Recursion

-By Nikhil Kumar

https://www.linkedin.com/in/nikhilkumar0609/

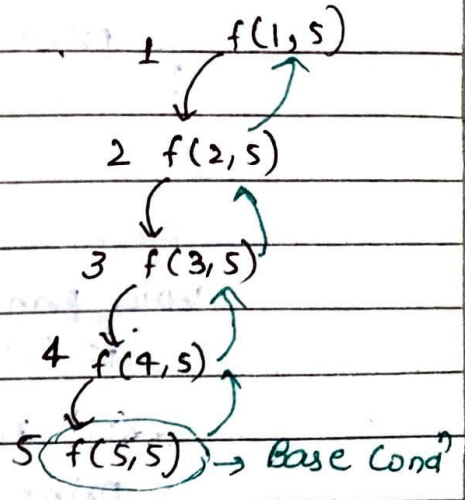
RECURSION

It is the method to call a function directly or indirectly. The function which is called itself is called a recursive function.

Q Print linearly 1 to N.

```
→ void print(int i, int n){
    if (i>n)
        return;
    cout << i << endl;
    print(i+1, n);
  }
```

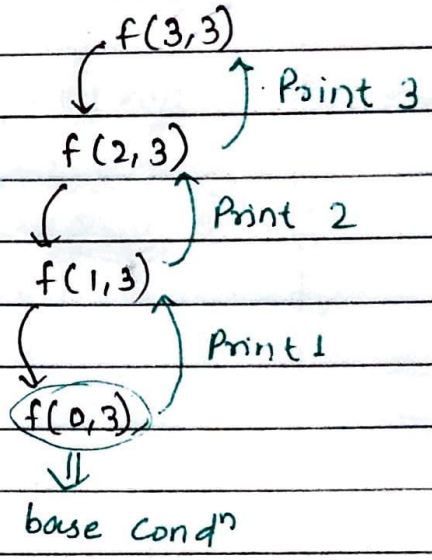
Let i=1, n=5



→ Backtracking

```
void print(int i, int n){
    if (i<1)
        return;
    print(i-1, n);
    cout << i << endl;
  }
```

Let i=1, n=3

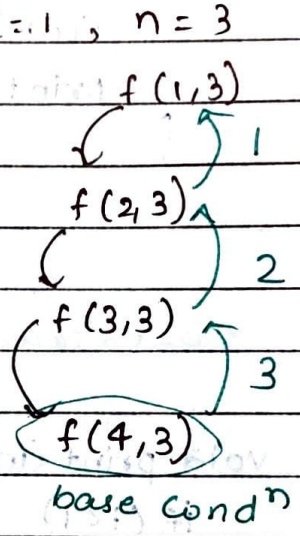


Q. Print linearly in reverse order (N to 1)

→
```
void print(int i, int n){
    if (i<1)
        return;
    cout << i << endl;
    print(i-1, n);
}
```

→ Backtracking

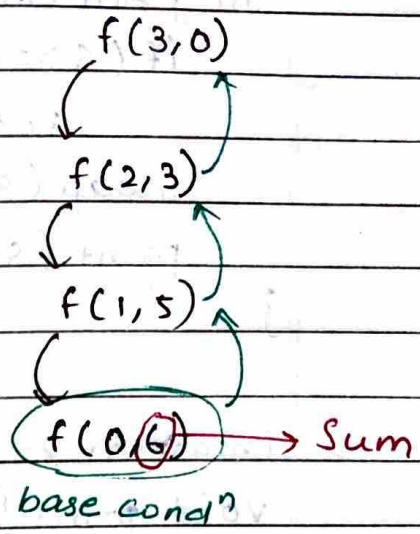
```
void print(int i, int n){
    if (i>1)
        return;
    print(i+1, n);
    cout << i << endl;
}
```



Q Print Sum of first n natural numbers.

→ using Parameterized recursion

```
void print (int i, int n){
    if (i<1) {
        cout << sum;
        return;
    }
    print (i-1, sum+i);
}
int main () {
    print (3,0);
}
```



→ using functional recursion

```
int print (int n) {
    if (n==0)
        return 0;
    return n+print (n-1);
}
int main () {
    cout << print (3);
}
```

return 3 + print(2) 3+3 = 6

return 2 + print(1) 2+1 = 3

return 1 + print(0) 1+0 = 1

return 0

Q Reverse array using Recursion.

Program 1 :-

```
int print (int s, int e, int arr[]){
    if (s >= e)
        return;
    swap (arr[s], arr[e]);
    print (s+1, e-1, arr);
}
```

Program 2 :- (Using Single Pointer).

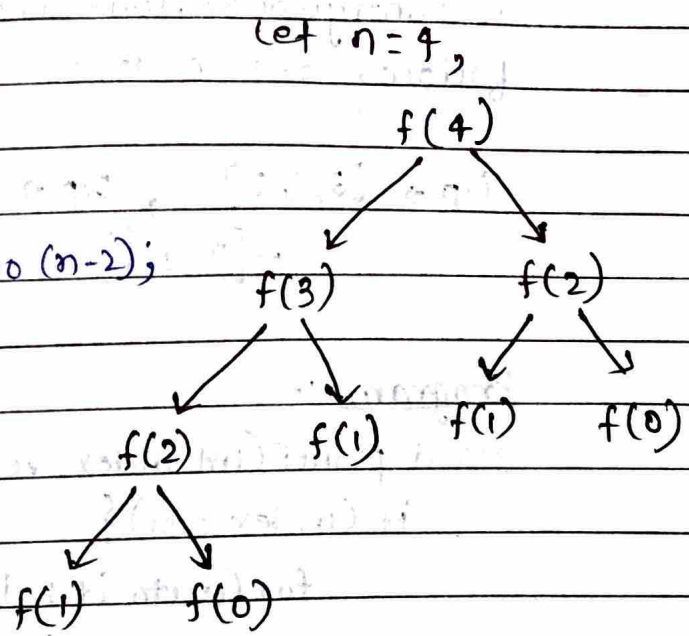
```
void print (int i, int n, int arr[]){
    if (i >= n/2)
        return;
    swap (arr[i], arr[n-i-1];);
    print (i+1, n, arr);
```

Q Check if a string is palindrome.

```
bool print (int i, int n, string s){
    if (i >= n/2)
        return true;
    if (s[i] != s[n-i-1])
        return false;
    return print (i+1, n, s);
}.
```

Q Find n^{th} Fibonacci number using recursion.

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



T.C → $2^n \times n$

S.C → O(N)

Q Print all Subsequences

A Contiguous/non-contiguous sequences, which follows the order.

i/p → [3,1,2] , o/p → [{}, 3, 1, 2, 3 1, 1 2, 3 2, 3 1 2]

~~→ [3 2 1]~~

Program :-

```
void printF(int index, vector<int>&ds, int arr[], int n){
    if(index == n){
        for(auto it : ds){
            cout << it << " ";
        }
        if(ds.size() == 0){
            cout << "{}";
        }
        cout << endl;
        return;
    }
    //not Pick
    printF(index+1, ds, arr, n);
    //Pick
    ds.push_back(arr[index]);
    printF(index+1, ds, arr, n);
    ds.pop_back();
}
int main(){
    int arr[] = {3,1,2}, int n = 3;
    vector<int> ds;
    printF(0, ds, arr, n);
    return 0;
}
```

Q Printing Subsequences whose Sum is K. (Sum)

```
void printSum(int index, vector<int> &ds, int sum, int K,
              int arr[], int n){
    if(index == n){
        if(sum == K){
            for(auto it : ds){
                cout << it << " ";
            }
            cout << endl;
        }
        return;
    }
    // Pick
    ds.push_back(arr[index]);
    sum += arr[index];
    printS(index+1, ds, sum, K, arr, n);

    sum -= arr[index];
    ds.pop_back();
    //not Pick
    printS(index+1, ds, sum, K, arr, n);
}
int main(){
    int arr[] = {1,2,3}, n = 3, K = 2;
    vector<int> ds;
    printS(0, ds, 0, K, arr, n);
    return 0;
}
```

Q. Print any subsequence whose Sum is K.

```
bool printSum(int index, vector<int>& ds, int sum, int K,
              int arr[], int n){
    if(index == n){
        //condition satisfied
        if(sum == K){
            for(auto it : ds){
                cout << it << " ";
            }
            return true;
        }
        //condition not satisfied
        else
            return false;
    }
    //pick
    ds.push_back(arr[index]);
    sum += arr[index];
    if(printSum(index+1, ds, sum, K, arr, n) == true)
        return true;
    sum -= arr[index];
    ds.pop_back();

    //not pick
    if(printSum(index+1, ds, sum, K, arr, n) == true)
        return true;

    return false;
}
```

Q Count the Subsequence with Sum = K.

```
int CountSum(int index, int sum, int K, int arr[], int n){
    if (index == n){
        // condition satisfied
        if(sum == K) return 1;
        // condition not satisfied.
        else return 0;
    }
    sum+=arr[index];
    int left = countSum(index+1, sum, K, arr, n);
    sum-= arr[index];

    int right = countSum(index+1, sum, K, arr, n);
    return left + right;
}
```

→ Count Question Trick

```
int f(){
    // base case
    return 1 → cond^n satisfies
    return 0 → cond^n not satisfies.

    left = f()
    right = f()        } for 2 recursion calls.
    return left + right;
}
```

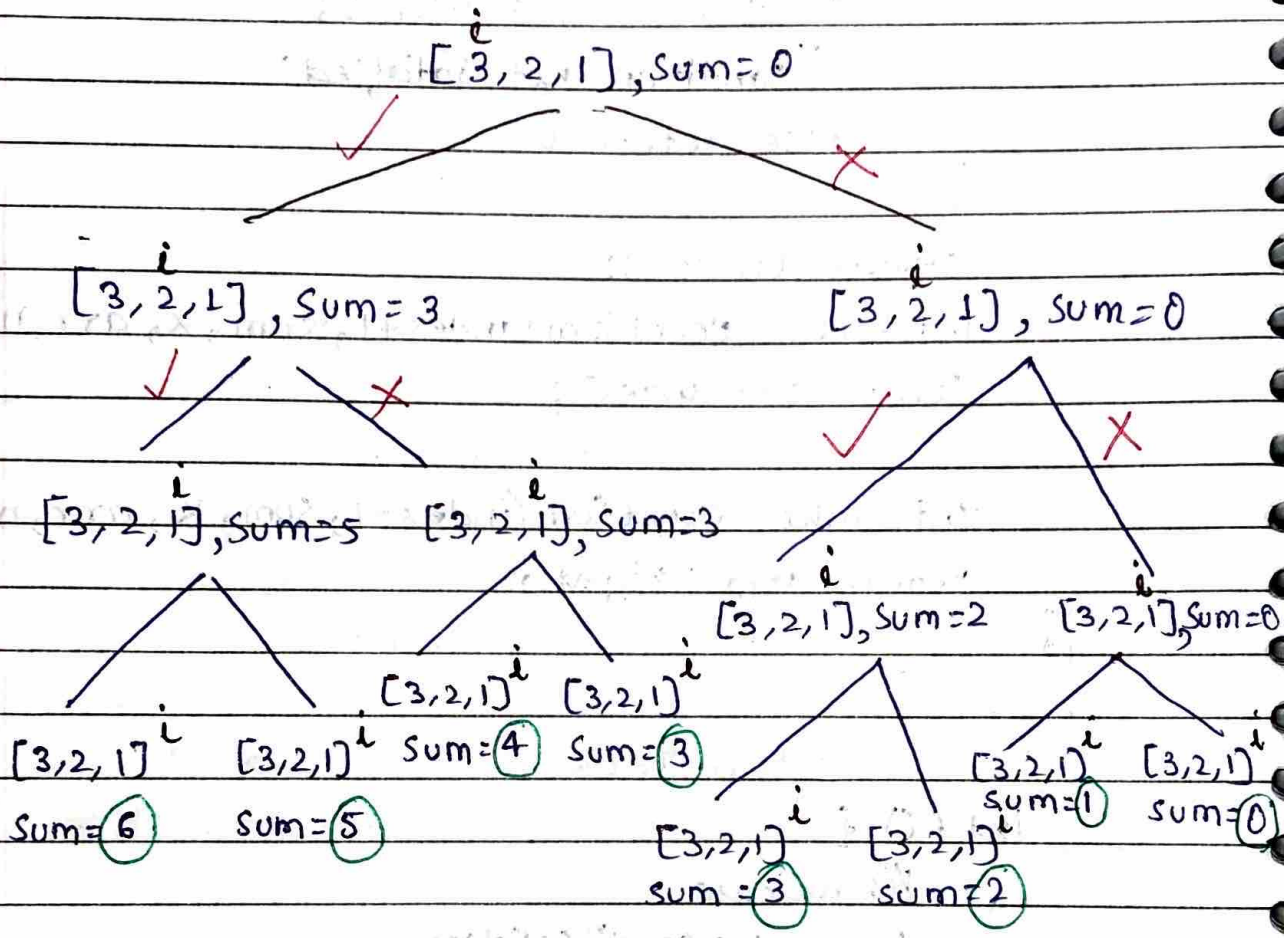
→ for n recursion call

```
s = 0
for(i = 1 → n)
    s+ = f();
return s;
```

Q Subset Sum : sum of all subsets

e.g.→ i/p → N = 3, arr[] = {5, 2, 1}

o/p = 0, 1, 2, 3, 5, 6, 7, 8



T.C → $O(2^n) + O(2^n \log 2^n)$

- $O(2^n)$ → for n index, Pick/not Pick
- $O(2^n \log 2^n)$ → for sorting 2^n elements.

S.C → $O(2^n)$, 2^n subsets can be generated for an array of size n.

Program :>

```
void solve (int index, int sum, int N, vector<int>&arr,
            vector<int>&ans) {
    //base case
    if (index == N){
        ans.push_back(sum);
        return;
    }
    //pick
    solve (index+1, sum+arr[index], N, arr, ans);
    //not Pick
    solve(index+1, sum, N, arr, ans);
}

vector<int> subsetSums (vector<int> arr, int N) {
    vector<int> ans;
    solve (0, 0, N, arr, ans);
    sort (ans.begin(), ans.end());
    return ans;
}
```

T.C → $O(2^n * k \log(x))$ ∵ $k \log(x)$ to insert every combination of avg. length k in a set of size x.

S.C → $O(2^n * k)$

Subset - II (Print all the Unique Subsets).

i/p :→ arr[] = [1,2,2]

o/p = [[], [1], [1,2], [1,2,2], [2], [2,2]]

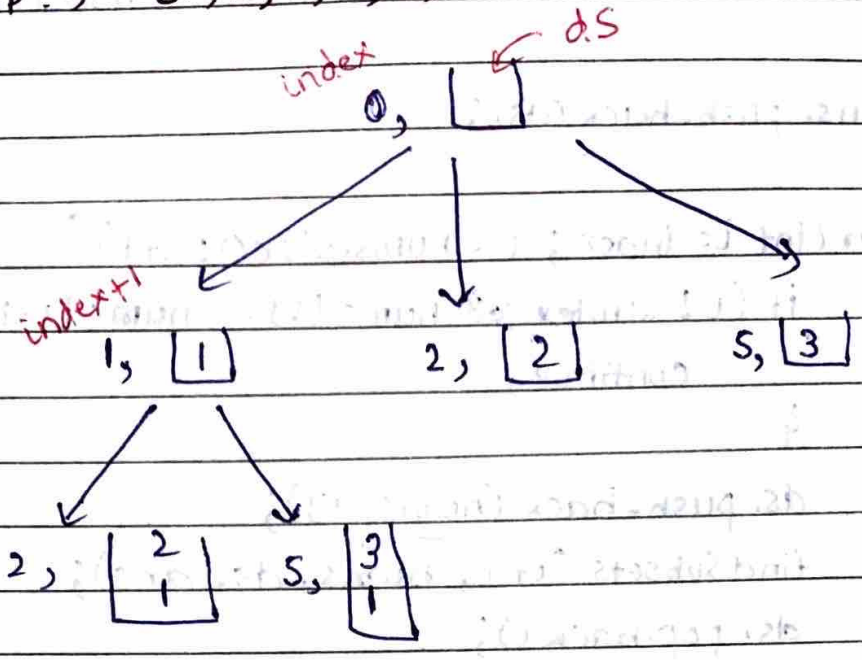
Approach 1 :→ At first generate all possible combinations, then we use a set to store all the combinations that will discard the duplicates.

Program :→

```
void solve(vector<int>& nums, int index, vector<int> ds,
           set<vector<int>>& res){
    if(index == nums.size()){
        sort(ds.begin(), ds.end());
        res.insert(ds);
        return;
    }
    ds.push_back(nums[index]);
    solve(nums, index+1, ds, res);
    ds.pop_back();
    solve(nums, index+1, ds, res);
}

vector<vector<int>> subsetDup(vector<int>& nums){
    vector<vector<int>> ans;
    set<vector<int>> res;
    vector<int> ds;
    solve(nums, 0, ds, res);
    for(auto it = res.begin(); it != res.end(); it++){
        ans.push_back(*it);
    }
    return ans;
}
```

→ Approach 2 :→

```
      0  1  2  3  4  5
i/p :→ [1, 2, 2, 2, 3, 3].
```



Program :⇒

```
void findSubsets(int index, vector<int>& nums, vector<int>&ds,
                 vector<vector<int>>& ans){

    ans.push_back(ds);

    for(int i= index; i<nums.size(); i++){
        if(i != index && nums[i] == nums[i-1]){
            continue;
        }
        ds.push_back(nums[i]);
        findSubsets(i+1, nums, ds, ans);
        ds.pop_back();
    }
}
vector<vector<int>> subsetDup(vector<int>& nums){
    vector<vector<int>> ans;
    vector<int> ds;
    sort(nums.begin(), nums.end());
    findSubsets(0, nums, ds, ans);
    return ans;
}
```

T.C → $O(K * 2^n)$, O^{2n} for generating every subset and $O(K)$ to insert every subset in ds, if avg. length of every subset is K.

S.C → $O(2^n * K)$

Q Combination Sum

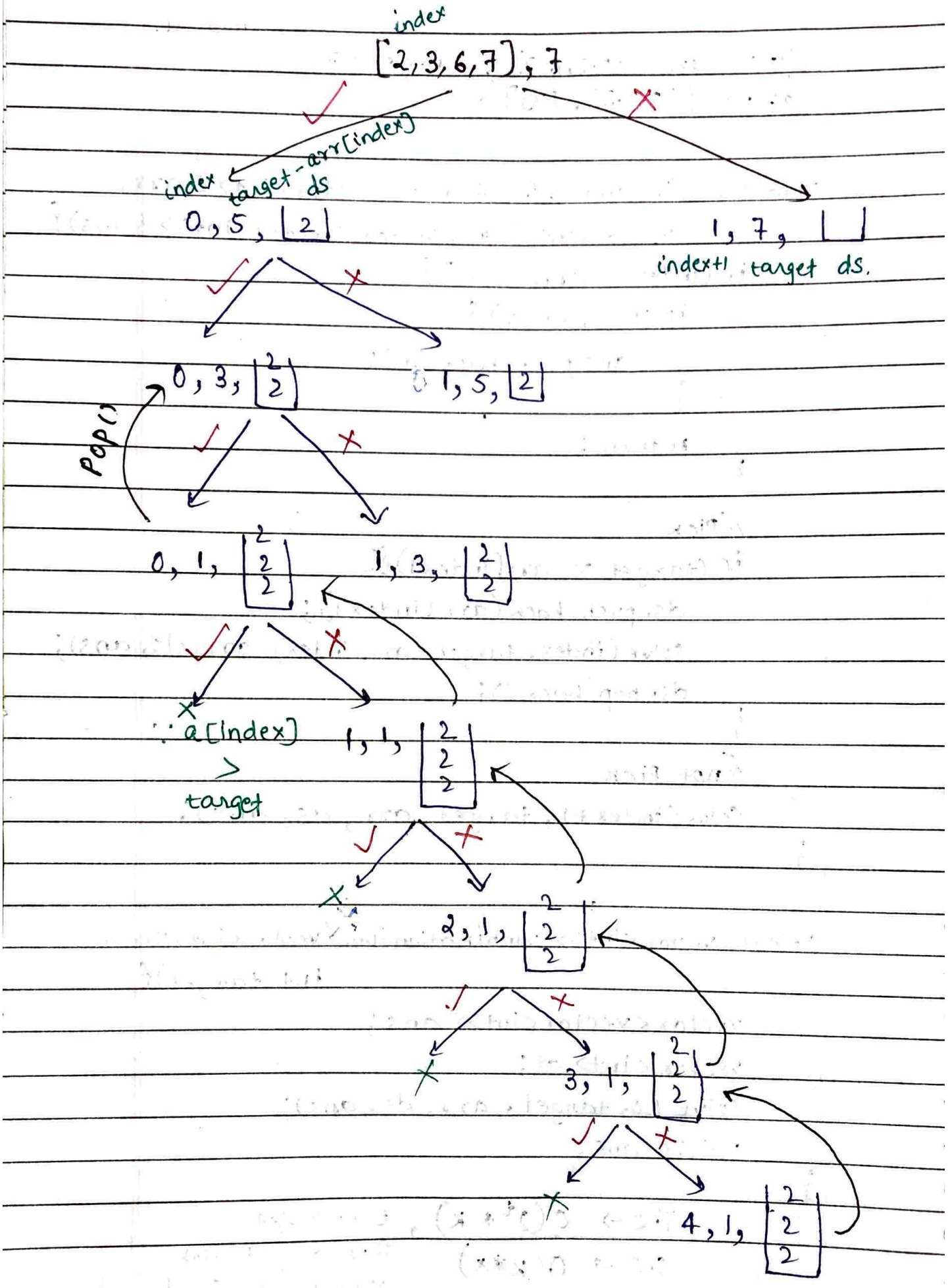
i/p :→ arr = [2,3,6,7], target = 7

o/p :→ [[2,2,3], [7]]

```
void solve(int index, int target, vector<int>& arr,
           vector<int>& ds, vector<vector<int>>& ans){
    if(index == arr.size()){
        if(target == 0){
            ans.push_back(ds);
        }
        return;
    }
    //Pick
    if(target >= arr[index]){
        ds.push_back(arr[index]);
        solve(index, target - arr[index], arr, ds, ans);
        ds.pop_back();
    }
    //not Pick
    solve(index+1, target, arr, ds, ans);
}

vector<vector<int>> combinationSum(vector<int>&arr,
                                   int target){
    vector<vector<int>> ans;
    vector<int> ds;
    solve(0, target, arr, ds, ans);
    return ans;
}
```

T.C → $O(2^t * k)$, t → target, k → avg. length

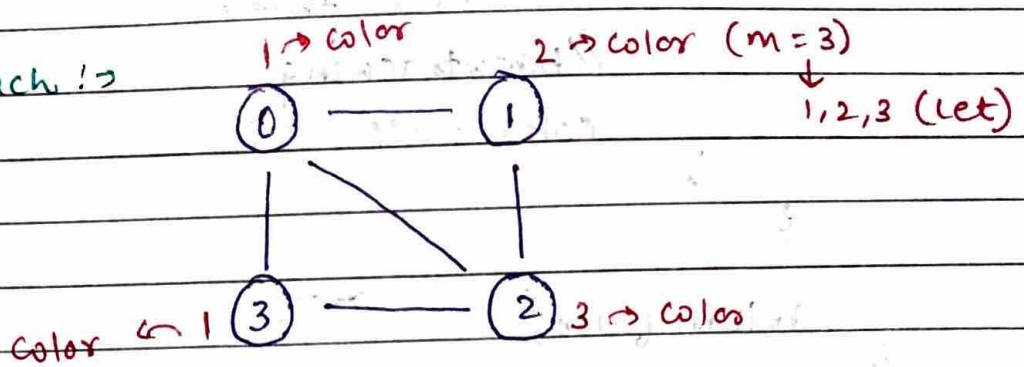
S.C → $O(k*x)$, x → no. of combinations.



Q M-Coloring Problem

i/p :→ n = 4, m = 3, E = 5
edges[] = {(0,1), (1,2), (2,3), (3,0), (0,2)}

Approach :→



Program :→

```
bool isSafe(int node, int color[], bool graph[101][101],
            int n, int col){
    for(int k=0; k<n; k++){
        if(k != node && graph[k][node] == 1 &&
           color[k] == col){

            return false;
        }
    return true;
}

bool solve(int node, int color[], int m, int n,
           bool graph[101][101]){
    //base case
    if(node == n)
        return true;
```

col → color

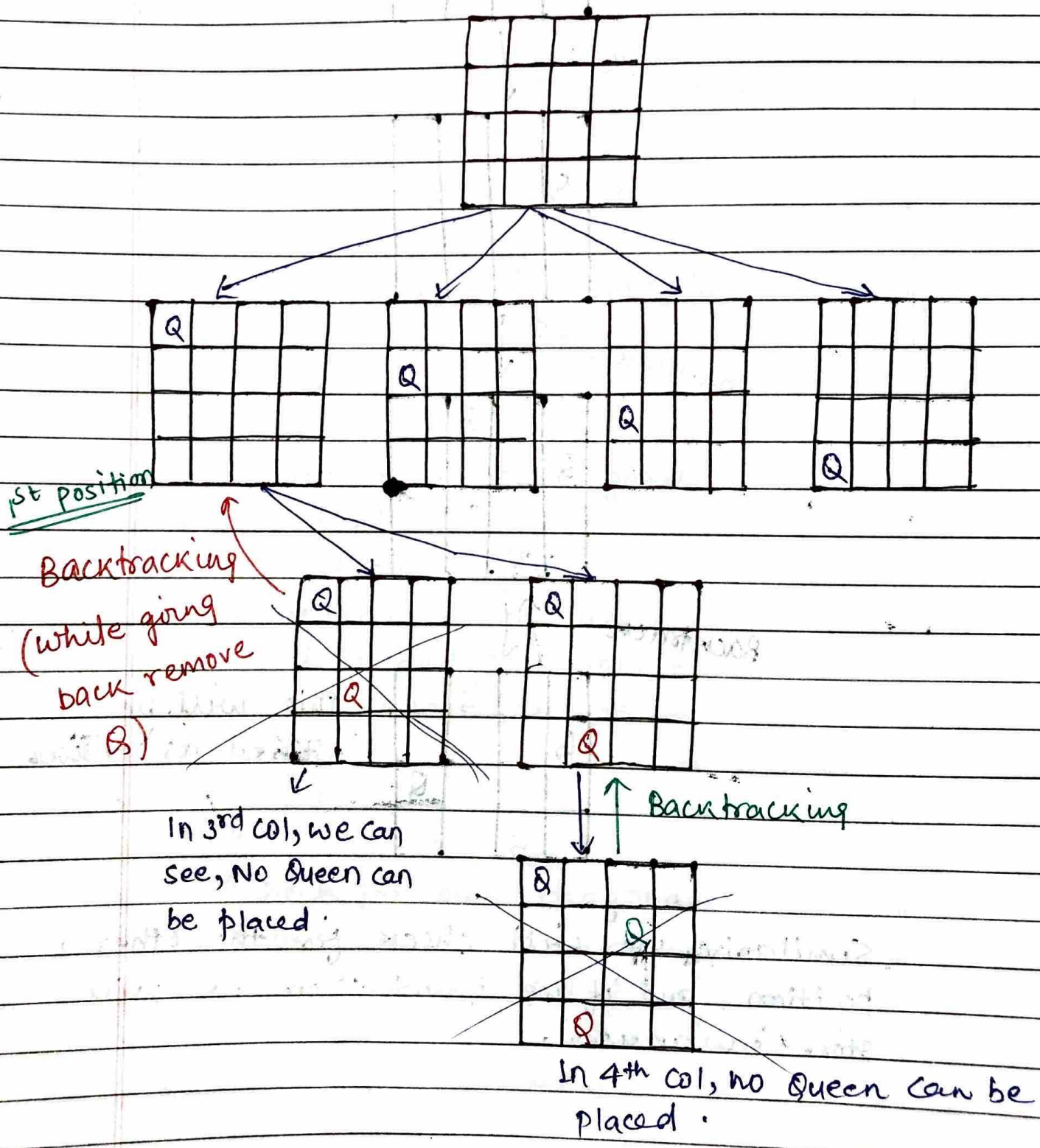
```
for(int i=1; i<=m; i++){
    if(isSafe(node, color, graph, n, i)){
        color[node] = i;
        if(solve(node+1, color, m, n, graph))
            return true;
        //backtracking
        color[node] = 0;
    }
}
return false;
}

bool graphColoring(bool graph[101][101], int m, int n){
    int color[n] = {0};

    if(Solve(0, color, m, n, graph))
        return true;
    return false;
}
```

T.C → $O(N^m)$

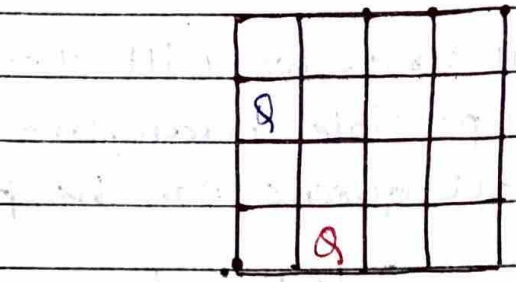
S.C → $O(n)$

Q N-Queens

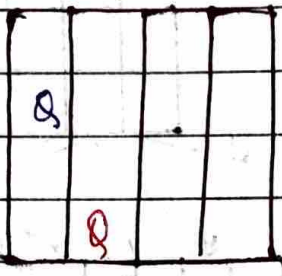
Approach :- At first, we will check for column 1 with all the possible arrangement in every row, where all queens can be placed.



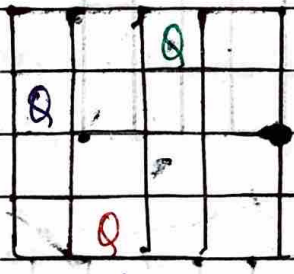
2nd position.



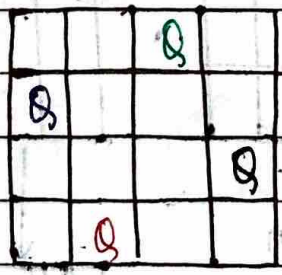
Backtrack



Backtrack



Backtrack



This will be stored as answer.

Similarly, we will check for the other 2 position, and if we found then we will store in answer..

T.C → $O(N^3)$

S.C → $O(N^2)$

Program :→

```
bool isSafe(int row, int col, vector<string> board, int n){

    int dupRow = row;
    int dupCol = col;

    //check for row
    col = dupCol;
    row = dupRow;
O(n) while (col >=0) {
        if (board [row] [col] == 'Q')
            return false;
        col--;
    }
    //check for upper diagonal
    col = dupCol;
    row = dupRow;
O(n) while (row >=0 && col >= 0){
        if (board [row] [col] == 'Q')
            return false;
        row--;
        col--;
    }
    //check for lower diagonal
    row = dupRow, col = dupCol;
    while (row <n && col >=0){
O(n)    if (board [row] [col] == 'Q')
            return false;
        row++, col--;
    }
```

```
    return true; //if none of three cond^n triggered
}                          ↓
                      means safe

void solve(int col, vector<string>& board,
           vector<vector<string>>& ans, int n){
    if(col==n){
        ans.push_back(board);
        return;
    }
    // Solve 1 case, baaki recursion dekh lega
    for(int row=0; row<n; row++){
        if(issafe(row, col, board, n)){
            // If placing Queen is safe
            board[row][col] = 'Q';
            solve(col+1, board, ans, n);
            // backtracking
            board[row][col] = '.';
        }
    }
}

vector<vector<string>> solveNQueens(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;
}
```

Program 2 :>

```
void solve (int col, vector<string> & board,
            vector<vector<string>> & ans,
            vector<int>& leftRow, vector<int>& upperDiagonal,
            vector<int> & lowerDiagonal, int n){

    //base case
    if(col == n){
        ans.push_back(board);
        return;
    }
    for(int row = 0; row < n ; row++){
        if(leftRow[row] == 0 &&
           lowerDiagonal[row + col] == 0 &&
           upperDiagonal[n-1 + col - row] == 0 ){

            board[row][col] = 'Q';
            leftRow[row] = 1;
            lowerDiagonal[row + col] = 1;
            upperDiagonal[n-1 + col - row] = 1;
            solve(col+1, board, ans, leftRow,
                  upperDiagonal, lowerDiagonal, n);
            //backtracking
            board[row][col] = '.';
            leftRow[row] = 0;
            lowerDiagonal[row + col] = 0;
            upperDiagonal[n-1 + col - row] = 0;
        }
    }
}
```

```
Vector<vector<string>> solveNQueens(int n){
    vector<vector<string>> ans;
    vector<string> board(n);
    string s(n, '.');
    for(int i=0; i<n; i++){
        board[i] = s;
    }
    vector<int> leftRow(n,0);
    vector<int> upperDiagonal(2*n-1, 0);
    vector<int> lowerDiagonal(2*n-1, 0);

    Solve(0, board, ans, leftRow, upperDiagonal,
          lowerDiagonal, n);
    return ans;
}
```

T.C → $O(N^2)$

S.C → $O(N^2)$

Sudoku Solver

```
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;
}

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))
                            return true;
                        else
                            board[i][j]='.';
                    }
                }
                return false;
            }
        }
    }
```

```
        for
        .
        .
        }
    return true;
}

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

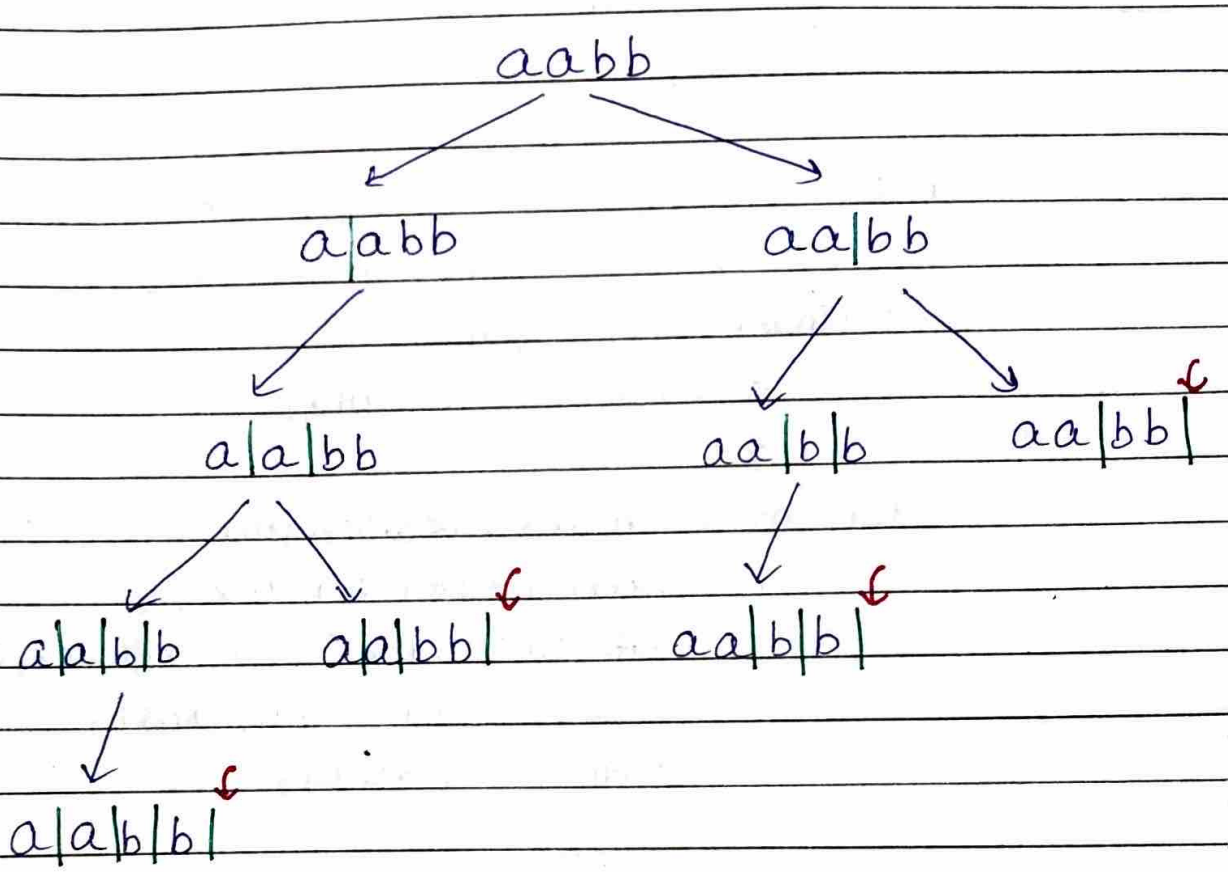
T.C → $O(9^{n^2})$

S.C → $O(1)$

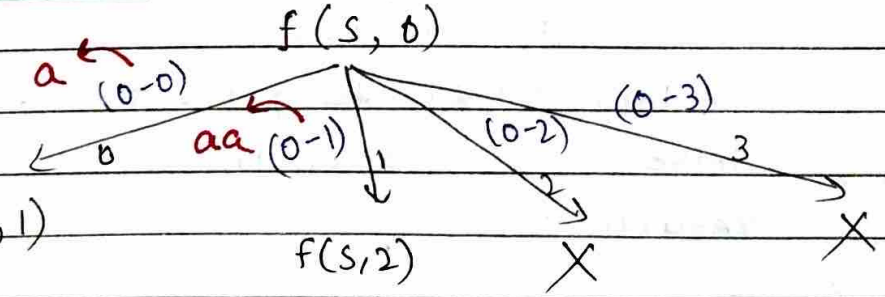
Q Palindrome Partitioning

i/p :→ s = "aabb"

o/p = [[a, a, b, b], [a, a, bb], [aa, b, b] [aa, bb]]



Pseudo code :-



Code :-

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

$T.C = O((2^n) * K * (n/2))$, $S.C = O(K*x)$

$O(2^n)$ → to generate every substring

$O(n/2)$ → to check if substring generated is palindrome.

$O(K)$ → inserting palindrome in data structure, where K is length of Palindrome list.

Leetcode 60.

Q = K-th Permutation Sequence

i/p :→ n=3, K=4

o/p :→ "231"

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

Brute-Force

Generate all the possible permutations using recursion. ⇒ O(N!)

and store all in a data structure (vector) ⇒ O(N)

Sort the data structure and return kth sequence from it. ⇒ O(N! log N!)

T.C = O(N! * N) + O(N! log N!)

S.C = O(N)

Optimised Approach :→

e.g.→ n=4, K=17 3 4 1 2

0-based indexing, K = 17-1 = 16.

3! = 6

0th block ⇒ 1, {2,3,4} 0-5
1st ⇒ 2, {1,3,4} 6-11
2nd ⇒ 3, {1,2,4} 12-17 ✓
3rd ⇒ 4, {1,2,3}. 18-23

16th Permutation will lie in (16/6) = 2nd block.

And now, K = (16 % 6) = 4th sequence.

$2! = 2$

0th block → 1 , {2,4}
1st → 2 , {1,4}
2nd → 4 , {1,2}. ✓

4th permutation will lie in (4/2) = 2nd block.
K = 4%2 = 0 sequence

$1! = 1$

0th block → 1 → {2} ✓
1st → 2 → {1}

0th permutation will lie in (0/1) = 0th block.
K = 0

0th block → 2 → { }. ✓

Code :-

```
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);
    }
    numbers.push_back(n);
    k = k-1;
    string ans = "";

    while (true) {
        ans += to_string (numbers [k/fact]);
        numbers.erase (numbers.begin() + k/fact);

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

        k = k % fact;
        fact = fact / numbers.size();
    }
    return ans;
}
```

T.C = $O(N) * O(N) = O(N^2)$

for placing N numbers in N position = O(N)
for erasing = O(N).
↳ search