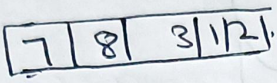
DSA in Java

Bubble sort
Selection sort
Insertion sort

Sorting :- highest to lowest or.
lowest to highest

Bubble sort :-

ex :-



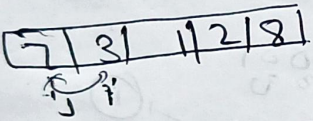
→ array

compare the adjacent elements and big value is should be interchange.

largest element → END

7, 8, 3, 8, 1, 8, 2, 8

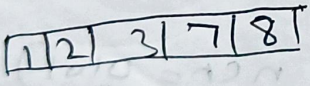
①



②

3 7 1, 7 2, 7, 8
j i

③ 1 2, 3, 7, 8



code

logic

```
for (int i=0 ; i<arr.length ; i++)
    // sorted elements were left out and removing unwanted loop
{
  for (int j=0 ; j<arr.length-i-1 ; j++)
                                  i=0
                                  i=1
  {
    if (arr[j] > arr[j+1])
    {
      if (arr[j] > arr[j+1]) {
                  Index
        // swap
```

```
int temp = arr[i];
arr[i] = arr[i+1];
arr[i+1] = temp;
```

i is just a counter here.

time complexity = $O(n^2)$

I have more time complexity

$n-1$
$n-2$
$n-3$
$n-4$
$\downarrow$
n^2

Selection Sort

In one iteration only one swap.

ex:- 7 8 3 1 2.

Here we take the smallest element and we push it to first place

0	1	2	3	4
7	8	3	1	2

① smallest = 0 [7]
= 2 [3] ✗
= 3 [1]

1	8	3	7	2

② smallest = 1 [8]
= 2 [3]
= 4 [2]

1	2	3	7	8

sorted

③ smallest > 2 [3] [i = 2]

code

logic

```
for(int i=0; i<arr.length-1; i++) {
    int smallest = i;
    for(int j = i+1; j<arr.length; j++).
    {
    if (arr[smallest] > arr[j]) {
        smallest = j;
    }
    swap
```

int temp = arr[smallest];

arr smallest = arr[i];

arr[i] = temp;

}

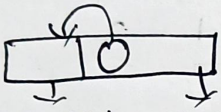
time complexity - n^2

$O(n^2)$

Insertion sort

ex:- 7 8 3 1 2

Take an element from the unsorted array, place it in its corresponding position in the sorted array part, and shift the elements accordingly



sorted unsorted

7 | 8 3 1 2

① 7, 8 | 3 1 2.

② 3 7 8 | 1, 2.

③ 1 3 7 8 | 2.

⑤ 1 2 3 7 8 → sorted.

Logic

```
for(i=1, i<arr.length; i++) {
    int current = arr[i];
    int j = i-1;
    while (j>=0 && arr[current] < arr[j])
    {
        arr[j+1] = arr[j];
        j--;
```

//placement
arr[j+1] = current;

time complexity: $O(n^2)$

Recursion

→ Iterational Loops } ① base f(n) → steps
– functions } ② assume ; inner + rest of work

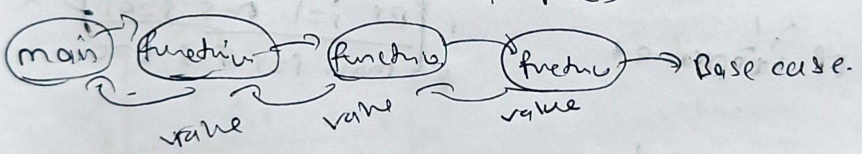
maths

$fx = x^2$ - given

$Rf(x) = f(x^2)$
↳ Recursion

function that calls itself

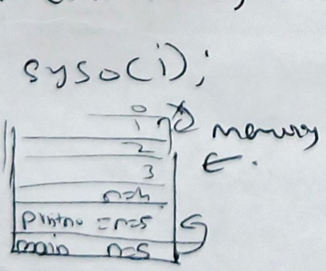
$x = 2$
$f(n) = 2^2 = 4$
$f\,f(n) = f(4) = 4^2 = 16$



Q) Print Number from 5 to 1

```
for (int i = 5; i > 0; i--)
sysO(i);
```

```
P.S.V printN(int n)
{ if (n == 0) } Base
   return;
  syso(n);    } Print
  printNu(n-1); } 
}
```
recursion



(1–5)
→ if (n == 6)
num (n+1)

Print Sum of first n natural numbers

sum {1+2+3+4 - - - n}

$\frac{n(n+1)}{2}$

① main f(n) ⇒)

② Base condt. → n (sum print)

③ Kaam - work — calculate sum

code

```
p.s.v sum(int i, int n, int sum
    sum += i;          → if (i==n) → sum += i;
                                     return;
    print sum(i+1, n, sum);
```

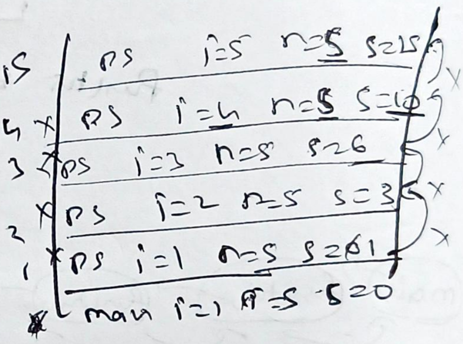
// call

sum(1 , 5 , sum)
↓ ↓ ↓
1 5 0

(15)

if i is printed

memory



---

factorial of a number n !-

$n! = n \times (n-1) \times (n-2) - - - \times 1$

$3! = 3 \times 2 \times 1 = 6 \Rightarrow 3 \times 2!$

info ⇒ n=5

Kaam ⇒ ① (n−1)!

② n×(n−1)!
↳ return.

base case

(n==1)
return 1

code

```
p.s.v fact(int n, int fact)
{
    if(n==0)
    {
        s.o.p.(fact);
        return;
```

fact * = n;

fact (n-1, fact)

}

⑥ Print fibonacci sequence till n^{th} term

a b c
c = a + b

0 1 2 3 4 5 ---

a = 2nd last term

b = last term

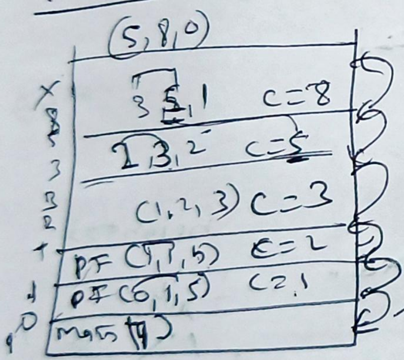
a + b → c

get given = 1st = 0
2nd = 1

② "kaam" execute the next term c = a + b

③ base → n^{th} term

memory



if (n == 0)

return

}

sys (a);

printFact (b, a+b, n-1);

}

1, 0+1, 5-1

c

(1, 1, 4) → c = 2

(0, 1, 5) c = 1

a b n-1

Q.) Print x^n (stack height = n)

$x^n = x \times x \times x \times x$ - - n times

① x, n

② $x^n = x \times (x^{n-1})$ $(x^{n-1}) = x^{n-1+1} = x^n$

③ $x^0 = 1$
x = 0 ⇒ 0 } base case

fact * = n;

fact (n-1, fact)

}

---

⑥ Print fibonacci sequence till nth term

a b c
  ↓
  c = a + b

0 1 2 3 4 5 ----

a = 2nd last term

b = last term

[a + b] → (c)

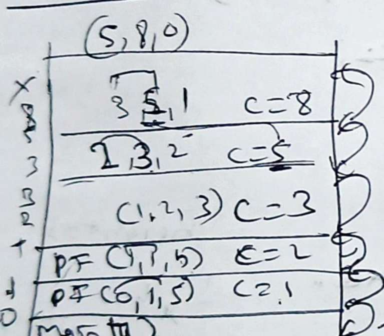
[illegible] given = 1st = 0

2nd = 1

② "kaam" execute the next term c = a + b

③ base → nth term

memory



(1, 1, 4) → c = 2

(0, 1, 5) c = 1

a b n-1

if (n == 0)

return

}

Sys (a);

printfact (b, a+b, n-1);

↓ ↓ ↓

1 0+1 5-1

↓

c

8.) Print x^n (stack height = n)

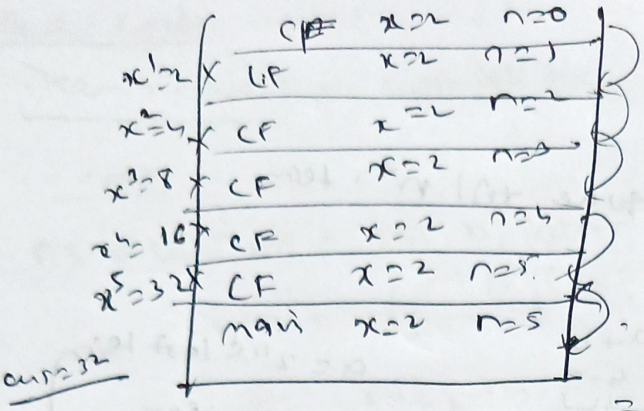
$x^n = x \times x \times x \times x$ --- n times

① x, n

② $x^n = x \times (x^{n-1})$ $\quad (x^{n-1}) = x^{n-1+1} = x^n$

③ $x^0 = 1$, $x = 0 \Rightarrow 0$ } base case

x=2 n=5 $2^5=32$



$= x^{n-1}$

$= 2^{5-3} = 2^4 = 16$

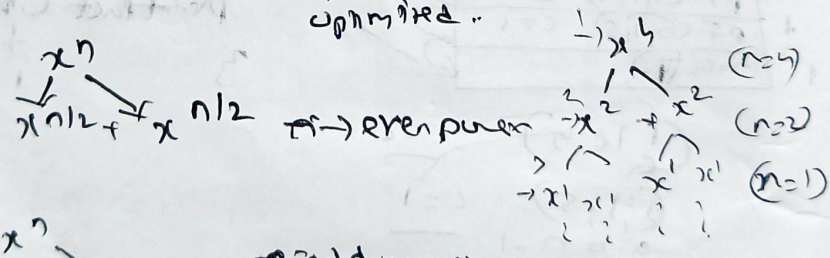
Logic

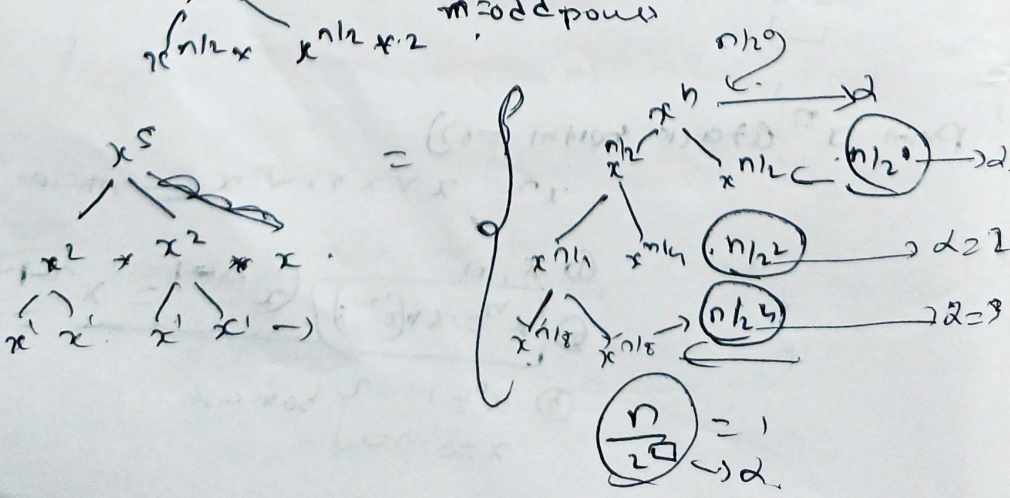
```
if (n==0)
    return 1;

if (x==0)
    return 0;

int x = pow(x, n-1);
int xn = x*x;
return xn;
```

Eg. Print x^n (stack height = log n)

↓

optimized.





logic

if (n%2==0)

return printpow(x, n/2) * printpow(x, n/2);

else

return x * " " " ;

Intermediate level recursion

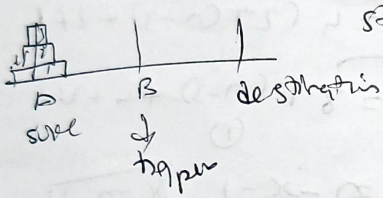
~~(1) Tower of Hanoi~~

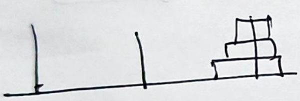
(2) Tower of Hanoi

Rules:-

(1) only one disk transferred to 1 step

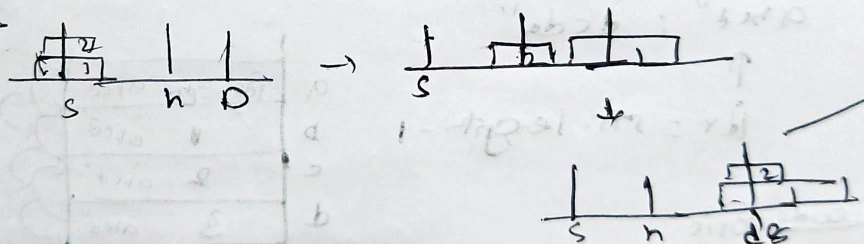
(2) smaller disks are always on top of larger disks





ex:-

n=2



If n=3

(3)+(1)+(3) steps

source → helper
helper → source } → n=2 → tyme.

logic code

n-1, dst, helper

Sy.o.l ("T.D" + n + "From" + src + "to" + dst);

((n-1), helper, src, dest);

$O(2^n - 1) \simeq (O(2^n))$

$T(n) = 2T(n-1) + 1 = 2(2T(n-2) + 1)$

$= 4T(n-2) + 2 + 1$

$2^x \leftarrow = 4(2T(n-3) + 2 + 1$

$2^{x.k} \leftarrow = 8T(n-3) + 4 + 2 + 1$

(1)

2^{n}

$2^{n-1} + (1) + 2^{n-2} + 2^{n-3} + \ldots$

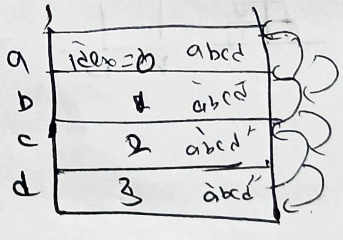
$n - x = 1$ $x = n - 1$

print a str in reverse

"abcd" ; "dcba"

↑

idx = str.length - 1



code logic

printrev (str, idx - 1)

O(n)

(str, str.length() - 1);

---

find the 1st and last occurance of an element in str

1st ← "abaacdaefaah" → last

↓ idx = 0

↓ idx = 0

Here we use static int first
last

String str, char element, int index

```
if (idx == str.length()){
    return;

  if (str.charAt(idx) == element)
  {
      if (first == -1)
      {
          first = index
      }
      else
          last = idx;
  }

  Indices (str, element, idx+1);
```

→ string of the length,, O(n)

Check if an array is sorted (strictly increasing)

{1, 2, 3, 5, 8} → return true.

e.g. 1 2 2 3 → return false

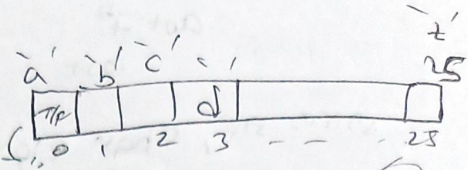
idx = 0 (idx+1) ↳ calls

return false X

```
if (idx == arr.length -1)
{
    true
}
if (arr[idx] < arr[idx+1])
    return isSorted(arr, idx+1);
else
    false.
```

Remove duplicates in a string

"abbccda" → "abcd"



currentchar - 'a' = ?

↓ map index

'a' - 'a' = 0

'b' - 'a' = 1

'c' - 'a' = 2

'z' - 'a' = 25

curr char → True
newstring ✗

curr char → False
newstring ✓
map[pos] = True.

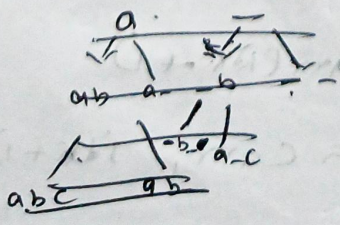
Logic

```
char currentchar = str.charAt(idx);
if(map[currchar - 'a'] == true){
    // already their
    removeDuplicate(str, idx+1, newstring);
}
else {
    // new char
    newstring += currentchar;
    map[current - 'a'] = true;
    print "
```

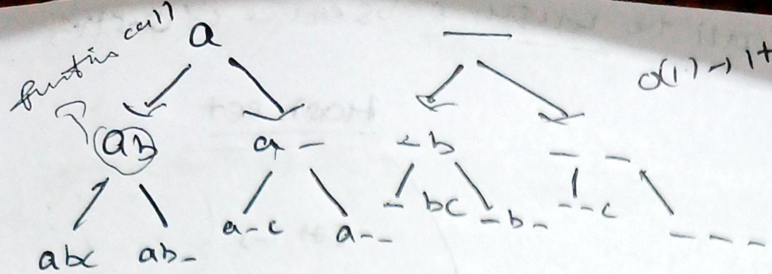
Print all the subsequences of a string

Important

"abc"



→



o(1) → 1 time.

a b c
bc
ac
a
b c
b
c
_

abc
ab
bc
ac
a
b
c
_

"abc" newString =
↑
idx = 0

call 1   newString + current char
call 2   newString

7 nodes
$= 2^n = 2^{(n-1)} \times 2$
$= 2^{n-1}$
$= 2^{n-3}$
⋮
1

GP = geometric progression

$[2^n + 2^{n-1} + 2^{n-2} - 2^{n-3} + \ldots 1]$

O(2^n)

task

```
char currchar = str.charAt(idx);

//to be
// call function
subseq(str, idx+1, newString + currchar);

//not to be added
subseq(str, idx+1, newString);

//base case
if(idx == str.length())
{
  print(newString
  return;

subseq(str, idx: 0, newString: "");
```

T.C

$= \frac{a(r^n - 1)}{r - 1} = \frac{1(2^{n+1} - 1)}{2 - 1} = (2^{n+1} - 1)$

$= O(2^n)$

Print all the unique subsequence of a string

"aaa"

- a a a
  a a
  a
  0

Hash set

from the same logic

add

set →

HashSet<String> set = new HashSet<>();

It can only store unique string

HashSet<String> set = new HashSet<>();

---

5) Print keyboard

"mobile key pad"

0 → 0

1 → abc

2 - def

3 - ghi

4 - jkl

5 - mno

6 - pqrs

7 - tu

8 - vwx

9 - yz

"23"

d → dg, dh, di

e → eg, eh, ei

f → fg, fh, fi

Logic

store the key map values as string

// recursive

char current char = str.charAt(idx);

string mapping = key[current char - '0'];

// if 2 - 0 = 2

for (int i=0; i<mapping[digit].length(); i++)

{

printComb(str, idx+1, combination+mapping[digit].charAt(i));

}

if 23

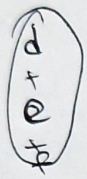
2

3

def

loop → d + e + f → level 1

level 2



O(4^n) → Time complexity

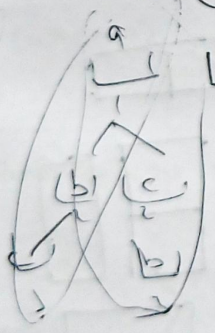
Print All permutation of a string

"abc" → all possible combination of letters

abc - 3! = 3×2=6

abc acb bac bca cab cba

n (n-1) (n-2)



O(n!) → T.C

Logic

```
for (i=0; i<str.length(); i++)
{
char currChar = str.charAt(i);
```

// "abc" → "ab"

```
String newStr = str.substring(0,i)
+ str.substring(i+1, end of str)
```

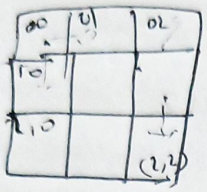
'a' i=1 (0-1) × (0-0) (i+1) → "c"

"ac"

print (newstr, permutation(str));

~~Count~~ Count total paths to move from (0,0) to (n,m)

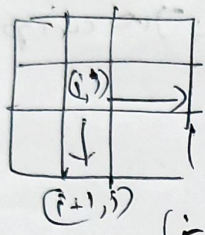
n=3, m=3



(3x3) (j) right

(i) downwards

Total paths = 6



(i,j)

(i, j+1) (i+1, j)

(i=n-1, j=m-1)

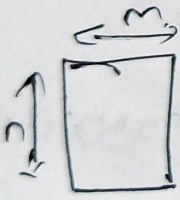
(i, j)

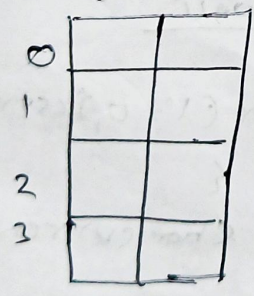
~~Count~~ count(i+1, j) + (count(i, j+1))

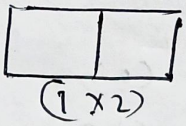
total paths backtracking

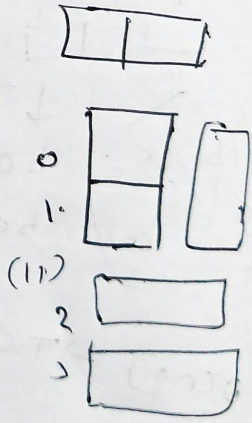
Place tiles of size 1xm in a floor of size nxm

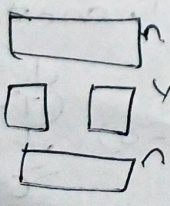
n=4 m=2

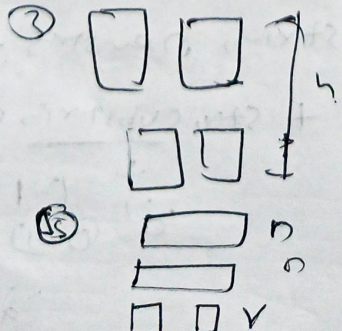


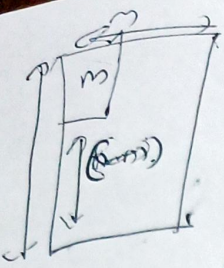






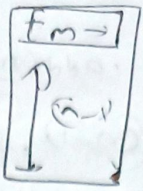






place (n)

place (n-m) | place (n-1)



// base case

n=m => (2)

n<m => (1)

---

find the most ways in which you can arrange n people in given pairs / single or in pairs.

ans: 1, 2, 3, 4.

n=1 => (1)

n=2 => (1,2), (1-2) => (2)

n=3 => 1 2 3
1-2 3
1-2 3
1 2-3
} 4

n=4 => (1), 2, 3, 4
1 2-3 4
(10)

guest call(n)

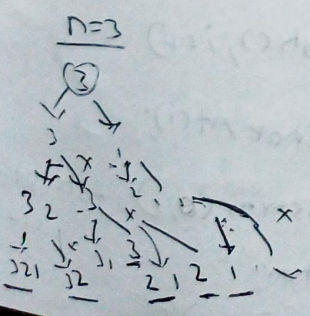
single / pair

call(n-1) (n-1) * call(n-2)

Print all the subsets of a set of first n natural numbers

n=3 (1,2,3)

(1,2,3) (1,2) (1,3) (2,3) (1) (2) (3) ( )




```
ArrayList<Integer> subset
for(i=0; i<subset.size(); i++)
sysO(subset.get(i) + " ");
}
sysO();
```

// add

subset.add(n)

subset(n-1, subset);

// Backtrack one do not add C.E

subset.remove(subset() - 1)

subset(n-1, subset);

ArrayList<Integer> subset = new ArrayList<>();

subset(n, subset);

}

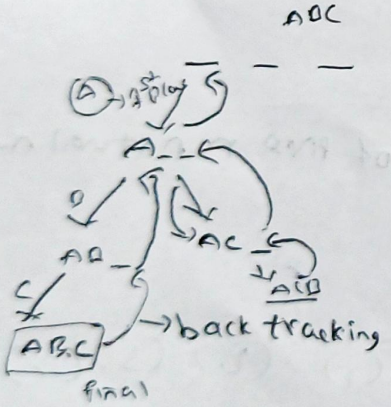
}

Backtracking

Used in advanced data structures

→ Find all Possible solutions & use the one you want.

Tree visualization



3! = 3x2 = 6

~~O(n^n)~~

= ~~O(n^2)~~

O(n x n!)

logic

```
for(int i=0; i<str.length(); i++)
{
char currchar = str.charAt(i);
char newstr = str.substring(0, i) + str.substring(i+1);
// perm = perm + currchar;
function(newstring, idx+1);
```

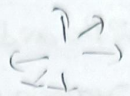
N-Queens

N Queens

Print all solutions where queens are safe.

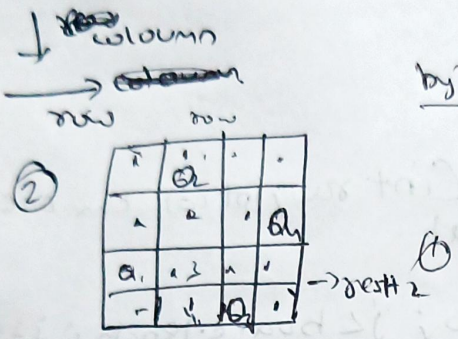
↓

~~chess~~ chess(ruled)

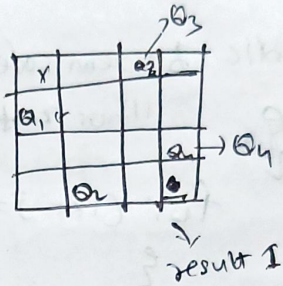


Q_1, Q_2, Q_3, Q_4

(4×4)



by back tracking



Input n=4

Output:

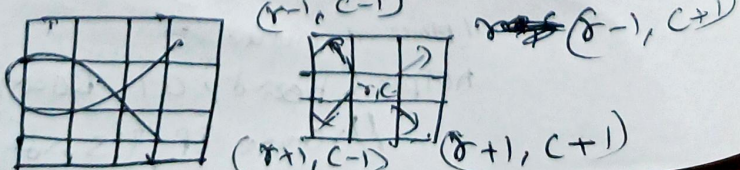
" . Q . . "

1 2 3 4 → column

```
List<List<String>> allBoard = new ArrayList<>();
public void helper(char[][] board, List... , int col)
{
    char [][] board = new board[n][n];
    // place queen.

    for(int row=0 ; row<board.length; row++)
    {
    if (isSafe(row, col, board))
        {
        board[row][col] = 'Q';
    // place in another
    helper(board, allBoard, col+1)
        //remove if its not safe.
```

```
board[row][col] = '.';

//base condition
if(col == board.length)
{
saveBoard(board, allBoards);
return;

// safe function
public boolean safe(int row, int col, char[][] board)
{
    //char board[][]
    for (int j=0; j< board.length; j++)
    {
      if(board[row][j] == 'Q') {
      // if row, j any queen is their return it as false
        return false;
      // Vertical.
      if (int i=0; i< board.length; i++)
      {
        if (board[i][col] == 'Q')
        {
          return false;
```

// Always have to check upper, lower (left & right)



```
//upper left
int r=row;
for(int c=col; c>=0 && r>=0; c--, r--)
    {
    if (board[r][c] == 'Q') {
        return false;
    }
}

//upper right
int r=row
for(int c=col, c<board.length && r>=0, c++, r--)
{
    if(board[r][c] == 'Q')
    {
        return false;
    }
}
```

Similarly, do for lower left and right. (6) → conditions

// to print / place (6) in all board

```
public void saveboard(char[][] board, List<List<String>> allBoards)
{
String now = "";
List<String> newBOard = new ArrayList<>();
    for(i=0; i<board[0].length; i++){
        if(board[i][j] == 'Q')
            now += 'Q';
        else
            now += '.'; }
```

newBoard.add(row);

}

allBoards.add(newBoard);

Time Complexity

$= O(n^n)$

Java Sudoku solver

Rules:

numbers can't be changed

empty cells can be filled with 1–9 any number

In one line (row) → if the number is used can't be used again

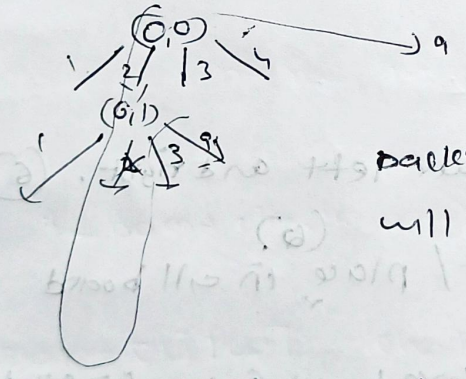
- same implies to column ↓

In grids (3×3) the number should not be repeated

Approach

→ Recursion part

cell (0,0) → (1–9)



backtracking will be done here

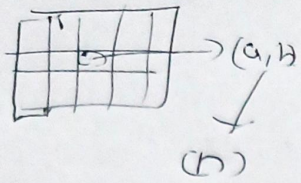
logic

board

```
boolean Helper(char[][] board, int row, int column)
{
    // what's the next row & col
    int row_new = 0;
    int col_new = 0;
    if (col != board.length - 1){
```

```
nrow
nrow = row;
ncol = (col)+1;
else {
nrow = row+1
ncol = 0; // again form new column col
}

if (board[r][c] != '.') {
    if (helper(board, nrow, ncol))
    {
        return true;
    }
else
for (int i=1; i<=9; i++) {
        if (isSafe(board, row, col, i+'0'))
        {                          // board is char
        board[r][c] = (char) (i+'0');  // string type
          helper(board, nrow, ncol)
            {return true;
             else {
                board[r][c] = '.';

            return false;
^
// base condition
    if (row == board.length)
        {
            return true;
        }
```

isSafe part; ① row

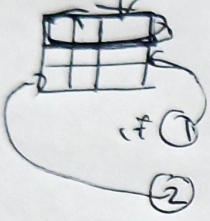


① row (9x9)
② col
③ grid (3x3)

→ take starting point on every grid

row / (r/3 * 3), (c/3 * 3)

quotient



if (0) → in top on one is grid

common grid take it

same to this

// safe function

```
public boolean isSafe(char[][] board,
                      int row, int col, int num)
{
    // row & col
    for (int i=0; i<board.length; i++)
    {
        if (board[i][col] == (char)(num + '0'))
            return false;

        if (board[row][i] == (char)(num + '0'))
            return false;
    }

    // grid
    int sr = row/3 * 3;
```

```
int sc = (col/3) * 3;

for (int i = sr; i < sr+3; i++)
    {
    for (int j = sc; j < sc+3; j++)
    {
    if (board[i][j] == (char) (number + '0'))
    {
        return false;
    //main:-
public void solveSudoku(char[][], board)
    {
    helper (board 0, 0);
}
}
```