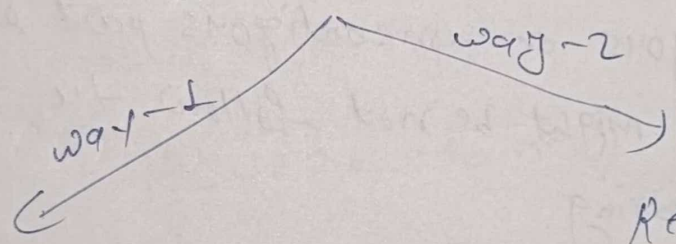
Ques: Subset Sum Equal to K

⇒ first thing that comes in our mind Generate all subsequences & check if any of them gives a sum equal to K or not.

How to generate all subsequences



Power Set
Bit manipulation

Recursion
include/exclude
method

But in this question we don't need to generate all the subsequences rather then we have to find a single subset when we found then ok stop.

Instead of generating all the subsequences we keep a check if there is at least one subsequence ~~then~~ whose sum equal to K then ok.

Recursion Method

By the recursion method we can generate subsequence & when we get that one subsequence/subset we stop.

Rules for writing Recurance

(1) Express every thing in term of indep & write the base case & check what other parameters are required

⇒ Here for every index we take care of target that we looking for

(idx, target)

In mostly dp on subsequences/subsets que. we always try to represent in (idx, target)

(2) Explore possibilites of that index

Their are two possibilities

include — arr[idx] part of subsequence/subset

exclude — arr[idx] is not the part of subsequence

(3) Return ans i.e. T/F acc to the question

Now questions arises how do I start with idx & how do I start with target.

(idx, target)

Now always remember what I'm looking for

↳ I'm looking for entire array & if their exist a subsequence with the target what target given to you.

f(n-1, target) ⟹ That means in the entire array till the index (n-1) does the exist a target.

ex n = 4 array → [1, 2, 3, 4] (0 1 2 3), target = 4

f(3, 4) ⟹ we are looking for in the given array till the index 3 their exist a subsequence with target = 4

Base Case)

① f(idx, target)

{

//Base cases → might happen we achived the target
→ we start from (n-1) & go till 0 suppose at 0-th idx required target = 4
if(arr[0] == 4) → true else false.

// Target achived
if(target == 0) return true;

// 0-th idx if target achived or not

if (idx == 0) return (arr[0] == target);

// Explore the possibilites of that idx.

bool nottake = f(idx-1, target)

↳ this means from 0 to idx-1 their exist any subsequence with the given target

bool take = false; // Initially false bcz if arr[idx] > target

ex: arr = [1, 3, 6] target = 2

if (target >= arr[idx])

{ take = f(idx-1, target - arr[idx]) take = arr[2] ⇒ 6

6 > 2

then we not achive target further

}

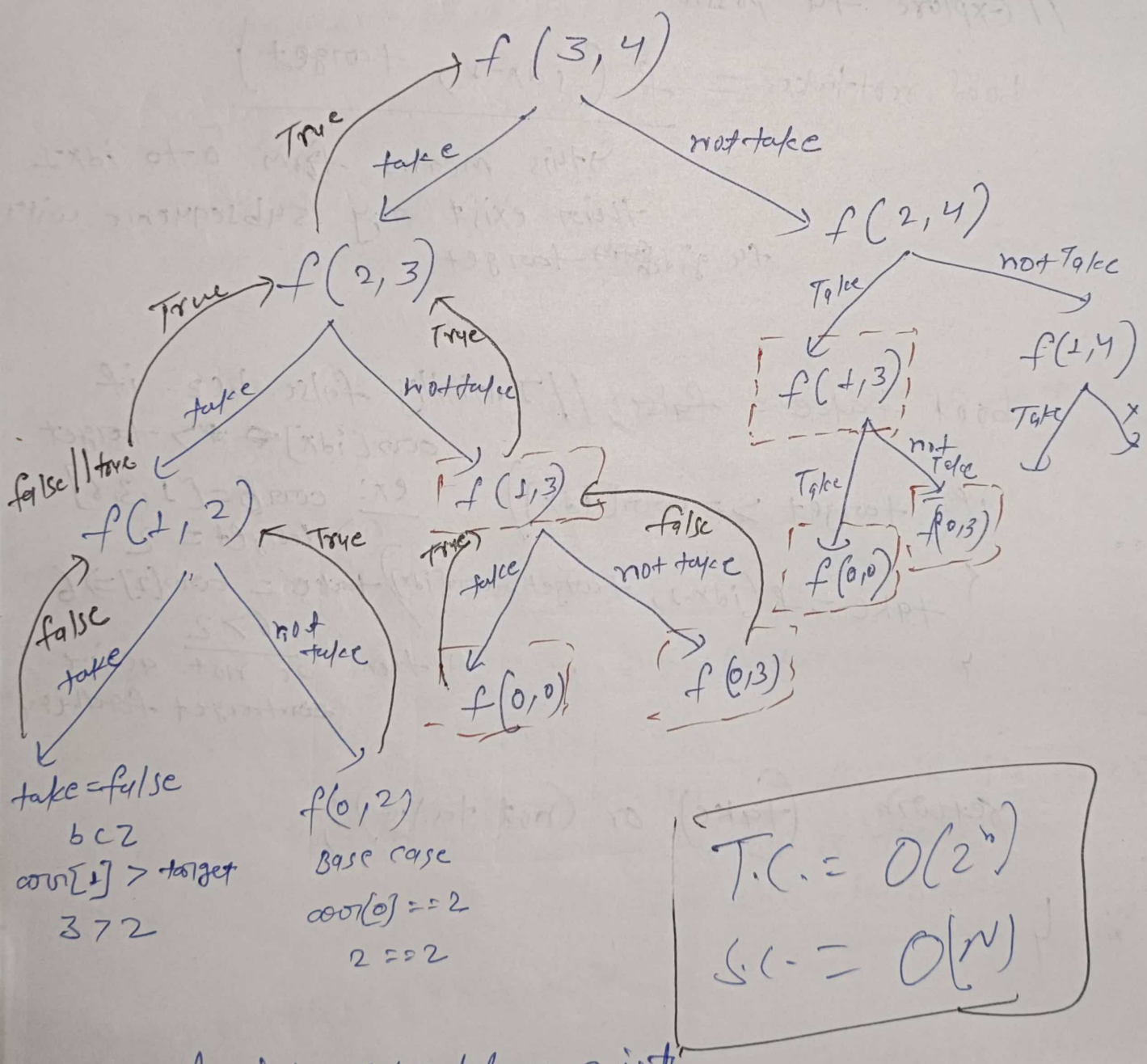
return (take) or (not take);

}

Recursion Tree

arr → [2, 3, 1, 1]
0 1 2 3

target = 4



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

S.C. = $O(N)$

Overlaping subproblems exist

Memoization Approach

Steps to convert a Recursive solution to a memoization

(1) first check what are the changing states in recursive solution

here → idx & target two changing states

(2) Choose a data structure acc to changing states.

Data structure → 2D Matrix bcz two changing states

(3) choose the size of DS acc to the range of changing states.

(4) Declare a DP DS.

(5) Store the result of subproblem in this DP DS

(6) If any subproblem result is previously stored then directly return it.

Tabulation Approach

Steps

(1) Declear a DP data structur.

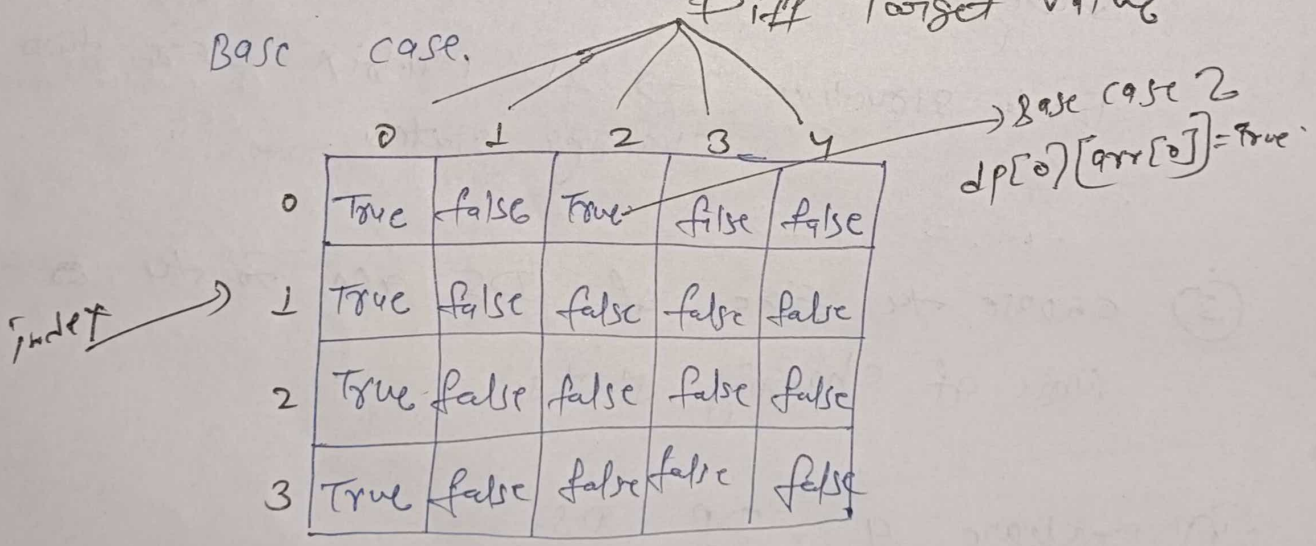
vector<vector<bool>> dp(n, vector<bool>(sum+1, false));

(2) Analyse the Base cases.

here 1st Base case ⟹

if(target == 0) return true;

Now fill the DP matrix acc to this Base case.



for(i=0 to n-1) dp[i][0] = true

2nd Base case → if(idx == 0)
return (arr[0] == target)

dp[0][arr[0]] = true;

(3) form Nested loop

Their are two nested loop.
idx & target

In Recurance index will go from (n-1) to 0

i.e. (n-1) → (n-2) → (n-3) - - - 0

so In tabulation index will go from
0 to (n-1)

0 → 1 → 2 → - - - (n-1)

also
target → (target - arr[idx]) → - - - →

Move opposite

In Tabulation always move opposite
from memoization

for(idx=1 to n-1)
{ for(target=1 to k)
{ copy paste the recurance