Qus. Subset sum

→ given array, print sum of all subsets in increasing order

Approach 1 Brute force approach

① use power set, takes $2^N \times N$ time to take all subsets

Approach 2. Recursive approach. $TC = 2^n + 2^n(\log 2^n)$ sorting

→ solution will follow DFS to find all the subsets

149. Subset ~~sum~~ 2

→ return all possible subsets of array that may contain duplicates.

→ solution must not contain duplicate subset

Approach 1. Brute force

use pick and non pick technique.

→ put all subsets in set and convert the set into list

→ set insertion takes time, need optimization

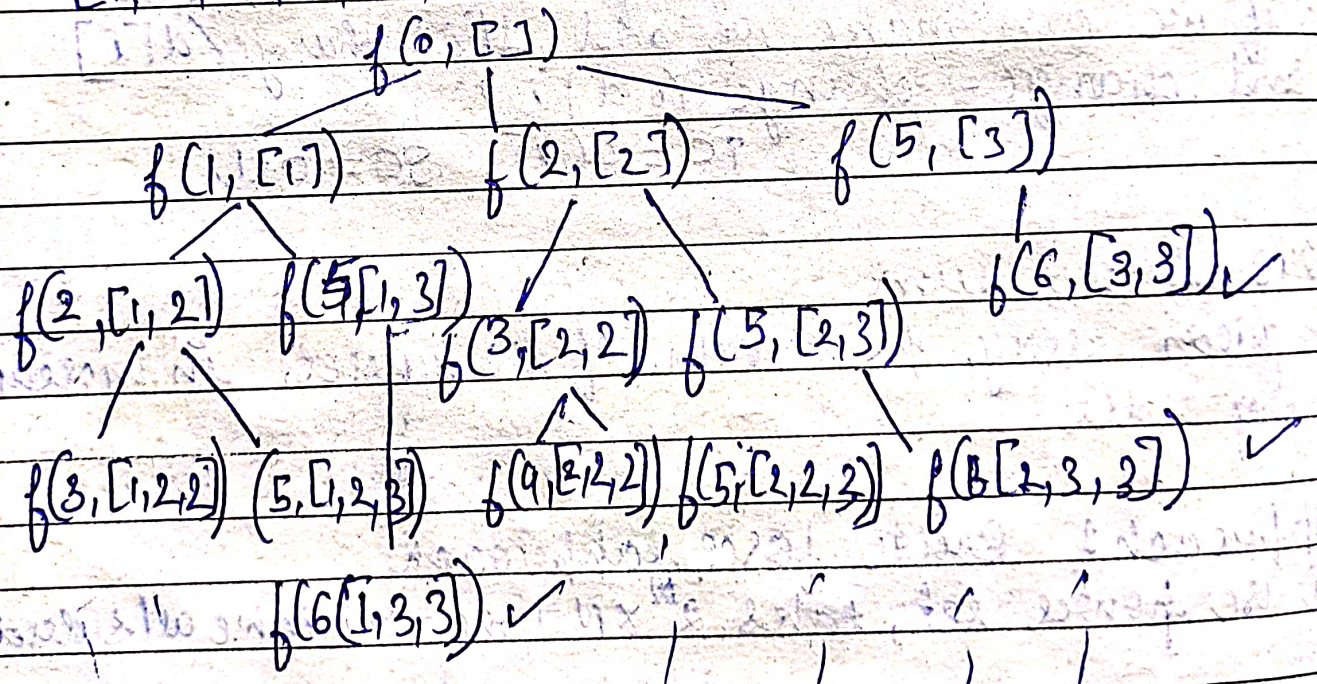
Approach 2 Recursive solution.

→ at every step it generates list

→ carry an empty list initially

→ dont take duplicates.

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



~~Pseudo code~~

~~f (ind, ds)~~

TC → $2^n \times n$

SC → $O(2^n) \times O(TC)$

Aux. SC → $O(n)$

Date Page

Q50 Combination sum 1

→ given array of distinct integer candidates
→ target integer (sum) -
→ return list of all unique combinations of candidates where sum of candidates = target
→ pick any no. of times

$TC \rightarrow 2^t \times K$
$SC \rightarrow K \times x$

Approach: recursion
for every int. we will call recursive f^n.
→ any sum will only be included if it's lesser or equal to target
→ if it's not we move index ahead
→ one conditions satisfies that index = size of array and target = 0, we will push our created data structure to the result 2D vector.
→ make sure to pop index from data structure if one were done.

Q51. combination sum 2

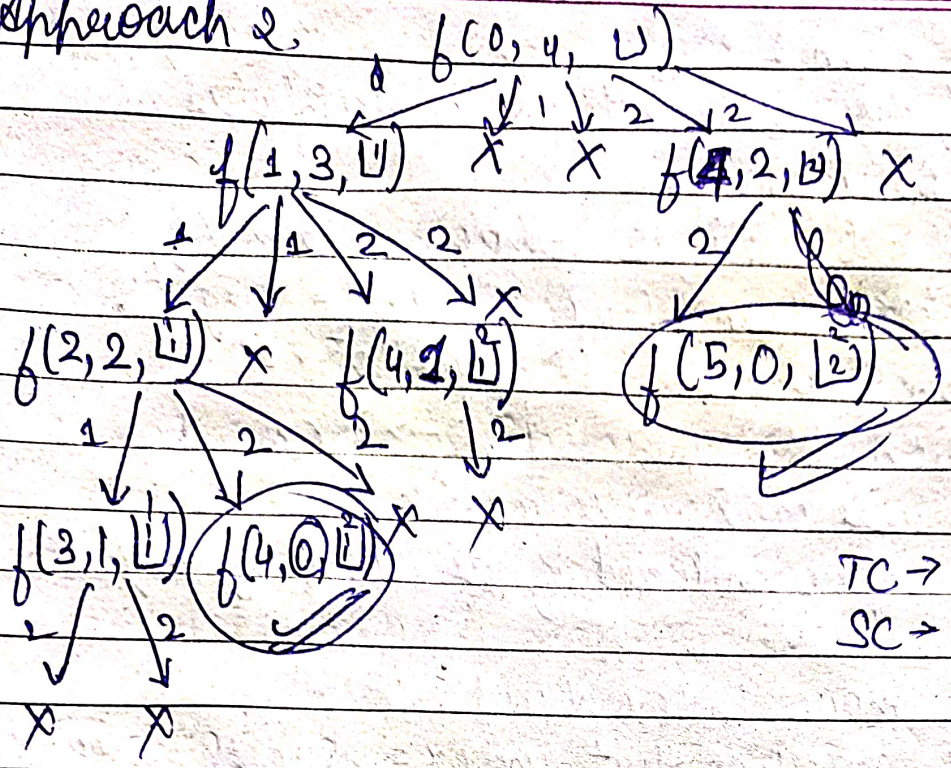
→ given candidate no, as well as target no.
→ find subset with sum = target
→ each no can be used only once.
→ no duplicate combinations (sorted)

Approach 1. Brute force (Q50. combination sum)
change, we don't remain at same index even after were included it to the subsequence.
→ it'll not include distinct values, thus we can make use of hash set to avoid duplicates.

$TC: (2^n) * K \log(\text{size of set})$

Date: Page:

Approach 2.

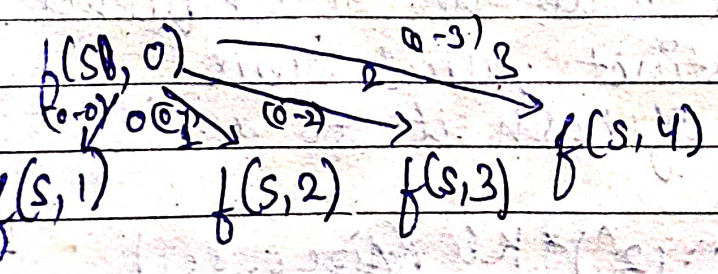


TC → $2^n \times K$.
SC → $K x$.

→ skip repeating elements after sorting
→ add element to ds if the value of element is lesser than target
→ else if greater, we break for loop.
→ make sure to pop values from ds.

153. Palindrome partitioning
→ split string s such that every part is palindrom
→ lower limit, thus brute force works

Approach 1. Brute force
→ find the position that could make all substrings at left palindrome.
→ do it for any call untill final position isn't reach.



looping.

general → f(S, index)
↓ loop → (ind+1)
(ind ... i) is pal
then partition

Q54. k^{th} permutation sequence.

→ find k^{th} permutation of any set with n numbers 1 - n.

Approach 1. Brute force

→ Recursive way for generating all sequence

$TC = O(n! \times n) + (n! \log(n!))$

Approach 2

$n = 4 \Rightarrow$ no. of permutations $= 4! = 24$.

→ divide problems into subproblems.

if start with 1 → 1 + (2, 3, 4) 6 perm (0 - 5)

" 2 → 2 + (1, 3, 4) 6 " (6 - 11)

" 3 → 3 + (1, 2, 4) 6 " (12 - 17)

4 → 4 + (1, 2, 3) 6 " (18 - 23)

24

we need to find 17th permutation

16th index

16 lies in 12 - 17, thus first number is 3.

$\frac{16}{6} = 2$ index → 3

$16 \% 6 = 4^{th}$ seq. in (1, 2, 4) → subprob.

general form.

$k = k - 1$ → 0 indexing

first time

set.insert(a[k/(n-1)!])

$k = k \% (n-1)!$

3 4 1 2.

TC → $O(N) \times O(N)$ ← crase function

$= O(N^2)$

SC → $O(N)$

Q 55.

~~Say~~ Print all permutation of string / array

Approach 1. generating all permutation through recursion (map)

→ at any recursion step, mark index of number inserted in ds, that resembles a permutation

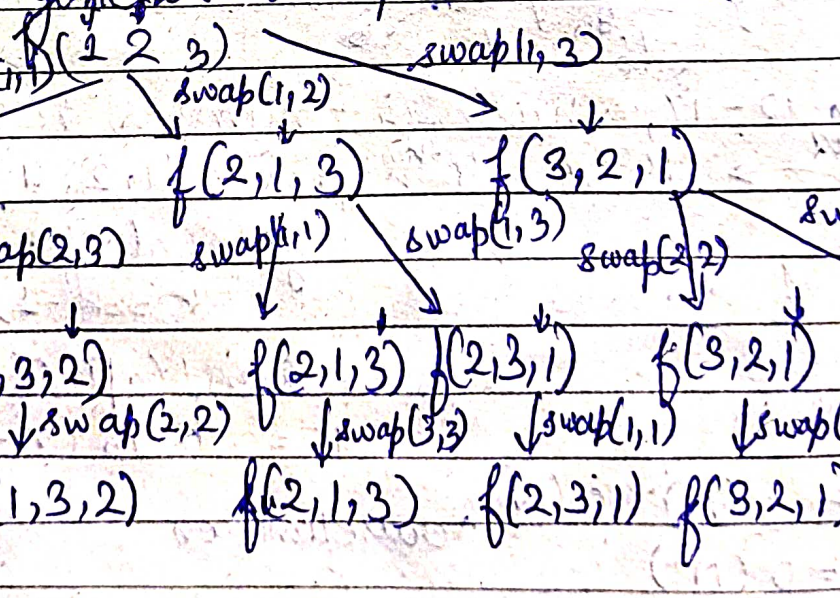
→ ~~mak~~ insert ds in answer as soon as all elements are inserted in DS

→ make sure to unmark the element that you've poped out of ds. as you backtrack

TC → $O(n! \times n)$ SC → $O(n) + O(n)$

Approach 2. (no map X)

→ swapping to generate all permutations



→ creating permutations by swapping all elements to the right with current one by one

→ pushing the permutation when index comes out of index.

→ make sure to swap same again for backtracking

TC → $O(n! \times n)$ SC → $O(1)$ *no extra space

Q56. N queen Problem.

→ given a chess board (n×n) and we have ~~to place n queens such that no queen~~ kills each other.

→ every row = 1 queen, every column = 1 queen ~~every diagonal = 1~~

→ use Backtracking technique.

* while you go back, remove queen if solution is completed coming its conclusion.

```
f(col)
{
    for(i = 0 → n-1)
      if (fill → v)
        row col → "Q"
        f(col+1)
        row col → "."
```

Q57 Sudoku solver

→ given sudoku puzzle.

→ follow the rules of sudoku and generate all solution.

3 rules → ① digit 1-9 should appear exactly once in row
② digit 1-9 should appear exactly once in column
③ " exactly once in 3x3 matrix

→ try every possible ways.

→ find empty place and find which digit can be placed.

→ return false for call that can't fill position.

```
for(i = 1 - 9){
    if (possible){
        f(next empty)
```

Q58 M colouring

→ given undirected graph and ~~integer M~~.

~~the task is to determining if graph can be coloured with at most M colours~~

~~Afterwards~~

Q59. Rat in a maze.

→ rat has to reach from (0,0) to (n-1, n-1)

→ it can move up, down, left or right

→ it can only move to a cell if value is 0.

→ no cell can be visited once.

→ print the path

→ ways should be printed in lexicographically order

Approach

→ try each and every direction possible

order ⇒ DLRU

f(0, 0, "")

D

f(1, 0, "D")

D

f(2, 0, "DD")

R

f(2, 1, "DDR")

D

f(3, 2, "DDRD")

R

f(3, 2, "DDRDR")

↓

f(3, 3, "DDRDRR")

in the similar way all recursive calls will be made.

→ make sure to mark cell visited