

Recursion

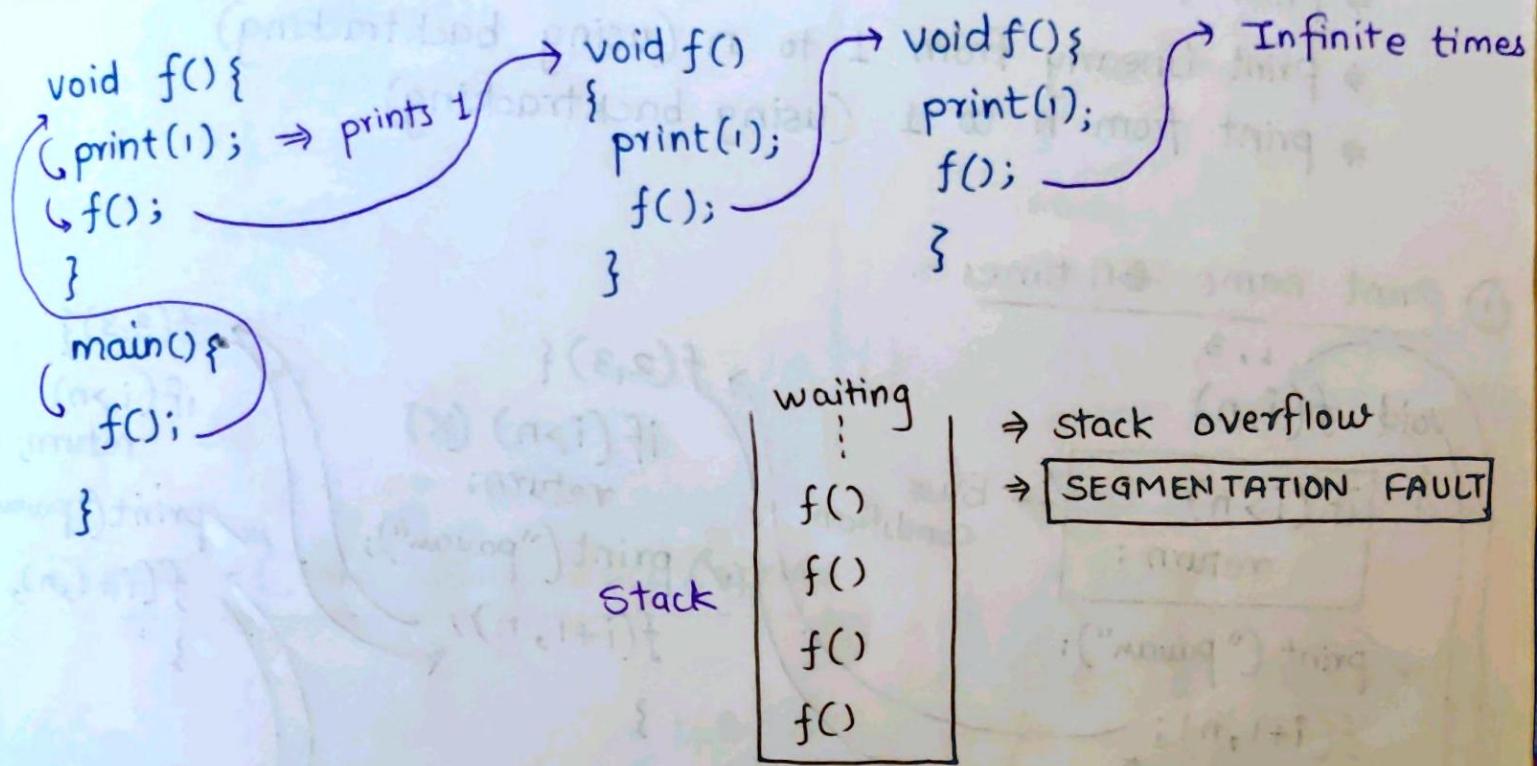
by

Striver

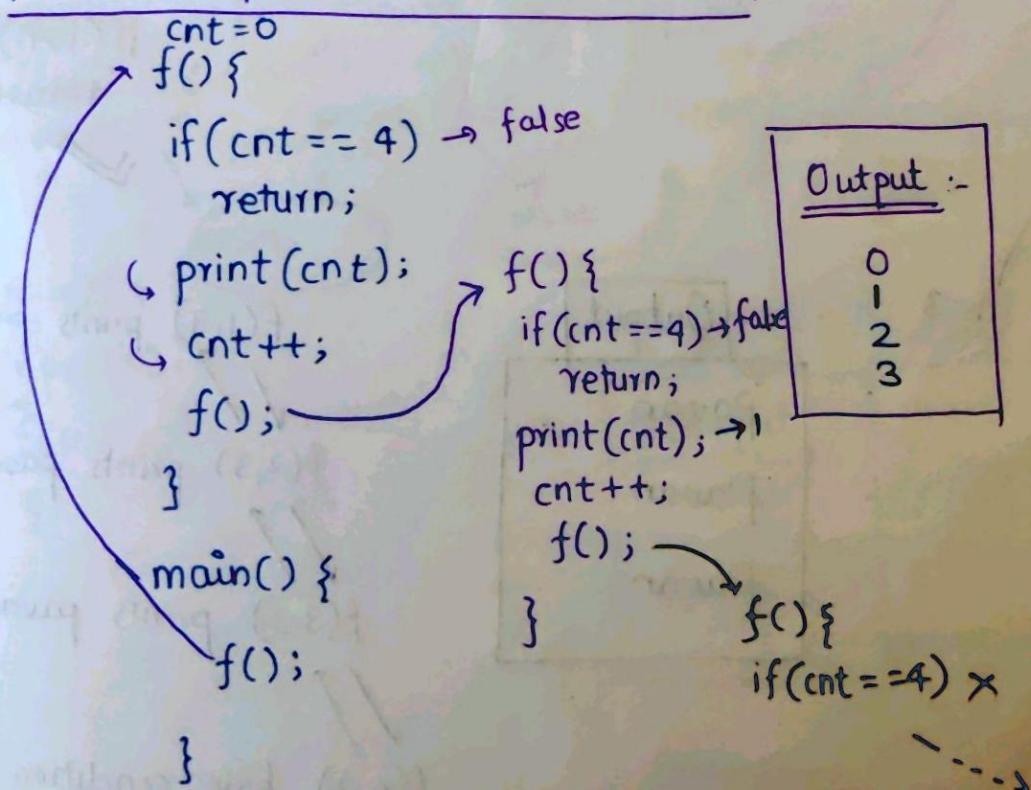
Recursion :-

When a function calls itself until a specified condition is met.

Example :-



Recursive function with base case:-



Basic Recursion Problems :-

- ⇒ Print Name 5 times
- ⇒ Print Linearly from 1 to n.
- ⇒ Print from n to 1.
- ⇒ print linearly from 1 to n (using backtracking)
- ⇒ print from n to 1 (using backtracking)

① Print name n times :-

```
void f(i,n)
{
    if(i>n)
        return;
    print("pavan");
    f(i+1,n);
}
```

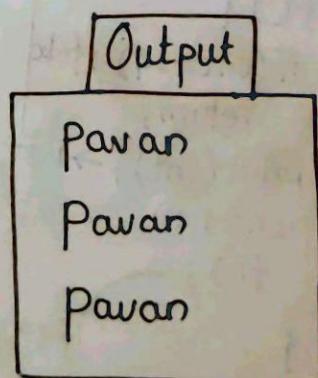
```
main()
{
    int n;
    cin >> n;
    f(1,n);
}
```

```
f(2,3) {
    if(i>n) (X)
    return;
    ✓ print("pavan");
    f(i+1,n);
}
```

```
f(3,3) {
    if(i>n) (X)
    return;
    ✓ print("pavan");
    f(i+1,n);
}
```

```
f(4,3) {
    if(i>n) (X)
    return;
    ✓ }
```

n=3



f(1,3) prints pawan
f(2,3) prints pawan
f(3,3) prints pawan
✓ f(4,3) base condition true
return;

② print 1 to n :-

```
f(i,n)
{ if(i>n)
    return;
    print(i);
    f(i+1,n);
}
main()
{
    input(n);
    f(1,n);
}
```

③ print n to 1

```
f(i,n)
{ if(i<1)
    return;
    print(i);
    f(i-1,n);
}
main()
{
    input(n);
    f(n,n);
}
```

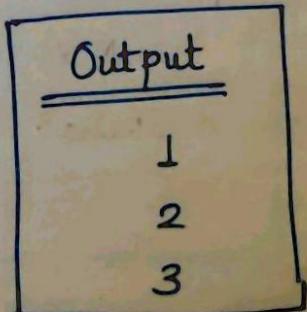
④ print 1 to n using backtracking :-

```
f(i,n)
{ if(i<1)
    return;
    f(i-1,n);
    print(i);
}
main()
{
    input(n);
    f(n,n);
}
```

```
f(0,3)
{ if(i<1)
    return;
    f(i-1,n);
    print(i);
}
prints "1"
```

```
f(1,3)
{ if(i<1)
    return;
    f(i-1,n);
    print(i);
}
prints "2"
```

```
f(2,3)
{ if(i<1)
    return;
    f(i-1,n);
    print(i);
}
prints "3"
```



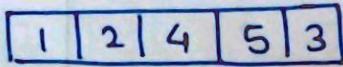
Reverse the array using recursion

① Method - I

```
f(l,r){  
    if(l>=r) return;  
    swap(arr[l],arr[r])  
    f(l+1,r-1);  
}
```

→ It will swap until ($l >= r$) this condition will get true.

```
main(){  
    input(arr);  
    f(0,n-1);  
}
```



② Method - II

```
f(i){  
    if(i >= n/2)  
        return;  
    swap(arr[i], arr[n-i-1]);  
    f(i+1);  
}
```

```
main(){  
    input(arr);  
    f(0);  
}
```

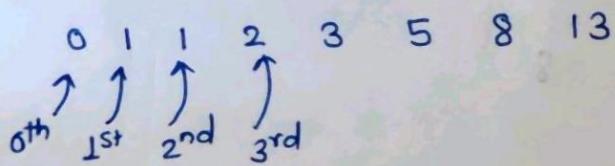
check if a string is palindrome

"MADAM" reverse "MADAM" \Rightarrow This is palindrome

```
f(i) {  
    if(i >= n/2)  
        return true;  
  
    if(s[i] != s[n-i-1])  
        return false;  
  
    return f(i+1);  
}
```

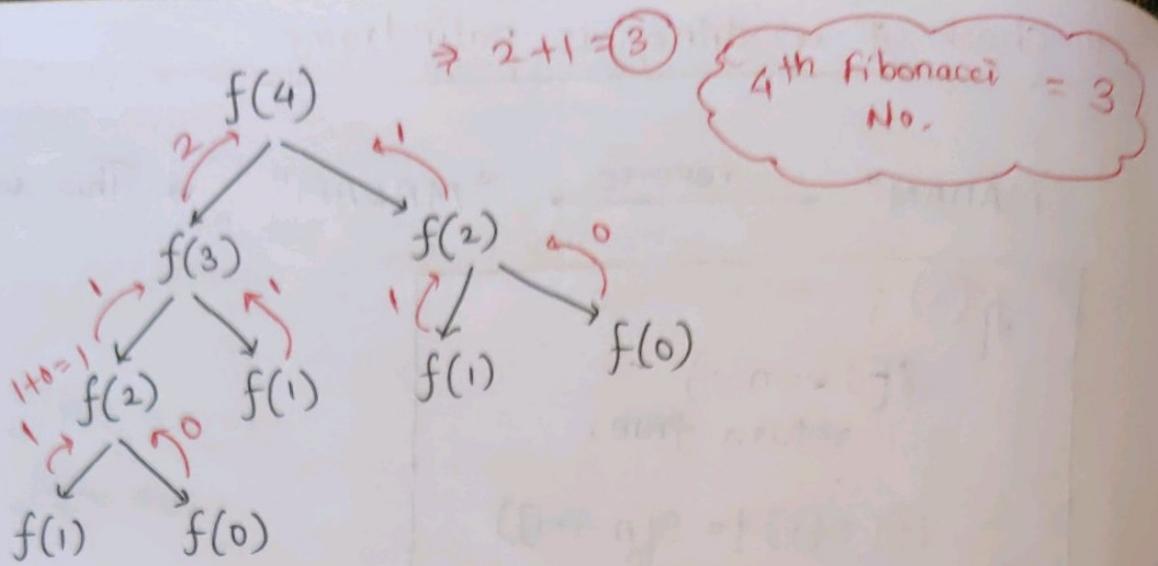
Multiple Recursion calls

① Fibonacci Number



$f(N)$ \longrightarrow N^{th} Fibonacci Number

```
f(n) {  
    if(n <= 1)  
        return n;  
  
    return f(n-1) + f(n-2);  
}
```



Print all Subsequence :-

A contiguous / non-contiguous sequence which follows the order

$$\text{arr} = [3, 1, 2]$$

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

#.Subsequence = 8

$$\underline{\#.\text{Subsequence} = 2^n}$$

arr =

| | | |
|---|---|---|
| 3 | 1 | 2 |
|---|---|---|

for [3, 2]

| | | | |
|---|------|---|------|
| i | Take | X | Take |
| | 0 | 1 | 2 |

Take | Non-take

↓
on indices

[1, 2]

| | | |
|---|---|---|
| X | ✓ | ✓ |
|---|---|---|

[3, 1]

| | | |
|---|---|---|
| ✓ | ✓ | X |
|---|---|---|

[3, 1, 2]

| | | |
|---|---|---|
| ✓ | ✓ | ✓ |
|---|---|---|

{ }

| | | |
|---|---|---|
| X | X | X |
|---|---|---|

arr

| | | |
|---|---|---|
| 3 | 1 | 2 |
| 0 | 1 | 2 |

N. Imp

f(ind, [])

{
 if (ind >= n)
 print([]);
 return;

[].push-back(arr[]);

f(ind+1, []); → Take

[].^{pop_back}remove(arr[i]);

f(ind+1, []); → Not-take

}

main() {

arr = {3, 1, 2}

f(0, []);

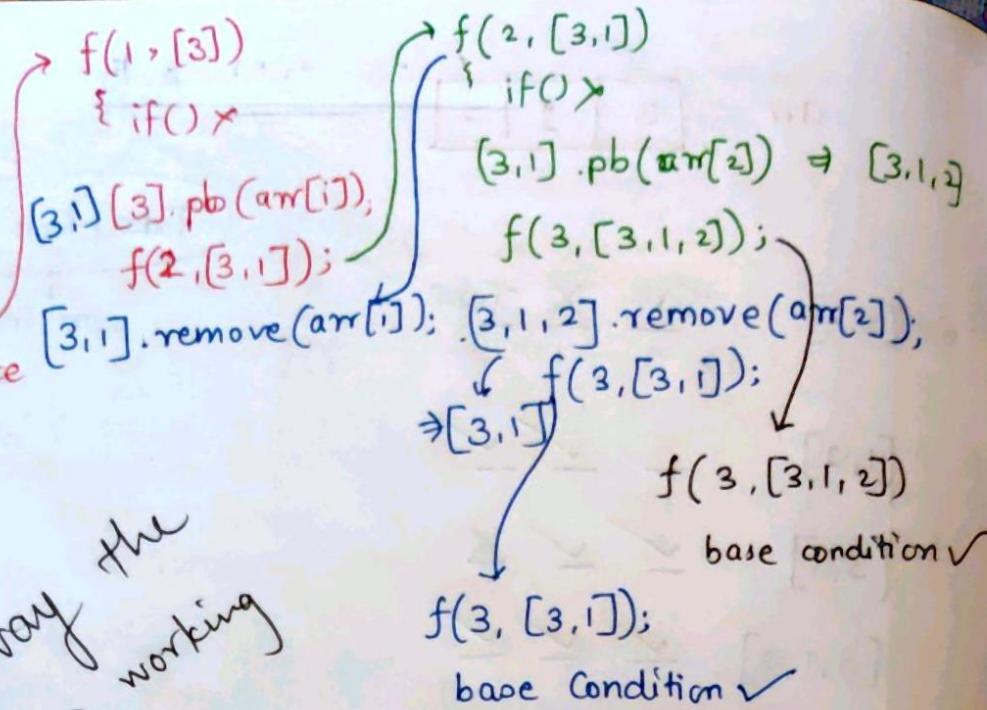
}

```

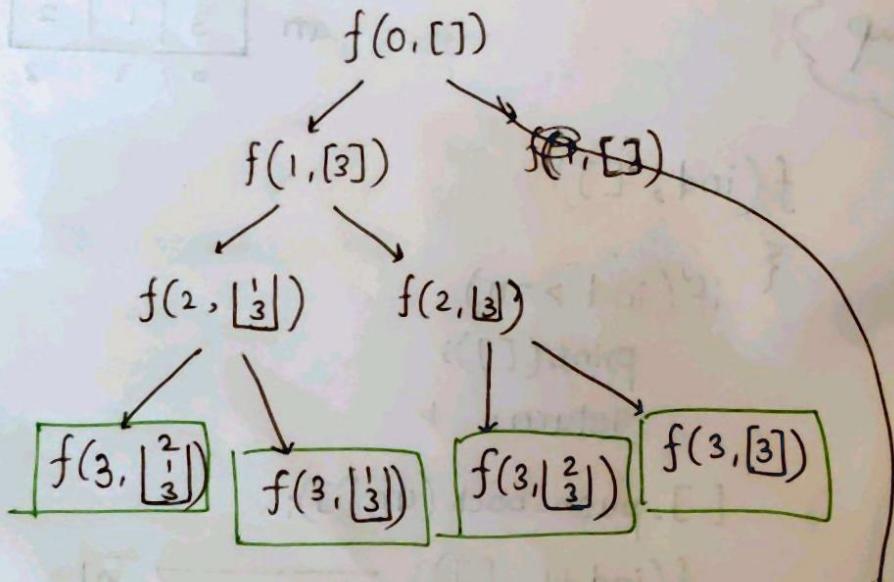
f(ind, []) {
    if (ind >= n)
        print();
        return;
    [3] .push_back(arr[i]);
    f(ind+1, []);
    [] .remove(arr[i]);
    f(ind+1, []);
}

```

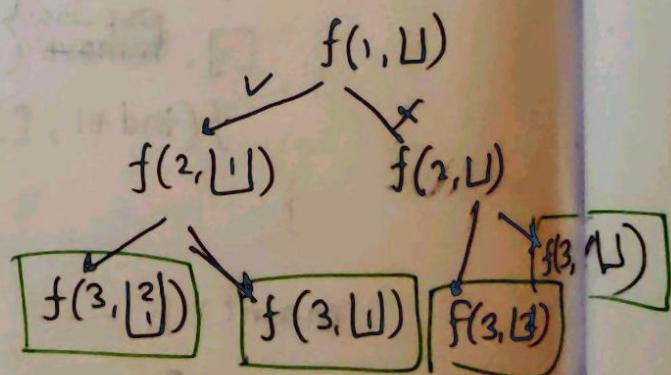
In this way the whole code is working



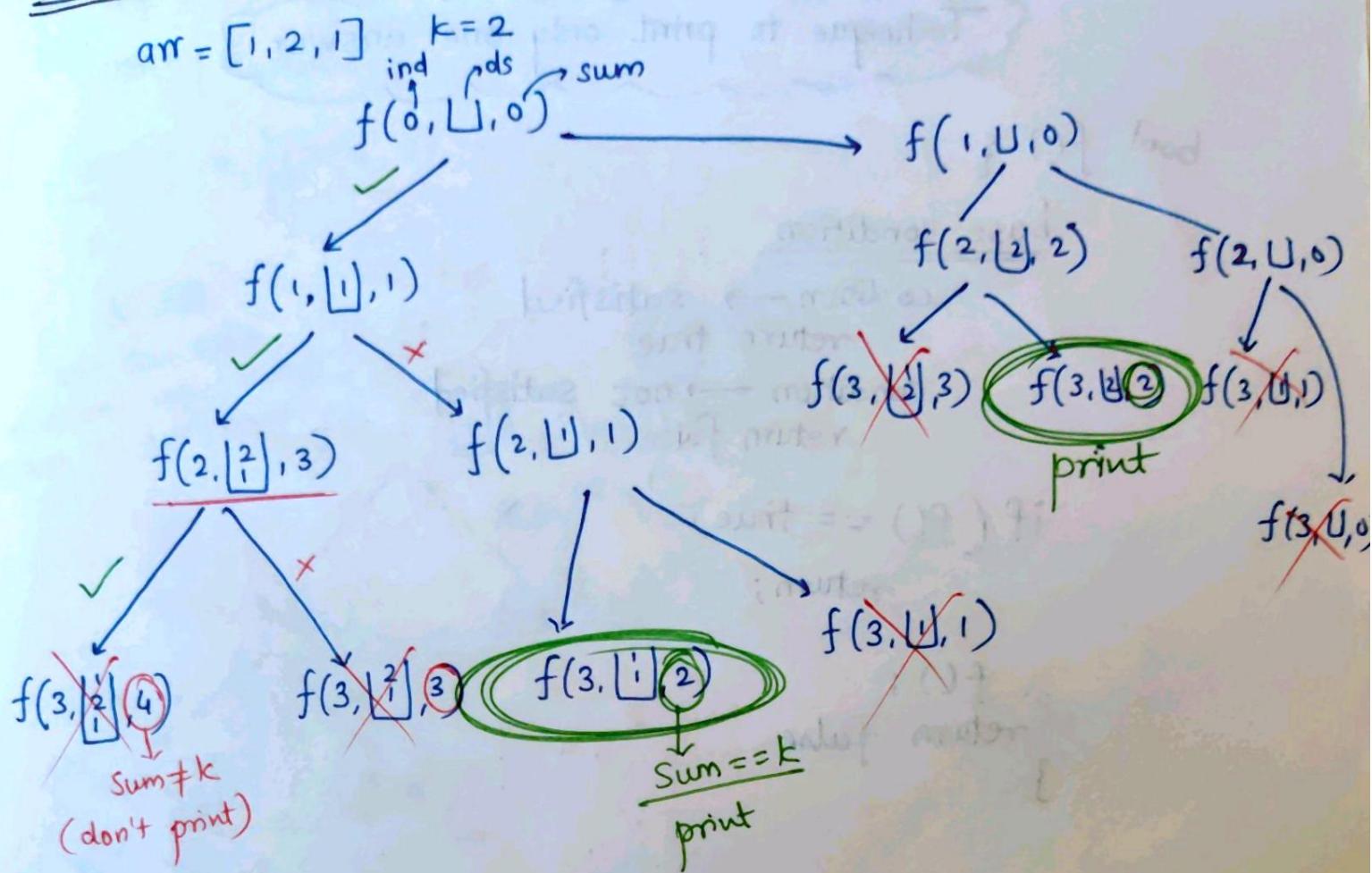
Output
 $[3, 1, 2]$
 $[3, 1]$



→ printed.



- Printing all subsequences whose sum is k



$f(\text{ind}, \cup, \text{sum})$

```
{ if (i == n)
    if (sum == k)
        print (ld);
    return;
```

```
ds.push-back (arr [ind]);  
sum += arr [ind];
```

$f(ind+1, ds, sum);$

```
ds.pop_back();
```

sum -= arr[ind];

`f(ind+1, ds, sum);`

3

• Print any subsequence whose sum is k .

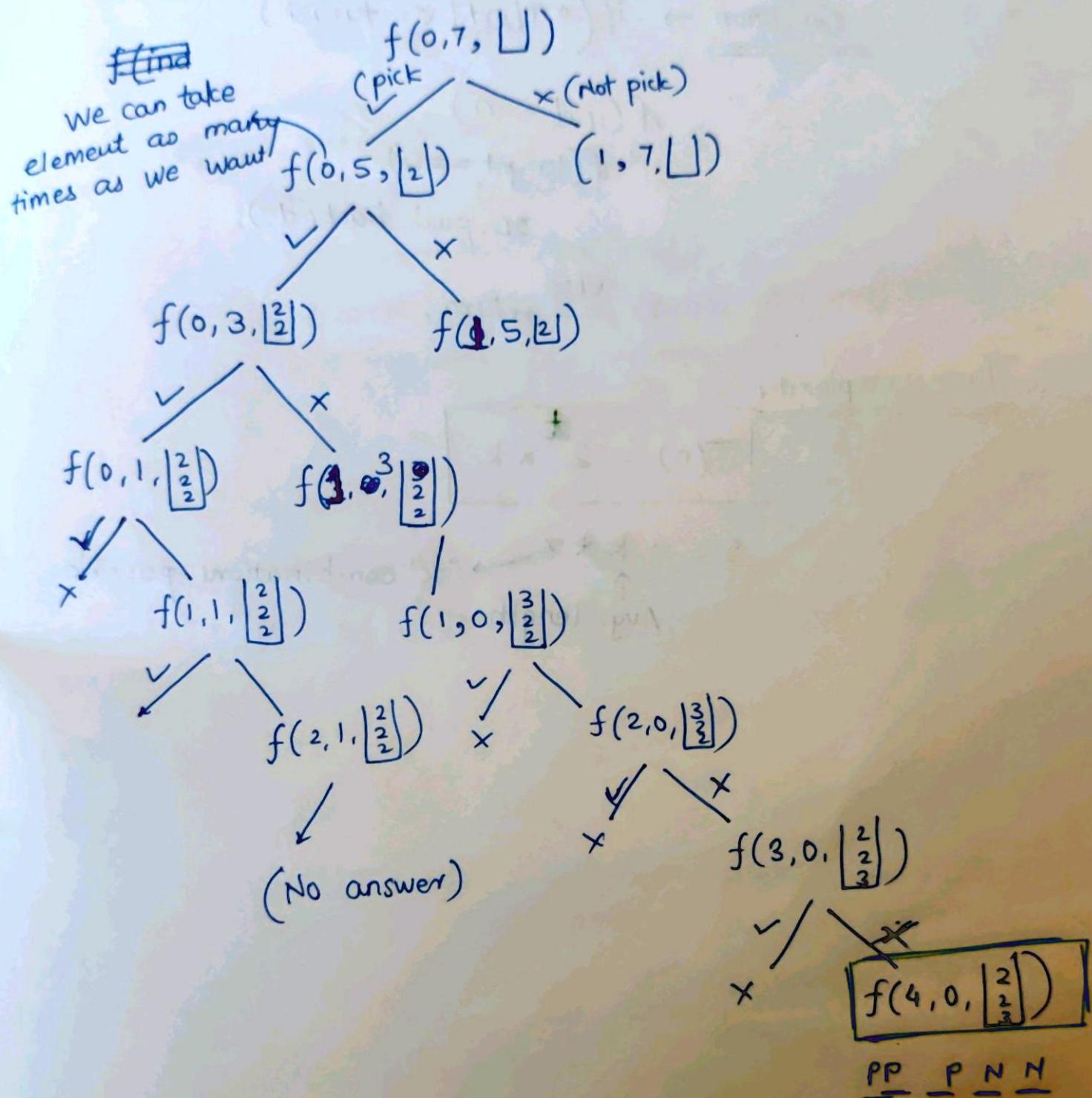
Technique to print only one answer

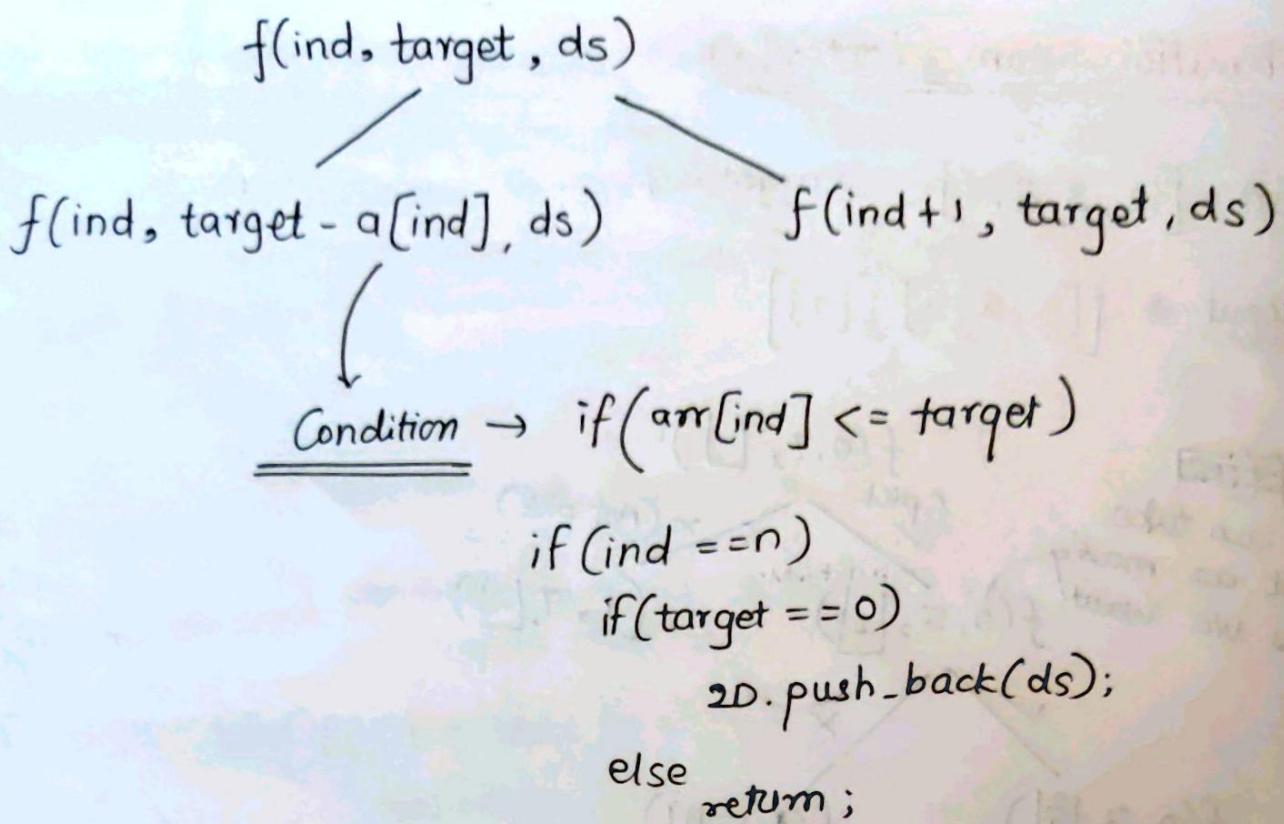
```
bool f() {  
    base condition  
    condition → satisfied  
    return true  
    condition → not satisfied  
    return false  
  
    if (f() == true)  
        return;  
  
    f();  
    return false  
}
```

• Combination Sum (Leetcode)

$\text{arr} = [2, 3, 6, 7]$ target = 7

Output $\Rightarrow [[2, 2, 3], [7]]$





Time Complexity :-

$$T(n) = 2^{\frac{n}{k}} \times k$$

S.C. = $k * x$ → x combinations possible
 ↑
 Avg. length

Code:-

```
findCombination(int ind, target, arr, vector<vector<int>>&ans, ds)
    if (ind == arr.size())
        if (target == 0)
            ans.push_back(ds);
            return;
        // pick the element
        if (arr[ind] <= target)
            ds.push_back(arr[ind])
            findCombination(ind, target - arr[ind], arr, ans, ds);
            ds.pop_back(); → When upper recursion is totally
                           executed and we have not got
                           our answer so we have to empty our
                           (pop-back()) that element) ds.
    }
    // not pick
    findCombination(ind+1, target, arr, ans, ds);
```

Combination Sum - II (Leetcode)

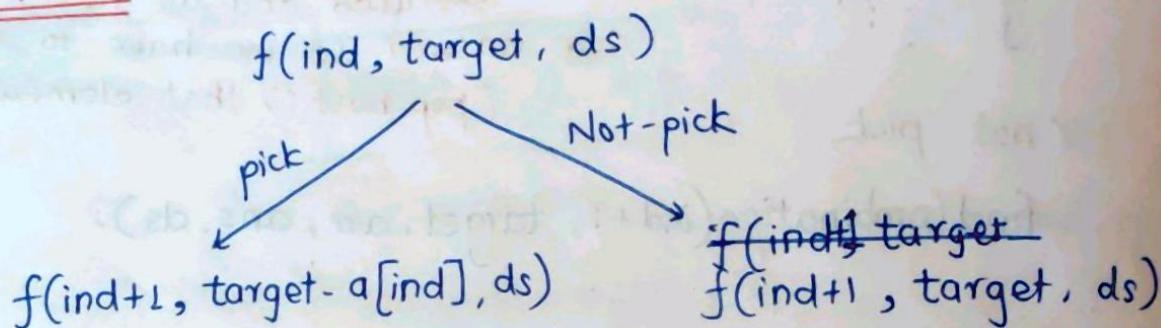
arr[] and target will be given to us.
 Find all unique combinations in "arr[]" where array elements sums to target.

- Each number can be taken only once.
- The solution set must not contain duplicate combinations.

$$\text{arr}[] = [1, 1, 1, 2, 2] \quad \text{target} = 4$$

Combinations \rightarrow $\begin{bmatrix} [1, 1, 2], \\ [2, 2] \end{bmatrix}$

Brute Force :-



- If we pick the element then we have to move the "ind" by 1 cause in this problem we can take ~~any~~ element only once.

```

findCombination(ind, target, arr, set<vector<int>> &ans, ds)
  if(ind == arr.size()) {
    if(target == 0) {
      ans.insert(ds);
    }
    return;
  }

  if(arr[ind] <= target) {
    ds.push_back(arr[ind]);
    findCombination(ind + 1, target - arr[ind], arr, ans, ds);
    ds.pop_back();
  }

  findCombination(ind + 1, target, arr, ans, ds);
  
```

modified approach :-

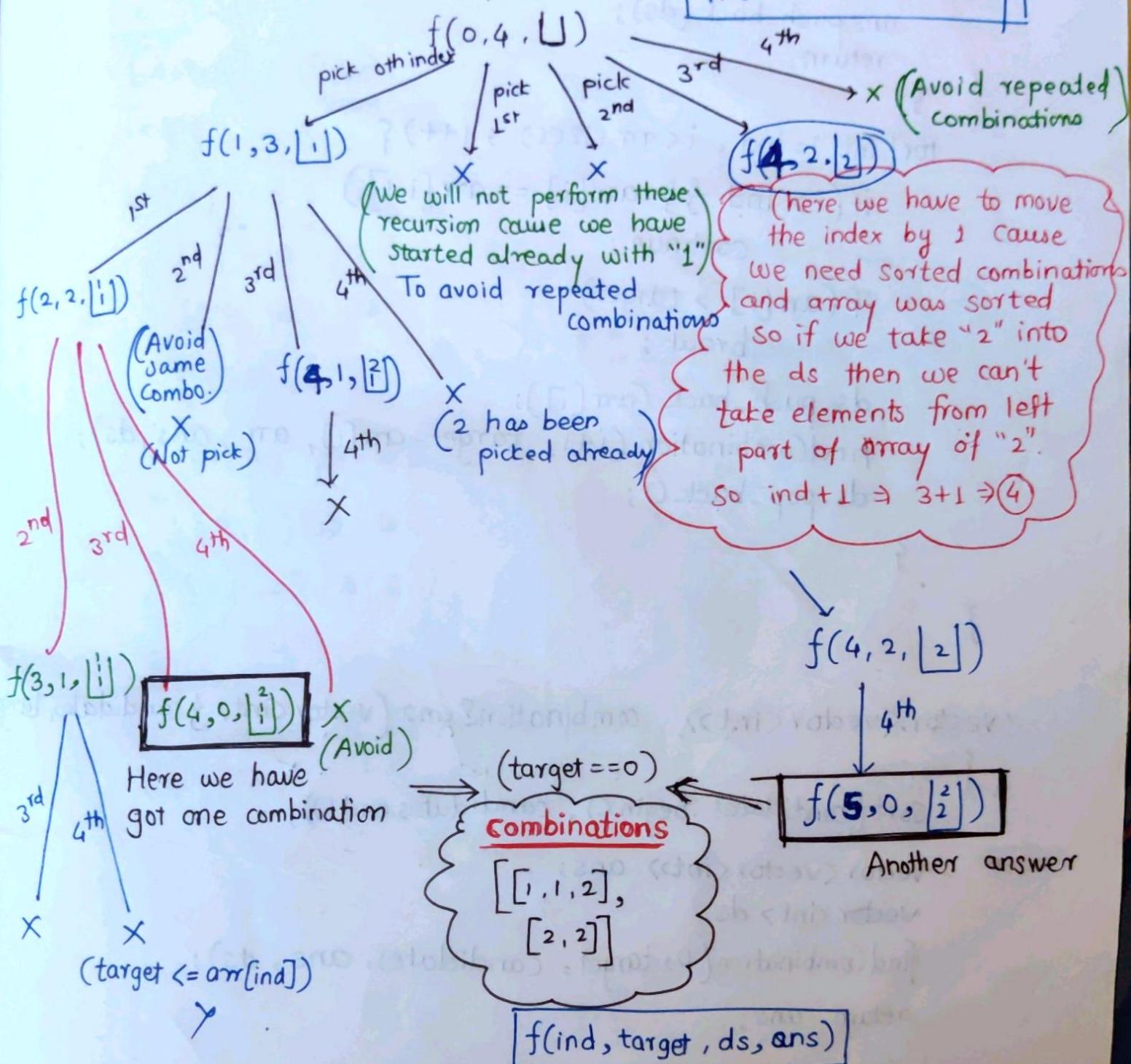
Instead of pick and non-pick we will pick subsequences.

$$\text{arr}[] = [1, 1, 1, 2, 2]$$

0 1 2 3 4

$$\text{target} = 4$$

first sort the array



if($\text{arr}[i] > \text{target}$) → break

looping from (ind to n-1) → (i)

possibility → $\text{ds.pop}(\text{arr}[i])$
 $f(\text{ind}+1, \text{target} - \text{arr}[i], \text{ds}, \text{ans})$
 $\text{ds.pop_back}()$

No possibility → $T.C. = 2^n \times k$
 $S.C. = k \times x$

Code

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

```
vector<vector<int>> combinationSum2(vector<int> &candidates, int target) {
    sort(candidates.begin(), candidates.end());
    vector<vector<int>> ans;
    vector<int> ds;
    findCombination(0, target, candidates, ans, ds);
    return ans;
}
```

Subset-Sum - I

Given an array. print sums of all subsets in it.
Output should be printed in increasing order of sums.

| | | |
|-----------|---------------------------|-------|
| <u>Ex</u> | [3, 1, 2] | n = 3 |
| | | sum |
| | { } \Rightarrow | 0 |
| | { 3 } \Rightarrow | 3 |
| | { 1 } \Rightarrow | 1 |
| | { 2 } \Rightarrow | 2 |
| | { 3, 1 } \Rightarrow | 4 |
| | { 1, 2 } \Rightarrow | 3 |
| | { 3, 2 } \Rightarrow | 5 |
| | { 3, 1, 2 } \Rightarrow | 6 |

O/P
[0, 1, 2, 3, 3, 4, 5, 6]

for any 'n'
The no. of subsets = 2^n

Brute force

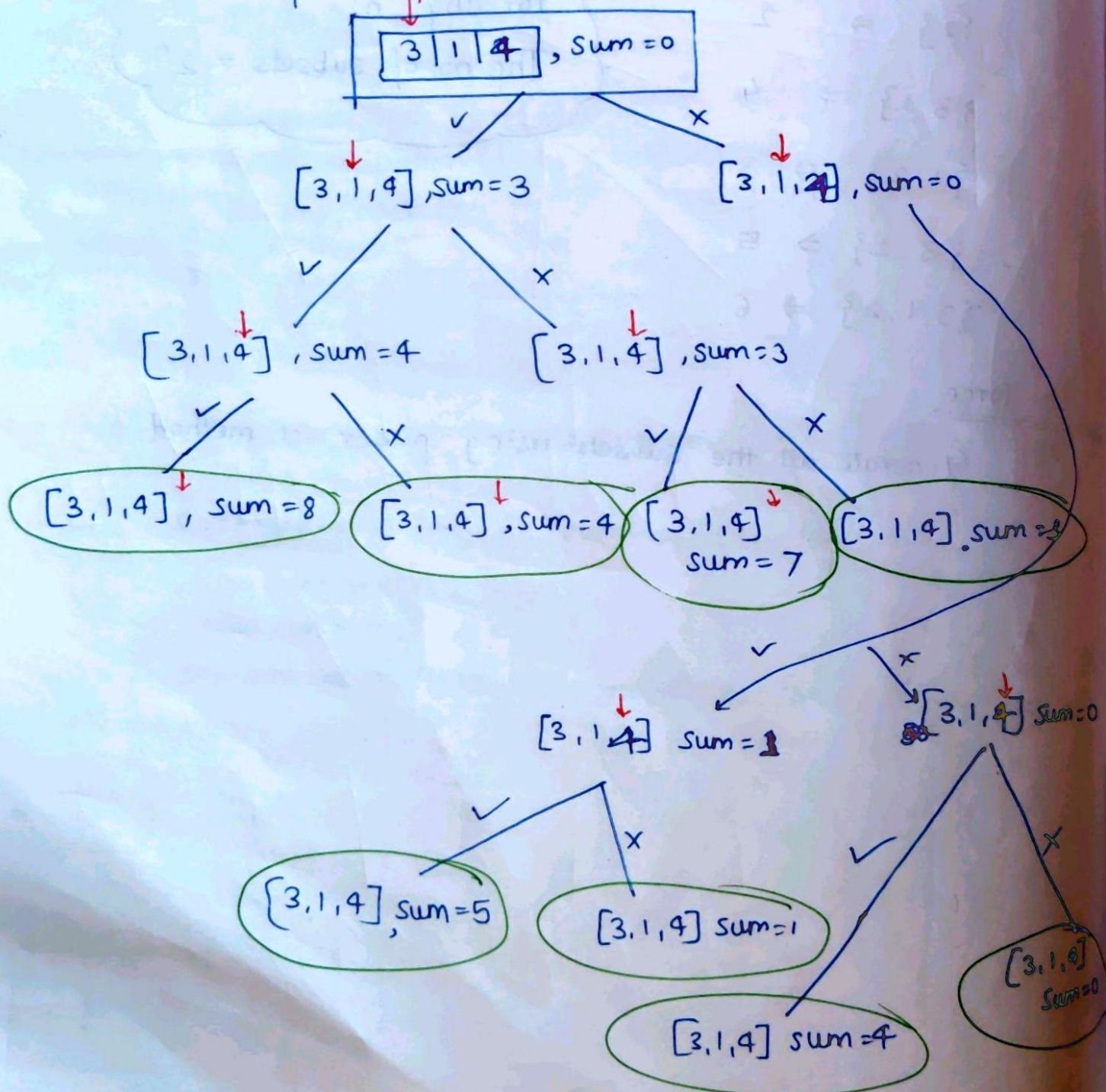
Generate all the Subsets using power set method

$[3, 1, 4]$

$$\begin{array}{r} \checkmark \\ \textcircled{0} \end{array} \quad \begin{array}{r} \times \\ \textcircled{1} \end{array} \quad \begin{array}{r} \checkmark \\ \textcircled{2} \end{array} \Rightarrow [3, 4]$$

$$\begin{array}{r} \checkmark \\ \textcircled{0} \end{array} \quad \begin{array}{r} \checkmark \\ \textcircled{1} \end{array} \quad \begin{array}{r} \times \\ \textcircled{2} \end{array} \Rightarrow [3, 1]$$

Select / Not-select
pointer

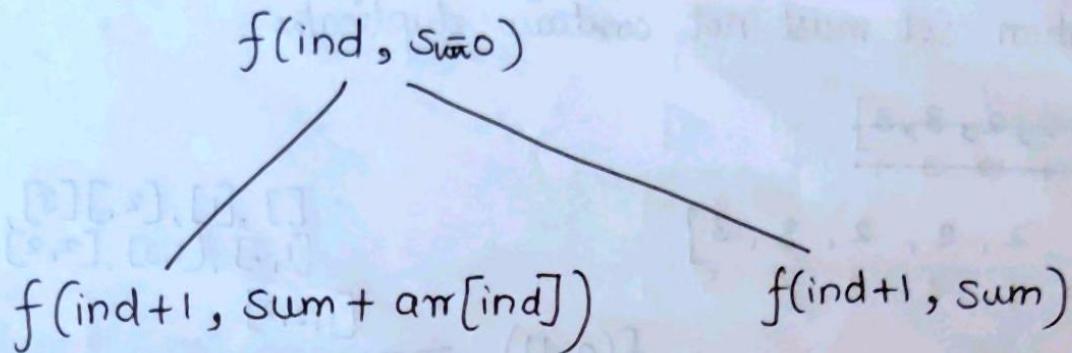


[8, 4, 7, 3, 5, 1, 4, 0]

Sort

0, 1, 3, 4, 4, 5, 7, 8

Recursive Calls



base case

```
if (ind == n)
    ds.push_back(sum);
```

$$\text{T.C.} = 2^n + 2^n \log(2^n)$$

Subset Sum - II

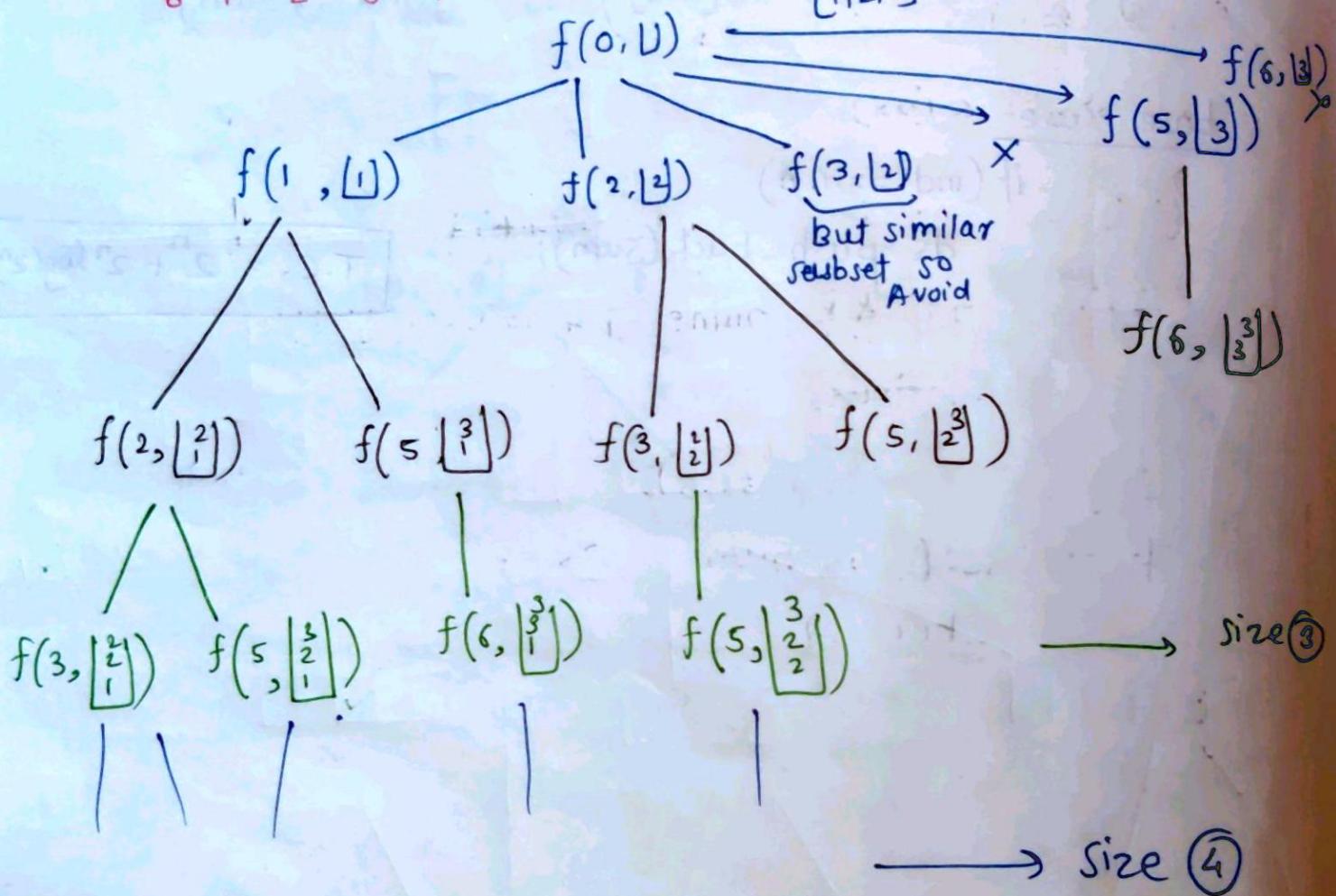
Given an integer array nums that may contain duplicates, return all possible subsets.

The solution set must not contain duplicates.

$$\text{arr} = [1, 2, 2, 3, 3]$$

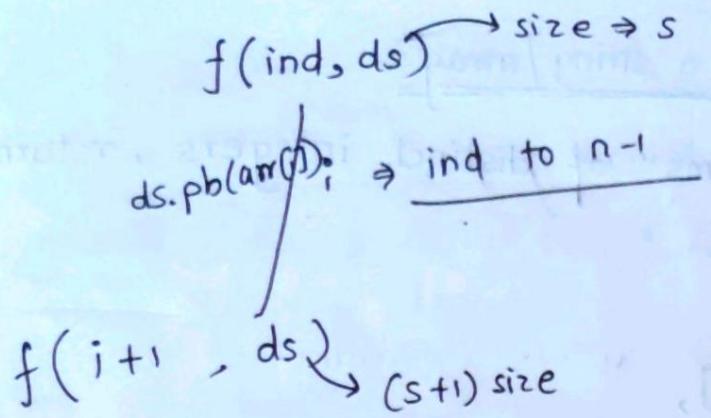
$$\text{arr} = [1, 2, 2, 2, 3, 3]$$

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



→ size ⑤

→ size ⑥



Code

```

void findSubsets(int ind, vector<int> &nums, vector<int> &ds, vector<vector<int>> &ans)
{
    ans.push_back(ds);

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

        ds.push_back(nums[i]);
        findSubsets(i + 1, nums, ds, ans);
        ds.pop_back();
    }
}
  
```

• Print all permutations of a string / array :-

Given an array "nums" of distinct integers, return all possible permutations.

Ex:- $\text{nums} = [1, 2, 3]$

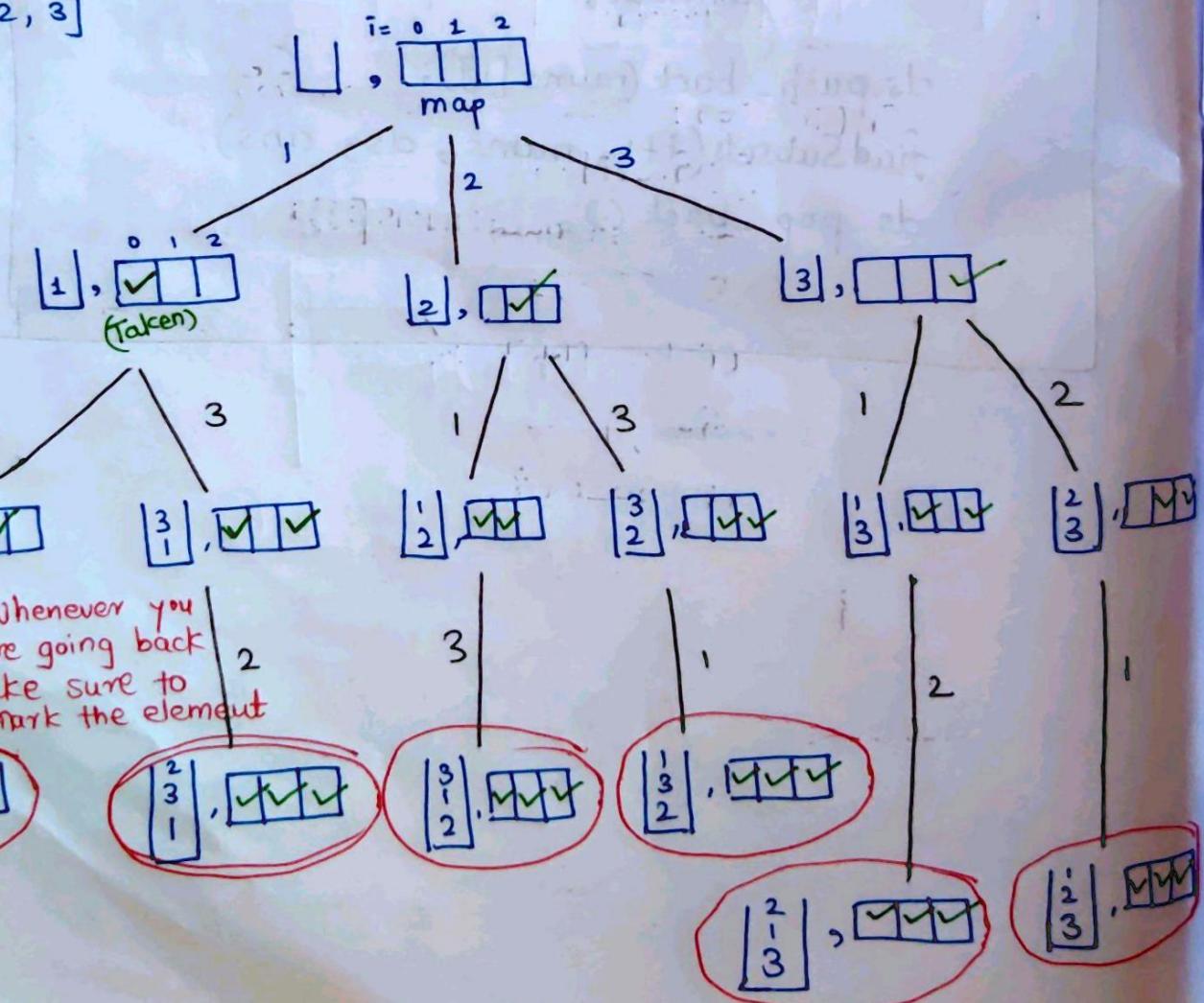
output \Rightarrow $[1, 2, 3],$
 $[1, 3, 2],$
 $[2, 1, 3],$
 $[2, 3, 1],$
 $[3, 1, 2],$
 $[3, 2, 1]$

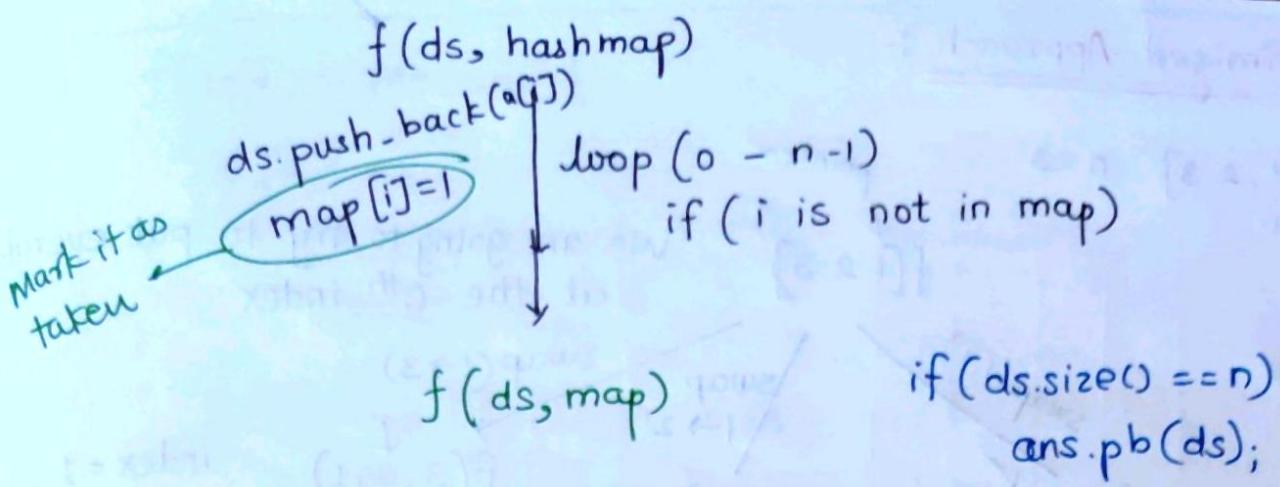
① Using Extra Space Complexity :-

If n elements is given then

No. of possible permutations = $n!$

$\text{arr} = [1, 2, 3]$





$$\text{Time Complexity} = O(n! \times n)$$

$$\text{Space Complexity} = O(n) + O(n)$$

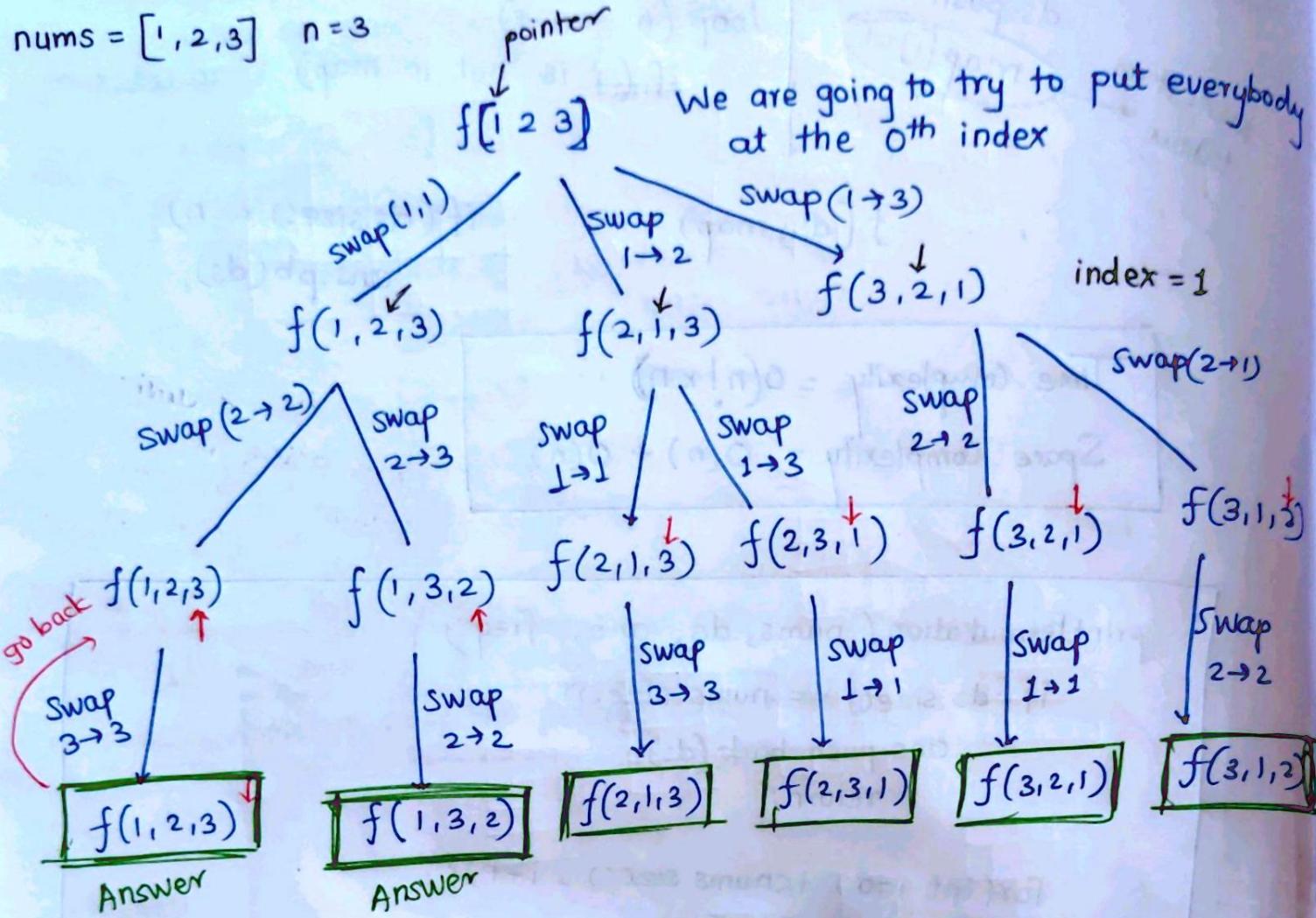
```

printPermutation ( nums, ds, ans, freq ) {
    if( ds.size() == nums.size() )
        ans.push_back (ds);
        return;
    for( int i=0 ; i<nums.size() ; i++ ) {
        if ( !freq[i] ) {
            ds.push_back (nums[i]);
            freq[i] = 1; → marked as picked
            printPermutation (nums, ds, ans, freq);
            freq[i] = 0; → unmark the position in freq
            ds.pop_back(); → Remove element from (ds);
        }
    }
}
  
```

```

main() {
    takeInput (nums);
    vector <int> ds;
    vector <vector <int>> ans;
    for int freq [nums.size()];
    for (int i=0 ; i<nums.size() ; i++)
        freq[i] = 0;
  
```

② Optimized Approach :-



f(ind, arr)

i → (ind → n - 1)
 swap(a[ind], a[i]);

f(ind + 1, arr)

Base Case :-

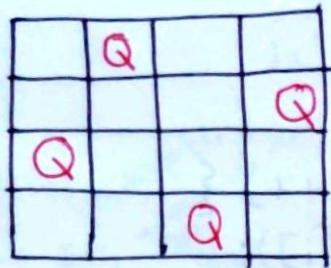
```
if(ind == n)
    ans.push_back(arr);
    return;
```

```
f(ind, nums, ans) {  
    if (ind == nums.size())  
        ans.push_back(nums);  
        return;  
    for (int i=ind ; i<nums.size(); i++) {  
        swap (nums[ind], nums [i]);  
        f(ind+1, nums, ans);  
        swap (nums[ind], nums [i]);  
    }  
}
```

After recursive call
when going backwards
we have to reswap
the elements.

N-Queens

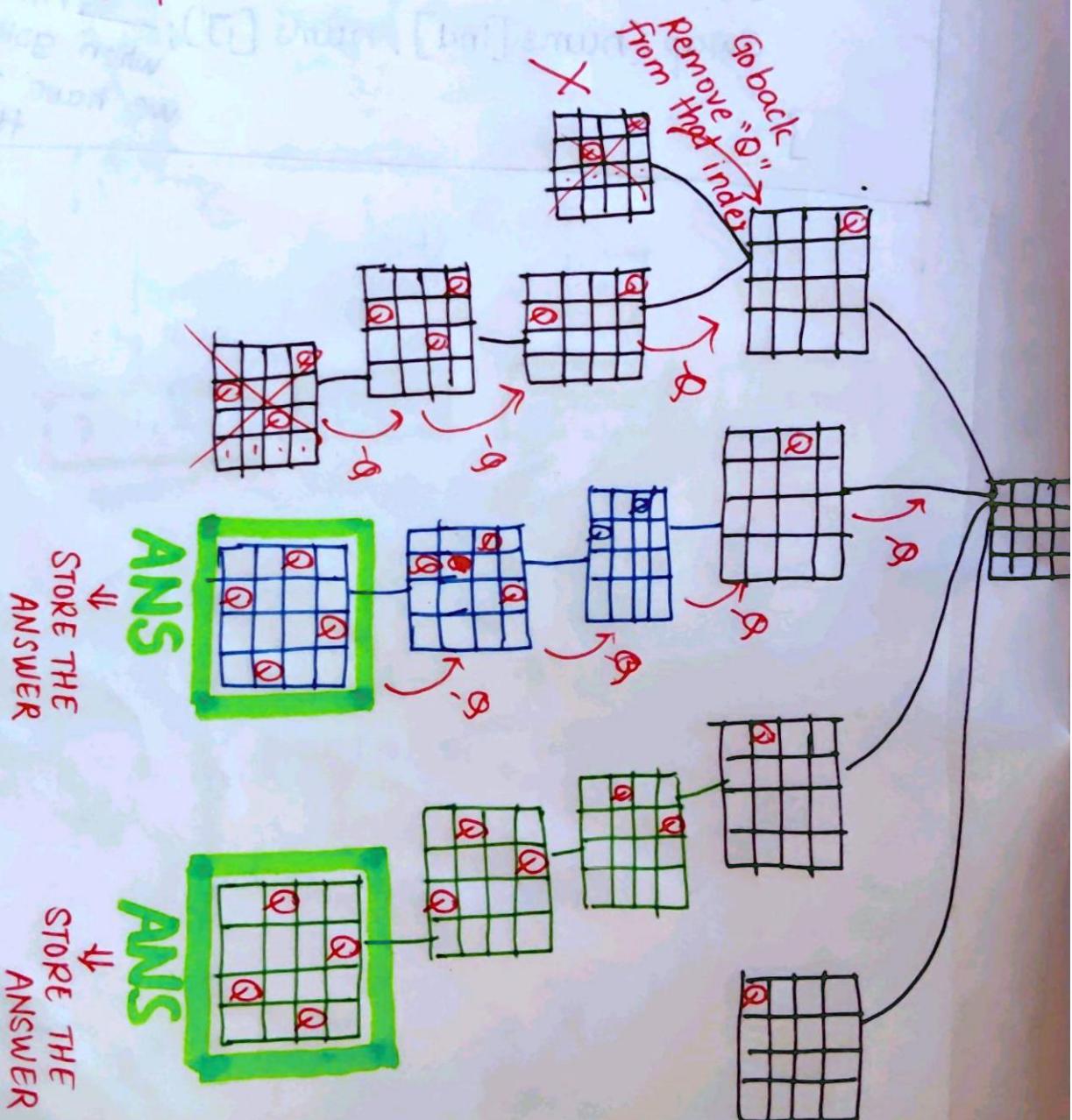
$n=4$

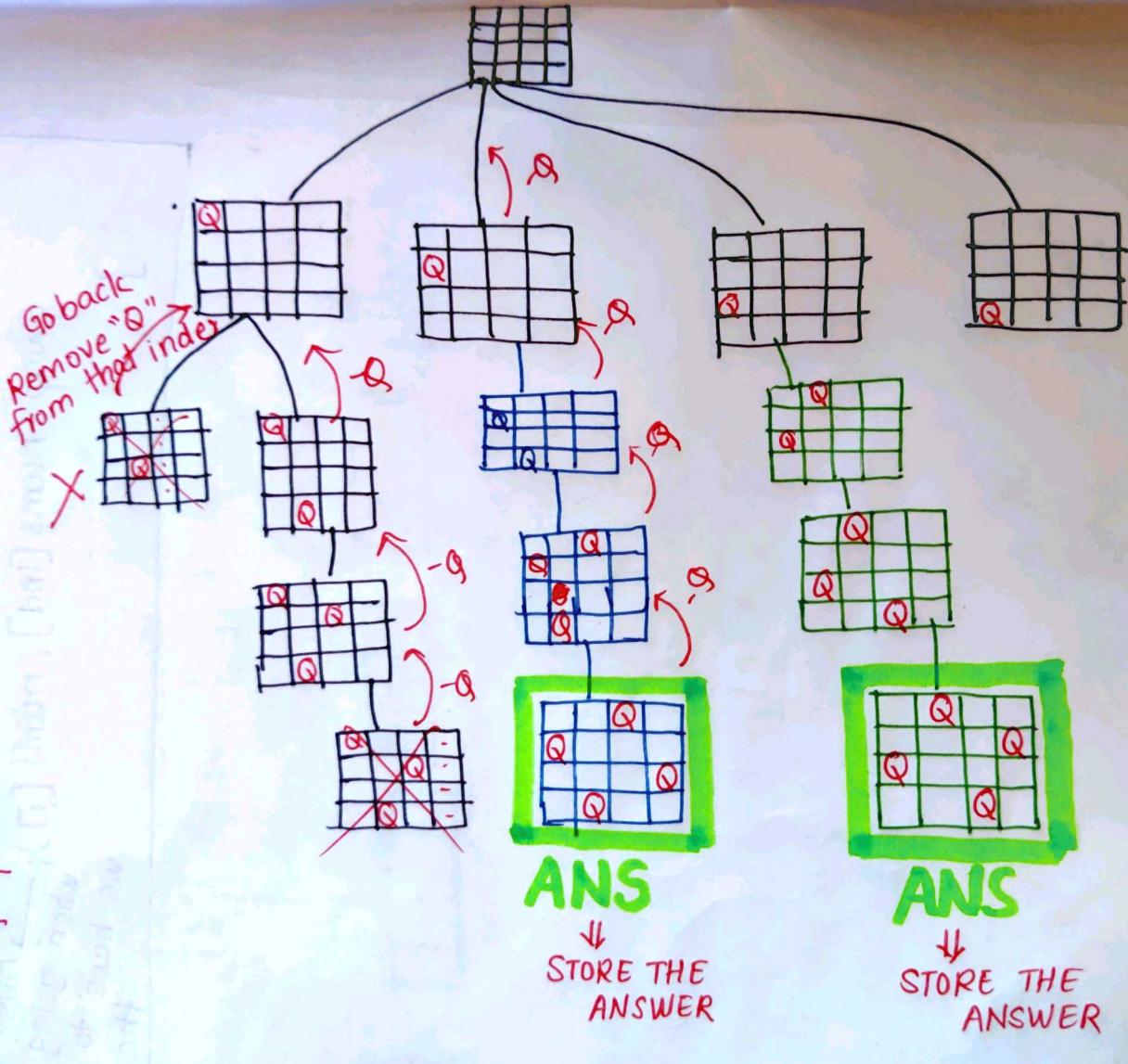


Rules

- ① Every row should have a queen.
- ② Every column should have a queen.
- ③ None of the Queen attack each other.

This is one of possible answer.





pseudo Code

filling each column

```

if (col) {
    for (i=0 ; i<row ; i++) {
        if (fill ✓) {
            Board [row] [col] = 'Q';
            solve (col +1);
            Board [row] [col] = '.';
        }
    }
}

```

Code :-

```

bool isSafe (int row, int col, vector<string> board, int n) {
    int duprow = row;
    int dupcol = col;
    while (row >= 0 && col >= 0) {
        if (board [row] [col] == 'Q') {
            return false;
        }
        row--;
        col--;
    }
    row = duprow;
    col = dupcol;
    while (row < n &&
           while (col >= 0) {
               if (board [row] [col] == 'Q') {
                   return false;
               }
               col--;
           }
    }
}

```

```
row = duprow;
col = dupcol;
while( row < n && col >= 0) {
    if (board[row][col] == 'Q') {
        return false;
    }
    row++;
    col--;
}
return true;
}
```

```
void solve(int col, vector<string> &board,
           vector<vector<string>> &ans, int n){
    if (col == n) {
        ans.push_back(board);
        return;
    }
    for (int row = 0; row < n; row++) {
        if (isSafe(row, col, board, n)) {
            board[row][col] = 'Q';
            solve(col + 1, board, ans, n);
            board[row][col] = '.';
        }
    }
}
```

```
vector<vector<string>> solveQueens(int n){  
    vector<vector<string>> ans;  
    vector<string> board(n);  
    string s(n, '.');  
    for (int i=0; i<n; i++) {  
        board[i] = s;  
    }  
    solve(0, board, ans, n);  
    return ans;  
}
```

Sudoku Solver

- 1) The digit 1-9 appears only once in a row.
- 2) The digit 1-9 appears only once in a column.
- 3) The digit 1-9 appears only once in 3x3 grid.

Code :-

```
void SolveSudoku(vector<vector<char>>& board){  
    solve(board);  
}
```

```
bool solve(vector<vector<char>>& board){  
    for(int i=0 ; i < board.size() ; i++){  
        for(int j=0 ; j < board[0].size() ; j++){  
            if(board[i][j] == '.'){  
                for(char c='1' ; c <= '9' ; c++){  
                    if(isValid(board, i, j, c)){  
                        board[i][j] = c;  
                        if(solve(board) == true)  
                            return true;  
                        else  
                            return false; board[i][j] = '.';  
                    }  
                }  
                return false;  
            }  
        }  
    }  
    return true;  
}
```

```
bool isValid(vector<vector<char>> &board, int row, int col, char c) {
    for(int i=0 ; i<9 ; i++) {
        if (board[i][col] == c)
            return false;
        if (board[row][i] == c)
            return false;
        if (board[3 * (row/3) + (i/3)][3 * (col/3) + (i%3)] == c)
            return false;
    }
    return true;
}
```

M-Coloring problem

Given an undirected graph and an integer M. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of graph means the assignment of colors of all vertices. Print 1 if possible to colour all the vertices and 0 otherwise.

Ex:-

$$N = 4$$

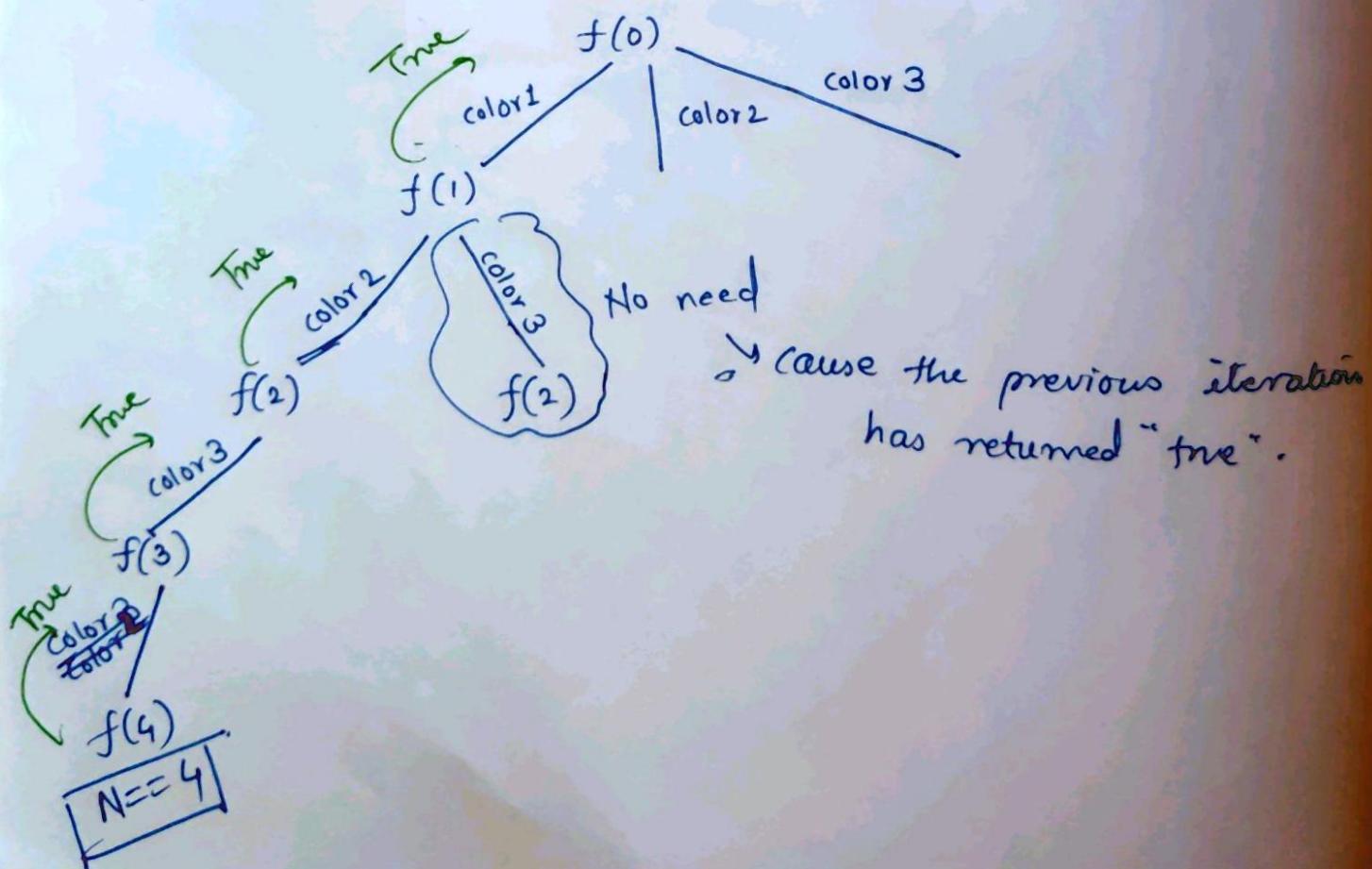
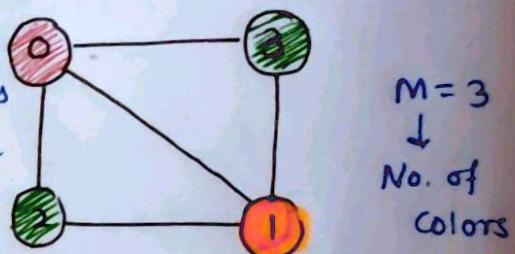
$$M = 3$$

$$E = 5$$

$$\text{Edges}[] = \{(1,2), (2,3), (3,4), (4,1), (1,3)\}$$

$$\text{o/p} = 1$$

None of the adjacent nodes have the same color. Hence we can color the graph.



pseudo code :-

```
f(node) {  
    if (node == N) → Base case  
        return true  
  
    for (color = 1 → M) {  
        if (possibleColor(node)) ✓ {  
            color[node] = col;  
            if (f(node + 1) = true) → if it giving ⇒ T  
                return true; then we can return true  
            color[node] = 0 → Remove the color  
        }  
    }  
    return false; → It's not possible to color the  
} graph.
```

Palindromic Partitioning

Given a string s , partition ' s ' such that every substring of the partition is a palindrome. Return all possible palindromic partition of string ' s '.

Ex:- $s = "aab"$

output = $\left[\left["a", "a", "b" \right], \left["aa", "b" \right] \right]$

Ex:- ① $s = "aabb"$

→ if $a|a|b|b$ → ^{then} _{partition} $\{ "a", "a", "b", "b" \}$
partitions

$a|a|b|b$ → $\{ "a", "a", "bb" \}$

$a|a|b|b$ → $\{ "aa", "b", "b" \}$

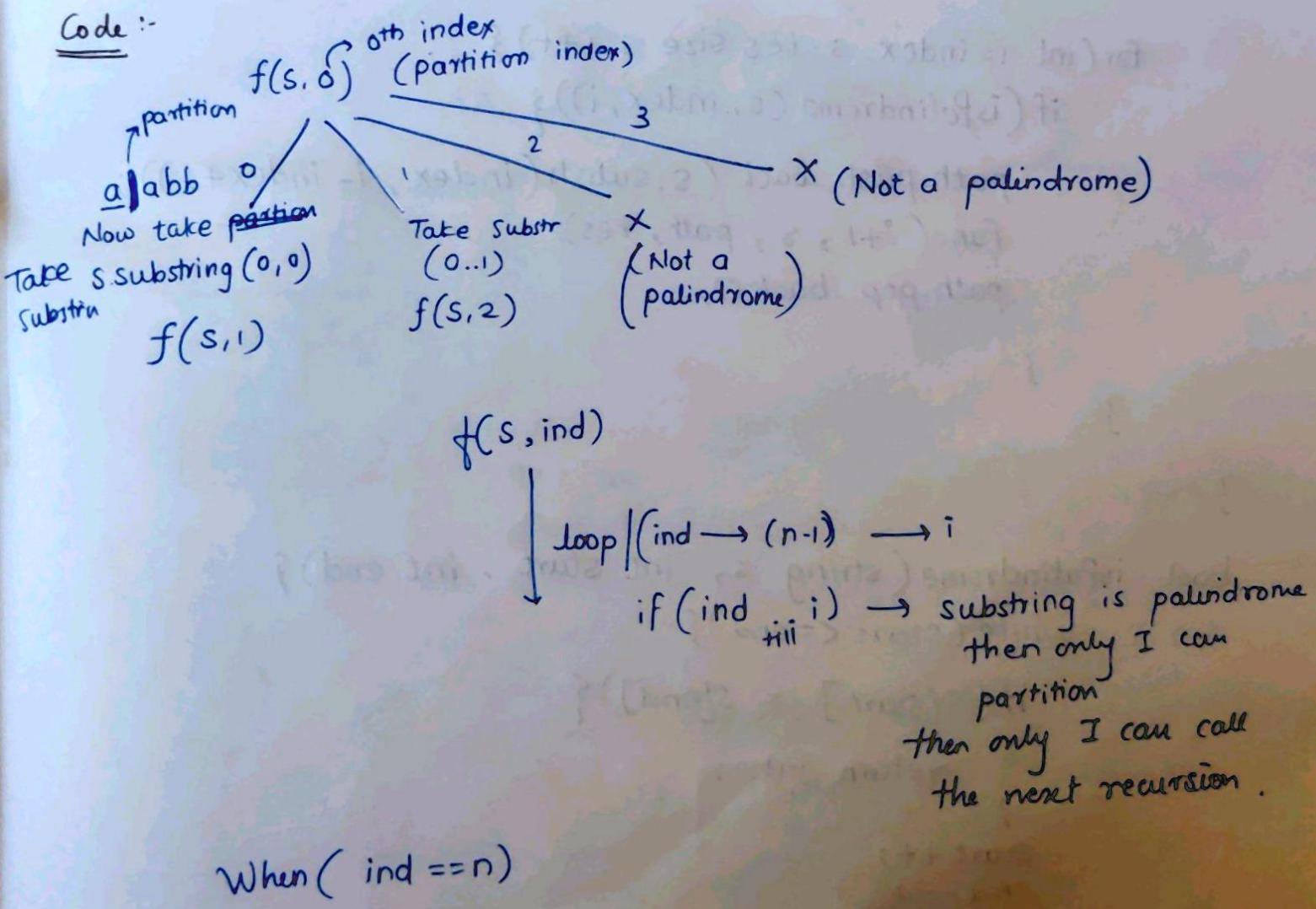
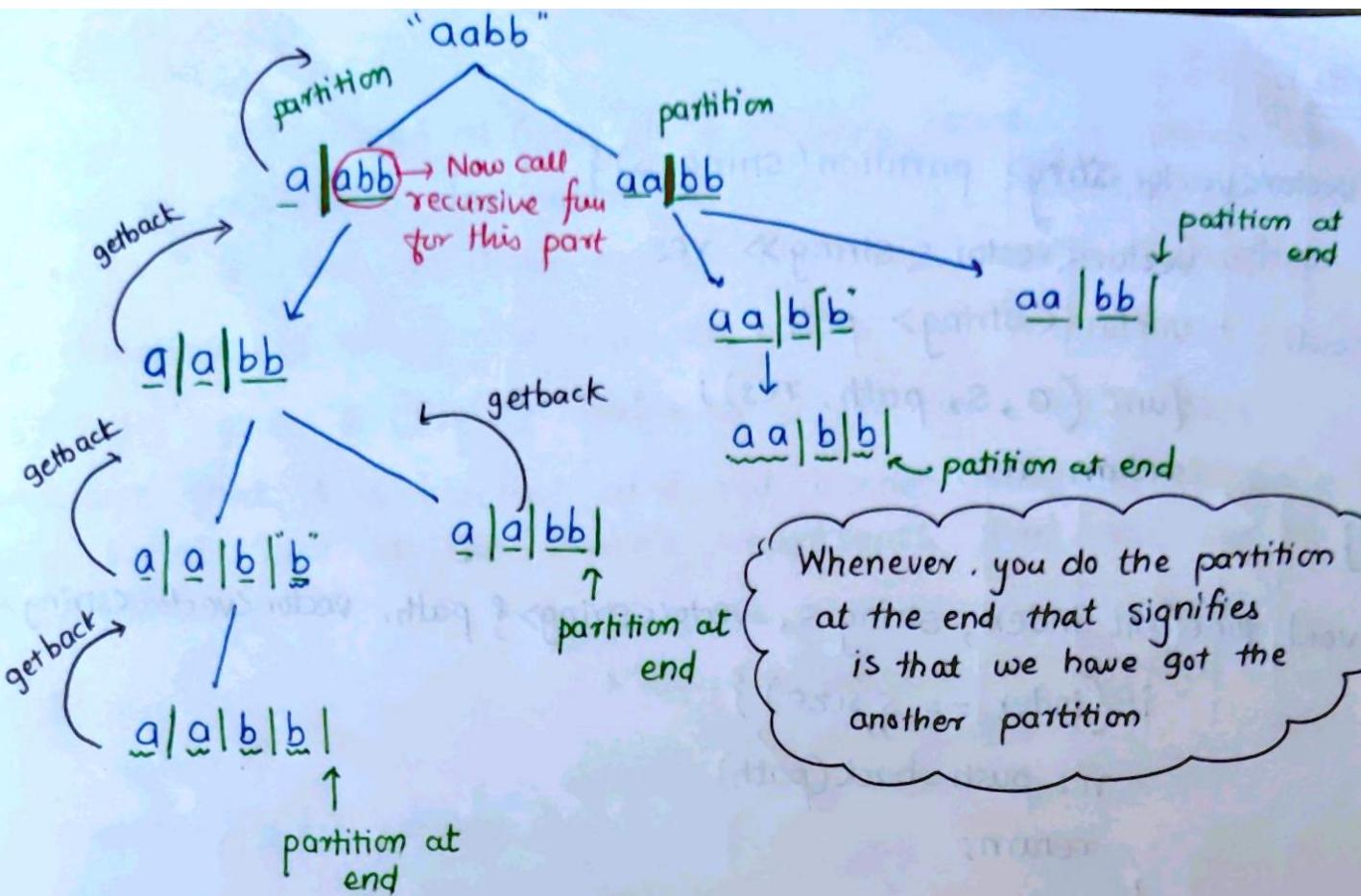
$aa|bb$ → $\{ "aa", "bb" \}$

Output:

$\{ \{ "a", "a", "b", "b" \},$
 $\{ "a", "a", "bb" \},$
 $\{ "aa", "b", "b" \},$
 $\{ "aa", "bb" \} \}$

⇒ as $1 \leq s.length() \leq 16$

so we can apply brute force.



Code :-

```
vector<vector<string>> partition(string s){  
    vector<vector<string>> res;  
    vector<string> path;  
    func(0, s, path, res);  
    return res;  
}  
  
void func(int index, string s, vector<string>& path, vector<vector<string>&> res){  
    if(index == s.size()) {  
        res.push_back(path);  
        return;  
    }  
  
    for(int i = index ; i < s.size ; i++) {  
        if(isPalindrome(s, index, i)) {  
            path.push_back(s.substr(index, i - index + 1));  
            func(i + 1, s, path, res);  
            path.pop_back();  
        }  
    }  
}  
  
bool isPalindrome(string s, int start, int end) {  
    while(start <= end) {  
        if(s[start] != s[end]) {  
            return false;  
        }  
        start++;  
        end--;  
    }  
    return true;  
}
```

Rat in a maze :-

Consider a rat placed at $(0,0)$ in a square matrix of order $n \times n$. It has to reach the destination at $(n-1, n-1)$. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U' (Up), 'D' (Down), 'L' (Left), and 'R' (Right). Value ~~at~~ 0 cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at cell in the matrix represents that rat can be travel through it.

Input :-

$n = 4$

matrix[][] = { {1, 0, 0, 0},
 {1, 1, 0, 1},
 {1, 1, 0, 0},
 {0, 1, 1, 1} }

The answer should be printed in lexicographically

Output \Rightarrow PPRDRR DRDDRR

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | |
| 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 |

DRDDRR
DDRDRD

Lexicographically \rightarrow

DDRDRR \leftarrow DRDDRR

| Source | 0 | 1 | 2 | 3 |
|--------|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |

$f(\text{start, end..}, 0, 0, "")$

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | ✓ | | | |
| 1 | ✓ | | | |
| 2 | ✓ | ✓ | | |
| 3 | | ✓ | ✓ | ✓ |

visited

Lexicographically order of moves
D L R U

$f(0, 0, "")$ (D, L, R, U)

$f(1, 0, "D")$ (D, L, R, U)

$f(2, 0, "DD")$ (D, L, R, U)

$f(2, 1, "DDR")$ (D, L, R, U)

$f(3, 1, "DDRD")$ (D, L, R, U) (Already visited)

$f(3, 2, "DDRDR")$ (D, L, R, U) (Already visited)

$f(3, 3, "DDRDRR")$

getback

$f(1, 1, "DDRU")$

Now after returning, we have to mark (3, 3) as unvisited → but we can't take any more



! We have reached to our destination !!

$f(1, 0, "DR")$

Right move

$f(1, 1, "DR")$ ~~D~~ L R U

$f(2, 1, "DRO")$ D LRU

$f(3, 1, "DRDD")$ ~~D~~ ~~L~~ ~~R~~ U

$f(3, 2, "DRDDR")$ ~~D~~ ~~L~~ ~~R~~ U ✓

$f(3, 3, "DRDDRR");$

Reached!

Pseudo Code

$f()$ {

 Down
 $f();$

 Left
 $f();$

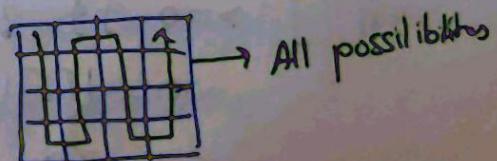
 Right
 $f();$

 Up
 $f();$

}

Time complexity = $O(4^{m \times n})$

Space Complexity = Depth of the maximum recursive tree
 $O(m * n)$



```

vector<string> findPath(vector<vector<int>> &m, int n) {
    vector<string> ans;
    vector<vector<int>> visited(n, vector<int>(n, 0));
    if (m[0][0] == 1)
        solve(0, 0, m, n, ans, "", visited);
    return ans;
}

void solve(int i, int j, vector<vector<int>> &a, int n, vector<string> &ans, string move,
           vector<vector<int>> &visited) {
    if (i == n - 1 && j == n - 1) {
        return ans.push_back(move);
        return;
    }

    // Downward direction
    if (i + 1 < n && !visited[i + 1][j] && a[i + 1][j] == 1) {
        visited[i][j] = 1;
        solve(i + 1, j, a, n, ans, move + 'D', visited);
        visited[i][j] = 0;
    }

    // Left direction
    if (j - 1 >= 0 && !visited[i][j - 1] && a[i][j - 1] == 1) {
        visited[i][j] = 1;
        solve(i + 1, j, a, n,
        solve(i, j - 1, a, n, ans, move + 'L', visited);
        visited[i][j] = 0;
    }
}

```

// Right direction

```
if (j+1 < n && !visited[i][j+1] && a[i][j+1] == 1) {  
    visited[i][j] = 1;  
    solve(i, j+1, n, ans, move + 'R', visited);  
    visited[i][j+1] = 0;
```

}

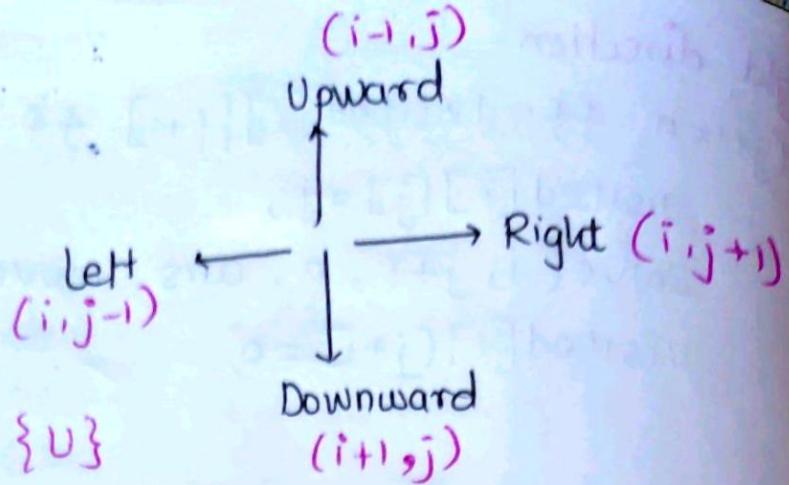
// upward direction

```
if (i-1 >= 0 && !visited[i-1][j] && a[i-1][j] == 1) {  
    visited[i][j] = 1;  
    solve(i-1, j, n, ans, move + 'U', visited);  
    visited[i-1][j] = 0;
```

}

→ This is a bit lengthy code.

→ We can shorten this code by using for loop.



$\{D\}$ $\{L\}$ $\{R\}$ $\{U\}$

$di[] = \{+1, +0, +0, -1\}$

$dj[] = \{+0, -1, +1, +0\}$

↓
current index gets $\rightarrow +1$
and 'j' gets $\rightarrow +0$

So we can do

$i + di[\text{index}]$
 $j + dj[\text{index}]$

Because of this you
don't need to write 4 different
if statements

```

void solve( int i, int j, vector<vector<int>> &a, int n, vector<string> &ans,
           string move, vector<vector<int>> &visited, int di[], int dj[] ) {
    if( i == n-1 && j == n-1 ) {
        ans.push_back(move);
        return;
    }

    string direction = "DLRU";
    for( int index=0 ; index < 4 ; index++ ) {
        int nexti = i + di[index];
        int nextj = j + dj[index];
        if( nexti >= 0 && nextj >= 0 && nexti < n && nextj < n && !visited[nexti][nextj] && a[nexti][nextj] == 1 ) {
            visited[i][j] = 1;
            solve(nexti, nextj, a, n, ans, move + direction[index],
                  visited, di, dj);
        }
    }
}
    
```

```
vector<string> findPath (vector<vector<int>> &m, int m, int n) {
    vector<string> ans;
    vector<vector<int>> visited (vector<int> (n, 0));
    int di[] = { 1, 0, 0, -1 };
    int dj[] = { 0, -1, 1, 0 };
    if (m[0][0] == 1)
        solve(0, 0, m, n, ans, "", visited, di, dj);
    return ans;
}
```

k^{th} permutation Sequence

The set $[1, 2, 3, \dots, n]$ contains a total of " $n!$ " unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for $n=3$.

- ① "123"
- ② "132"
- ③ "213"
- ④ "231"
- ⑤ "312"
- ⑥ "321"

Given 'n' & 'k' find the k^{th} permutation sequence.

Example 1 :-

Input :- $n=3, k=3$

Output :- "213"

① Brute Force:-

- ① Generate all the permutations by using recursion.
- ② Store all the permutation into a data structure.
- ③ Sort the data structure.
- ④ Return the $(k-1)^{\text{th}}$ element from the ds.

Time Complexity :- $O(n! \times n)$

To generate all
permutation

Store the permutation
into a ds

② Optimal Solution :-

Example :- $n=4$, $k=17$

$$\# \text{. of permutations} = n!$$

$$= 41$$

$$= 24$$

[1, 2, 3, 4]

indexing $k=17 \rightarrow 16^{\text{th}}$ permutation

(0-5) pick 1 + (2, 3, 4) → 6 permutations

(6, 11) will pick 2 + (1, 3, 4) → 6 permutations

$(12, 17)$ In this range pick $3 + (1, 2, 4) \rightarrow 6$ permutations

$(18-23)$ pick 4 + $(1, 2, 3)$ \longrightarrow 6 permutations

1 2 3 4 → 0^{th} permutation

(0-based indexing)

$\begin{array}{cccc} & & & \\ & & & \\ & & & \\ & & & \\ \hline 4 & 3 & 2 & 1 \end{array} \longrightarrow 23^{\text{rd}} \text{ permutation}$

$k=17 \rightarrow 16^{\text{th}}$ permutation

Each no. has 6 permutation

$$\text{So } 16/6 = \boxed{2} \frac{\text{index}}{}$$

s means

It will start from
this number.

Scanned by Scanner Go

3 - - -

$16/.6 = 4^{\text{th}}$ sequence out of

$$3 + (1, 2, 4)$$

this sequence

then our job will done.

$[1, 2, 4] \rightarrow$ Remaining numbers
and we want 4^{th} permutation

So Now question is

$$[1, 2, 4] \quad k=4$$

Again Same process

(0-1) pick $\rightarrow 1 + \{2, 4\} \rightarrow$ 2 permutations

(2-3) pick $\rightarrow 2 + \{1, 4\} \rightarrow$ 2 permutations

(4-5) pick $\rightarrow 4 + \{1, 2\} \rightarrow$ 2 permutations

k^{th} permutation $\rightarrow 4/2 = 2$

$$[1, 2, 4]$$

permutation is going to start with 4

$$\underline{3} \underline{4} - -$$

$4/.2 = 0 \rightarrow$ Now question boils down to

$$[1, 2] \quad k=0$$

Again Same process

$$[1, 2] \quad k=0$$

(0-0) pick $\rightarrow 1 + \{2\} \rightarrow 1$ permutations

(1-1) pick $\rightarrow 2 + \{1\} \rightarrow 1$ permutation

$$k^{\text{th}} \text{ permutation} = 0!_1 = \underline{0}$$

$$\begin{matrix} & 1 \\ [1, 2] & \end{matrix}$$

means permutation is going to start with 1.

$$\begin{matrix} 3 & 4 & 1 & - \\ \cancel{0} & & & \end{matrix}$$



$$k = 0 \% \cancel{0} = 0 \rightarrow [2] \quad k=0$$

Of again same process

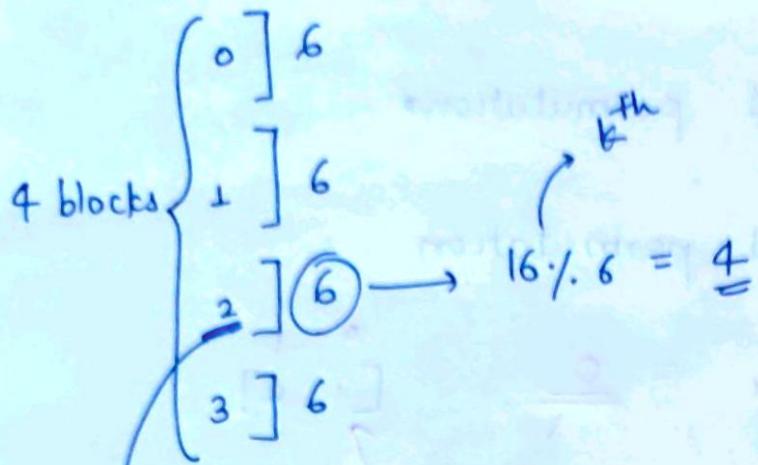
we will get

3 4 1 2

\Rightarrow This is our answer !!

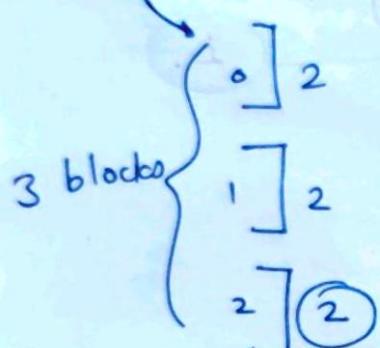
Summary :-

At first,



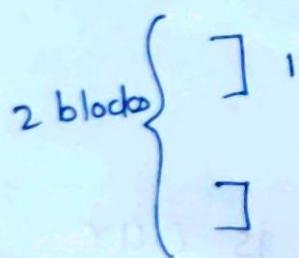
$$6+6+6+6=24$$

$$4! = 24$$



$$2+2+2=8$$

$$3! = 8$$



$$2! = \underline{\underline{2}}$$

In each pass, we are taking/picking 1 element out till $\underline{\underline{n}}$

↑ so $O(N)$

$\Rightarrow O(N^2)$

like this.

Time Complexity : $O(N) * O(N)$

$n=4 \rightarrow$ we are looking for 4 numbers

so $O(N)$ will take

Space Complexity = $O(N)$

Code

WHY?

because initially we have ~~n blocks~~
if $n=4 \rightarrow 24$ permutations but we had $4!$ blocks
of $[6]$ permutations

```
string getPermutation (int n, int k){
```

```
    int fact = 1;
```

```
    vector<int> numbers;
```

```
    for(int i=1 ; i<n ; i++) {
```

```
        fact = fact * i;
```

```
        numbers.push_back(i); → This will store  
    }
```

Hence (n) is taken
↑ This is the factorial.
for $n=4 \rightarrow$ we have to compute $3!$

```
    numbers.push_back(n); → This will add 'n' to last position
```

```
    string ans = " ";
```

$[1, 2, 3, 4]$

```
    k = k - 1; → (0-based) indexing hence we have reduced a value by 1.
```

```
while(true) {
```

```
    ans += to_string(numbers[k/fact]); k=16
```

↓
Initially we had 4 blocks of
numbers.erase(numbers.begin() + (k/fact)); $\textcircled{6}$

erase '3'
so $16/6 = 2$

$16/6 =$ we have done
we got '2' and then we have taken the 2nd number

$[1, 2, \textcircled{3}, 4]$

Taken ✓

Add it to ans.

numbers.begin() + 2

$0+2$
 $(2)^{\text{nd}}$ element
get erased.

```
if(numbers.size() == 0) break;
```

$k = k \% \text{fact}$ → After adding, our next 'k'

$\text{fact} = \text{fact} / \text{numbers.size}();$ → Now, block size will

This $\textcircled{3}$ is after adding 3 element to ans
 $\text{size} = 2$ Hence we have to divide $6/3 = 2$

```
}
```