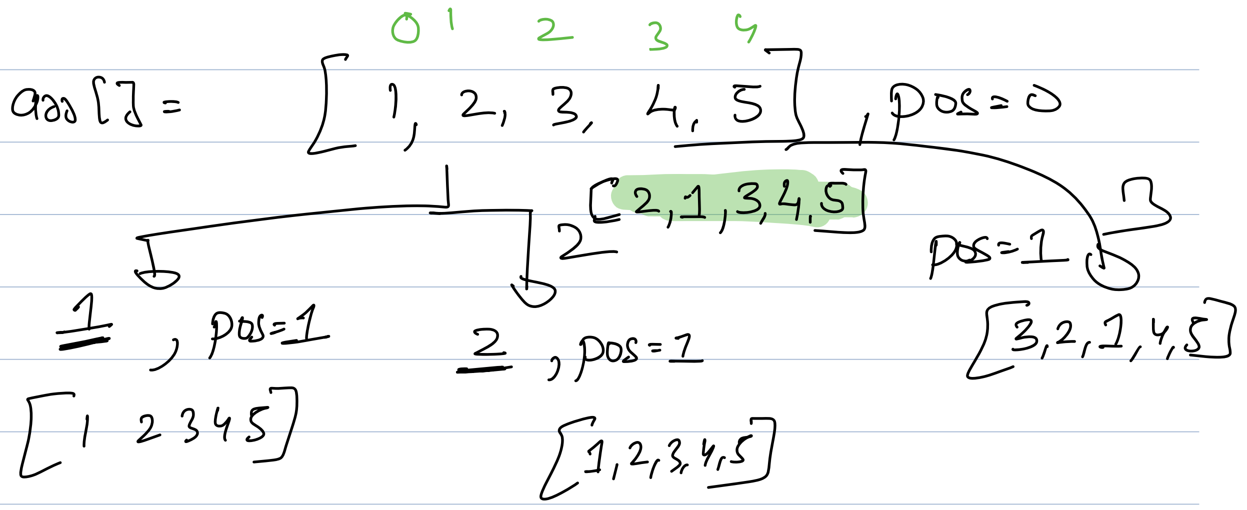
pos=1

2345

⇒ Boolean array.

⇒ Set

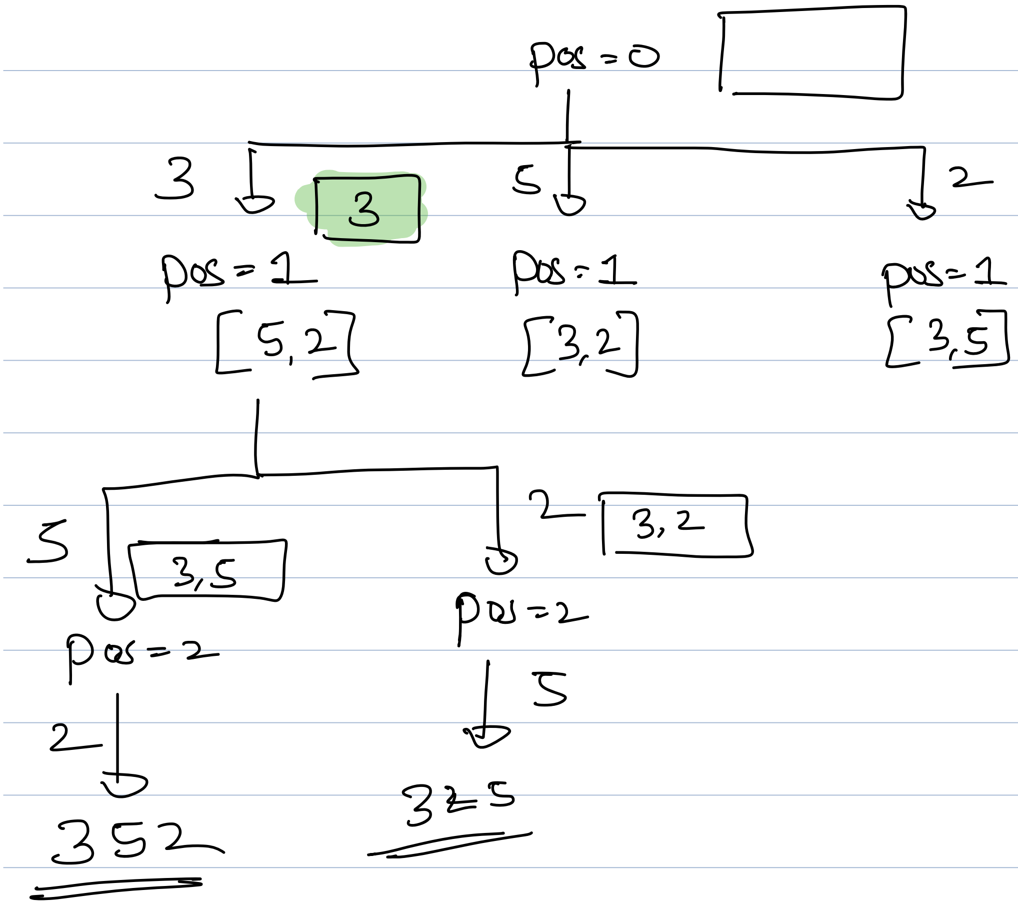
⇒ Bit masking



Pseudo Code

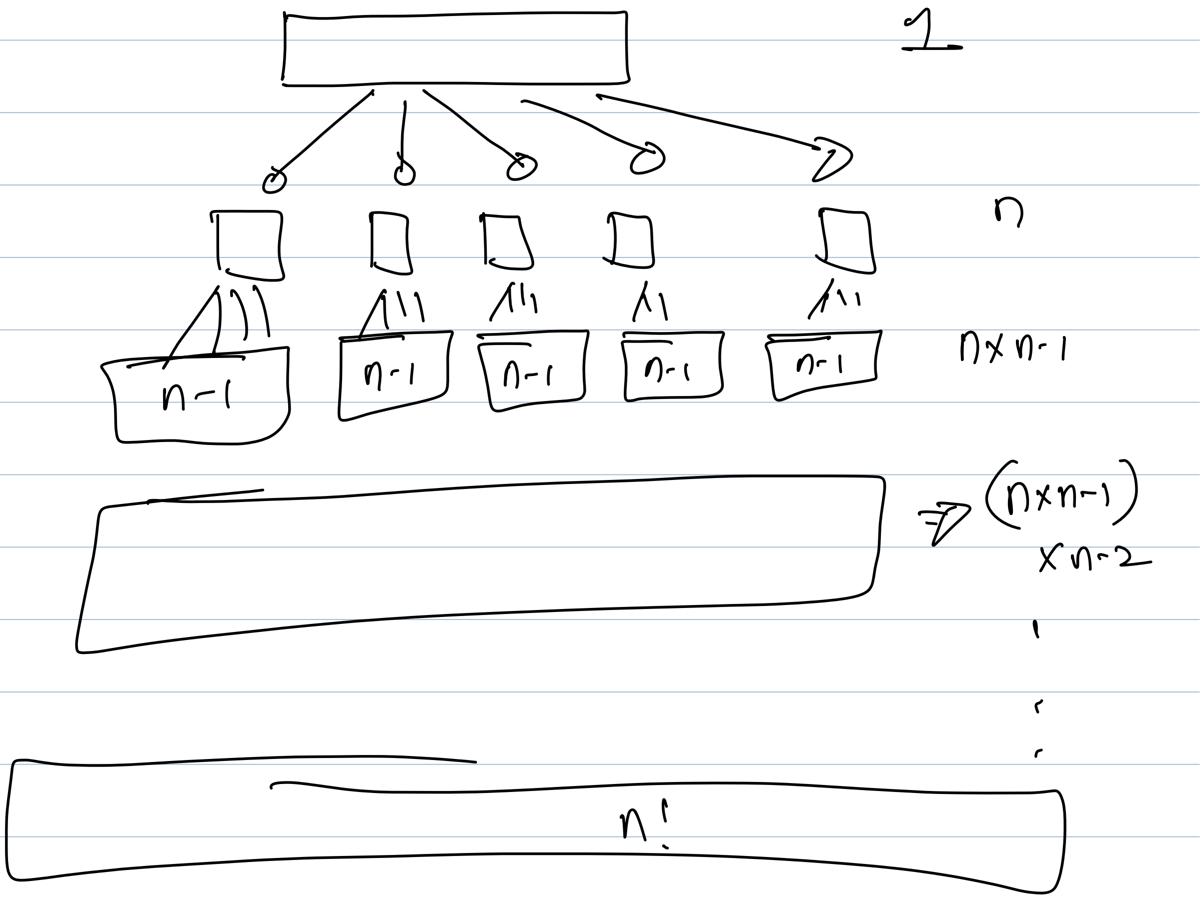
TC

arr = [3, 5, 2]



Pseudo Code !

```
void gen (int ans[], int n, int arr[], HS) {
    int pos = HS.size();
    if (pos == n) {
        print ans;
        return;
    }
    for (int i=0 ; i<n ; i++) {
        if (HS.contains (arr[i]) == False) {
            ans[pos] = arr[i];        DO
            HS.insert (arr[i]);       DO
            gen (ans, n, arr, HS);
            HS.remove (arr[i]);       UNDO
        }
}
```



Total non base ⇒ n!

Total base calls ⇒ n!

Time taken for non base calls ⇒ n

" " " base " ⇒ n.

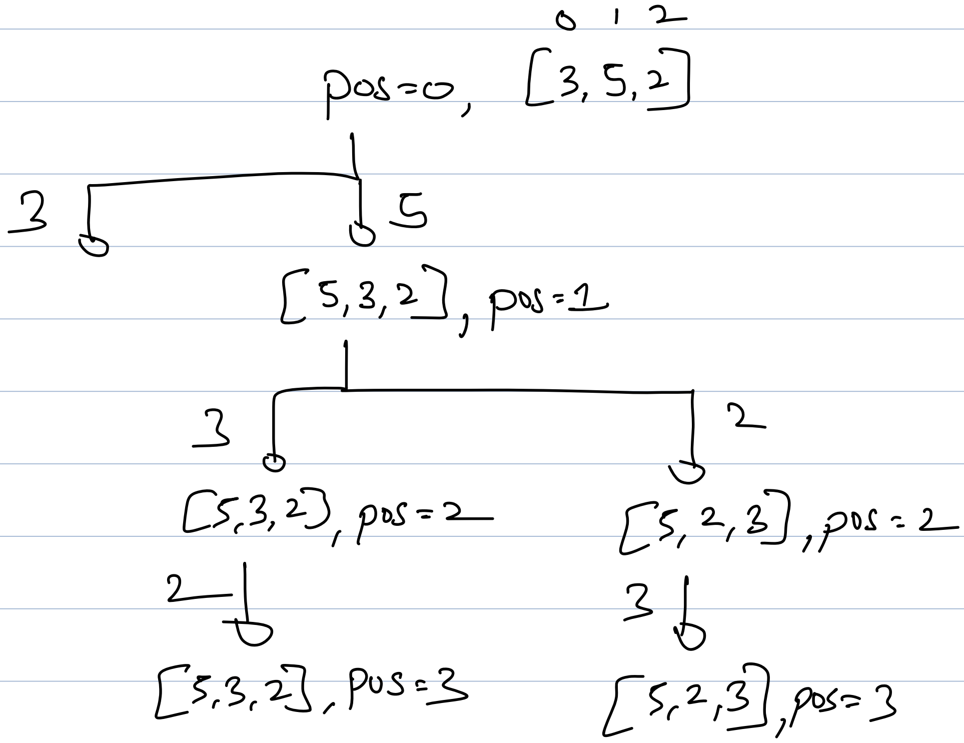
Total time = n×n! + n×n!

TC ⇒ $\boxed{n \times n!}$

SC ⇒ $\underbrace{O(n)}_{\text{Hashset}} + \underbrace{O(n)}_{\text{ans list}} + \underbrace{O(n)}_{\text{call stack}}$

SC ⇒ O(n)

arr = [3, 5, 2]


0 1 2
pos=0, [3, 5, 2]
3
5
[5, 3, 2], pos=1
3
2
[5, 3, 2], pos=2
[5, 2, 3], pos=2
2
3
[5, 3, 2], pos=3
[5, 2, 3], pos=3

Pseudo Code!

```
void gen ( int n , int arr[] , int pos ) {
    if (pos == n) {
        print arr;
        return;
    }

    for (int i=pos; i<n ; i++ ) {
        swap ( arr[pos], arr[i] );
        generate (n, arr, pos+1);
        swap (arr[pos], arr[i] );
    }
}
```

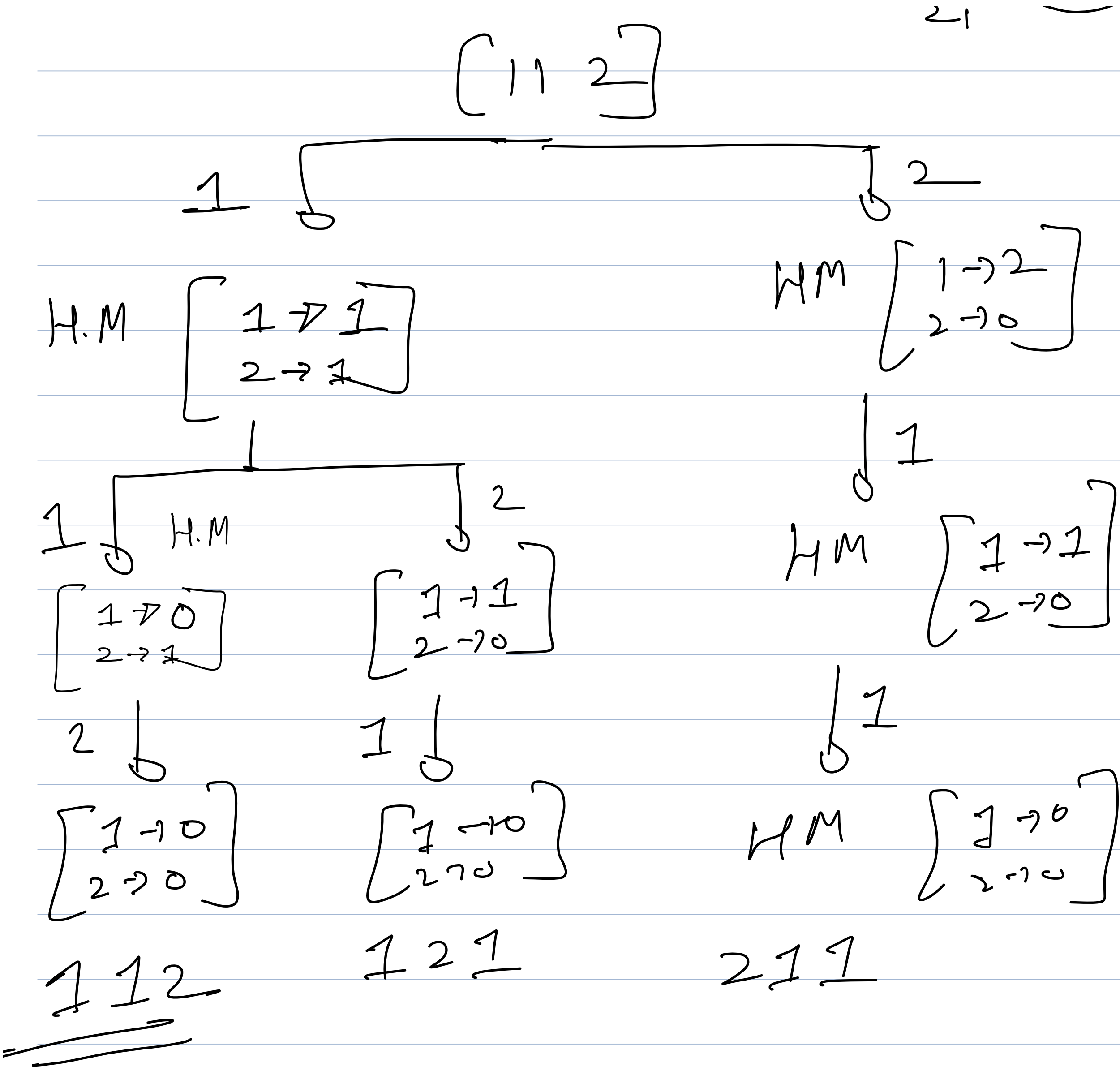
TC : $O(n! \times n)$

SC : $O(n)$

Non distinct elements!

H.M $\begin{bmatrix} 1 \rightarrow 2 \\ 2 \rightarrow 1 \end{bmatrix}$

$\frac{3!}{\;}$ (3)


21
[1 1 2]
1
2
H.M [1 → 1, 2 → 1]
HM [1 → 2, 2 → 0]
1
2
H.M
1
[1 → 0, 2 → 1]
[1 → 1, 2 → 0]
HM [1 → 1, 2 → 0]
2
1
1
[1 → 0, 2 → 0]
[1 → 0, 2 → 0]
HM [1 → 0, 2 → 0]
1 1 2
1 2 1
2 1 1