3) Recursion :-

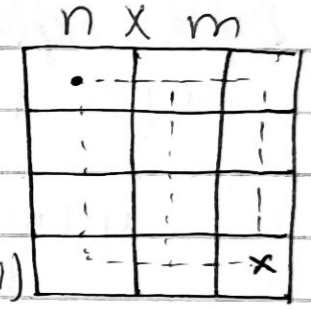
i) Steps for recursion :-

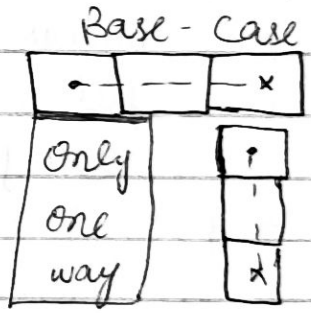
- Find the base case.
- Find the relation b/w problem and sub-problem.
- Generalize the relation.

•) No. Of ways to reach from 1 point to end in n * m :-

```
function countWays (rows, col)
{
    if (row == 1 || col == 1)
        return 1;
    return countWays (row-1, col) + countWays (row, col-1)
}
```

→ Base case n = 1 ; return 1

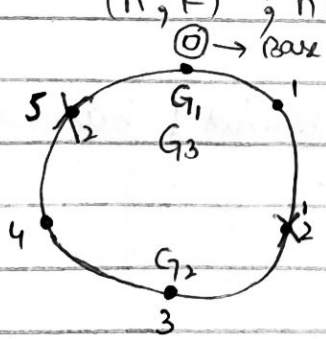




- If we have only 1 row then their will be only one way. Similar to column.

• Josephus Problem :- (Death Game)

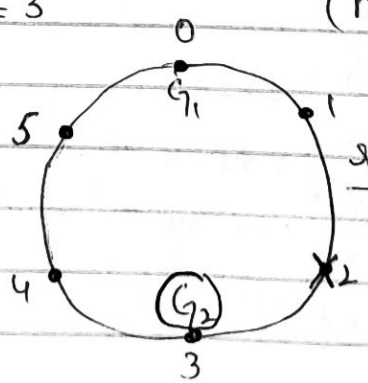
(n, K) ; n = 5, K = 3



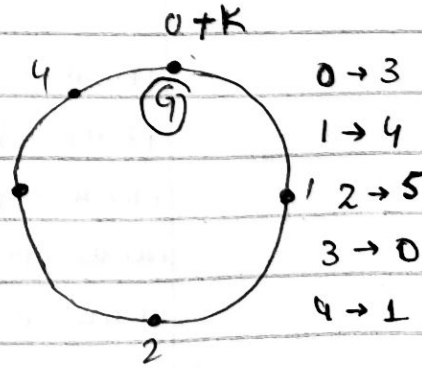
→ Base - case

```
if n == 1
    return 0;
```

(n-1, K)



0 + K



∴ relation b/w $f(n,K) = (f(n-1,K)+K)\%n$

- **Palindrome String :-**

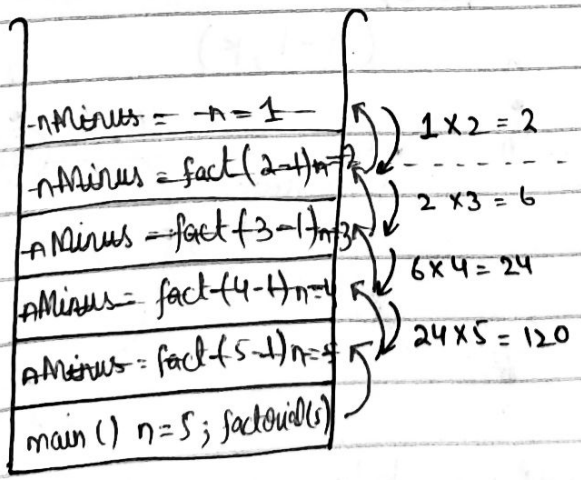
Base Case
Left == Right

```
function stePal(string, left, right)
{
    if (left == right) return true;
    if (string[left] != string[right]) return false;
    return stePal(string, left+1, right-1);
}
```

- **Depth Explanation of recursion :-**

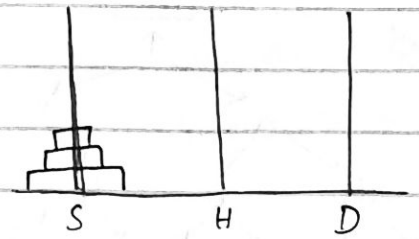
```
1. function factorial(n)
2. {
3.     if (n==1 || n==0)
4.         return 1;
5.     nMinus = factorial(n-1);
6.     fact = n * nMinus
7.     return fact;
8. }
```

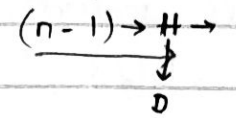
Stack :-

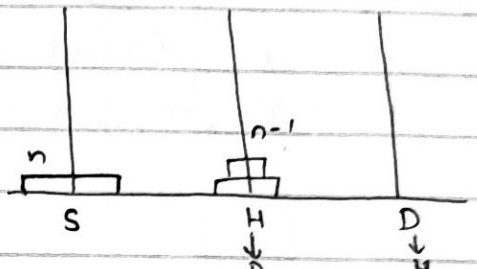


..... as it reaches 1 basecase hits.

• Tower of Hanoi :-

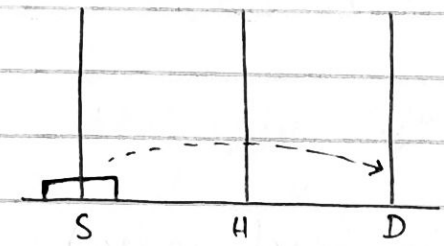
① 



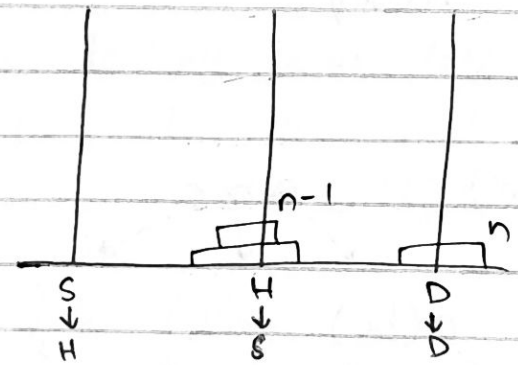


from source to Helper we used H → D and D → H in (n-1)

② BaseCase n == 1



③



In this case helper is used as source and source as helper.

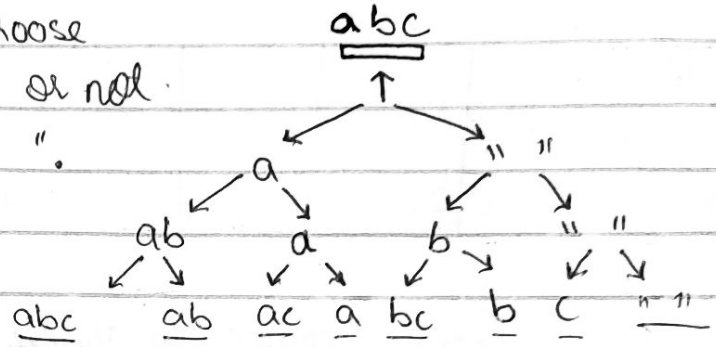
Code

```
function towerOfHanoi(n, s, h, d){
    if (n==1) console(plate from s → d ); return;
    towerOf Hanoi (n-1) source, destination, helper );
    console.log( ... )
    tower of Hanoi ( n-1) helper, source, destination);
```

- **Print subsequences of string :-**

we have eight substrings to choose if next element will include or not.

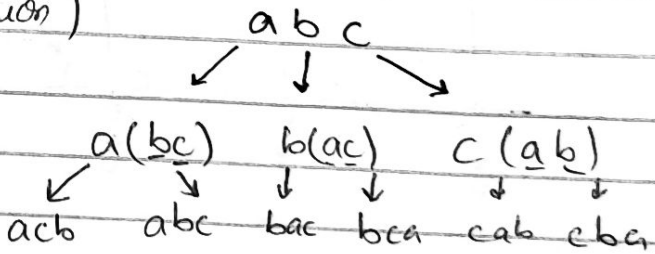
abc, ab, ac, a, bc, b, c, " ".



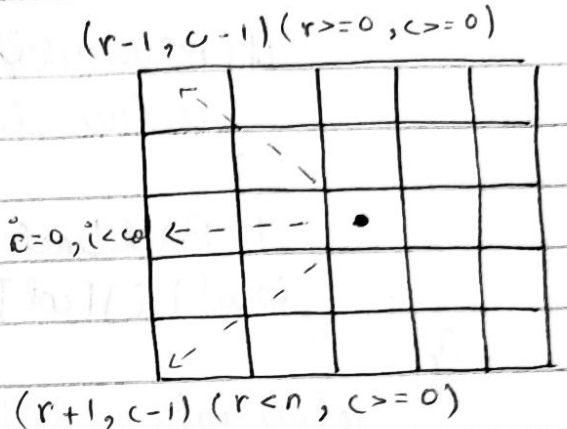

```
function subsequences (string, current = " ", index = 0)
{
    if (index == string.length) {
        console.log (current)
        return;
    }
    subsequences (string, current + string[index], index+1);
    subsequences (string, current, index+1);
}
```

- **Print permutations of string :-**

```
function strPermutation (string, permutation)
{
  if( string.length == 0)
     console.log (Permutation)
     return;
  for( let i=0; i< string.length; i++)
     {
       curChar = string[i];
       string2 = string.slice (0,i) + (i+1)
       strPermutation ( string2, permutation + curChar);
     }
}
```



- **N-Queens Problem :-**

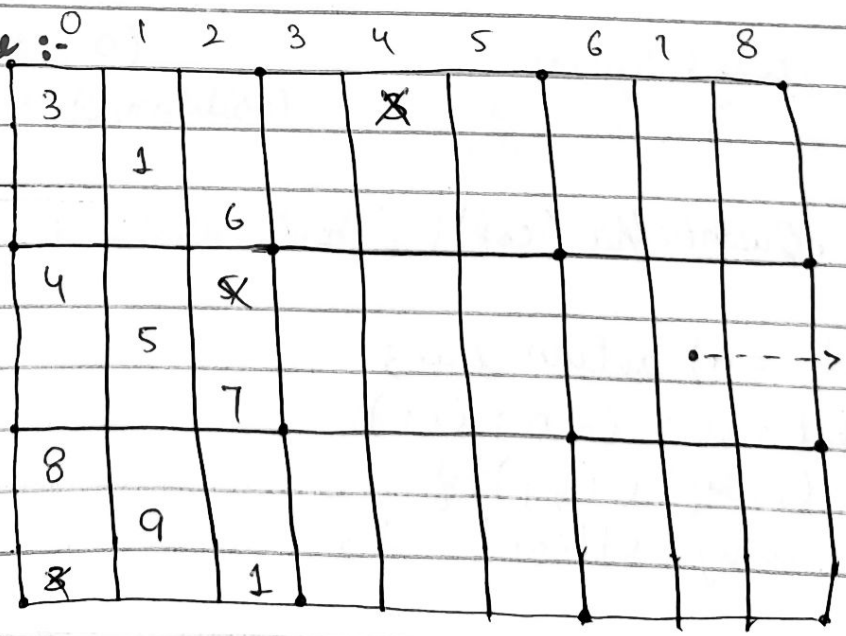



```
Let n = 4;
Let Board = new Array(n).fill(0).map
            (() => new Array(n).fill(0))

function isSafe(row, col)
{
  // checking column wise.
   for(let i=0; i<col; i++)
      if(board[row][i]) return false;

  // checking upper left side
   for(let i=row; j=col; i>=0 && j>=0; j--, i--)
      if(board[i][j]) return false;

  // checking lower left side
   for(let i=row; j=col; i<n && j>=0; i++, j--)
      if(board[i][j]) return false;

   return true;
  }

function nQueenHelper(col)
{
   if(col == n) return true;
   for(let i=0; i<n; i++)
      if(isSafe(i, j)){
         array[i][col] = 1;
```

```
        if (nQueenHelper(col+1)){
            return true;
        }
        // Backtrack if condition is false
        board[i][col] = 0;
    }
    return false;
}

function solveNQueen()
{
    if(nQueenHelper(0)){
        console.log(board[i].join(" "));
    }
    else{
        solution doesnot exist
    }
};
```

- **Sudoku Solver :-**

rules
col, grid and row must not have the same no.



```
Let grid = 9 x 9;

function isSafe (grid, row, col, num)
{
  // check row and column
  for (let x = 0; x < 9; x++)
     if (grid[x][col] == num || grid[row][col] == num)
        return false;

  // check grid 3 x 3
  Let nRow = row - (row % 3);
  Let nCol = col - (col % 3);
  for (i = 0; i < 3; i++)
     for (j = 0; j < 3; j++)
        if (grid[i + nRow][j + nCol] == num)
              return false;

  return true;
}

function solveSudokuHelper (grid)
{
  Let row = 0, col = 0, emptySpace = false
  for (let i = 0; i < 9; i++)
     for (let j = 0; j < 9; j++)
        if (grid[i][j] == 0)
           row = i, col = j;  emptyspace = true
           break;
     if (emptyspace)
        break;
```

```
for (let num = 0; num <= 9; num++)
    if (isSafe(grid, col, row, num){
        grid[row][col] = num;
        if (solveSudokuHelper(grid))
            return true
        else // backtrack
            grid[row][col] = 0;
    }
    return false;
```