Recursion / Backtracking.

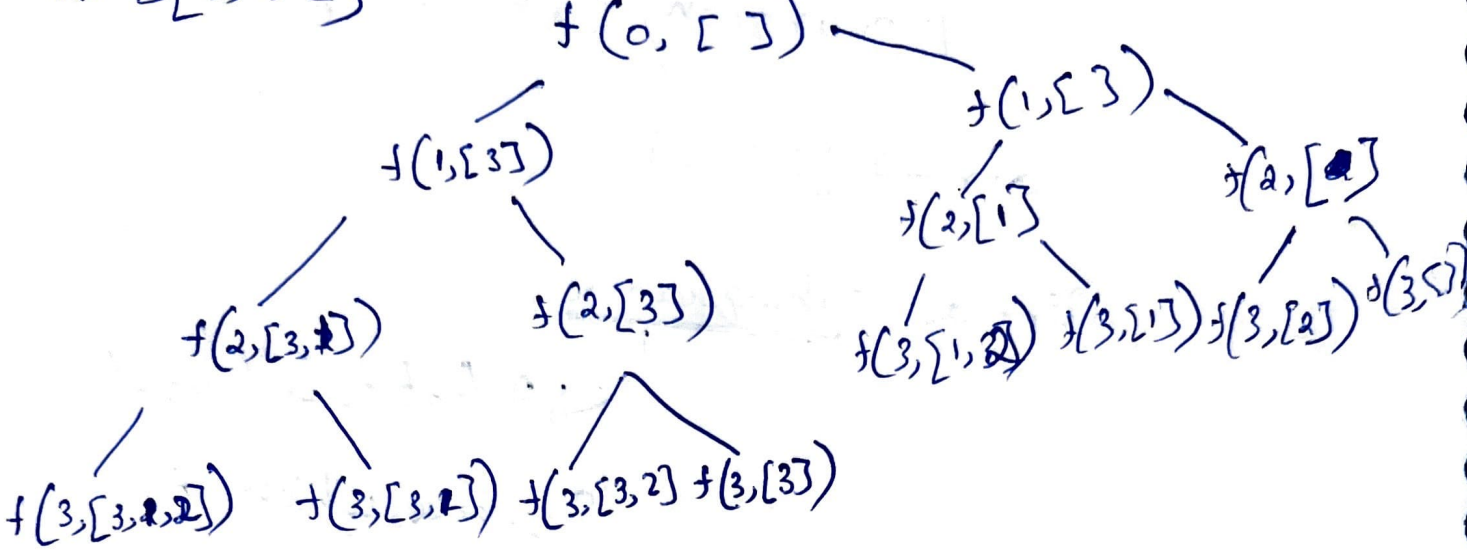
arr → [3, 1, 2]

Print all subsequence → A contigoues / non-contigous sequance, which follows the order

arr → [3 1, 2]

```
f(index, [])
{
  if (index >= n)
    { print([]);
      return;
    }
  [].add(arr[i]);
  f(index+1, []);
  [].remove(arr[i]);
  f(index+1, [])
}
```

arr → [3, 1, 2]



if (index >= arr.length) Print [ ]

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

↓

Java Code:

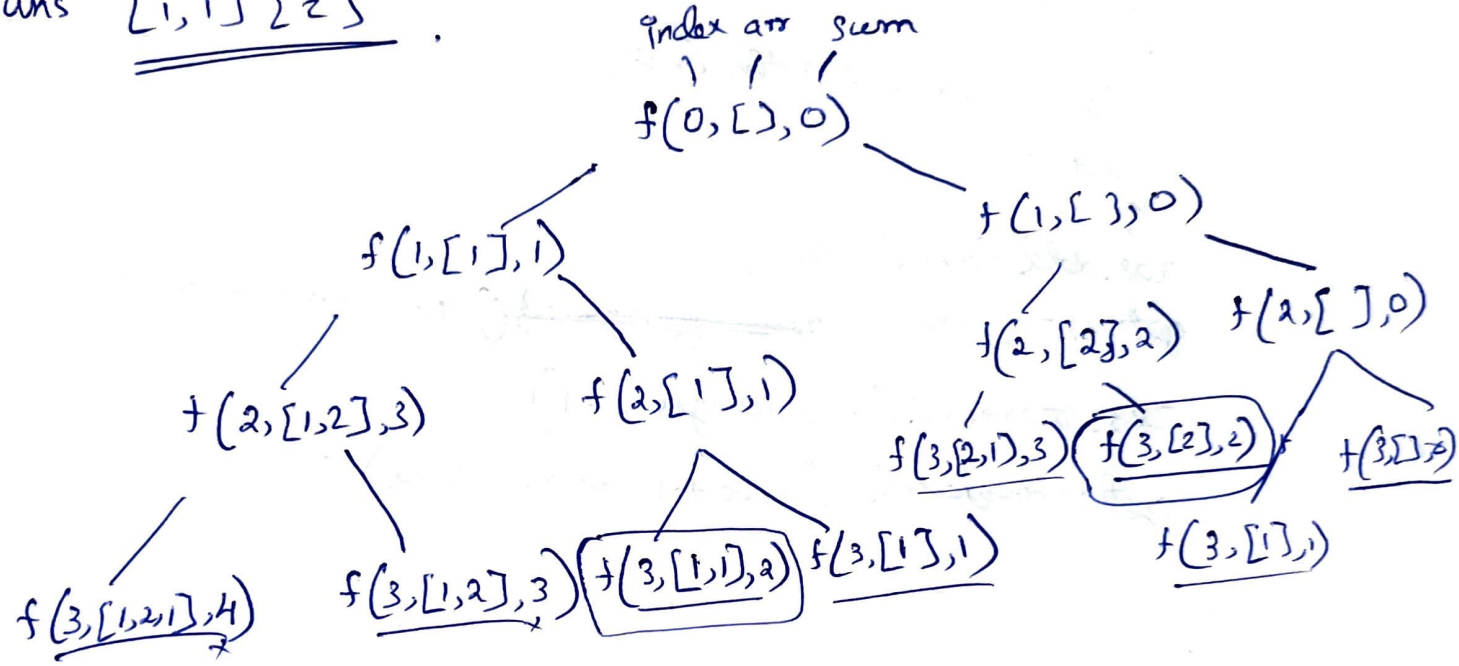
```
getsebsequance (index, arr, List<>res){
  if(index .> = arr.length)
  { system.out.println (res);
    return;
  }
  res.add (arr[index]);
  getsubsequance (~~resize()-1~~( index+1, arr, res);
  res. remove(res.resize()-1)
  getsubsequance (index+1, arr, res);

}
```

Print Subsequence whose sum is k

take / not take.

arr → [1, 2, 1] Sum = 2

ans [1, 1] [2] .



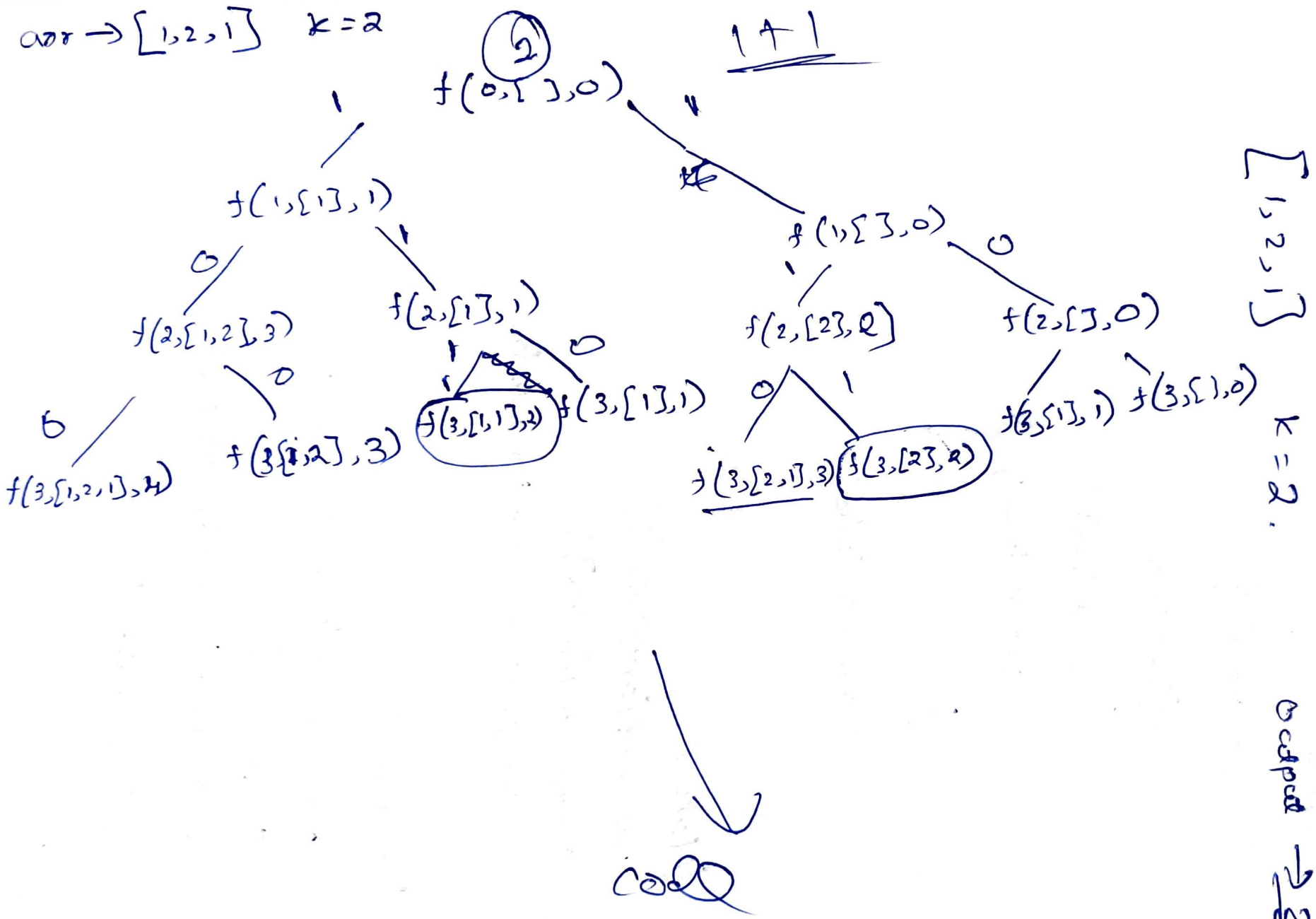
[1, 1]

[2]

```
f(index, [], sum, k)
{ if(index >= arr.length)
   { if(k == sum) return res [ ]
   }
  [ ].add(arr[index);
   sum = arr[i];
  f(index+1, [ ], sum, k)
  [ ].remove(arr[index])
   sum -= arr[i];
  f(index+1), [ ], sum, k)
```

Print first subsequence whose sum is equals k.

```
getfirstSubsequence ( ) {
    if (index >= n) {
        if (k == sum)
        { sout (res) ;
          return true ;
        }
        else
        { return false;
        }
    }
//take//
res.add (arr[index]);
sum+ = arr[index];
if (getfirstsubsequence ) return true;

// no take //
res. remove (res.size() - 1) ;
sum - = arr [index];
if (getfirst (index+1,a)) return true
return false;
}
```


arr → [1,2,1] k=2
2
1+1
f(0,[ ],0)
f(1,[1],1)
f(1,[ ],0)
f(2,[1,2],3)
f(2,[1],1)
f(2,[2],2)
f(2,[ ],0)
f(3,[1,2,1],4)
f(3,[1,2],3)
f(3,[1,1],2)
f(3,[1],1)
f(3,[2,1],3)
f(3,[2],2)
f(3,[1],1)
f(3,[ ],0)
[1,2,1]
k=2.
Count Subsequence sum = k.
Output → 2
code

```
Public Stati int getcount(int index, int [] arr, int k, int n, List<I> res, int sum)
{
    if(index >= n)
    {
        if(k == sum)
        {
            return 1;
        }
        else {
        return 0;
        }
    }
```

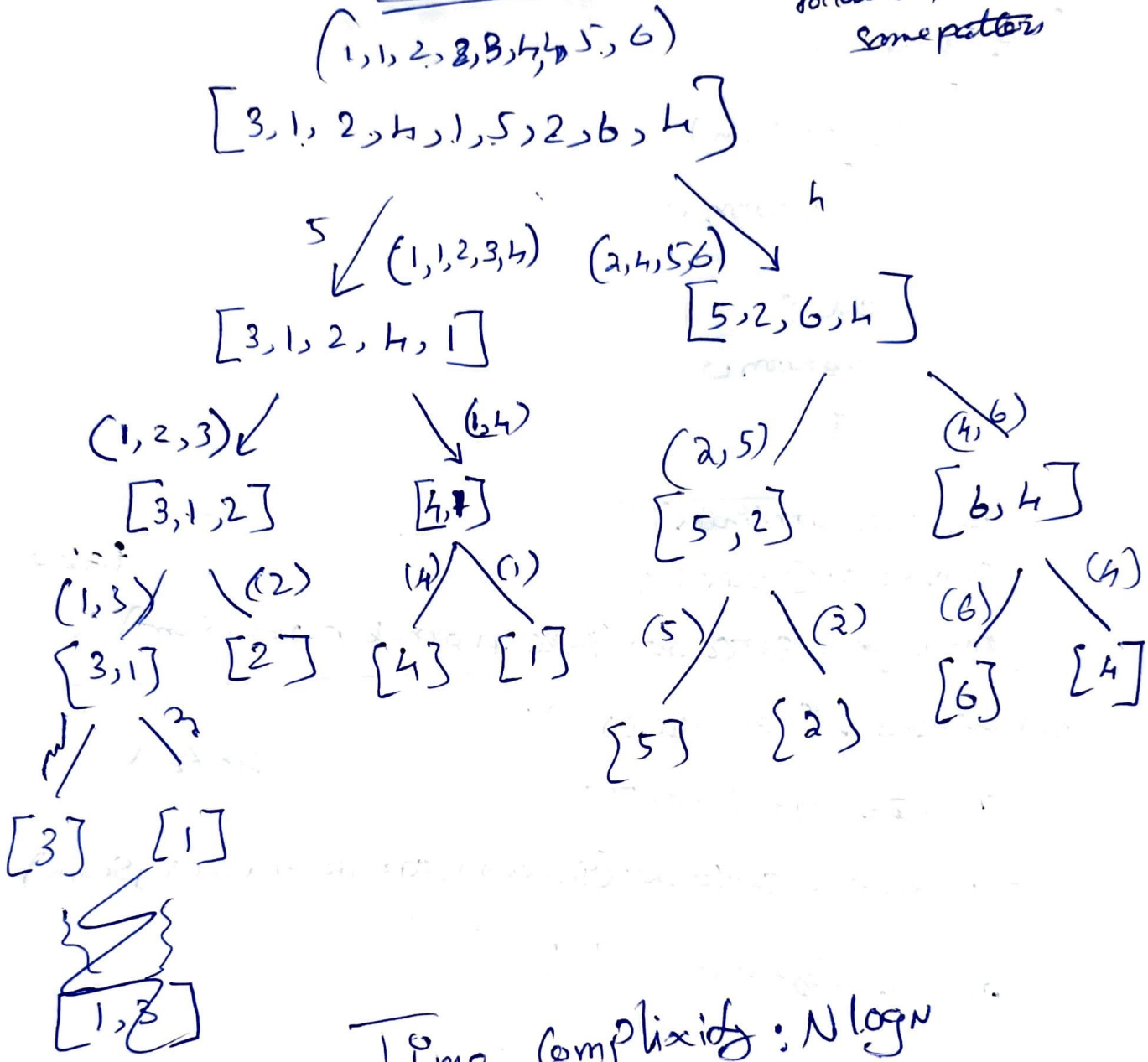
```
res.add(arr[index]);                                    fake
sum += arr[index];
int left = getcount(index+1, arr, k, n, res, sum);
```

```
res.remove(res.size()-1);                               backtrack
sum -= arr[index];
int right = getcount(index+1, arr, k, n, res, sum);
return left + right;
```

```
}
```

Merge Sort

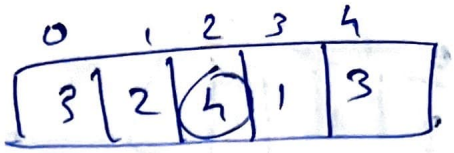
follow the same pattern



Time Complixity : N log N

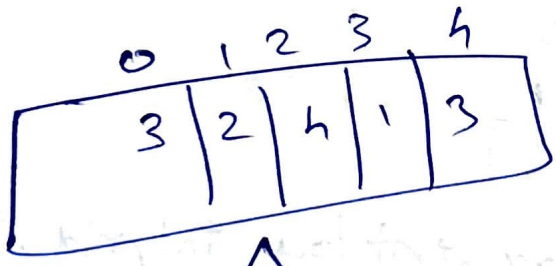
Space Complixity : O(N)

mergesort :- Pseudocode

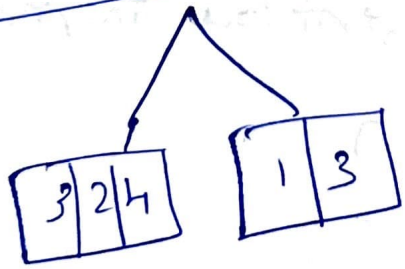


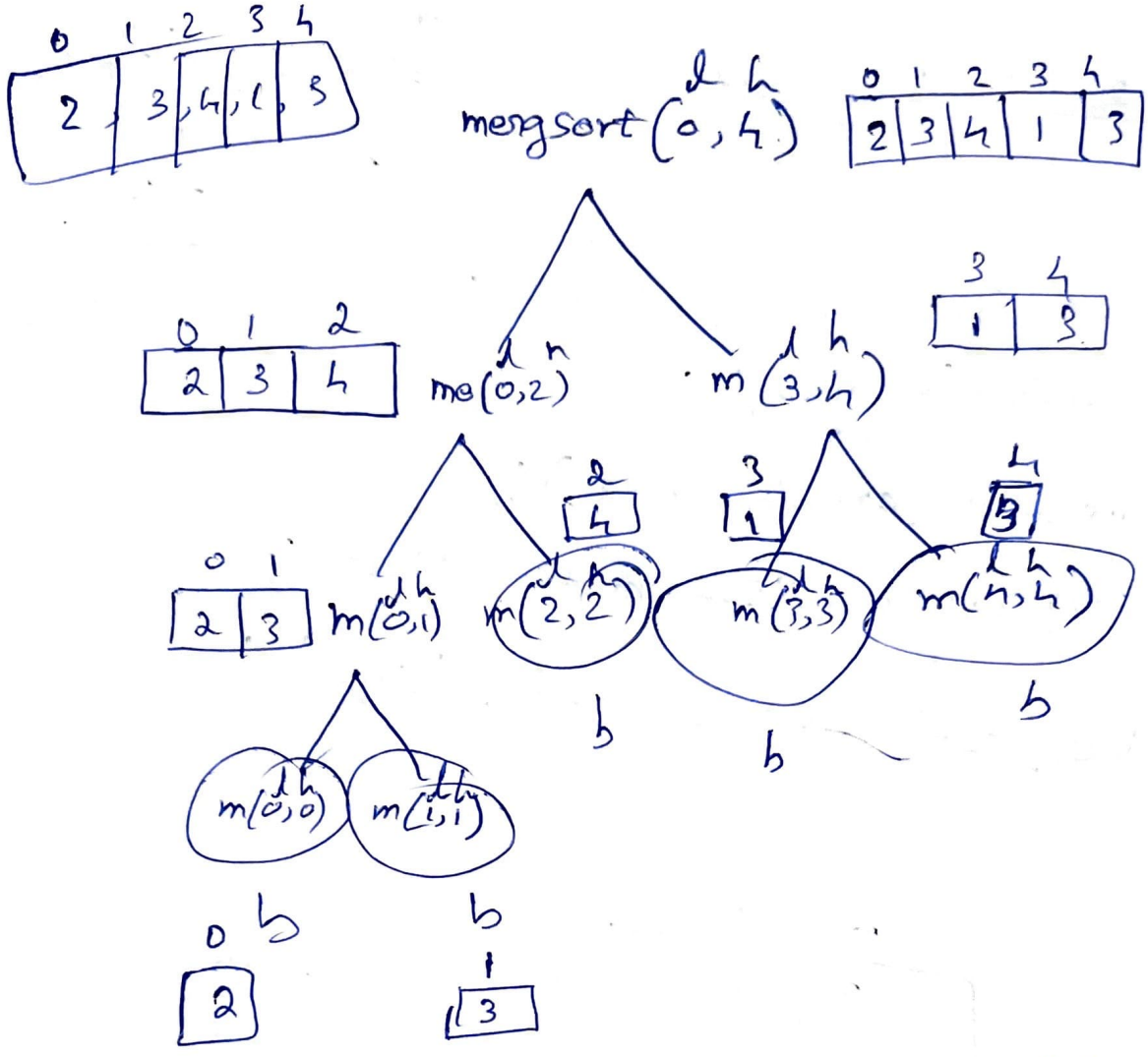
$\frac{0+4}{2}$ = 2

```
mergeSort (arr, low, high)
{   if (low == high) return;
    mid = (low + high)/2;
    mergeSort (arr, low, mid);
    mergsort (arr, ~~low~~, mid+1, high);
    merg sort (arr, low, mid, high);
}
```



l = 0
h = 4
~~m = 2~~





merge (int [] arr, int low, int mid, int high) {

int[] temp = new [ ]

i = 0; p

~~left~~ = low

~~right~~ high

while (p <= mid && j <= high) {

if (arr[i] <= arr[j])

{

temp[i++] = arr[i]

i++;

}

```
        }
        else
        {
            temp[t++] = arr[j];
            j++;
        }
    }
    while (i <= mid)
    {
        temp[t++] = arr[i];
        i++;
    }
    while (j <= high)
    {
        temp[t++] = arr[j];
        j++;
    }
    for (int k = 0; k < temp.length; k++) {
        arr[low + k] = temp[k];
    }
}
```

Java Code:

```
Public static Void mergsort (int[] arr, int low int high) {
    if (low == high) return;
    int mid = (low+high) /2 ;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid+1, high);
    merge(arr, low, mid, arr);
}
Public static Void merge (int arr, int low int mid,
                          int high) {
    int[] tem = new int [high - low+1];
    int t=0;
    int i=low;
    int j= mid+1;
    while (i<=mid && j <=high){
        if(arr[i] <=arr[j]){
            temp[t++] =arr[i];
            i++;
        }
        else{
            temp[t++]=arr[j];
            j++;
    }  }
```

Rest order

```
while (i <= mid) {
    temp[t++] = arr[i];
    i++;
    }
while (j <= high) {
    temp[t++] = arr[j];
    j++;
    }
for (int k=0; k<temp.length; k++) {
    arr[locat k] = temp[k];
    }
}
```

Quick Sort

Time complexity: $O(N(\log N))$

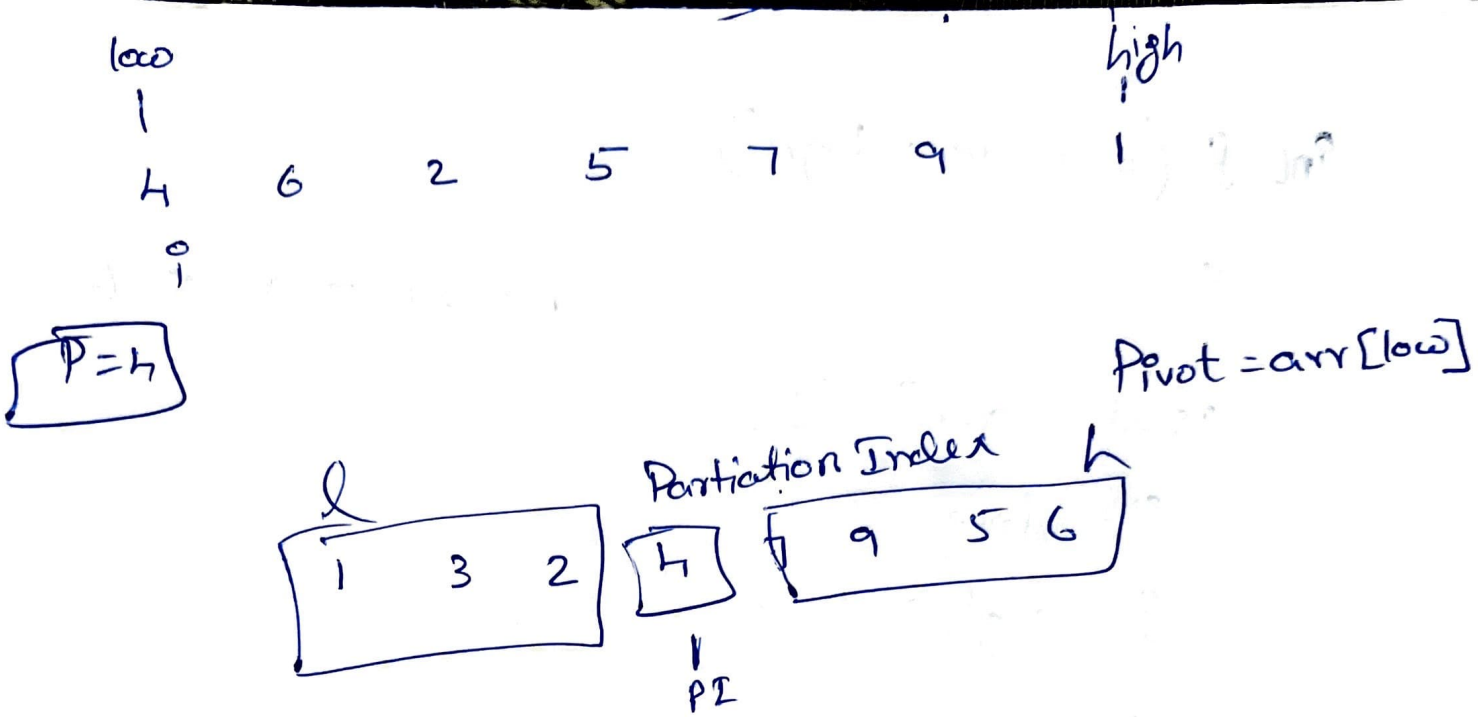
Space complexity: $O(N)$

$arr[3] = [4, 6, 2, 5, 7, 9, 1, 3]$ (P = 4)

~~$arr = [2, 3, 4]$~~

$arr[3] = [4, 6, 2, 5, 7, 9, 1, 3]$ (P = 4)

1. Take a pivot & place it in its correct place in its sorted array.
2. Smaller on the left and larger on the right



D. Pseudocode:-

```
quicksort (arr, low, high)
{
  if (low < high)
  {
    Partition index = f (arr, low, high);
    quicksort (arr, low, partitionIndex-1);
    quicksort (arr, partitionIndex+1, high);
  }
}
```

```
int f (arr, low, high)
{
                                    1 3 2 4 7 9 5 6
pivot = arr[low],
  i = low;
  j = high;
while (i < j)
  {
   while (arr[i] <= arr[pivot] && i <= high)
    {
      i++;
    }
   while (arr[j] >= arr[pivot] && i >= low)
   {
     j--;
   }
  if (i < j) swap(arr[i], arr[j]);
  }
 Swap [arr[low], arr[j]);
 return j;
}
```

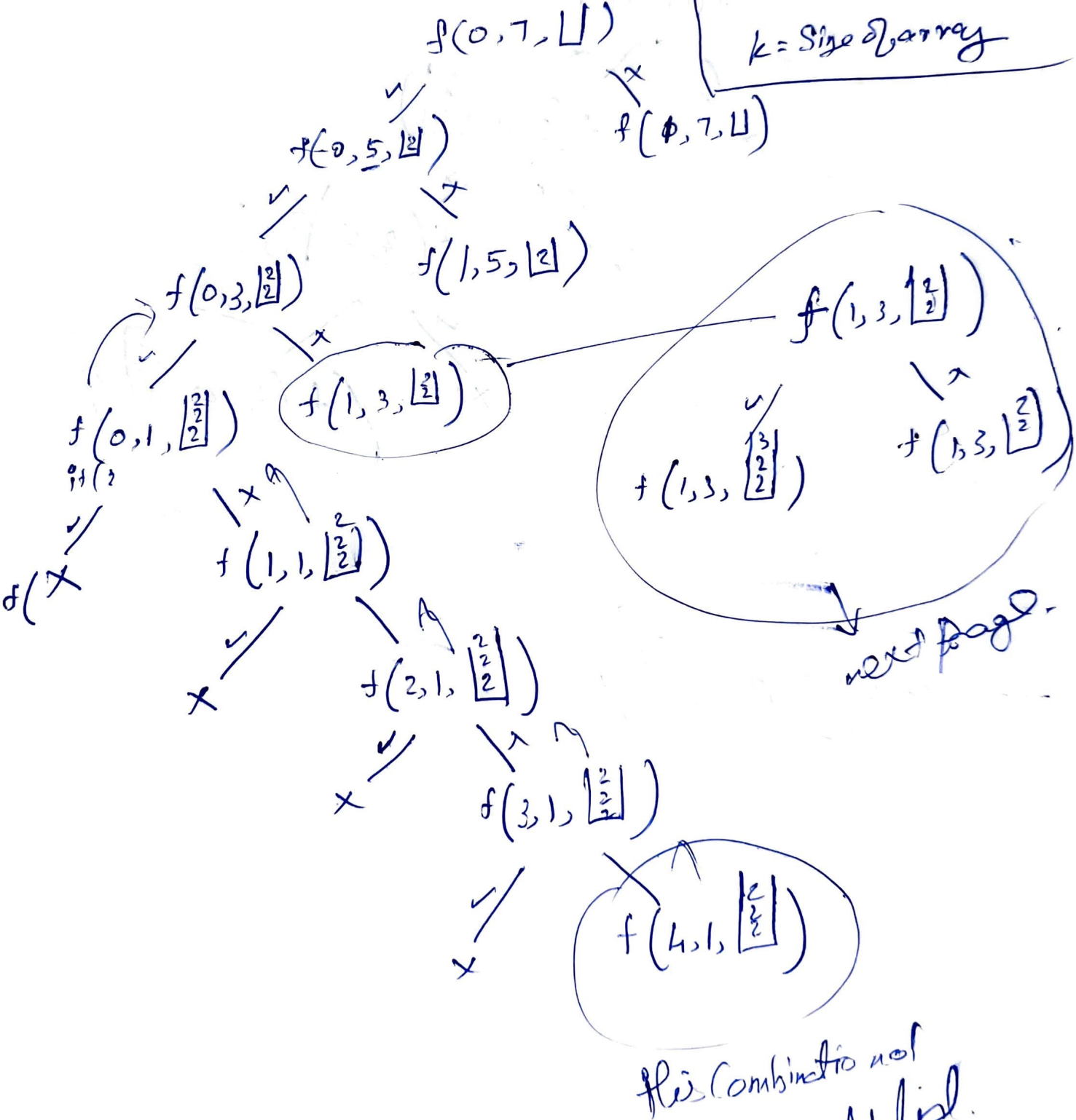
Combination Sum

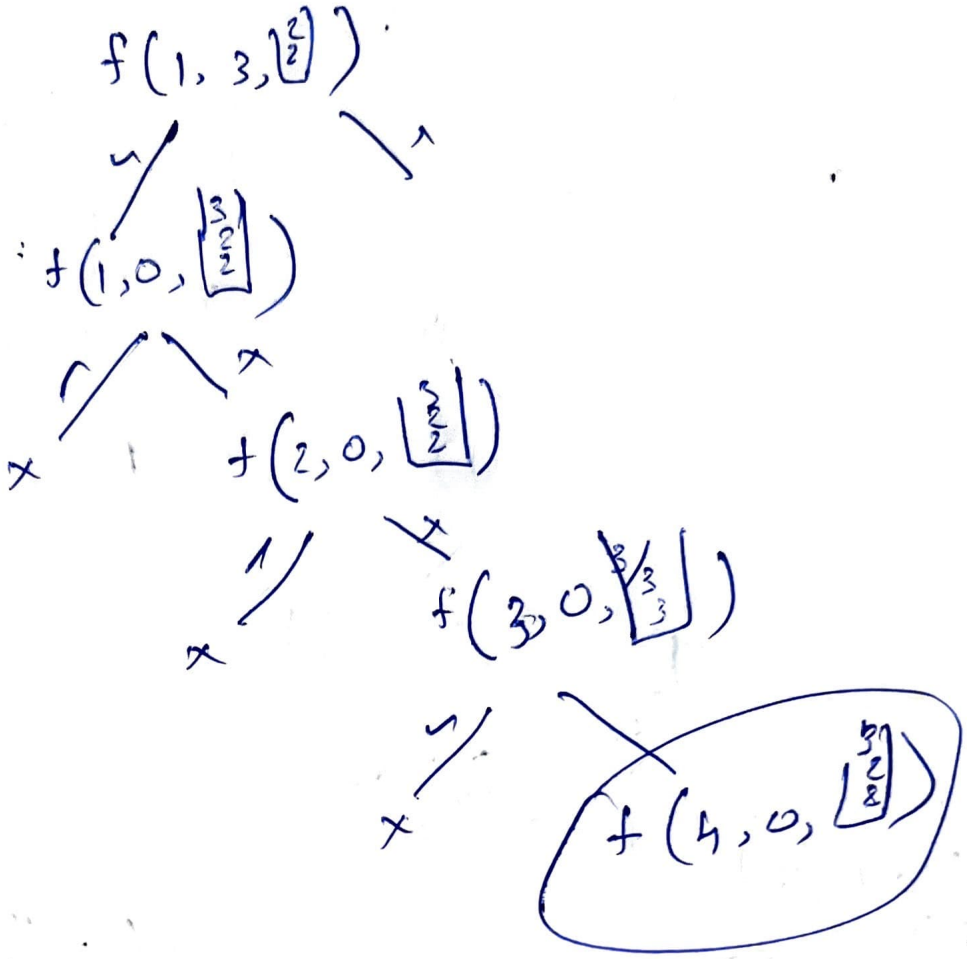
$TC = 2^t * k$

arr = [] = [2, 3, 6, 7] target = 7

t = take / notake

k = Size of array





res = [2, 2, 3]

f(index, target, ds)

ds.remove(ds.size()-1) ✓ ds.add(a[index]) ✗

f(index, target-a[index], ds) f(index+1, target, ds)

↓

if(a[index] <= target)

```
combinationSum(i, arr, target, list, res) {
    if (index >= arr.length) {
        if(target == 0)
        { res.add(new ArrayList<>(list));
          return;
        }
    }
// take
    if (arr[index] <= target) {
        list.add(arr[index]);
        getCombination(index, arr, target-arr[index], list, res);
        list.remove(list.size()-1);
    }
```

c/ Notake

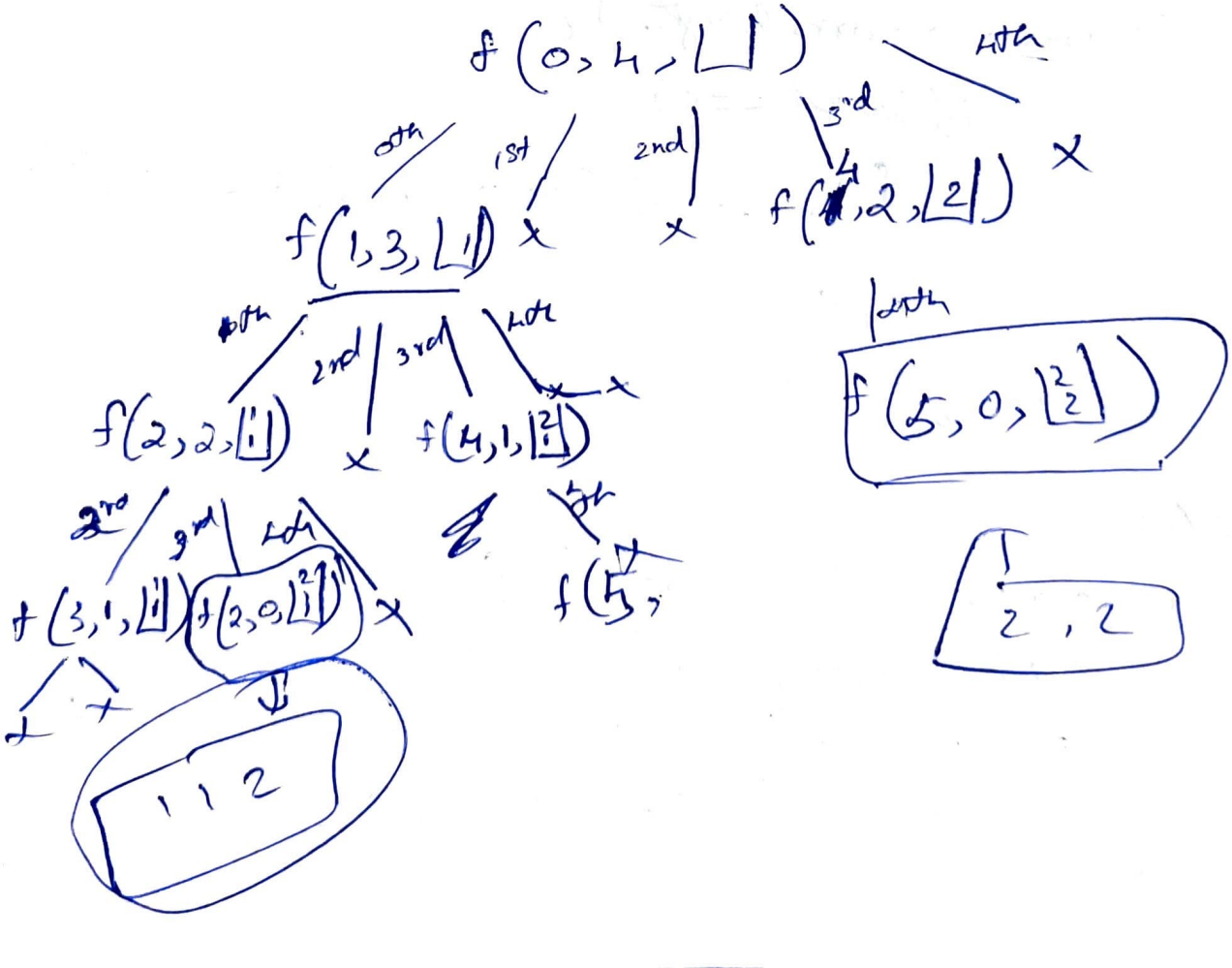
getCombinationSum(index+1, arr, target, list, res);

}

Combination Sum - II

arr: [1, 1, 1, 2, 2] target = 7

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



```
getCombination (int i, int arr, int target, List, res){

    if (target ==0)
      {
        res.add (new ArrayList<>(list));
        return;
      }

    for(int i =index; i<arr.length; i++)
      {
        if (i >index && arr[i] == arr[i-1]) continue;
        if (target <0 || arr[i] > target) break;
        list.add (arr[i])
        getCombination.(i+1, arr, target-arr[i], list, re
        list.return (list.size()-1);

      }
}
```

Subset

Brute force approach – Power Set.

↓

Time Complixity $2^n \times n$

Print all subset

$n = 3$ nums = $[1, 2, 3]$ (0, 1, 2)

$n = 2^n = 2^3 = 8$ [ ], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]

↓

2^n

2	1	0	
0	0	0	[ ]
0	0	1	[1]
0	1	0	[2]
0	1	1	[1,2]
1	0	0	[3]
1	0	1	[1,3]
1	1	0	[2,3]
1	1	1	[1,2,3]

Subsets

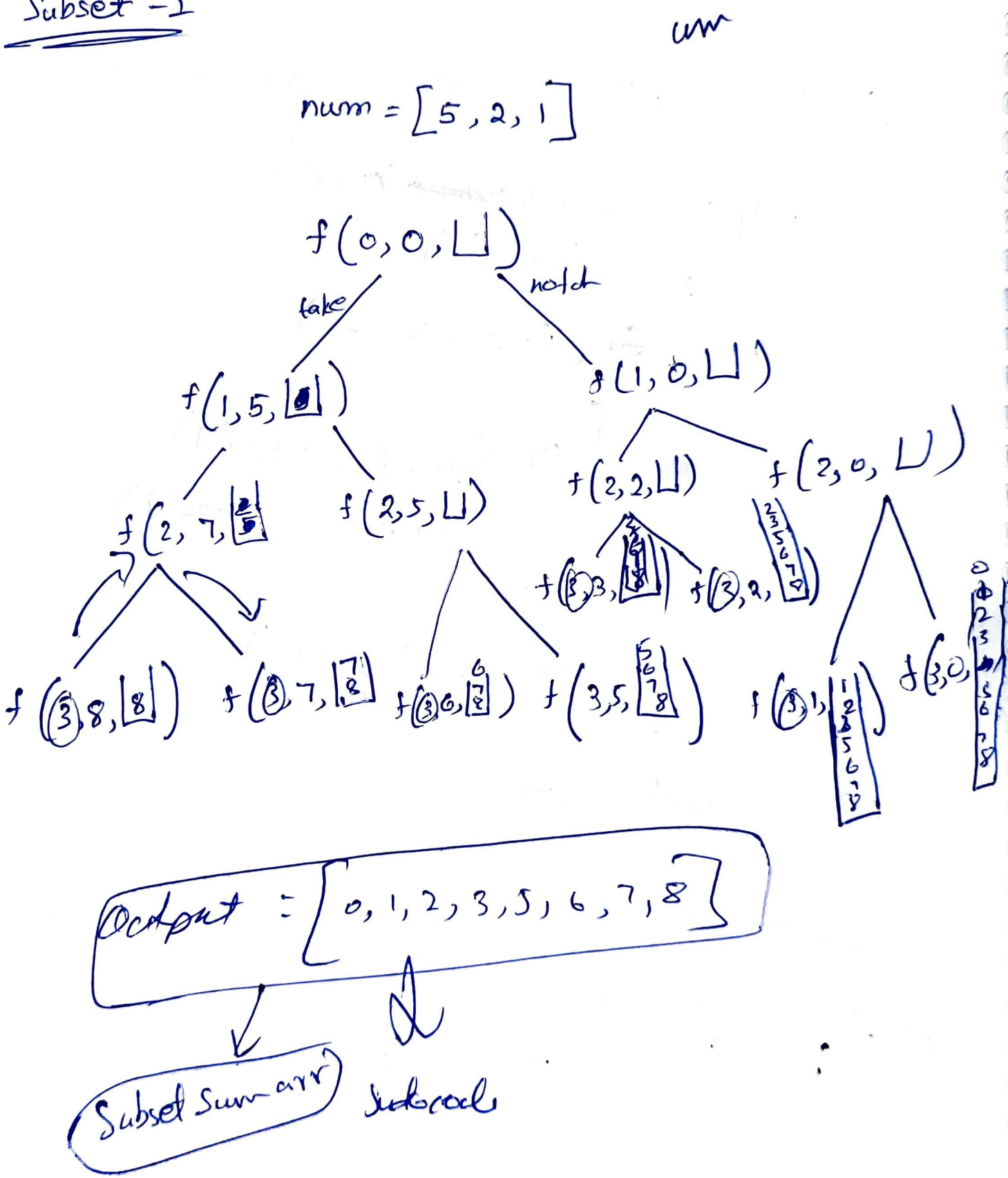
n = 3 $\quad 2^n = 2^3 = 8 \quad$ ans = [ ]

```
for (ncem = 0 -> subset(n-1))
        List = [ ]
    {
      for (i = 0 -> n-1)
      {
        if (ncem & (1<<i))
            List add (ncem[i])
      }
      ans.add(list);
}
return ans.
```

Time Complixt = $2^n \times n$

Spac Complixity = $\leq 2^n \times n$

Subset -I


num = [5, 2, 1]
f(0, 0, [])
take
not take
f(1, 5, [5])
f(1, 0, [])
f(2, 7, [2 5])
f(2, 5, [])
f(2, 2, [])
f(2, 0, [])
f(3, 8, [2 5 8])
f(3, 7, [2 7])
f(3, 6, [5 8])
f(3, 5, [5 7 8])
f(3, 3, [2 5 8])
f(3, 2, [2 5 7])
f(3, 1, [1 2 5 7])
f(3, 0, [0 2 5 7 8])
Output = [0, 1, 2, 3, 5, 6, 7, 8]
Subset sum arr
Leetcode

```
Public static getsumlist (index, nums, res, Sum)
{
    if (index >= num.length)
    {
        res.add (scum) ;
        return
    }
    // take
    scumList (inde + 1, nums, res , Sum + numsCindex)

    //no take
    SumList (index+1 , nums, res, scum) ;
}.
```

Subset - II

arr = [1, 2, 2]

Brute force - TC - 2^n * n → for all generating subsets
storing subset in a list.

Use set to avoid duplicates

Set res = [[1,2,2], [1,2], [1], [2,2], [2], []]

to avoid duplicate.

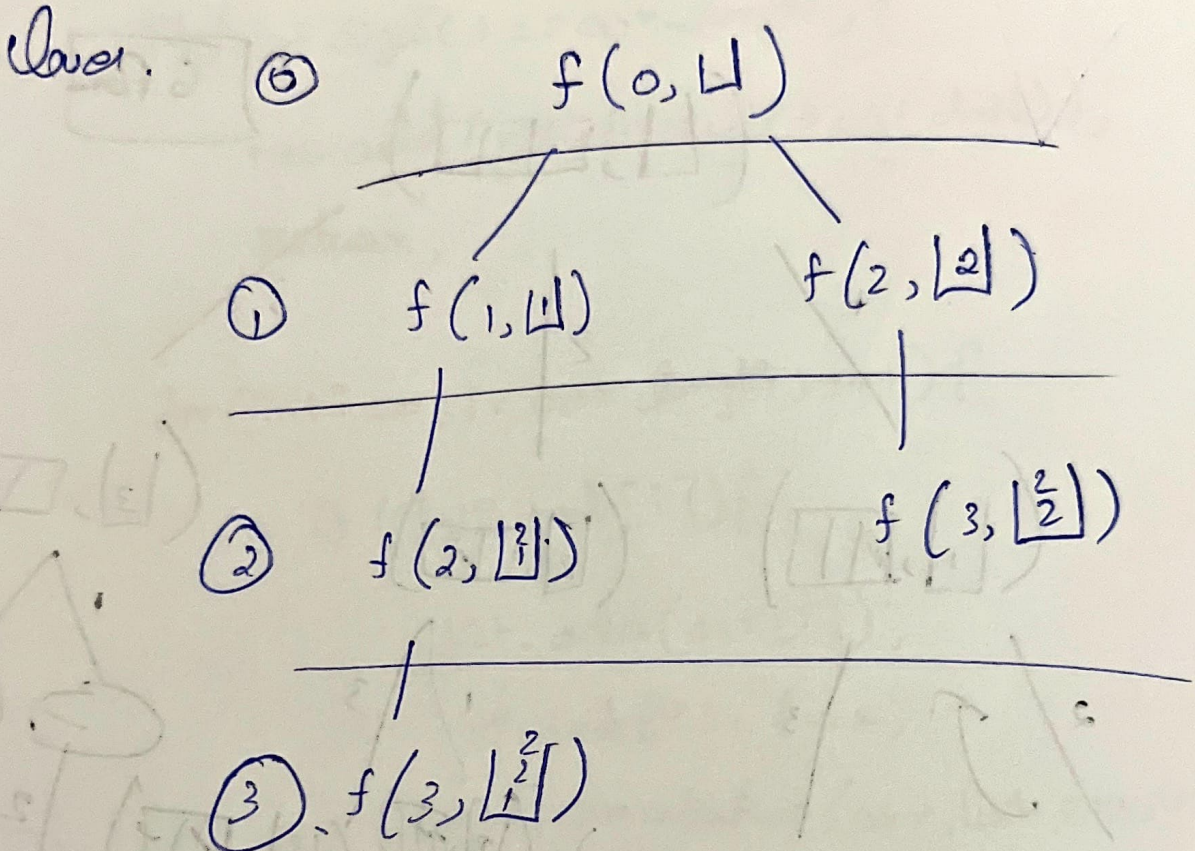
* In this approach first we are generating all the subset and then we are skiping the duplicates.

* After that Converting Set<List> to List<List>

```
getsubset (i, [], list, res) {
    if (i >= nums.length) {
        res.add (new ArrayList<>(list));
        return;
    }
    // take
    list.add()
    get (i+1, [], ls, res);
    list.remove()
    // not take
    get (i+1, [], ls, res);
```

Optimal approch

arr = [1, 2, 2] res = [[ ],



Code

```
getsubsedSum (index, nums, list, res) {
    res.add (new ArrayList <> (list));
    for (int i = index; i < nums.length; i++) {
        if (i > index && nums[i] == nums[i-1]) continue;
        list.add (num[index]);
        getsubset (i+1, nums, list, res);
        list.remove (list.size() - 1);
    }
}
```

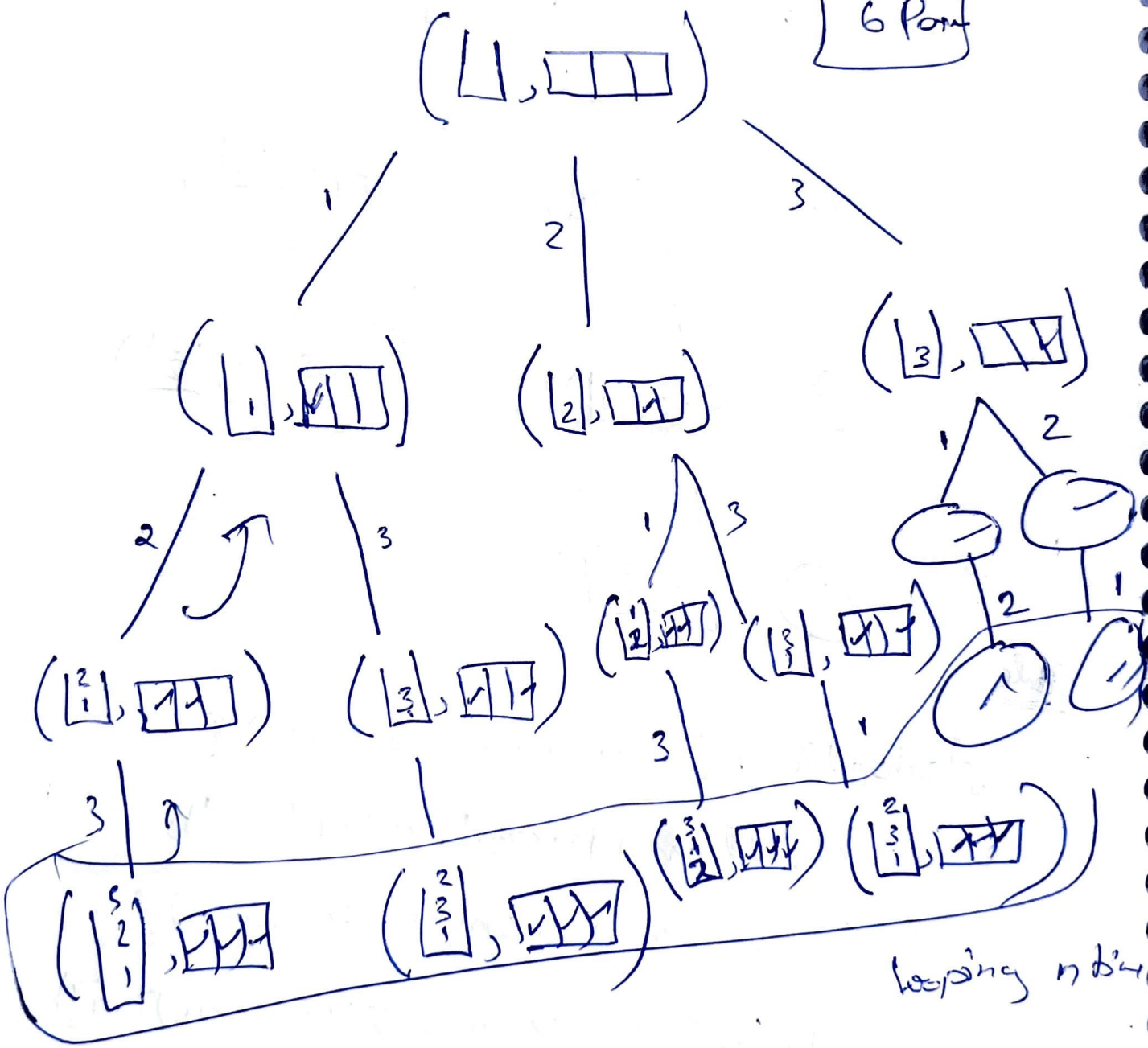
Permutation:

number of permutatio

Brute force [1, 2, 3]

n!

6 Perm



Time Complixity = n! x n

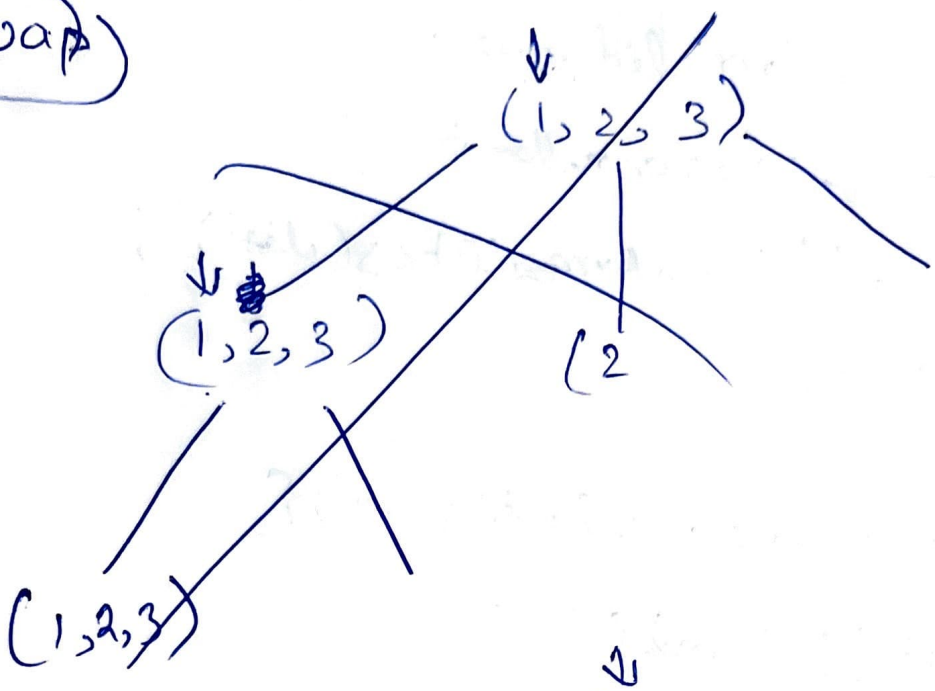
Space Complixity = O(n) + O(n)

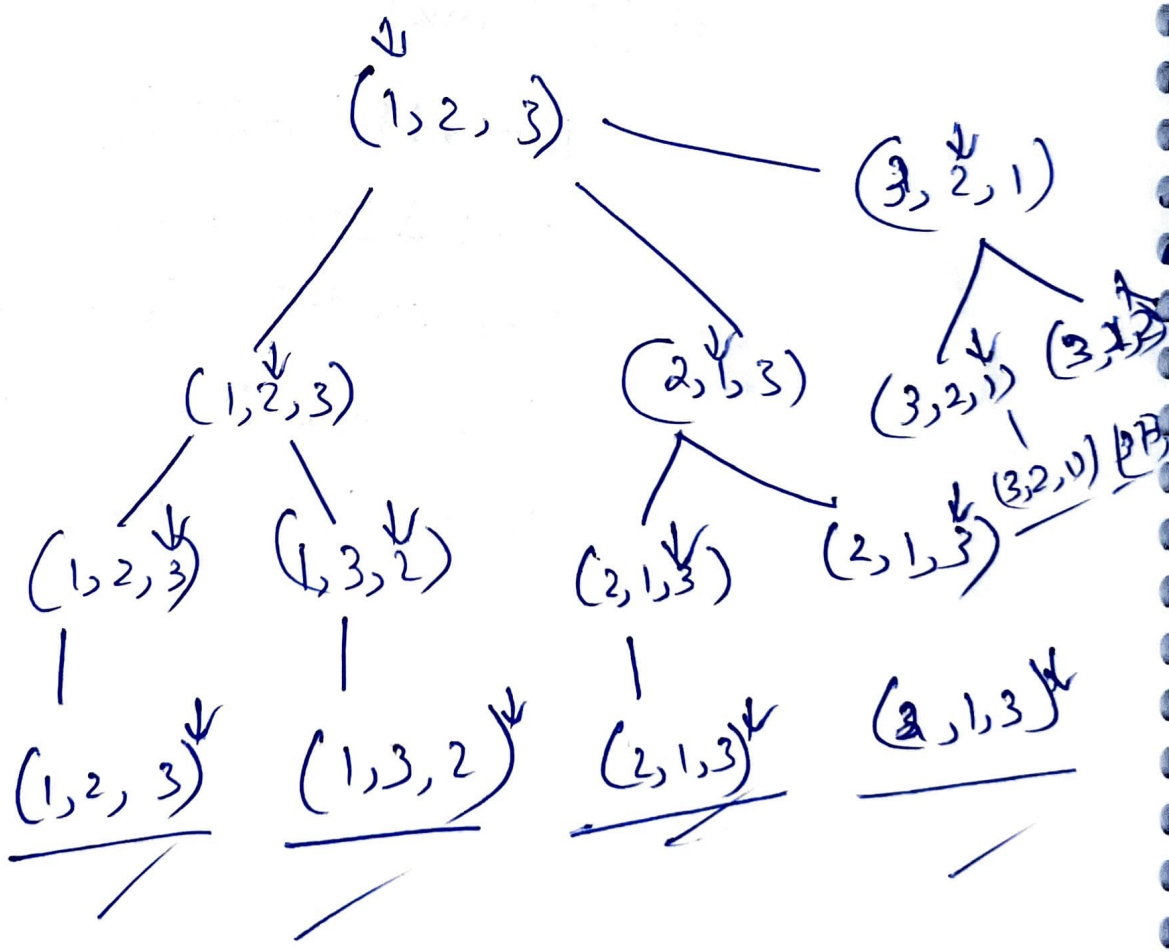
Permutation - Code - Approth (1)

```
getpermuation (arr, list, res, visited) {
    if (list.size() == arr.length) {
        res.add (new ArrayList<>(list));
        return;
    }
    for (int i=0; i<arr.length; i++) {
        if (!visited[i]) {
            list.add(arr[i]);
            visited[i] = true;
            getpermutation (arr, list, res, visited);
            list.remove (list.size() - 1);
            visited[i] = false;
        }
    }
}
```

~~Part~~ Permutation - II Approach (2).

Swap



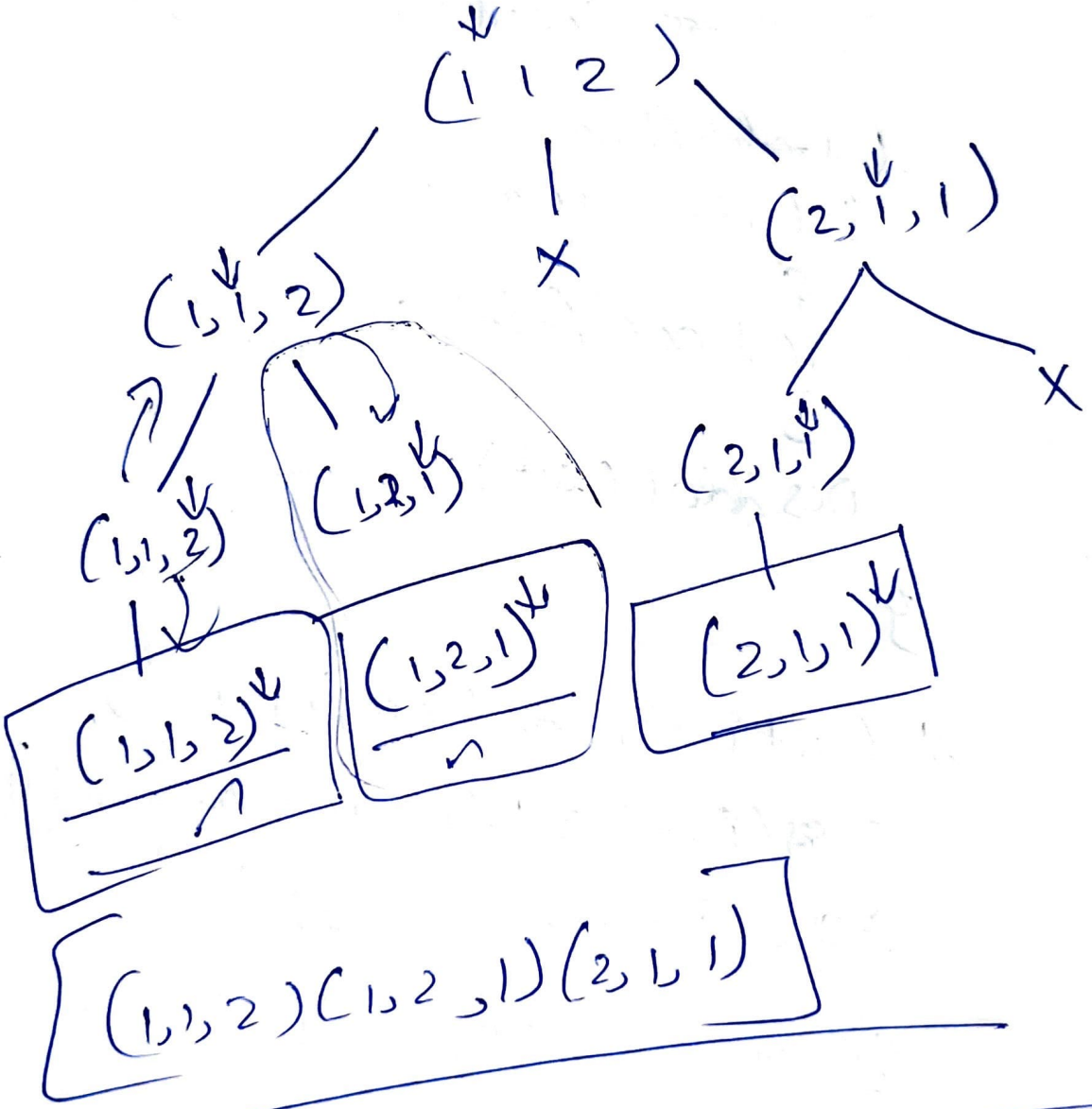


Permutation Combination – II

~~(1,1,2)~~ ~~(1,1,2)~~ arr = [1,2,1]

To remove duplication

(1,2,1) → sort



(1,1,2) (1,2,1) (2,1,1)

if (i > index && arr[i] == arr[i-1]) continue;

```
Array. sort(num);
get Combination (index , nums, res);

{ n = ~~length~~ 
  if (index >= nums.len) {
  List <Integer> list = new ArrayList<>();
  ~~list.add (Arra~~
  for(int ele : nums)
    {
    list.add (ele) ;
    }
  res.add (list);

  }

for(int i = index; i<n; i++ )
  { if (i > index && arr[i] == arr[i - 1]) continue;
  swap(index , i, nums) ;
  get Combination(index+1 , nums, res);

  swap (index, i , nums) ;

  }
}
```

N - Queens n = 4