SUBSET SUM - 1

arr = {5, 2, 1}

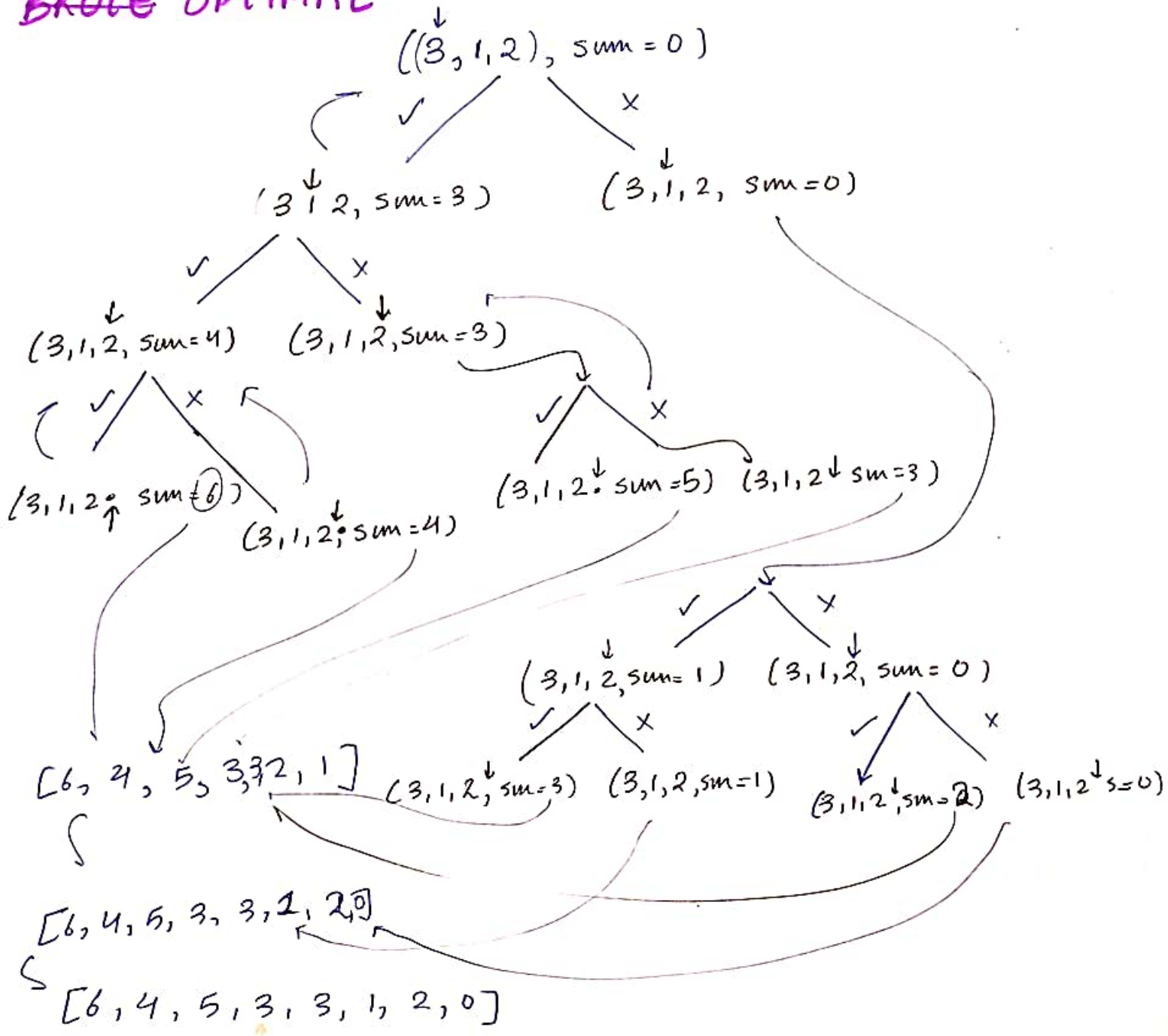
print all the sum of the subset generated from it, in increasing order.

arr = {5, 2, 1}

output = 0, 1, 2, 3, 5, 6, 7, 8

from → {[], [1], [2], [2, 1], [5, 1], [5, 2], [5, 2, 1]}

~~BRUTE~~ OPTIMAL



COMBINATION SUM - 2 ★ (try using Set data structure to store list and remove duplicate)

candidate = {10, 1, 2, 7, 6, 1, 5} target = 8

Each number in candidate may only be used once in combination

Solution must not contain duplicate combinations.

output: {

{1, 1, 6}

{1, 2, 5} ← now

{1, 7}

{2, 6}

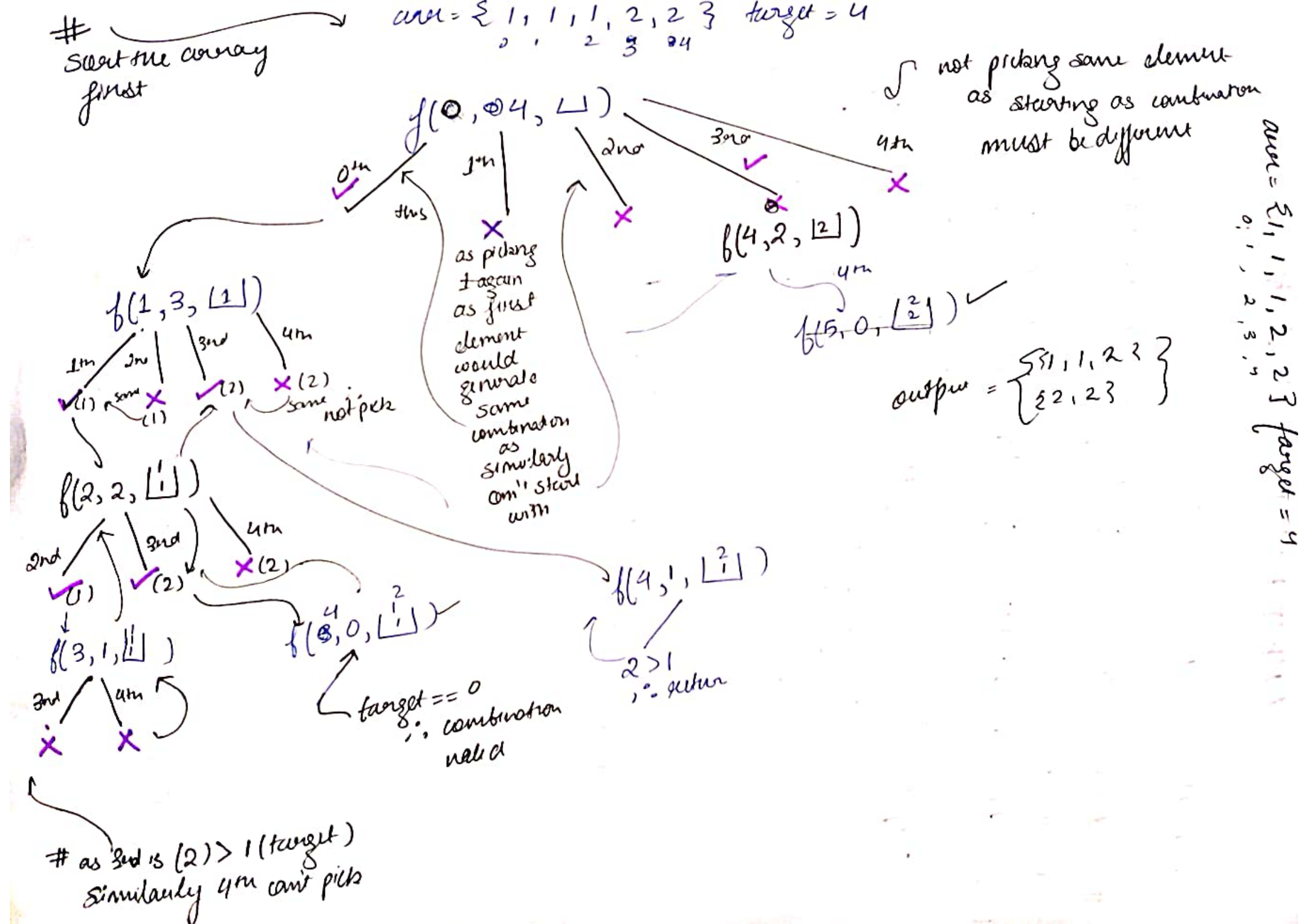
}

10, 1, 2, 7, 6, 1, 5

{1,2,5} but {1,2,5} / {2,1,5}

same combination but only one should be added

arr = {1, 1, 1, 2, 2} target = 4


arr = {1, 1, 1, 2, 2} target = 4
0 1 2 3 4
# sort the array first
f(0, 4, [])
0th
1st
2nd
3rd
4th
this
as picking 1 again as first element would generate same combination as similarly can't start with
not picking same element as starting as combination must be different
f(1, 3, [1])
1st
2nd
3rd
4th
(1)
same
(1)
(2)
same
not pick
f(4, 2, [2])
4th
f(5, 0, [2,2])
output = {{1, 1, 2}, {2, 3}}
f(2, 2, [1,1])
2nd
3rd
4th
(1)
(2)
(2)
f(3, 1, [1,1])
3rd
4th
f(4, 0, [1,1,2])
target == 0
∴ combination valid
f(4, 1, [1,2])
2 > 1
∴ return
as 3rd is (2) > 1 (target)
similarly 4th can't pick
arr = {1, 1, 1, 2, 2} target = 4
0 1 2 3 4

```
findCombination(ind, arr, target, ans, ds) {
    if(target == 0){
        ans.add(ds);
        return;
    }
    for(int i = ind; i < arr.length; i++) {
        if(i > ind && arr[i] == arr[i-1]) continue;
        if(arr[i] > target) break;
        ds.add(arr[i]);
        findCombination(i+1, arr, target - arr[i], ans, ds);
        ds.remove(ds.size() - 1);
    }
}
```

if element previous then check checking for duplicate

if value can't be picked then as array is sorted then no adv. in going forward.

next element

if looping to call for only distinct element

① 1, 1, ② 2

SUBSET - II

return all possible subsets (the power set).

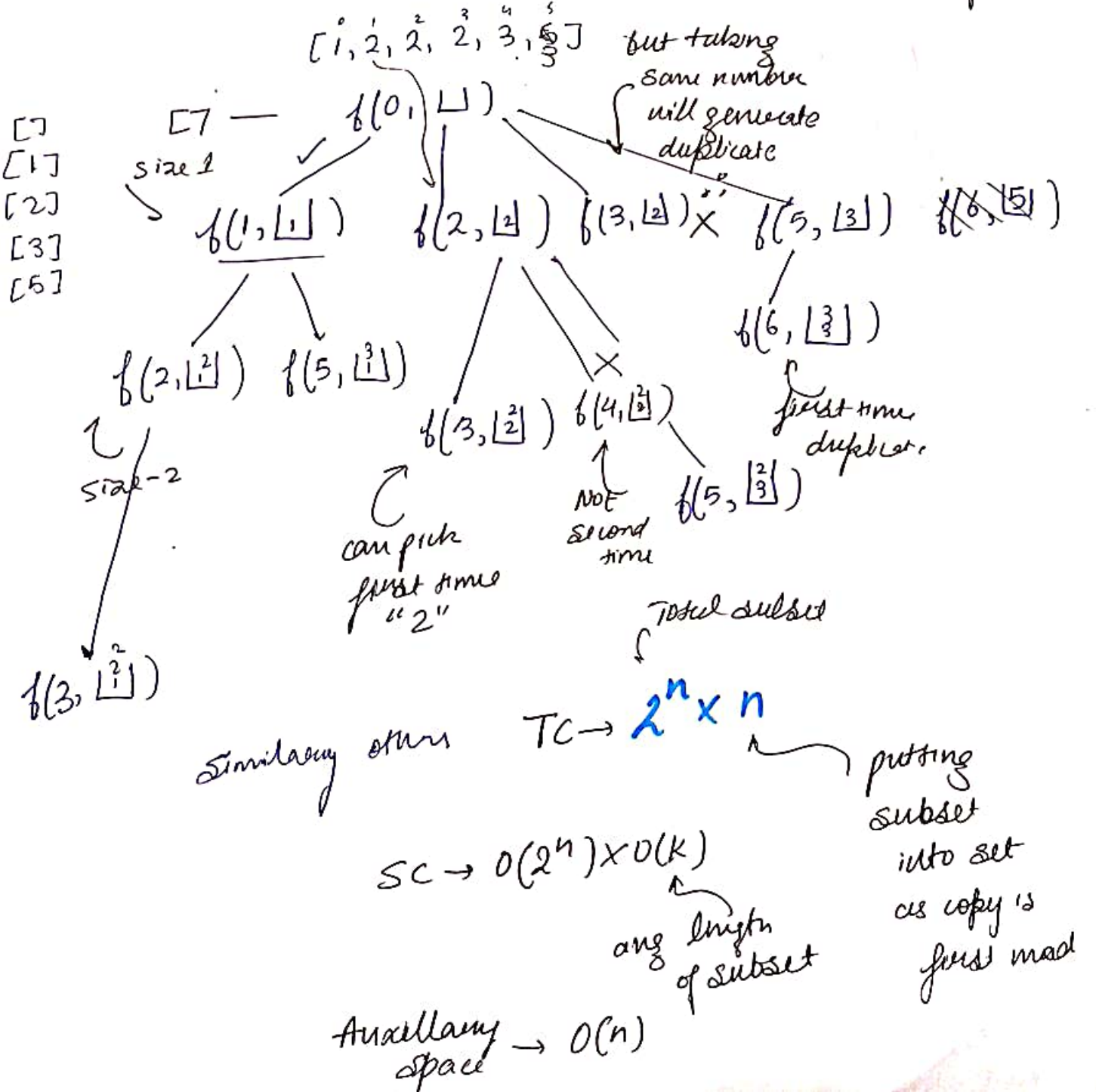
The solution set must not contain duplicate subsets

nums = {1, 2, 2}

output = {[], [1], [1,2], [1,2,2], [2], [2,2]}

now {1,2,2} → {1,2} but only one

#2,2 repeat will be picked first time not after that



Total subset

Similarly others TC → $2^n \times n$

putting subset into set as copy is first made

SC → $O(2^n) \times O(k)$

any length of subset

Auxillary space → $O(n)$

code

1D list ⊆ 2D list

```
void findSubsets(int ind, int[] nums, List ds, ans){

    ans.add(new ArrayList<>(ds));

    for(int i = ind; i < nums.length; i++){
        if(i != ind && nums[i] == nums[i-1]) continue;
```

(i != ind: when not at ind in first time (something to compare); nums[i] == nums[i-1]: whenever value equal)

```
        ds.add(nums[i]);
        findSubsets(i+1, nums, ds, ans);
        ds.remove(ds.size()-1);
    }
}
```

PALINDROME PARTITIONING

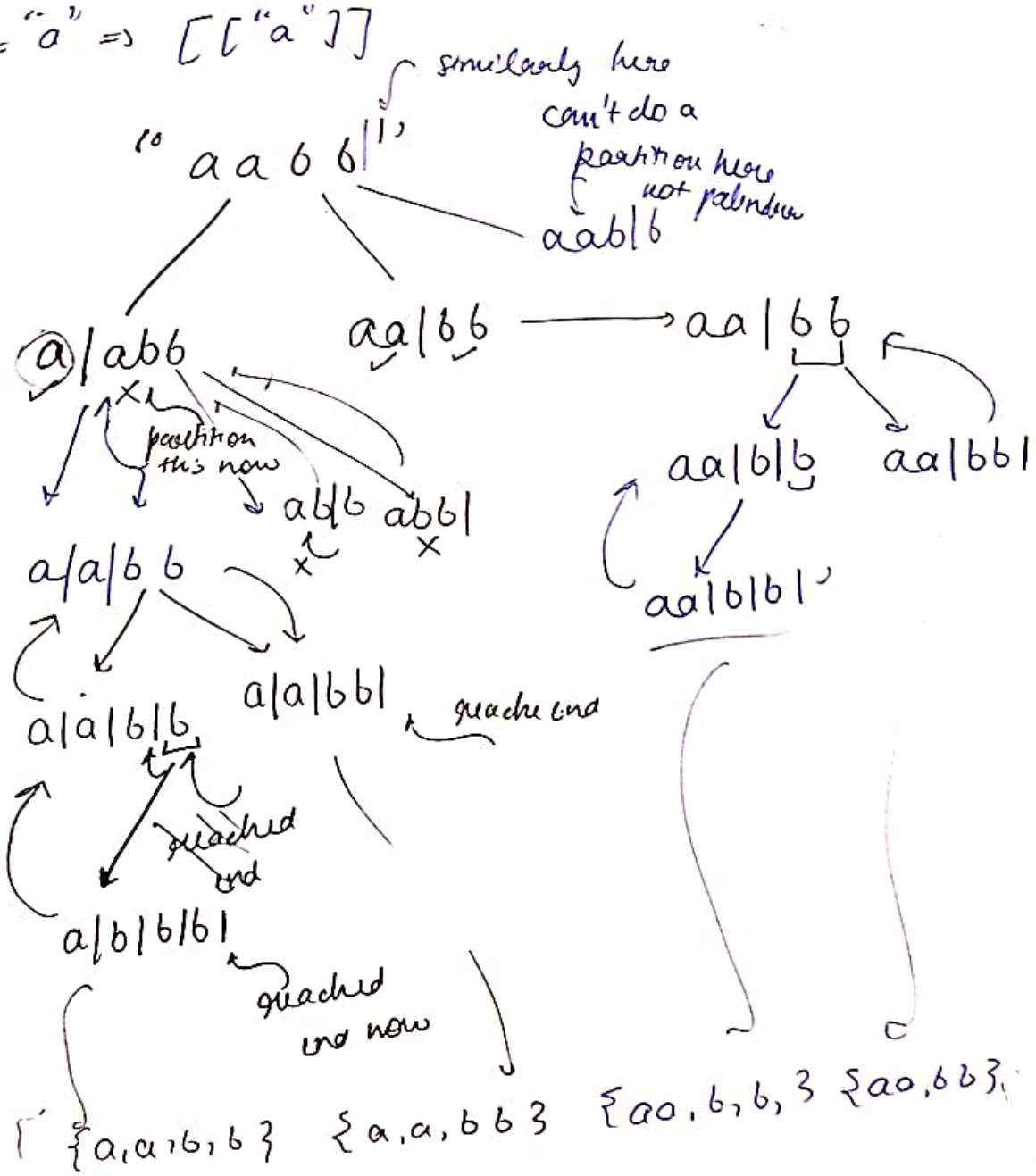
given a string "s", partition, s such that every substring of the partition is a palindrome.

Input = s = "aab"

a|a|b [['a','a','b'],

a a|b ["aa",'b']]

s = "a" => [["a"]]

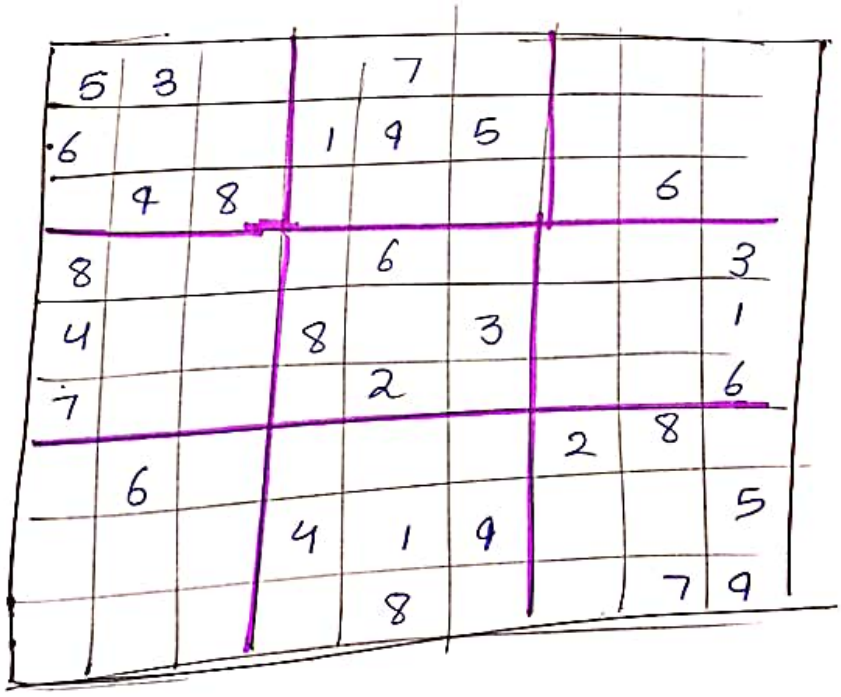


SUDOKU SOLVER

Each of digit 1-9 must occure exactly once in each a row

Each of digit 1-9 must occure exactly once in each column.

Each of digit 1-9 must occur exactly once in each of the 9 (3x3) sub boxes of grid.



```
boolean solveSudoku(char[][] board){
    for(0 ⟶ <9){
        for(0 ⟶ <9){
            if(board[i][j] == '.'){
                for(char c = '1'; c <= '9'; c++){
                    if(isValid(board, i, j, c)){
                        board[i][j] = c;
                        if(solveSudoku(board)) return true;
                        else board[i][j] = '.';
                    }
                }
                return false;
            }
        }
    }
    return true;
}
```

if true then no recursion calls are made the out of loop but only true. when return false

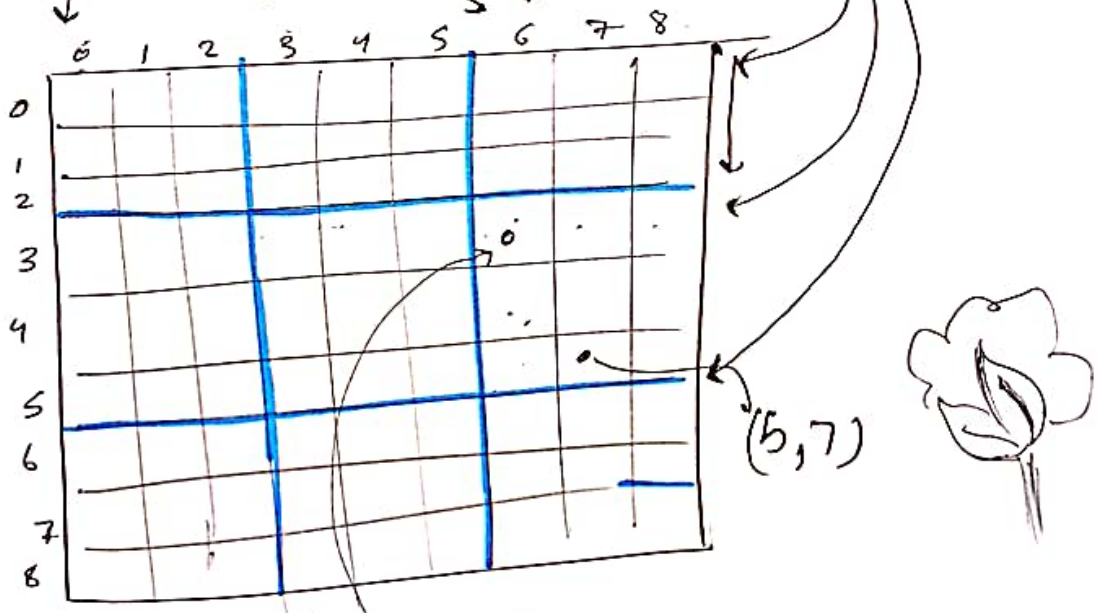
as 81 iteration is complete

```
boolean isValid ( board, row, col, char c ) {
    for (int i=0; i<9; i++) {
        if (board[i][col] == c) return false
        if (board [row][i] == c) return fals;
        if (board [3 x (row/3) + i/3 ] [3x(col/3) + i%3] == c)
            return false;
    }
    return true;
}
```

3x

this tell us which of 3 col it belong as start

this tell which horizontal row it belongs



Now $i = 0 \rightarrow 9$

$i = 0$

Now

$3 \times \left(\frac{5}{3}\right) + \frac{0}{3}$ & $3 \times \frac{7}{3} + \frac{0}{3}$

$3 \times 1 + 0$ $\quad$ $6 + 0$

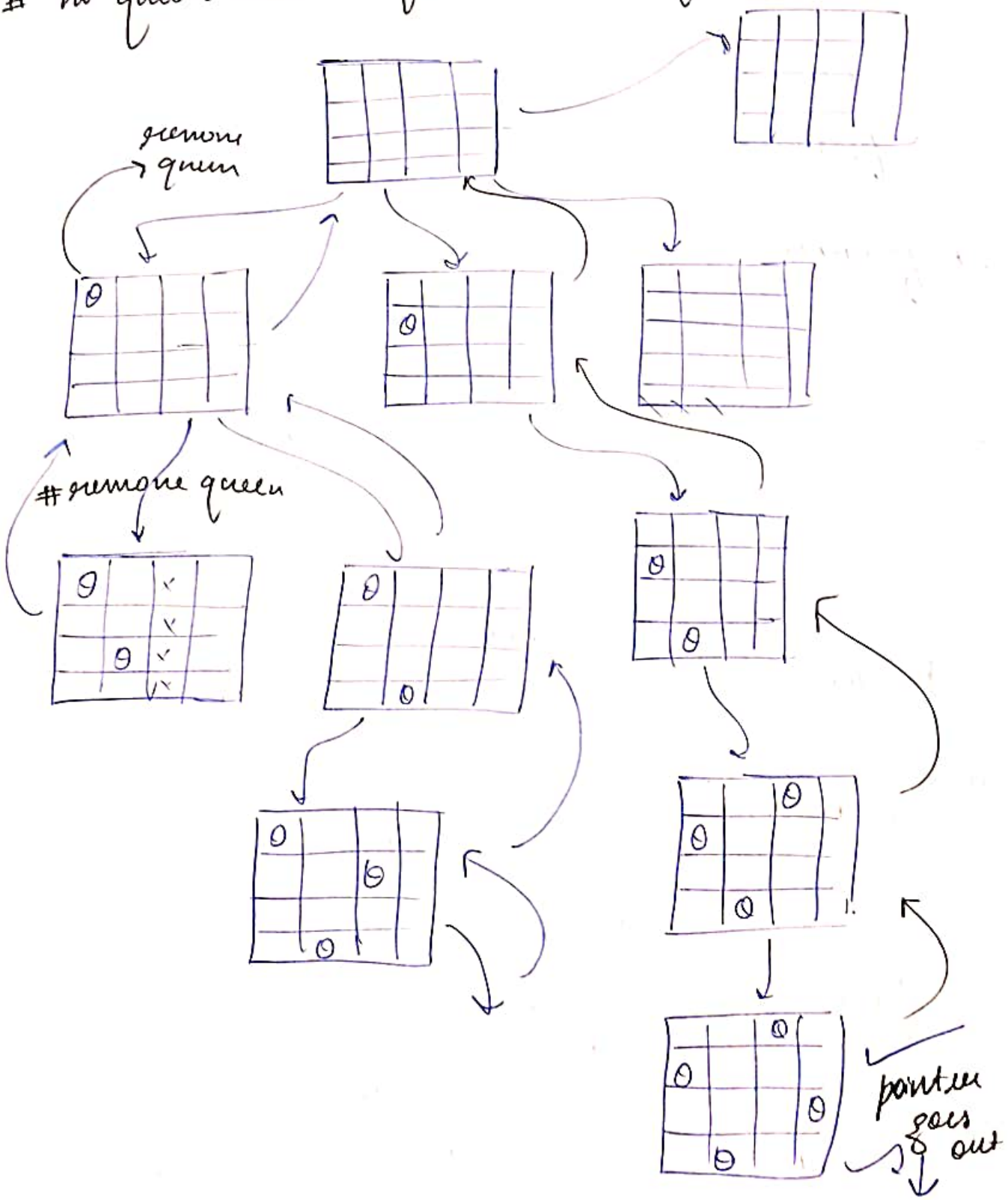
$3 + 0 = 3$ $\quad$ 6

(3,6)

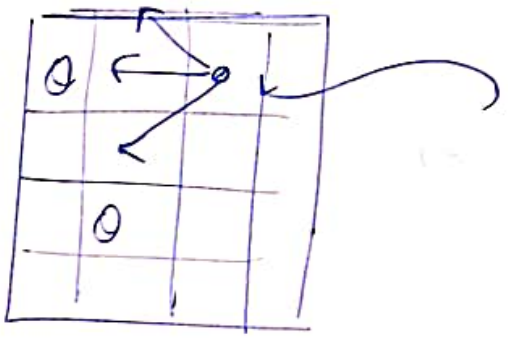
Similarly for other

N-QUEENS

N×N chessboard
every row should have 1 queen
every colum " " "
no queen should attack each other



To find if queen attacks two ways.

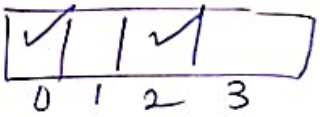


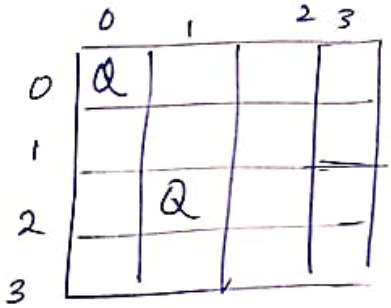
traverse in all three directions one by one.

Or

Storing

first for row





maintain array to store row position

for ↘ & ↙ diagonal

add row + col (index)

let

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	8
2	2	3	4	5	6	7	8	9
3	3	4	5	6	7	8	9	10
4	4	5	6	7	8	9	10	11
5	5	6	7	8	9	10	11	12
6	6	7	8	9	10	11	12	13
7	7	6	9	10	11	12	13	14

then mark row + col as occupied

array size $\frac{1}{2}(2n-2)$

if queen placed at this the mark

for ↖ diagonal

	0	1	2	3	4	5	6	7
0	7	8	9	10	11	12	13	14
1	6	7	8	9	10	11	12	13
2	5	6	7	8	9	10	11	12
3	4	5	6	7	8	9	10	11
4	3	4	5	6	7	8	9	10
5	2	3	4	5	6	7	8	9
6	1	2	3	4	5	6	7	8
7	0	1	2	3	4	5	6	7

similar for this out

$(n-1) + (col - row)$ formula for testing

0	- - - - - -	14

COMBINATION SUM

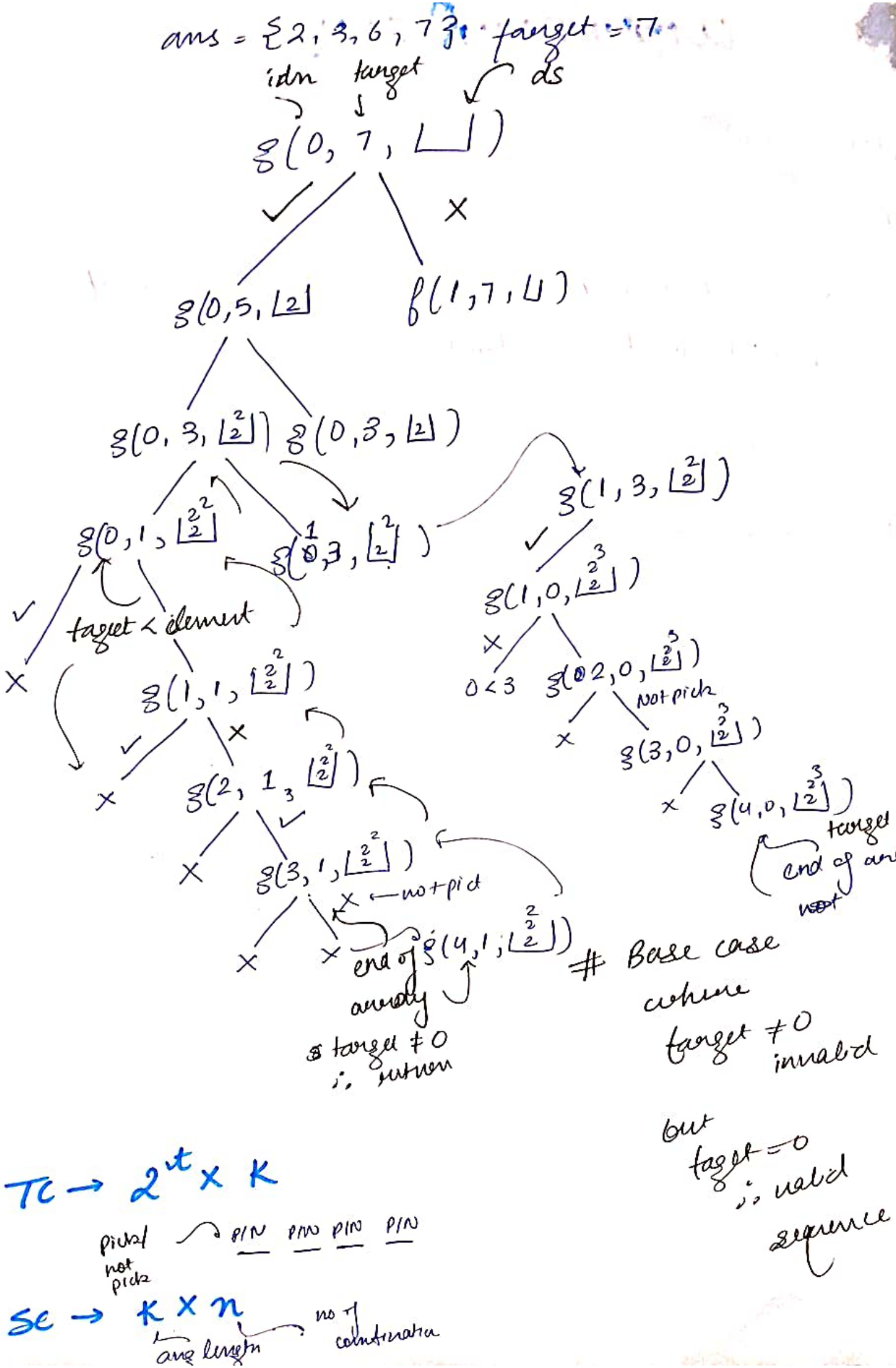
candidate = {2, 3, 6, 7} target = 7

output = { {2, 2, 3}, {7} }

The same number may be chosen from candidates are unlimited number of times.

{2, 3, 5} target = 8

output: {{2, 2, 2, 2}, {2, 3, 3}, {3, 5}}


ans = {2, 3, 6, 7}
target = 7
idx
target
ds
f(0, 7, [ ])
f(0,5,[2])
f(1,7,[])
f(0,3,[2])
f(1,3,[2])
f(1,0,[2])
f(2,0,[2])
f(3,0,[2])
f(4,0,[2])
target < element
not pick
target = 0
end of array
not
f(2,1,[2])
f(3,1,[2])
f(4,1,[2])
no+pick
end of array
target ≠ 0
∴ return
Base case
when
target ≠ 0
invalid
but
target = 0
∴ valid
sequence
TC → 2^t x K
pick/not pick
PICK PICK PICK PICK
SC → K x n
avg length
no of combinations

Pow(n, n)

n raised to power n

{ n = 2.000 n = 10

output = 1024

n = 2.1000 n = 3

output = 9.261e0 }

n^n

example

$2^{10} = (2 \times 2)^5 = (4)^5$ → 1024

$(4)^5 = (4) \times (4)^4$ → 4×256

$(4)^4 = (4 \times 4)^2 = (16)^2$

$(16)^2 = 16 \times (16^1) = (256)^1$

$(256)^1 = 256 \times (256)^0$ ($=1$)

(n%.2 = 0) → (n x n) and (n/2)

(n%.2 = 1) → (ans = ans x n) and n = n - 1

Tc → log(n)

code:

```
power ( double x, int n ) {
    double ans = 1;
    long nn = n;
    if (nn<0) nn = -1 * nn;
    while (nn>0) {
        if (nn % 2 == 1) {
            ans = ans * x;
            nn = nn - 1;
        }
        else {
            x = x * x;
            nn = nn / 2;
        }
    }
    if (n<0) ans = (double) (1.0) / (double) (ans);
    return ans;
    }
}
```

making power +ve

yaha make it long if directly done (-1 x n) this will still be integer not ∴ will exceed the range and overlap around.

Recursive

```
long power ( long base, long e , long ans ) {
    if (e == 0) return ans;
    if ((e % 2) == 1)) return power (base, e-1, (ans * base))
    else return power ((base * base), e/2 , ans);
}
```