Backtracking - 1

Question: Generate all arrays of size N with 0's / 1's in lexicographical order (Dictionary order)

N ≤ 20

N = 3

$\frac{0/1}{2} \quad \frac{0/1}{2} \quad \frac{0/1}{2}$ $\left\lceil \frac{3}{2} \right\rceil$

ans: [→ [0,0,0], [0,0,1], [0,1,0], [0,1,1],
→ [1,0,0], [1,0,1], [1,1,0], [1,1,1]]

a b
0 1

ans = [ ]

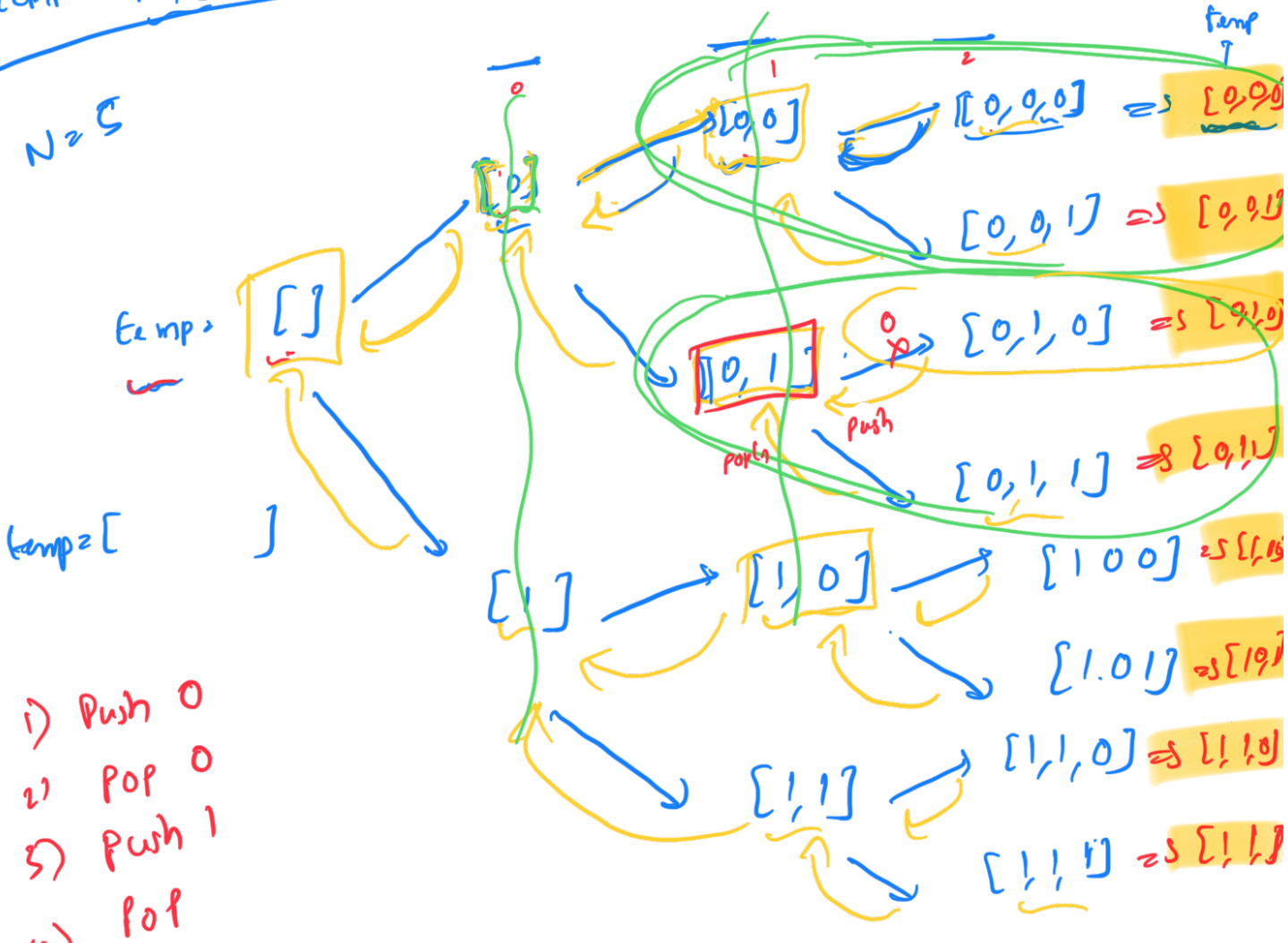
array of array

Dynamic Array

ind

temp: [ ]

N = 3



1) Push 0
2) Pop 0
3) Push 1
4) Pop

For ith element :

1) select 0
2) select 1

→

```
temp.push(0);
F(ind+1);
temp.pop();
temp.push(1)
F(ind+1);
temp.pop()
```

Parameters :

1) ind
2) temp (Global variable)
3) N : size of array

temp => global vector , ans => global

```
void allArrays(ind, N) {
    if(ind == N) {
        ans.push(temp);
        return;
    }
    // select 0                          // DO
    temp.push(0);
    allArrays(ind+1, N)
    temp.pop();                          // UNDO
    // select 1
    temp.push(1);
    allArrays(ind+1, N);
    temp.pop();
}
```

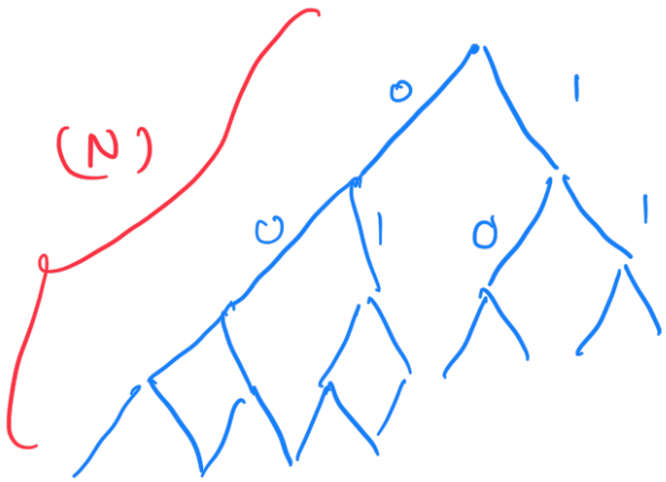
T.C : $O(2^n)$

S.C : $O(N)$ ✓

$O(N) + O(N) = O(N)$

↓ Recursion stack ; ↓ temp

Depth of Tree : (N)



$= O(2^n)$

Question: Given an array of size N, print all subsets in lexicographical order

$2 \times 2 \times 2 \cdots N$ times $= 2^n$

A = [7, 3, 6]

subsets = [7], [3], [6], [7,3], [7,6], [3,6], [7,3,6], [] => 8 = 2^n

Output = [[] [3] [3,6] [3,6,7] [3,7] [6] [6,7] [7]]

2^n

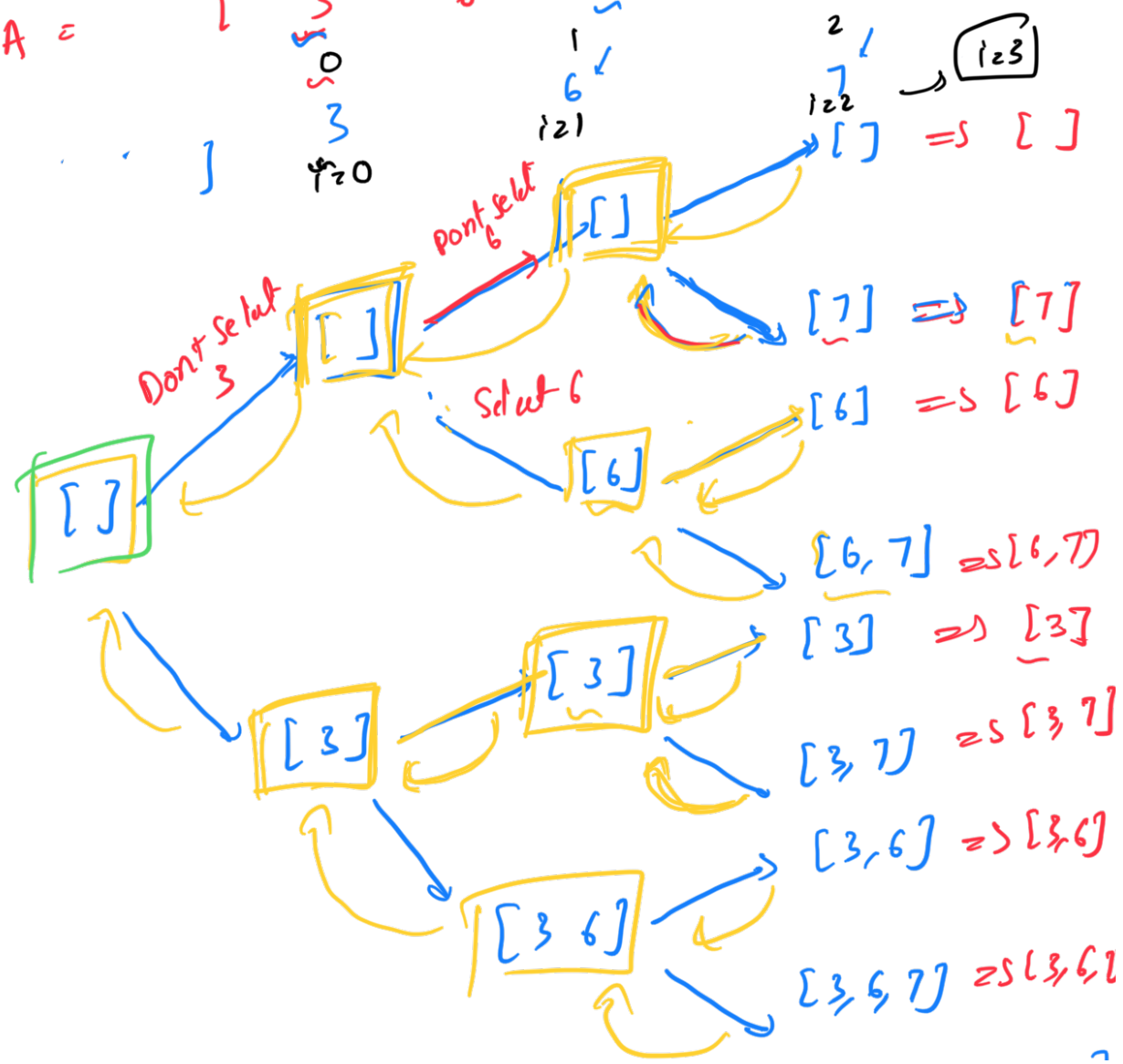
ans = []

[2][2]...N times

N=3

A = [3 6 7]

temp = [ ]



Ans? [[], [7], [6], [6,7], [3], [3,7], [3,6], [3,6,7]]

sort(ans);

Parameters:

1) ind
2) N
3) temp

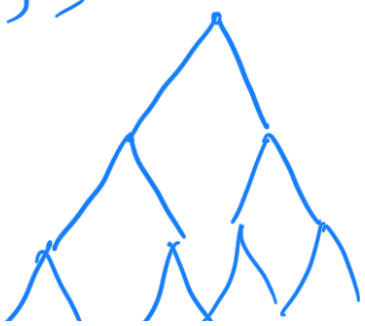
```
allSubsets( ind, N, temp) {
    if(ind == N){
        ans.push(temp);
        return;
    }
    // Don't include A[ind]
    allSubsets( ind+1, N, temp);
    // Include A[ind]
    temp.push(A[ind]);                    // DO
    allSubsets(ind+1, N, temp);
    temp.pop()                            // UNDO
    return;
}
main() {
    sort(A);                              // sort the given
    allSubsets(0, A.size()-1);
    sort(ans);
}
```

ans = hashset
↓ output

ans = hashset( );

T.C: $O(2^n)$

S: $O(n)$



i^{th} position

h

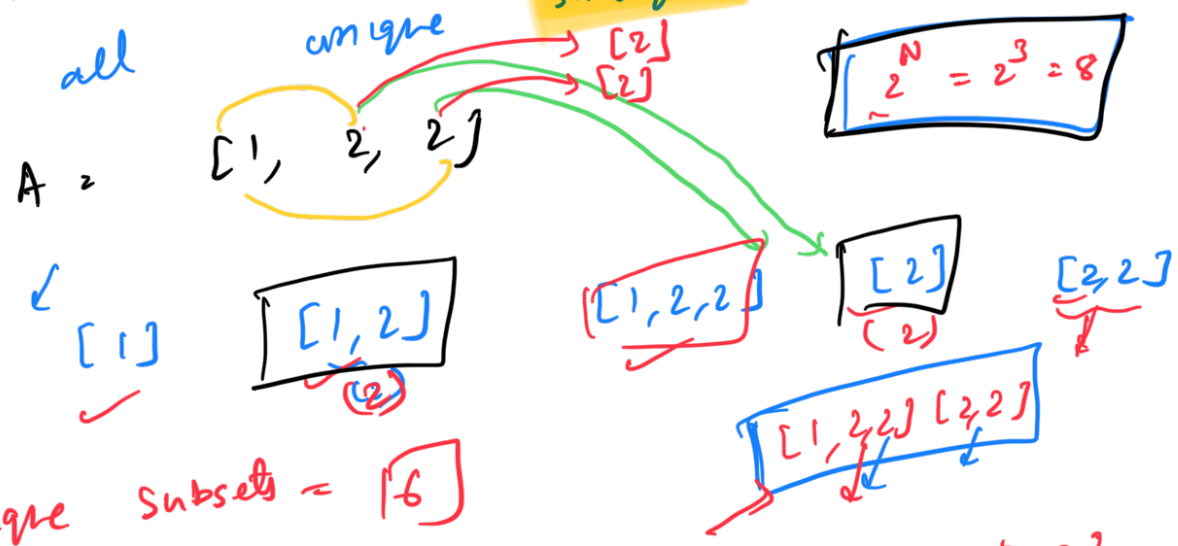
A = [3, 4, 5, 7, 7, 8]

a^b

$n!$

$n+k-1 \; C \; n-1$

Question: Given a sorted array, generate all unique 'subsequences' of it.



Unique subsets = 6

Solution1: Hashset

{1, 2}
{2, 1}
{1, 2} = {2, 1}

Solution2:

→ Can We do without using hashset

→ Dont even generate these duplicate subsets.

A = [1, 1, 2, 3]

unique subsets = 12

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

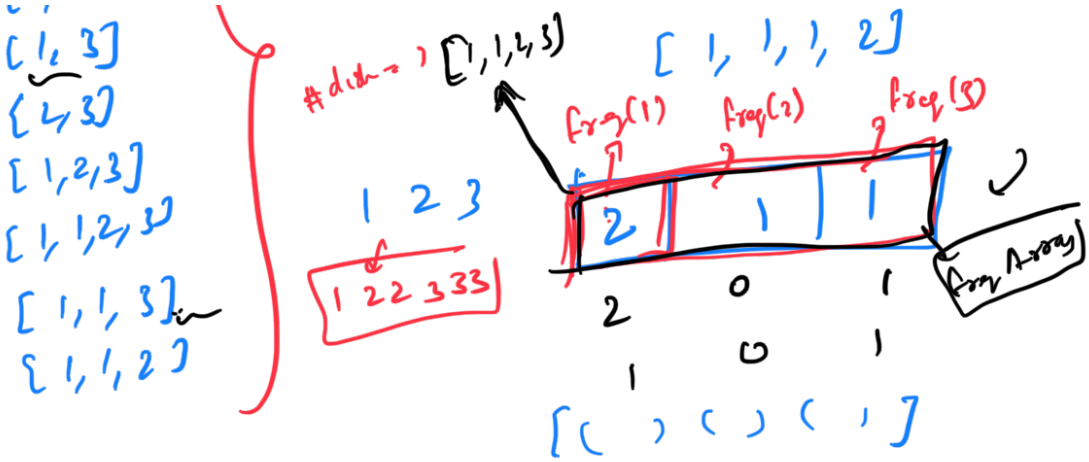
→ All these 12 subsequences have to be in sorted order

N = 4

$2^4 = 16$

Distinct: 1, 2, 3 (3)

3 1 0



Size of Array:

No. of distinct/unique elements

N = 6

distinct = 3

A = → [1, 1, 1, 2, 2, 4]

F = → [(1, 3), (2, 2), (4, 1)]

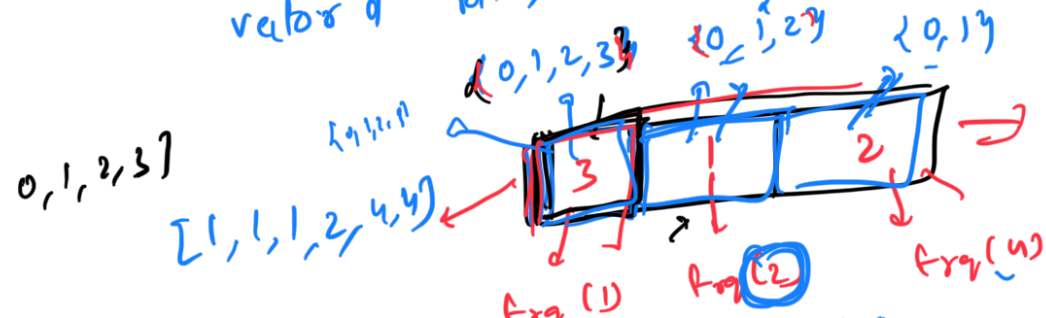
val freq

F.size()

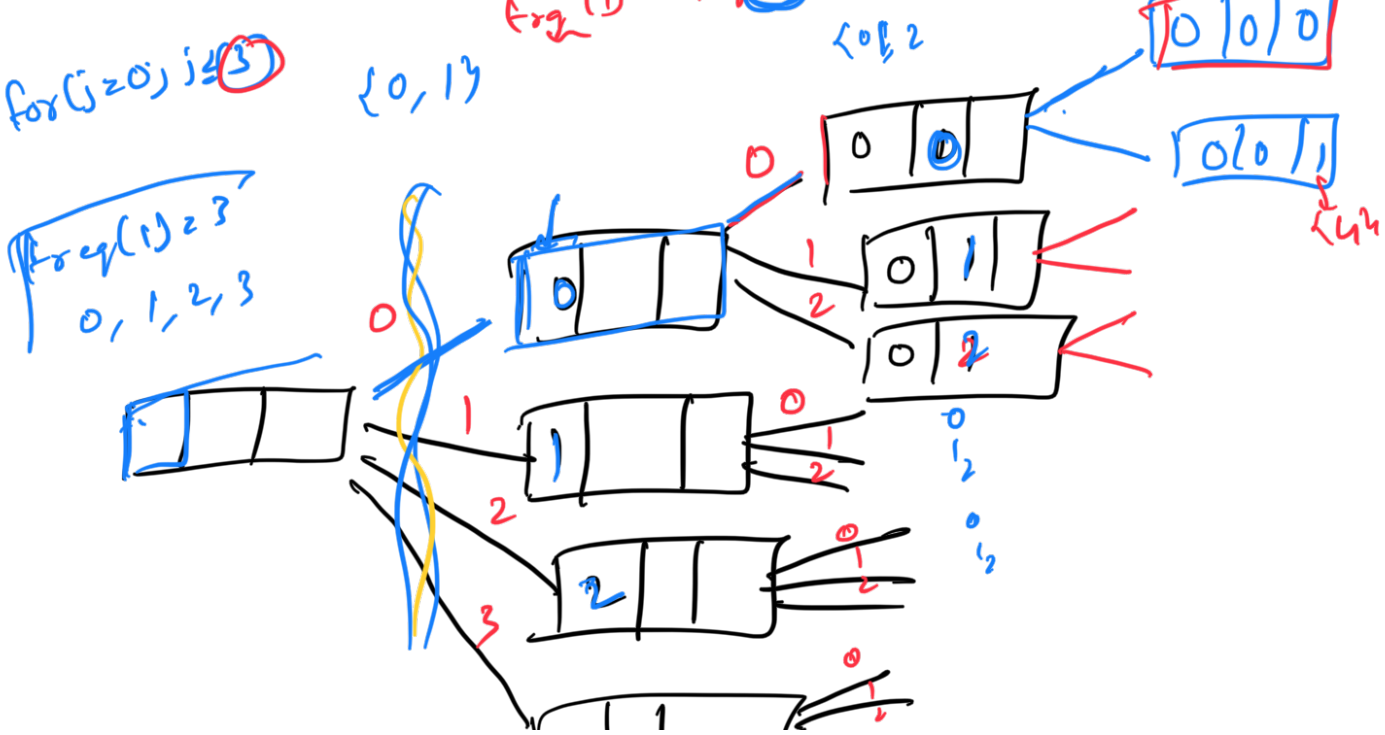
size = #distinct

vector of pair, struct

struct / enum / hashmap



Generate all possible frequency arrays;



```
generate (ind, F, temp) {
    if(ind == F.size()) {
        // Generate the subsequence using temp
    }
    for(j=0; j < F[ind].freq; j++) {
        temp.push(j);
        generate(ind+1, F, temp);
        temp.pop();
    }
    return;
}
```

X Hashset

Question: Given an array, generate all the permutations

A = [4, 7, 8]

[4, 7, 8] [4, 8, 7] [7, 4, 8] [7, 8, 4] [8, 4, 7] [8, 7, 4]

N!

3 2 1 = N x (N-1) x (N-2)... = N!

4/1/8

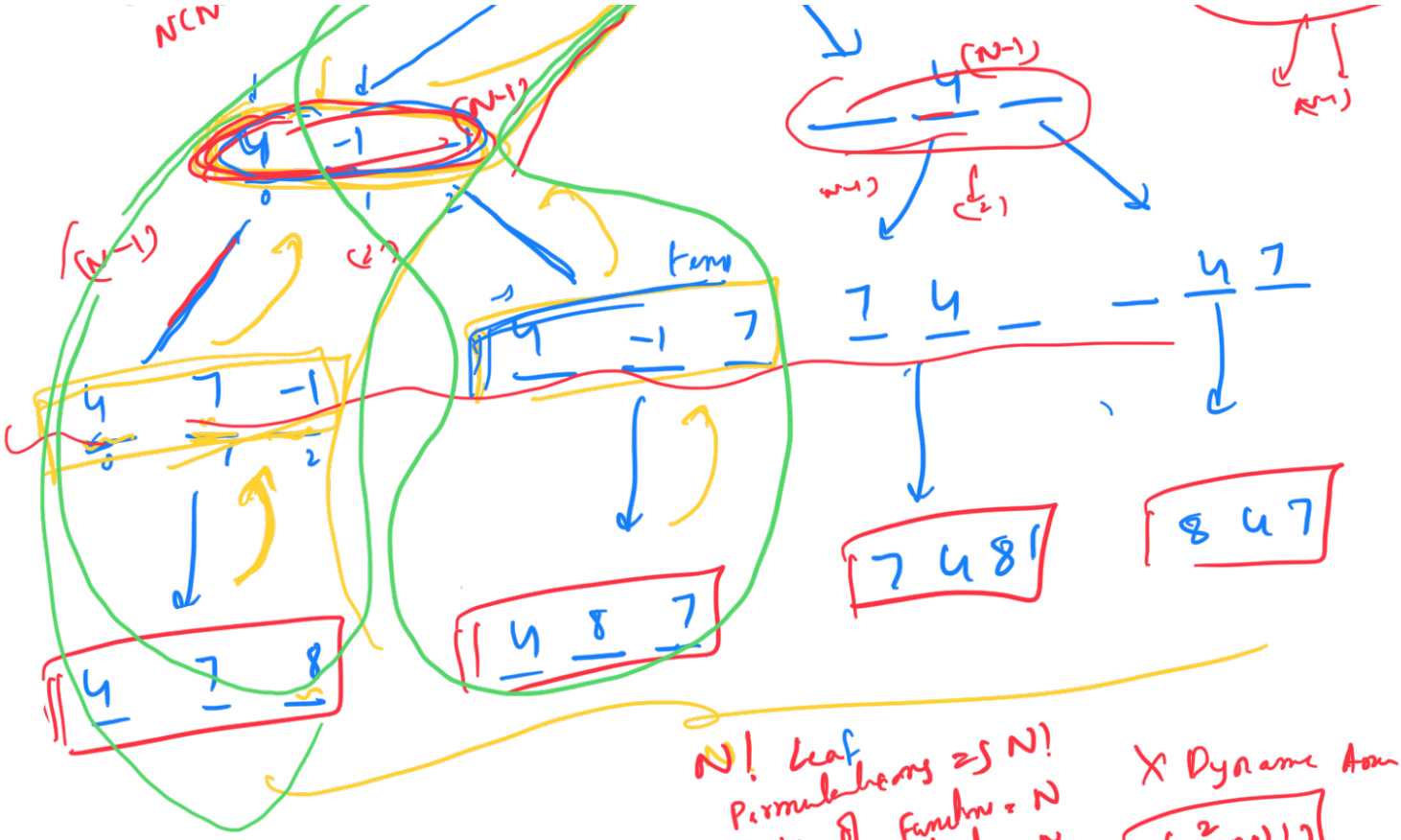
A = [4, 7, 8]

-1, -1, -1

temp = [-1, -1, -1]

temp = [-1, -1, -1]

(N) + N(N-1) ...



N! Leaf
Permutations = N!
No. of Function = N
T.C. Per function = N

X Dynamic Array

$O(N^2 * N!)$

Parameters:

1) ind
2) N → size of Array
3) temp → vector / ArrayList

[-1, -1, -1]

```
void permute( ind, N, temp) {
    if (ind == N) {
        ans.push(temp);
        return;
    }
    for (j = 0; j < N; j++) {
        if ( temp[j] == -1) {
            temp[j] = A[ind];        // DO
            permute (ind + 1, N, temp);
            temp[j] = -1;            // UNDO
        }
    }
}
```

T.C Per Function = O(N)

T.C: (#Function calls * T.C. per Fun)

O(n)

Generate the unique Permutations

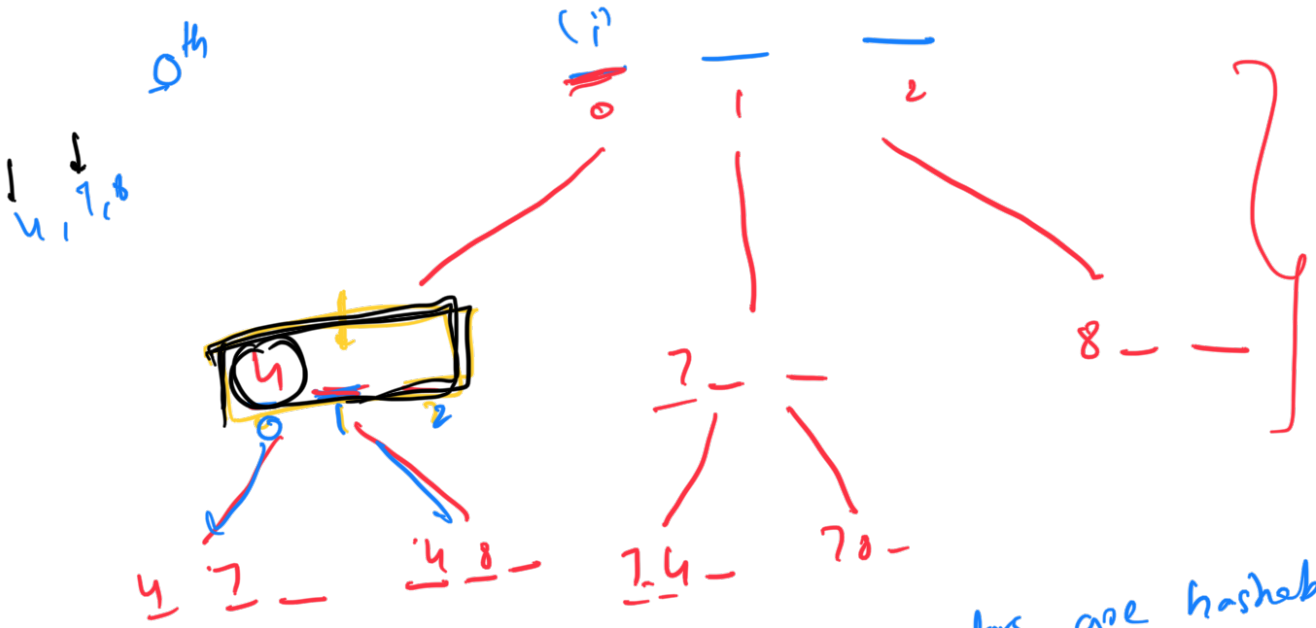
Extension:

solution1: HASH SET

solution2:

UTKARSH's Approach

A = [4, 7, 8]



vectors are hashable.

Ordered_set / Treeset (C++, Java) | dict (Python)

Exact T.C

functions = $C_n + n(n-1) + n(n-1)(n-2) + \dots + n!$

T.C = $O(n!)$

T.C Per

Upper bound: