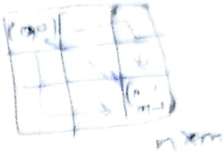
BACKTRACKING :-

Before discussing
→ Revise recursion
→ Divide & Conquer

Types of Backtracking :-

1) Decision Ex:- grid
2) optimization
3) Enumeration.



Backtracking - Arrays

```
public static void change Arr (int arr[], int i, int val)'

    // base case
    if (i == arr.length)
        {                                  TC = O(n)
        printArr (arr);                    SC = O(N)
        rtn;
    }

  //recursion.

    arr[i] = val;
    change Arr (arr, i+1, val+1);
        arr[i] = arr[i] - 2 // back tracking.

public static print Arr (int arr[])
    {
    for (int (i = 0 (i < n ; i++;)){
        syso (print (arr[i]);
    }

PSV main SArgs ( )
    {
    int arr[] = new int [5];
    int val = 1;
    print Arr (Arr);
    change Arr (arr, 0, 1);

Output→   1 2 3 4 5     starting
         -1 0 1 2 3     back tracking.
```

Find Subsets :-

Find & print all subsets of a given string'

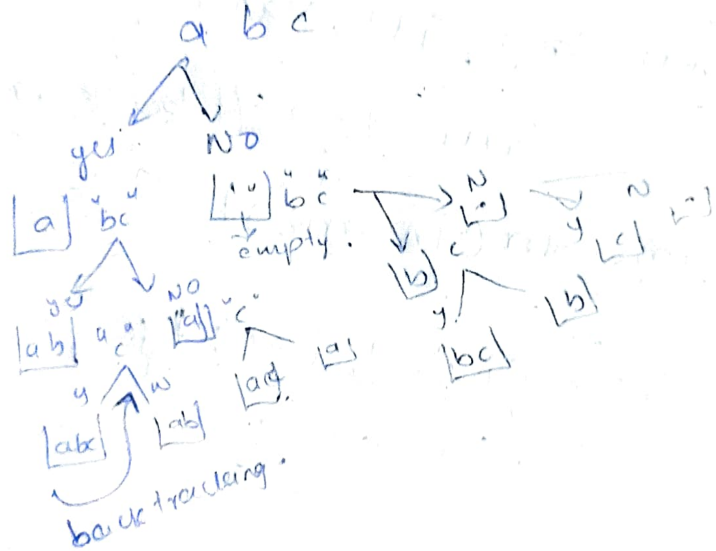
"abc":

a, b, c, ab, bc, ac, abc, "" emptyset
null set
ϕ

Total = 8

no of subsets = 2^n
$n=2 \rightarrow 2^2 = 4$

Aproach :-



find subset (str, ans) {

→ BC
print ans

{ ans + str }

```
public static void findsubset (strig str, string ans,
                                        int i) {

    // base case.
  if ( i = = str.length()) {
       syso( ans);
         return;

  }
   // yes choice recursion
  findsubsets ( arr, ans+str.charAt(i), i+1)
      // no choice recursion

  findsubsets ( arr, ans , i+1);

 }
 public static void main {

 String  str = "ab.c";
                          ans,  i);
  find subsets ( str,
                  str,   "  " , 0
```

Time Complexity = O(n x 2^n)

SC = O(n)

Find permulations :-

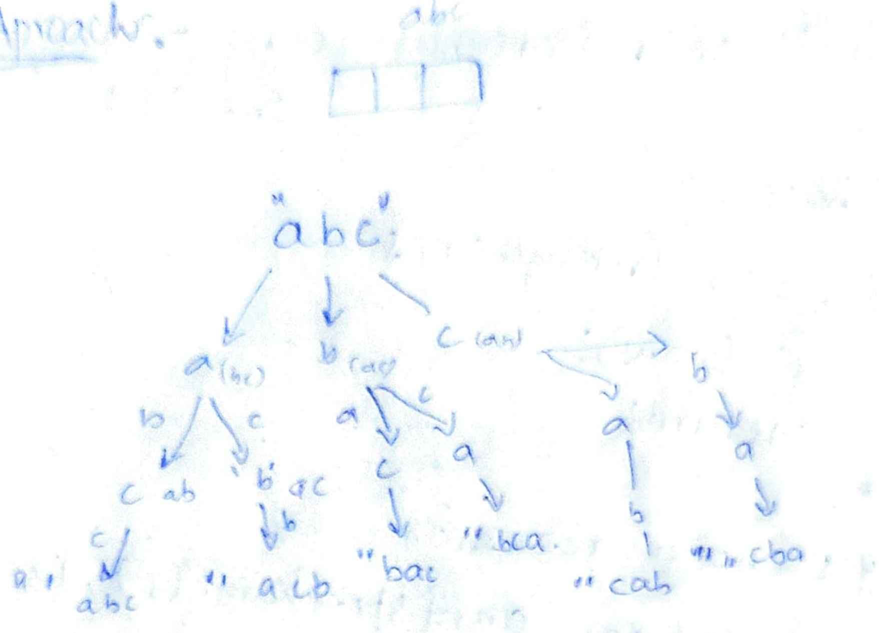
find all the permutation's of string.

"abc" → for n elements permulation = n!
array length = n → n! ways

"abc"

abc , acb, bac , bca, cab, cba.

Aproach.-



Code :-

```
public static void fidpermutation(String str, String ans)
    {
    // Base case
    if (str.length ( ) = =0) {
          sysO (ans);
    rtn; }
  // recursion
    for (int i=0; i< str.length ; i++) {
          int char curr = str.charAt(i);
String NStr = str.substring (0, i) + str.substring (i+1, str.length)
                      New
        find permut (str, ans+curr);
    }
  }

  psvmain ( ) {
          String str = "abc";
            find permutation (str, "");
      }
   }
```

Time Complexity = O (n×n!)
ang.

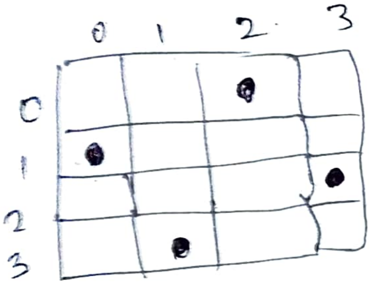
N-Queens problem

place N- queens on a N X N chessboard such that
queens can attack each other.

all. sollutions → count sollutions.

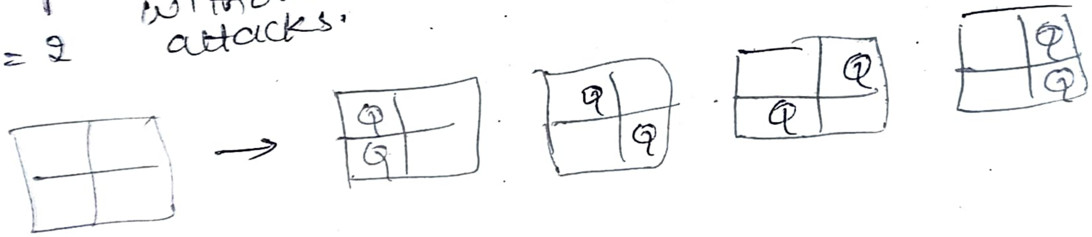
yes/no → sollution.

N=4



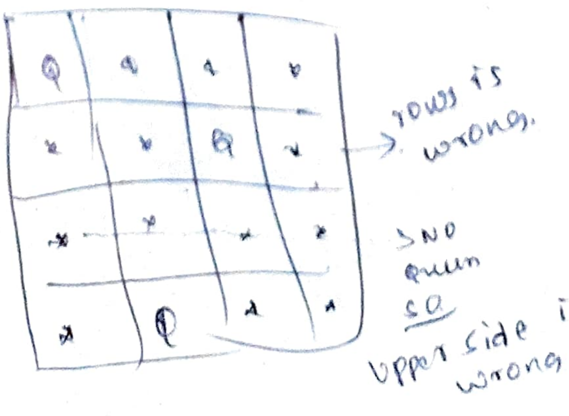
Logical work :-

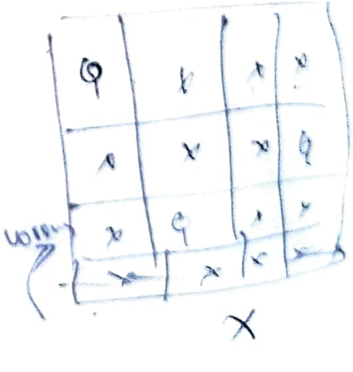
N = 2 without attacks.



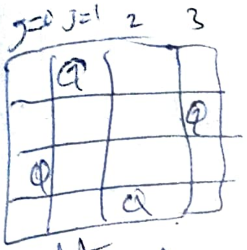
```
Public static void. n Queens (char board[][], int row)

   // base case
   if ( row == board. length) {
        printBoard ( board);
        rtn;
   }
   // column loop
   for (int j=0 ; j < board. length ; j++){
     board[row][j] = 'Q';
     nQueens.(board, row+1) // recursion call.
     board [row][j] = '.' ; // back tracking step
   }
```

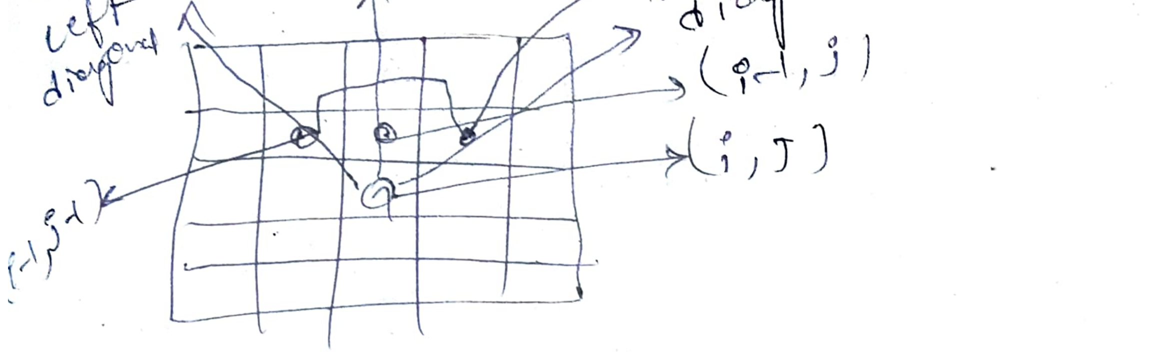
```
public static void printBoard (char Board[][],
for (int i=0 ; i< b.length; i++) {
        for (int j=0 ; j<b.length ; j++ {
                syso (board[i][j] + " ");
        }
        syso pln();
}
}
public static void main (S A) {
        int n=2 ;
        char board[][] = new char[n][n];
        // initialize
        for (int i=0; i<n; i++) {
                for (int j=0 ; j<n ; j++) {
                        board[i][j] = "." ;
                }
        }
        nQueens (board, 0);
}
}
```





n Queens p. safely.
n rows





<u>In column Loop</u> :-

```
for(int j=0 ; j< board.len; j++) {            if true
    if (issafe (board, row, J)) {
        .board [row][j] = 'Q';
        nQuens (board, row+1);
        board [row] [j] = 'x';

}
```

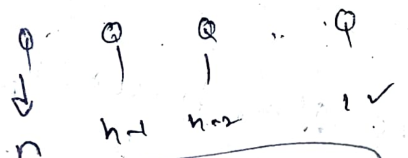
```
              boolean
public static ~~void~~ isSafe(char board[], int row,
                                            int col){

    // vertical up
    for(int i = row-1 ; i>=0 ; i--) {
        if (board[i][col]=='Q'){
            rtn false;
        }
    }

    // diag up
    for(int i= row-1, j= col-1 ; i>=0 && j>=0 ;
                                          i-- ; j--){
        if(board[i][j] == 'Q' {
            rtn false;
```

// diagonal right = (i-1 , j+1)

```
// diag right
for (int i = row-1 ; j = col+1 ; i ≥ 0 && j < board.length
                                  i-- , j++) {

        if (board[i][j] == 'Q'; {

            return false ;

        }

    }

    return true ;   // if not diagonal and vertical.

}
```

int n = 4 ; for 4/4

Time complexity :-



$\boxed{Tc = O(n!)}$ Important.

T(n) = 1 Queen place * T(n-1) + issafe()

Count the number of ways :-

// Base case

recursion → done by call by value

```
if (row == board.length) {
    count ++;
}

static int count = 0;
```

N-Queens -> print 1 solution.

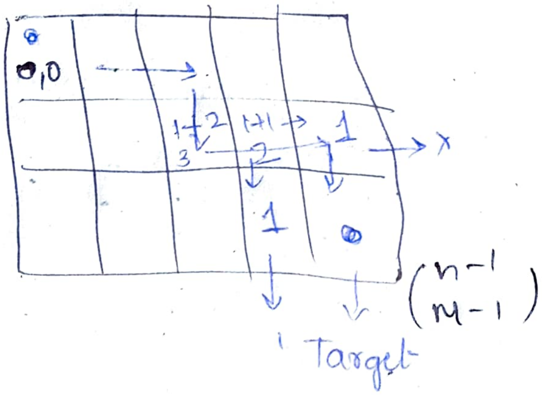
check if the problem can be solved & print only one 1 sollution to N-Queens problem.

→ place
→(n-1)Q
→ unplace (return)

```
public static boolean nQueens (char board[][], int row)

  {
    // base case
    if (row == board.length) {
            count ++;
         return true;

    //column loop
    for (int j = 0 ; j < board . length; j++) {

        if ( is safe ( board, row, j)) {
            board [row][j] = 'Q';

            if (nQueens (board, row+1)) {
                  return true;

            board[row] [j] = 'x'; // backtraking
                                     unplace
                                     step
        }

    return false;

Main() {
    if (nQueens (board,0)) {
            syso( soln is posible);
            printBoard (board); }

    else { syso ("soln is not posible");
```

GRID ways (Imp)

find number of ways to reach from (0,0) to [(N-1),(M-1)] in a N×M grid. Allowed moves right or down:



Cell (x,y)

Right (x,y+1) w1

Down (x+1,y) w2

Total ways = w1 + w2

Base case:- if we are at target

$$\underset{i\ \ j}{f(x,y)} = \underset{down}{f(x+1,y)} + \underset{right}{f(x,y+1)}$$

Code:-

```
public class X{
   public static gridways (int i, int j, int n, int m)
      // base case
      if (i == n-1 && j == m-1){ rtn 1 };
      else if (i ≥ n || j > n) { rtn 0; }
public & recursion call.
 // Left   gridways ( i, j+1 , n , m ) ; = int w1
 // down  grid ways ( i+1 , j , n , m ) ; = int w2

   return w1 + w2 ;

public static void main( S a[]) {
      int n = 3 ;
      int m = 3 ;
    syso ( gridways (0, 0 , n , m)); } }
        } }
```

Dryrun

6	3	1
3	2	1
1	1	1 2,2

Time complexity =

right turns = m
down turn = n

Total (n+m)

$TC = O(2^{n+m})$

Math trick for Linear Time :-

ways = (n-1) D
(m-1) R

$\left\{ \begin{matrix} D\,D\,D\,D \\ R\,R\,R\,R \end{matrix} \right\} \begin{matrix} n-1 \\ m-1 \end{matrix}$

total characters = (n-1+m-1)

Permutations

Repeating (n-1) D
(m-1) R

$\frac{((n-1)+(m-1))!}{(n-1)(m-1)!}$ = tot ways.

n, m ways

$= \frac{(n+m)!}{(n!\ m!)}$

n = 3 n.
m = 3

Permutations formula = $\frac{(n-1+m-1)!}{(n-1)!+(m-1)!}$ total ways (repeated)

```
public static int fact(int n) {
        if(n==0) {
            rtn 1;
        }
        rtn n*fact(n-1)
    }

public static gridways (int n, int m) {
    int f1 = fact(n-1);
    int f2 = fact(m-1);
    int f3 = fact(n-1+m-1);
    rtn f3/(f1+f2)
    }
    public static void main (S*)
{
    int n=3 , m = 3;
    syso(grid ways(n,m));
```

Time Complexity using permutations
= O(n)