How to build Strong Programming Logic ?

"If I had 1 hour to Chop Down a tree, I'd take 45 min to Sharpening my Axe"

— Abhram Lincon

* (1) Concepts >> Contests * Rank Ratings

* min 375 Ques from excel sheet provided (till medium level questions)
* Practise & do them only then pact for contests

(2) THINK >> CODING

→ Read the Question carefully
→ make algo with (pen + paper)
→ Corner Cases (ye sai Kaha Kaha fas sakta hai)
→ Dry Run with 2 sample Inputs

(3) STEP - BY- STEP

(i) Basic Approach

↓

(ii) Time reduce (DSA apply)

↓

(iii) Reduce space

(4) PRACTISE

(i) 3 days Rule → Max. 3 din se jada don't leave the concept/logic for practise

(ii) 1 hr each day

(5) MOVE - ON

Company Expectation

Easy : 5 - 10 min

Medium : 15 - 20 min

Hard : 40 - 60 min

5 Basic Principles

(SMART WORK)

(1.) ODYSSEUS METHOD

→ Cut off all your Distraction

→ Remove all the things that makes u feel distracted

(2.) PARETO PRINCIPLE

⇒ 20% Input —result→ 80% Output

⇒ Read the syllabus
↓
Read the topic
↓
But search from PYQ
↓
class Notes

→ Combine them "common" part

(3.) SPACED REPETITION (POMODORO)

⇒ 1½ hr (focus) + 5-10 min (Break)

(4.) PARKINSON'S LAW

⇒ Set deadline for your Each work

⇒ Pre-plan your work

⇒ Pre-planned includes:

- What to study
- When to study

& follow your Deadline

(5.) Active Recall

→ 1st Read your topic + Analyse them

⇓

then go for notes making

(think about that topic & write notes by yourself)

Coding Platforms

Beginners :

→ Hacker Rank

→ Hacker Earth

→ Geeks for Geeks

Placement / Interview Prep :

→ Leet code

→ Interview Bit

Competitive Programming

→ Codeforces

→ Codechef

---

① Github

② IIT Bombay → EdX → Online courses platform.

Java Lecture
– Shradha Mam (Apna College)

Lect - 1
Introduction to Java Language

* Psudocode :→ sample code containg only logic part

↓

to represent flow of code in simple english language.

Psudocode
↓
1) Start
2) Input 2 NO.
3) calculate sum = nuber 1 + nuber 2
4) Print sum
5) Exit

Installation

1.) Java Development Kit (JDK)

2.) Code Editor (IDE → Integrated Development Environment)
- V.S. code
- Intellij
- Eclipse

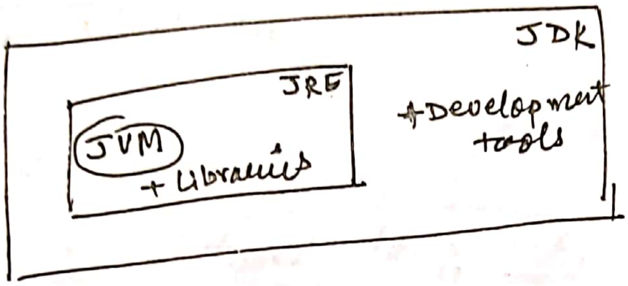
".java"

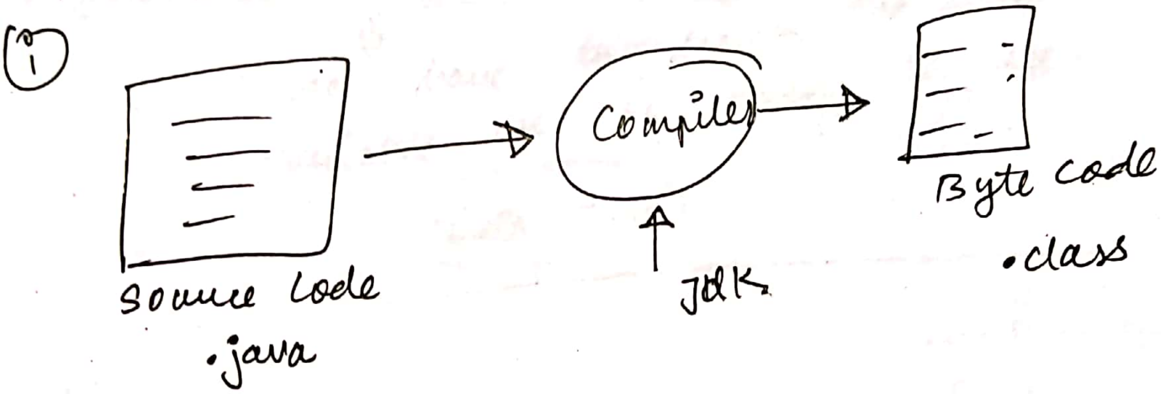
Q. How is my Code Running?

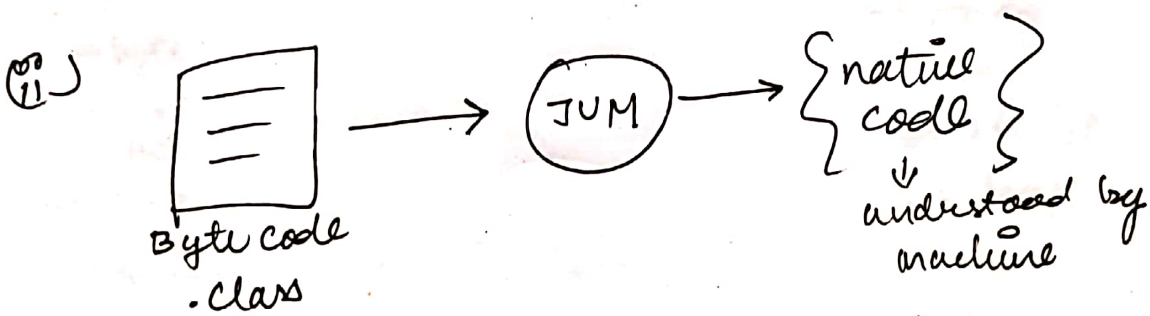
⇒ (i) Compilation (ii) Execution

* JDK contains JRE → Java Runtime Environment

JVM : Java Virtual Machine







function

```
void main()
{

}
```

J, main fn se sequence se compile + execute hota hai.

Lecture - 2

Output

System . out . print( );

↓

System class

* Boiler code ⇒ jo code by default editor se mila ho

Variables

* Java is a typed language
  ↓
  we have to declare the type of variable we are going to use.

Data Type

Primitive
- byte
- short
- int
- long
- double
- float
- char
- boolean

Non-Primitive
- String
- Array
- class
- object
- Interface

1 byte = 8 bits.

* sc.next() → read only 1st word
* sc.nextLine() → read whole sentences / paragraph.

Lecture -3

Conditionals Statements

Keywords → if, else, else if, switch, break, continue

% → calculate Remainder (modulo)

break → terminate the condition or exit the condition

Lecture - 4

Loops

→ for

→ while

→ do...while

(1)

```
for ( initialize ; condition ; updation )
{
    ≡≡≡
}
```

(2)

```
while (condition)
{
    ≡≡≡
}
```

(3)

```
do
{
    ≡≡≡
} while (condition);
```

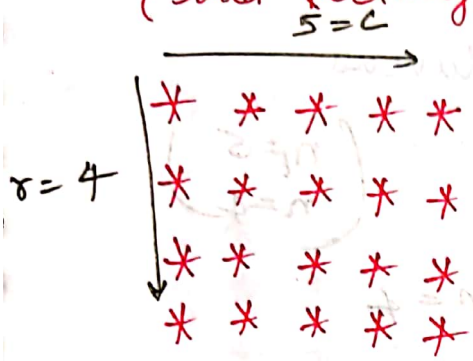
even if condition becomes false ↓ execute once

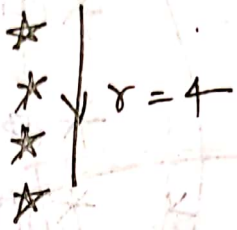
Lecture - 5

9 Best Pattern Question in Java

(1) Pattern - 1

(Solid Rectangle)

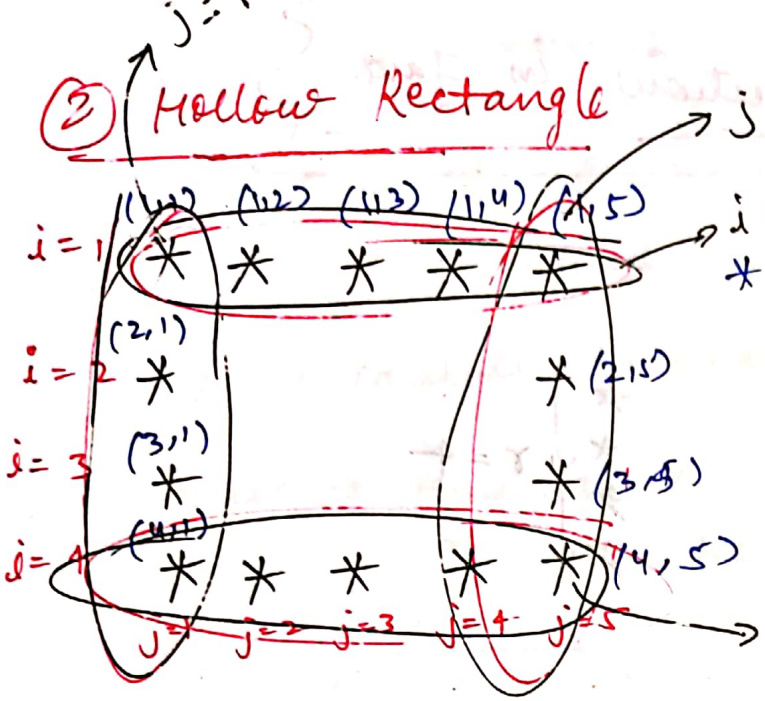




Nested loop:

```
for(i=1 ; i<=4 ; i++)
  {
     for (j=1 ; j<=5 ; j++)
       {
          sout("*");
       }
       sout("\n");
  }
```

② Hollow Rectangle



* observe any logical pattern with rows & columns

(m=5, n=4)

→ columns (inner loop)

↓ rows (outer loop)

* Boundary

```
int n=4, m=5;
for(i=1; i<=n; i++)
{
    for(j=1; j<=m; j++)
    {
        if(i==1 || j==1 || i==n ||
           j==m)
        {
            sout("*");
        }
    }
    sout
}
```

```
else
{
    sout(" ");
}
}//for - inner
sout("\n");
}//for - outer
```

(3) Half Pyramid Pattern

```
*
* *
* * *
* * * *
```

n = 4

1st row → 1 star
2nd → 2 star
3rd → 3 star
4th → 4 star

* No. of row = No. of star *

```
for(i=1; i<=n; i++)
{
    for(j=1; j<=i; j++)
    {
        sout("* ");
    }
    sout("\n");
}
```

(4) Inverted Half Pyramid

```
* * * *
* * *
* *
*
```

n = 4

```
for(i=n; i>=1; i--)
{
    for(j=1; j<=i; j++)
    {
        sout("*");
    }
    sout("\n");
}
```

(5) Inverted Half Pyramid (rotated by 180°)

n = 4

space

```
- - - *    <- 3 space + 1 star = 4
- - * *    <- 2 space + 2 star = 4
- * * *    <- 1 space + 3 star = 4
* * * *    <- 0 space + 4 star = 4
```

n = 4

* Print Space *

row	space	star
row 1	3 = n - i	i = 1
2	2 = n - i	i = 2
3	1 = n - i	i = 3
4	0 = n - i	i = 4

3 loop
- 1 loop → no. of rows
- 1 loop → space
- 1 loop → star

```
// row
for (i = 1; i <= n; i++)
{   // space
    for (j = 1; j < n - i; j++)
    {
        sout(" ");
    }
    // star
    for (j = 1; j <= i; j++)
    {
        sout("*");
    }
    sout("\n");
}
```

(6) Half Pyramid with Number

```
1                  → row-1   no. 1 to 1
1 2                → row 2   no. 1 to 2
1 2 3              → row 3   no. 1 to 3
1 2 3 4            → row 4   no. 1 to 4
1 2 3 4 5          → row 5   no. 1 to 5
```

n=5

```
// for row
for (i=1; i<=n; i++)
{
    for (j=1; j<=i; j++)
    {
        cout(j+" ");
    }
    cout("\n");
}
```

(7.) Inverted Half Pyramid with Nos.

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

n=5

i → row = 1 to 5

→ r = 2 — 1 to 4

→ r = 3 — 1 to 3

→ r = 4 — 1 to 2

→ r = 5 — 1 to 1

```
//row
for(i=1 ; i<=n ; i++)
  {
    for(j=1 ; j<=(n-i+1) ; j++)
      {
        sout(j+" ");
      }
    sout("\n");
  }
```

(8.) Floyd's Triangle

```
1                  -> r=1
2  3               -> r=2
4  5  6            -> r=3
7  8  9  10        -> r=4
11 12 13 14 15     -> r=5
```

1 val (1)
2 val (2 3)
3 val (4 5 6)
4 val (7 8 9 10)
5 val (11 12 13 14 15)

n=5 (total no. of rows)

```
int number = 1;
// row
for(i=1; i<=n; i++)
{
    for(j=1; j<=i; j++)
    {
        sout(number+" ");
        number++;
    }
    sout("\n");
}
```

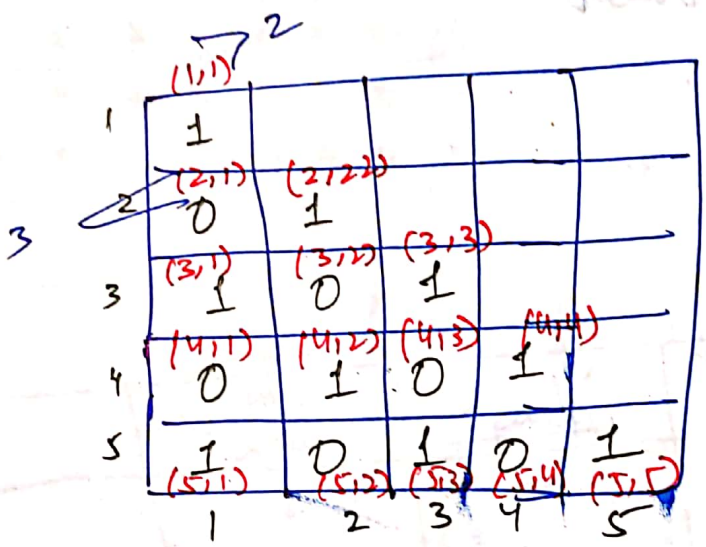
(90) 0-1 Triangle

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

(i) → r = 1, → r = 2, → r = 3, → r = 4, → r = 5

(j) 1 val (1)
2 val (0,1)
3 val (1,0,1)
4 val (0,1,0,1)
5 val (1,0,1,0,1)

n = 5

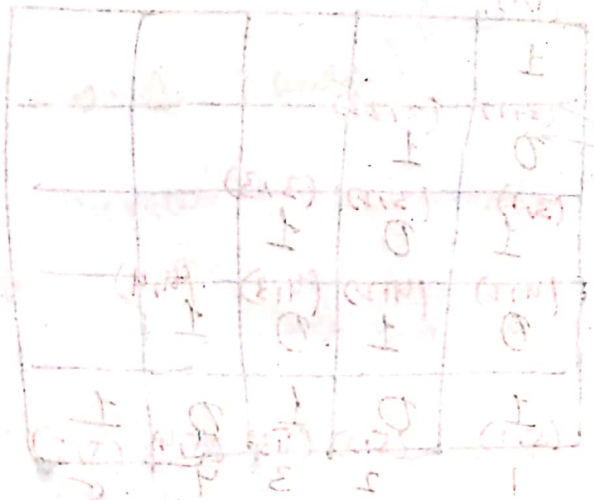


(i+j) → odd sum = print → 0

(i+j) → even sum = print → 1

```
for(i=1 ; i<=n ; i++)
{
    for(j=1 ; j<=i ; j++)
    {
        int sum = i+j;
        if(sum%2==0)
        { //even
            sout("1");
        }
```

```
else
 {
   //odd
   sout("0");
 }
}//for
sout("\n");
}// four-outer
```

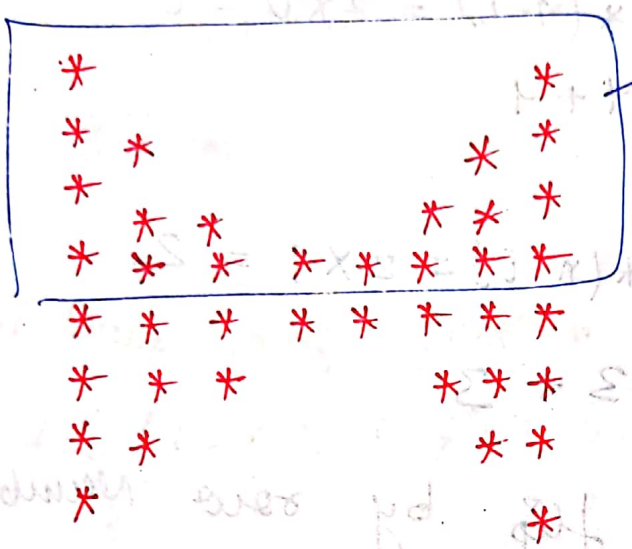


Scanned with CamScanner

Lecture - 6

Advanced Pattern Questions

(10) Butterfly Pattern



→ focus on this pat
→ just reverse the loop.

row (i)	1st part star	spare	2nd part star
1	1	6	1
2	2	4	2
3	3	2	3
4	4	0	4

row no. = no. of star in 1st part
= no. of star in 2nd part

★ How to print spaces = ?

Spaces

2*3 → (n-i) (4-1)
2*2 → (n-i) (4-2)
2*1 → (n-i) (4-3)
2*0 → (n-i) (4-4)

1st half ⇒ (1 to n) ⇒ row loop

2nd half ⇒ (n to 1) ⇒ row loop

i = n = 4

space = 2*(n-i) = 2×0 = 0

star = 4+4

i = 3

space = 2*(n-i) = 2×1 = 2

star = 3+3

* Condition gets flip by row number

```
int n = 5;
//uper half
for(i=1; i<=n; i++)
  {
    //star - 1st part
    for(j=1; j<=i; j++)
      {
        cout("*");
      }
    //space
    int space = 2*(n-i);
    for(j=1; j<=spaces; j++)
      {
        sout(" ");
      }
```

```
// star - 2nd part
for(j=1 ; j<=i ; j++)
  {
     cout("*");
  }
  cout("\n");
}// for
```

// lower half pattern

```
for(i=n ; i>=1 ; i--)
  {
     :
     :
     remain same as above loop :-
     :
  }
```

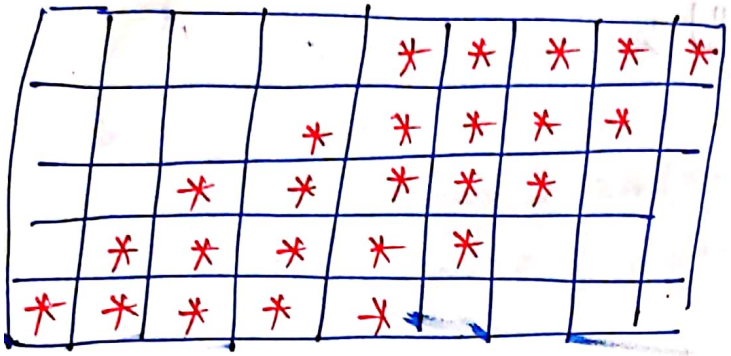
(11) Solid Rhombus

$n = 5$

row = 5

↓

each row = 5 star



Matrix form

Convert

i	Space	Star
1	4 Space	5 star
2	3 Space	5 star
3	2 Space	5 star
4	1 Space	5 star
5	0 Space	5 star

↓ i = row

↓ (n - i)

↓ const.

```
for (i = 1; i <= n; i++)
{
    // spaces
    for (j = 1; j <= (n-i); j++)
    {
```

```
        sout(" ");
    }
    //stars
    for(j=1;j<=5;j++)
        {
            sout("*");
        }
    sout("\n");
}
```

(13.) Number Pyramid

□ => space

n = 5

```
    1
  2 □ 2
3 □ 3 □ 3
4 □ 4 □ 4 □ 4
5 □ 5 □ 5 □ 5 □ 5
```

i	step
1	1 → 1 time
2	2 → 2 time
3	3 → 3 time
4	4 → 4 time
5	5 → 5 time

front space → at the beggining wala space

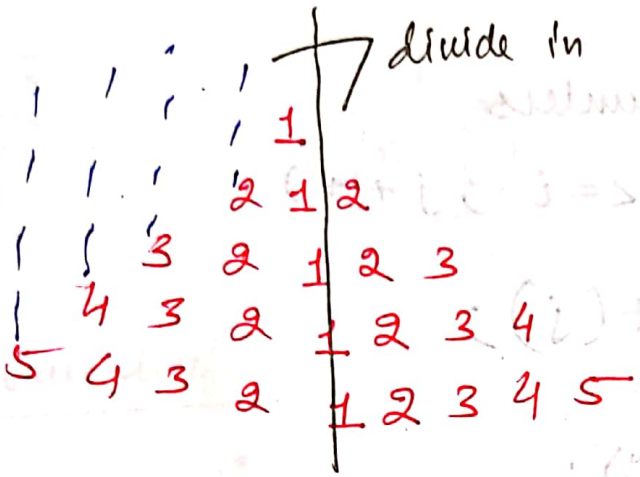
row	space	print
1	4	1 (1x)
2	3	2 (2x)
3	2	3 (3x)
4	1	4 (4x)
5	0	5 (5x)
i	(n-i)	row no. times (i-times print)

```
int n=5;
for(i=1; i<=n; i++)
{
    //spaces
    for(j=1; j<=(n-i); j++)
    {
        sout(" ");
    }
    // numbers → print row no → row times
    for(j=1; j<=i; j++)
    {
        sout(i+" ");
    }
    sout("\n");
}//for
```

(13:) Palindromic Pattern

→ BOB
→ 121

front + back se same dikte hai



i
1
2
3
i

space
4
3
2
n-i

1st half

NO.
1 to 1
2 to 1
3 to 1
i to 1
j = i to 1

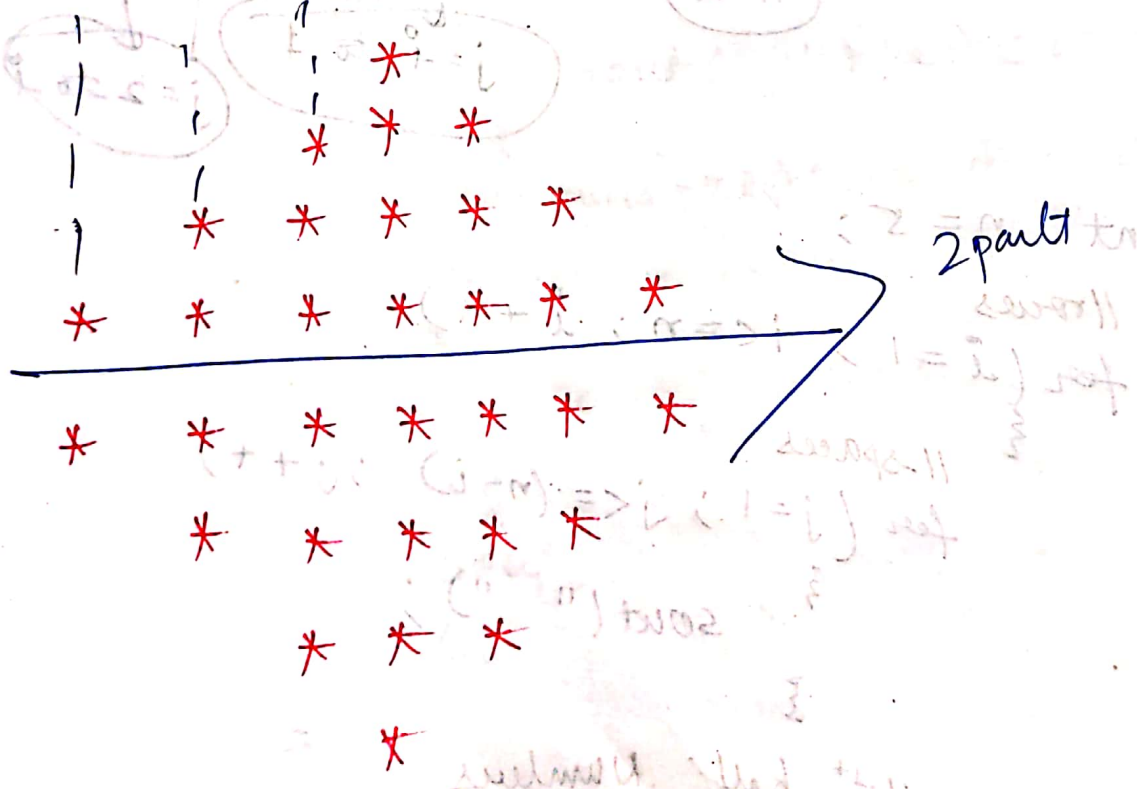
2nd half

NO.
X
2.
2 to 3
2 to 4
2 to 5
j = 2 to i

```
int n = 5;
//rows
for (i = 1; i <= n; i++)
{
    //spaces
    for (j = 1; j <= (n - i); j++)
    {
        sout(" ");
    }
    //1st half Numbers
```

```
for (j=i ; j>=1 ; j--)
{
    cout(j);
}
//2nd half - numbers
for (j=2 ; j<=i ; j++)
{
    cout(j);
}
cout("\n");
}//for
```

(14) Diamond pattern



Upper half

i	spaces	stars
1	3	1
2	2	3
3	1	5
i	(n-i)	(2*i-1)

// odd no. stars

lower half

⇒ just flip the row no. → remaining will be same as upper half.

```
int n = 4;
//Uper Half
for (i=1; i<=n; i++)
{
    //spaces
    for (j=1; j<=(n-i); j++)
    {
        cout(" ");
    }
    //star
    for (j=1; j<=(2*i-1); j++)
    {
        cout("*");
    }
```

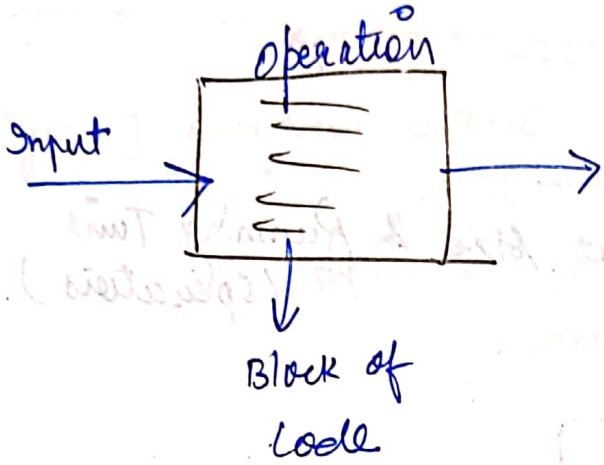
```
sout("\n");
}//for

// lower half
for(i=n; i>=1 ; i--)
{
   // spaces
   for(j=1 ; j<=(n-i) ; j++)
   {
      sout(" ");
   }
   //star
   for(j=1 ; j<=(2*i-1) ; j++)
   {
      sout("*");
   }
   sout("\n");
}
```

space

Note:- To replicate any pattern (flip) ; jis variable par ao depend kar rahi hai, us value ko opp. flip kar do.

Lecture - 7

* Functions & Methods *



Syntax :

arguments (inputs)

```
public static returnType functionName ( . . . . )
{
    ≡≡≡
}
```

void ⇒ no return type

functions

(we call them directly .)

method

(we call ↓ by ↓ Class ke objects ↕ creating objects .)

Lecture - 8

Time Complexity & Space Complexity

Time Complexity

Rel. b/w. Input size & Running Time (Operations)

* Times less
* Space less

Lecture - 9

Array Introduction (1-D - Array)

to provide space

```
type[] arrayName = new type[size];
```

keyword

new → non-primitive data type ke space ko lene ke liye

⇒ index starts from ⇒ "0" (zero)

⇒ address of memory stored in hexadecimal

```
int [] marks = new int[5];
```

```
// display array
for(i=0 ; i< arr.length ; i++)
  {
     sout(arr[i] + " ");
  }
```

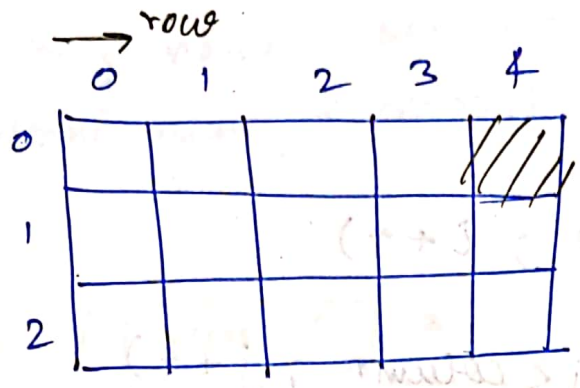
→ linear fashion memory allocation (contigous)

→ by default null values gets initialize

```
// input array
for(i=0 ; i< size ; i++)
{
    arr[i] = sc.nextInt();
}

// Linear Search
for(i=0 ; i<arr.length; i++)
{
    if (arr[i] == val)
    {
        sout (i);
    }
}
```

Lecture -10

2-D - Array



row = 3

colum = 5

(row x colum) x datatype

```
int[][] arr = new int[3][5];
```

row ↑ column ↑

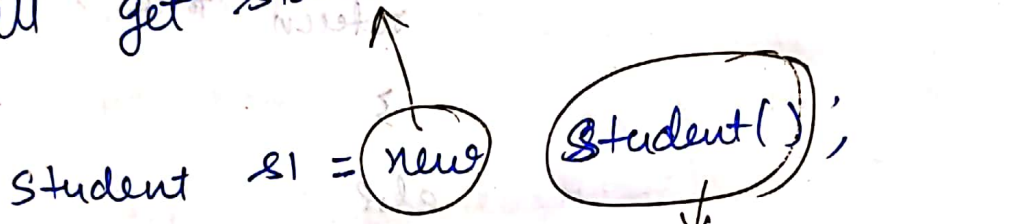
```
// Input :
                    // row
for (i=0 ; i<row ; i++)
  {
                    //clounmn
     for (j=0 ; j<column ; j++)
       {
          arr[i][j] = sc.nextInt();
       }
  }
// Output of 2-d array → Matrix format
for (i=0; i<row; i++)
  {
     for (j=0; j<column ; j++)
       {
          sout(arr[i][j]+"  ");
       }
```

```
sout("\n");
}//for

//linear search
for(i=0; i<row; i++)
{
    for(j=0; j<column; j++)
    {
        if(arr[i][j] == val)
        {
            sout(i+" "+j);
        }
    }
}
```

Lecture - 22

Java OOPs in 1 shot

* "new" Keyword ⇒ memory ke ander ;
  ⇒ memory heap me ek jagah allocate ho jayegi ; us jagah ke ander, hamari puri - ki puri object will get stored there.



Buzz Java - Class

Eg:-

```
class Pen
 {
    string color ;
    public void printColor ()
     { sout(" Colour of this Pen" + this.color) ;
     }
}// class

// main class
```

```
public class Main
{
    public static void main(String args[])
    {
        Pen p1 = new Pen();
        Pen p2 = new Pen();

        p1.color = blue;
        p2.color = black;

        p1.printcolor();
        p2.printcolor();
    }//main
}//class
```

---

* Java has garbage collector
  (NO destructor unlike C++, C)

* Polymorphism
  - may / can
  - → f^n overloading
  - → f^n overridding

* see package → @ 39.00 min in this video.

* main class → public → access to compiler.

* if access modifier used ⇒ protected
  we can use them only by creating its getter & setter in other class.

Encapsulation

↓

class - object (identity)

(methods / properties)

→ Data Hiding is possible with the help of encapsulation.

→ with the help of Access Modifier

Abstraction ⇒ user ko imp. chize dikana & non-imp things chupa lena.

* instantiation ⇒ to create

~~* instance~~

Static - Keyword

⇒ it can be access anywhere means the class or outside the class

⇒ properties common to all

⇒ static ko 1 baar hi memory di jati hai lekin object ke liye baar baar memory create hoti hai.

Lecture 11

String

String name = sc.next(); → only 1st word of string

String name = sc.nextLine(); → whole line / para

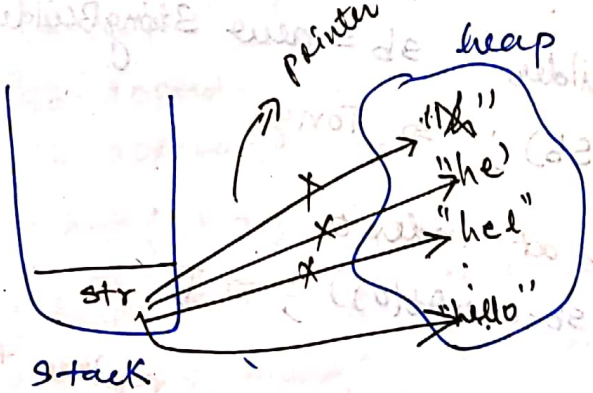
Lecture 12

String Builder

⇒ when we need to modify over string

→ String in Java are Immutable.

String str = "h";



str + "e" ⇒ he

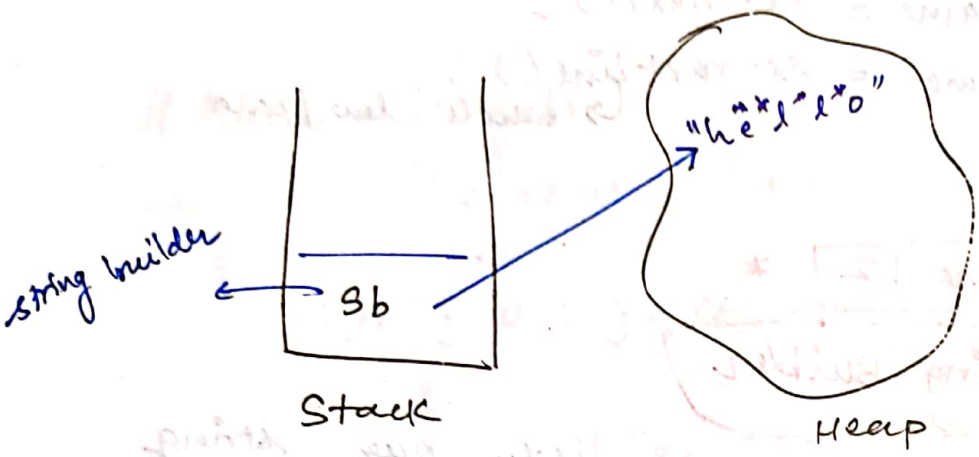
Str + "l" ⇒ hel

str + "l" ⇒ hell

str + "o" ⇒ hello

* Any process in heap is little time taking process ⇒ (User has to wait a little)

→ uses expensive memory
so, to tackle it, string Builder class came into existence.



In case of sb, it will add directly there only in the heap.

```
//Syntax :
StringBuilder sb = new StringBuilder("Tony");
sout(sb); ⇒ Tony

// character at index 0
sout(sb.charAt(0)); ⇒ T

//set char at index → replace the prev wala character with 'P'
sout(sb.setCharAt(0, 'P')); ⇒ Pony
                    index ↑  ↑ character

sout(sb);

//insert any character → 'index jaha par insert karna hai
                       → char
sb.insert(0, 'S'); ⇒ STony
sout(sb)
```

```
sb.insert(2,'n');        => Tonny
sout(sb);

// deleting any substring from string
sb.delete(2,3);          => Tony
sout(sb);
```

[2,3)

```
sb.delete(2,4);          => Toy
sout(sb);
```

```
// add at the end
StringBuilder sb = new StringBuilder("h");

sb.append
sb.append("e");
sb.append("l");
sb.append("o");          => helo
sout(sb);
```

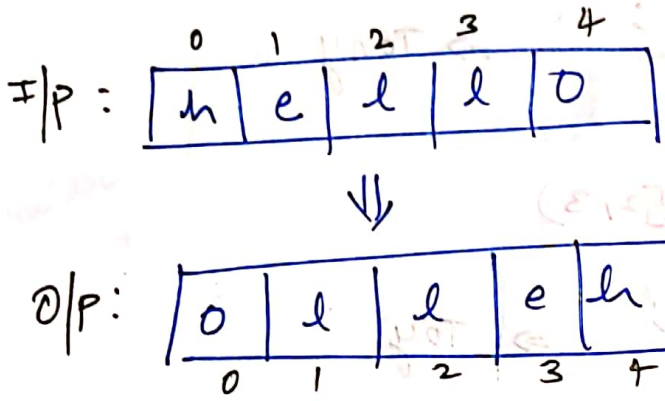
Same string builder me change ho raha hai
but ensure of string me have to make new string
in each case.

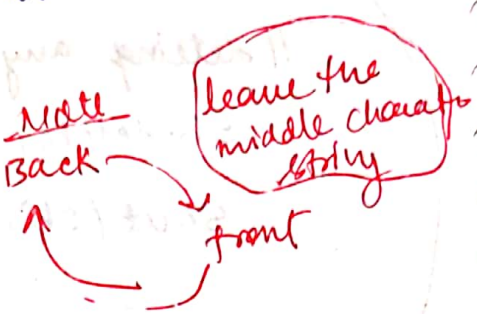
```
//length of string
sout(sb.length());       => 4
```

// Reverse a String

"hello" ⟹ "olleh"

I/p → "o/p"






```
StringBuilder sb = new StringBuilder("hello");

for(i=0; i<sb.length()/2; i++)
{
    int front = i;
    int back = sb.length()-i-1;

    char frontchar = sb.charAt(front);
    char Backchar = sb.charAt(back);
    //setting the character
    sb.setCharAt(front, backChar);

    sb.setCharAt(back, frontchar);
}

sout(sb);
```

front index pe back wala char ←

back index ↓ front wala char. ←

Lecture -12

Operators & Binary No. System

operand ← a + b → operand

operator

* Symbols that tell compiler to perform some operations

(1) Arithmetic Operator

Binary (2 operands)

+ , - , * , / , %

↑ modulo (remainder)

Unary (1 operands)

++ , --

Pre-Increment

++a

(1) change value

(2) use value / assign value

```
int a = 10;
int b = 0;
b = ++a;
sout (a); → 11
sout (b); → 11
```

Post-Increment

a++

(1) use value

(2) change value / assign value

```
int a = 10;
int b = 0;
b = a++;
sout (a); → 11
sout (b); → 10
```

(2) Relational Operators → output boolean value

==, !=, >, <, >=, <=

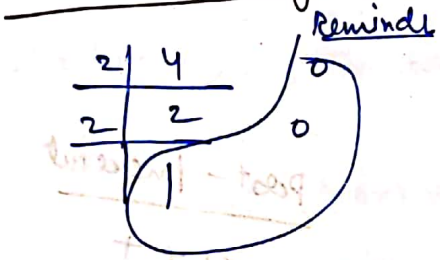
(3) Logical Operators

&&, ||, !

Binary Number System ⇒ Base 2

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 → decimal no. system

* 4 to Binary

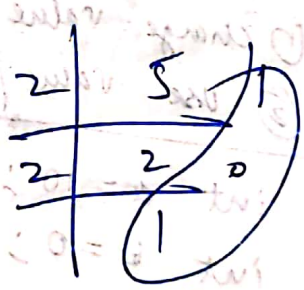


100

1 0 0

2^2 2^1 2^0

$= 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 0 = 4$



101

1 0 1

2^2 2^1 2^0

$4 + 0 + 1 = 5$

$(101)_2 = (5)_{10}$

- Octal → Base 8
- Hexadecimal → Base 16

④ Bitwise Operator

(i) & Binary And

(ii) | Binary OR

(iii) ^ Binary XOR

(iv) ~ Binary One's Complement

(v) << Binary left shift

(vi) >> Binary Right shift

True → 1

false → 0

A = 0101

B = 0110

(i)
```
  0101
& 0110
------
  0100
------
```
A & B = C

∴ C = 0100

(ii)
```
  0101
| 0110
------
  0111
------
```
A | B = C

∴ C = 0111

(iii) XOR (^)

Similar value → false

diff values → true

```
  0101
^ 0110
------
  0011
------
```
A ^ B

∴ C = 0011

(iv) Complement

0 → 1

1 → 0

A = 0 1 0 1

~A = 1 0 1 0

(vi) Binary Right shift

A >>1

⇒ (0 1 0 1)

⇓

[0 0 1 0]

B >>1

⇓

[0 0 1 1]

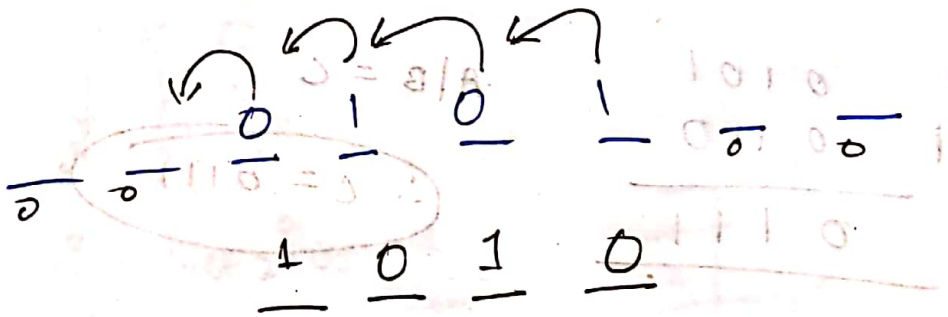
(v) Binary left shift

≪

Syntax :⇒ Number Operator Position

A ≪ 1

0 1 0 1 (bit shift towards left)



1 0 1 0

new No = (1010)

⑤ Assignment Operator

= ; += ; -= ; *= ; /=

a = a + b

a += b

Lecture - 13

Bit_manipulation in Java

true ← current flow => 1 (high state)

false ← no current flow => 0 (low state)

Left shift

N<<i

3<<1

(010 <<1)

(100)

Right shift

N>>i

3>>1

(010 >>1)

(001)

1. Get
2. set
3. clear
4. Update

① Get Bit Operation

Ques) Get the 3rd bit (position = 2) of a number.

n. (n = 0101) = (5) in decimal

$n = 0\,(1)\,0\,1 = 2^0$

3 2 1 0 (positions)

* Bit Mask : 1 << i
* Operation : AND (&)

i = 2 (here) ← position of your bit

(i) 1 << (2) (left shift)

I/P ← 0001 << 2 → bit mask

0100 → O/P

(ii) 0100 AND 0101

= 0100

final NO.

if final no. ≠ zero ⇒ bit = 1
final no. = zero ⇒ bit = 0

Code :

```
int n = 5 ;
int pos = 2 ;
int bitMask = n<< pos ;
if( ( bitMask & n) == 0)
   {
     cout(" bit was zero") ;
   }
else
   {
     cout (" bit was non-zero") ;
   }
```

(2) Set Bit Operation

Ques:) Set the 2nd bit (pos = 1) of a no. n (n = 0101) = $(5)_{10}$

(i) Bit mask : 1<<1

(ii) Operation : OR

(i) 1 << 1

0001 <<1

0010 → o/p

(ii) 0010 OR 0101

$(0111)_2 = (7)_{10}$

Code :-

```
int n = 5;
int pos = 1;
int bitMask = n << pos;
int newNo = (bitMask) | (n);
sout (newNO);
```

(3) Clear Bit

Q) Clear the 3rd bit (pos = 2) of a number.
(n = 0101)

(i) Bit Mask : 1 << 1

(ii) Operation : AND with NOT

Sol)

$n = 0101 = (5)_{10}$
3 2 1 0

o/p find ⇒ 0 0 0 1 $= (1)_{10}$

(i) 1 << 2

0001 << 2

0100 → Bit Mask

(ii) ~(0100) ⇒ (1011)

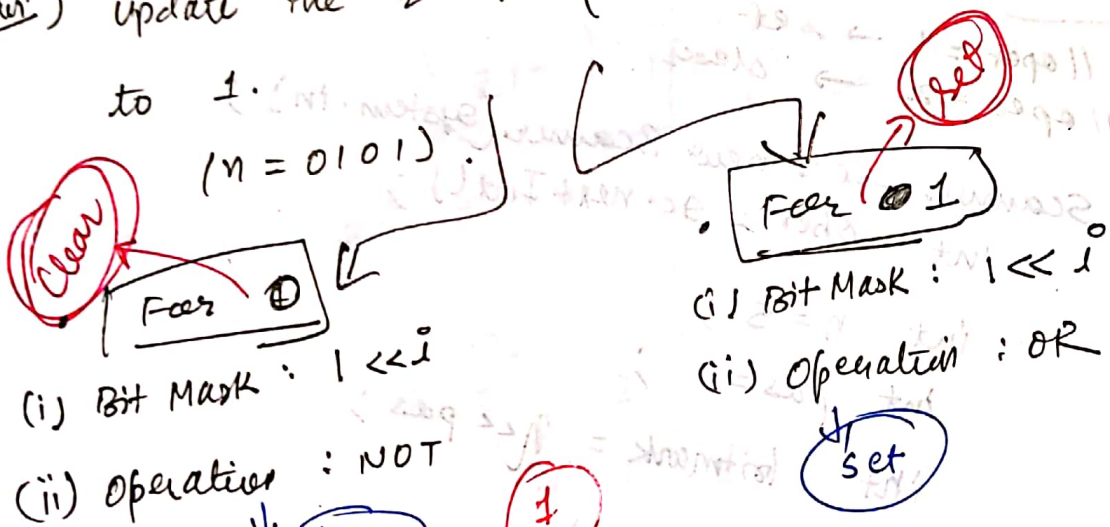
(1011) & (0101) ⇒ (0001) $= (1)_{10}$

Code :

```
int n = 5;
int pos = 2;
int bitMask = n << pos;
int notbitMask = ~(bitmask);
int newNO = (notbitMask) & (n);
Sout(newNO);
```

(4) Update Bit

Ques) update the 2nd bit (pos = 1) of a number n to 1.
(n = 0101)



Sol^n

1

(i) 1 << 1
0001 << 1
0010

(ii) (0010) | (0101)
=> 0111 → o/p

0 (i) 1 << 2
0001 << 2
0100
↳ BM

(ii) ~(0100) & (0101)
(0001) o/p

Code

```
int n = 5 ;
int pos = 1 ;
int oper = 1 ;
//update bit to 1 else update bit to 0

//set operation
int bitMask = 1<<i ;
int newNO = (bitMask) | (n) ;
```

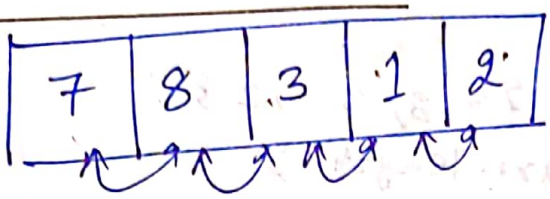
Code

```
// oper = 1 → set
// oper = 2 → clear

Scanner sc = new Scanner(System.in) ;
int oper = sc.nextInt() ;

int n = 5 ;
int pos = 1 ;
int bitmask = n<<pos ;

if (oper == 1)
{
    int newNO = (bitMask) | (n) ;
    sout(newNO) ;
}
else
{
    int new_mask = ~(bitMask) ;
    int newNO = (new_mask) & (n) ;
    sout(newNO) ;
}
```

Lecture - 14

Sorting in Java

(1) Bubble Sort



(Unsorted array)

length = n

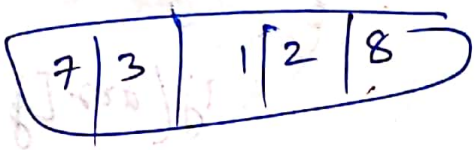
no of pass = (n-1)

(1) 7 8 3 1 2

1st loop

7 3 8 1 2

7 3 1 8 2

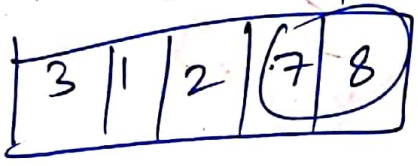


(largest element ⇒ end.

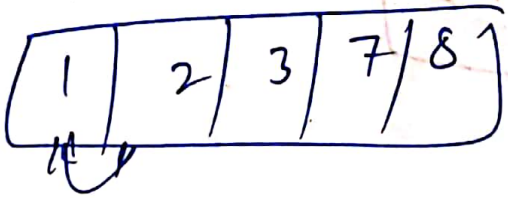
(2) 2nd loop

3 7 1 2 8

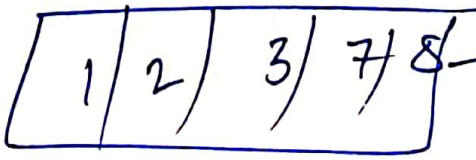
3 1 7 2 8



(3) 1 3 2 7 8



(4)



→ sorted array

* n = length of array

* no. of comparison = $(n-1) \rightarrow (n-i-1)$

Code :

```
int arr[] = {7, 8, 3, 1, 2};
int n = arr.length;
for (i = 0; i < (n-1); i++)          // (n-1)
{
    for (j = 0; j < (n-i-1); j++)    // (n-i-1)
    {
        if (arr[j] > arr[j+1])
        {
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

$T_n = O(n^2)$

(20) Selection Sort

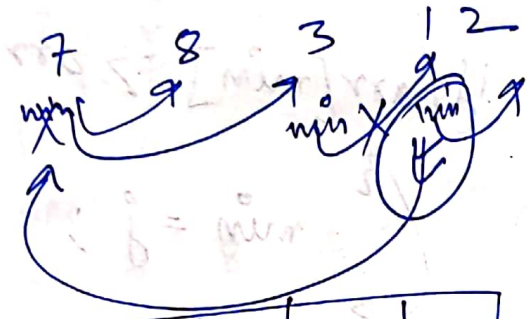
7	8	3	1	2

unsorted array

→ 1 swap per iteration

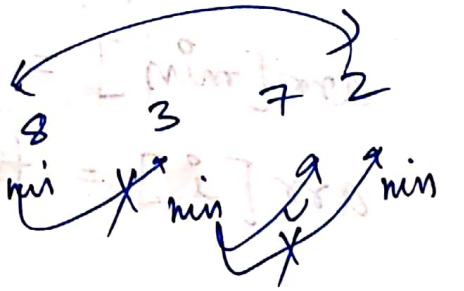
→ select any 1 as min / smallest elemt and then compare it with other array element

(1) $i = 0$



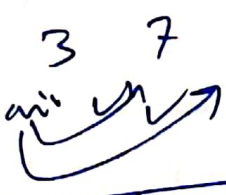
1	8	3	7	2

(2) $i = 1$



1	2	3	7	8

(3) $i = 2$



1	2	3	7	8

(4) $i = 3$

no swaps.
7 is min

1	2	3	7	8

Code

```
int arr[] = { 7, 8, 3, 1, 2 };
    int n = arr.length;
for(i=0 ; i<n ;i++)
  {
      int min = i ;
for(j=i+1; j<n ;j++)
  {
    if(arr[min] > arr[j])
      {
        min = j ;
      }
  }
int temp = arr[min];
arr[min] = arr[i];
arr[i] = temp;

}//for
```

min elem ko min assume & compare with rest

$T_n = O(n^2)$

(3) Insertation Sort

7	8	3	1	2

Divide Array into 2 parts

Sorted ↙ ↘ Unsorted

Sorted | Unsorted

(1) 7 | 8 3 1 2

place

compare 8 & 7 and place accordingly

(2) 7 8 | 3 1 2

3 7 8 | 1 2

(3) 3 7 8 | 1 2

(4) 1 3 7 8 |

1	2	3	7	8

sorted

Code

```
int arr[] = {7, 8, 3, 1, 2};
int n = arr.length;

for(int i=1; i<n; i++)
{
    int curr = arr[i];
    int j = i-1;

    while(j>=0 && curr < arr[j])
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = curr;
}
```

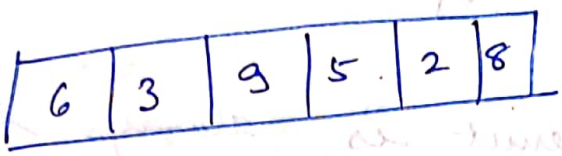
i=1, i=0 sorted part

* $T_n = O(n^2)$

Lecture -19

Quick Sort

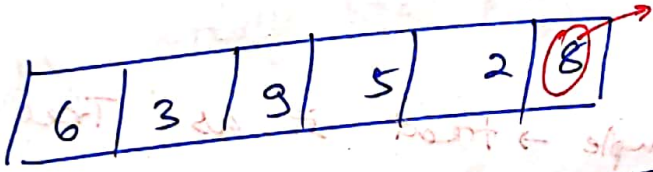
Unsorted array :



Pivot & Partition

To choose Pivot ?

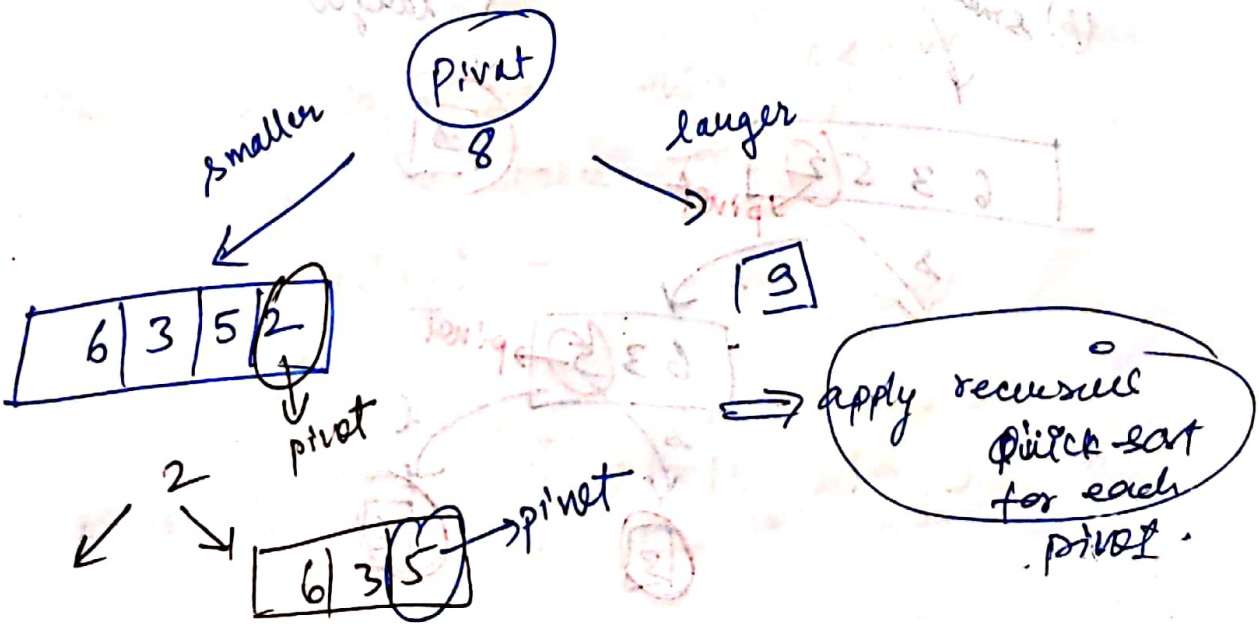
1) random
2) median
3) 1st elemt
4) last elemt



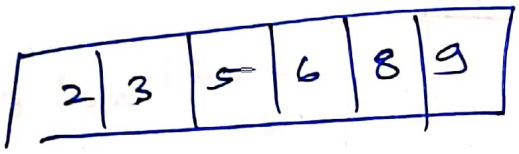
** Karan.

element < pivot ⇒ keep them at left

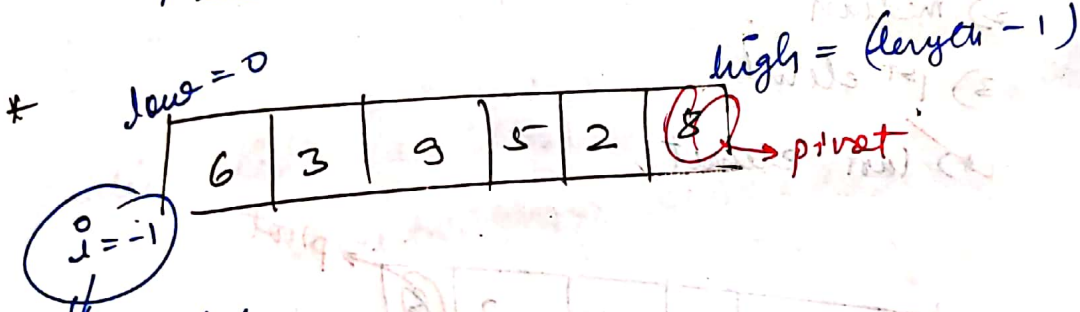
element > pivot ⇒ send them Right



* Single element is always sorted
* see from left to right in this recursive tree

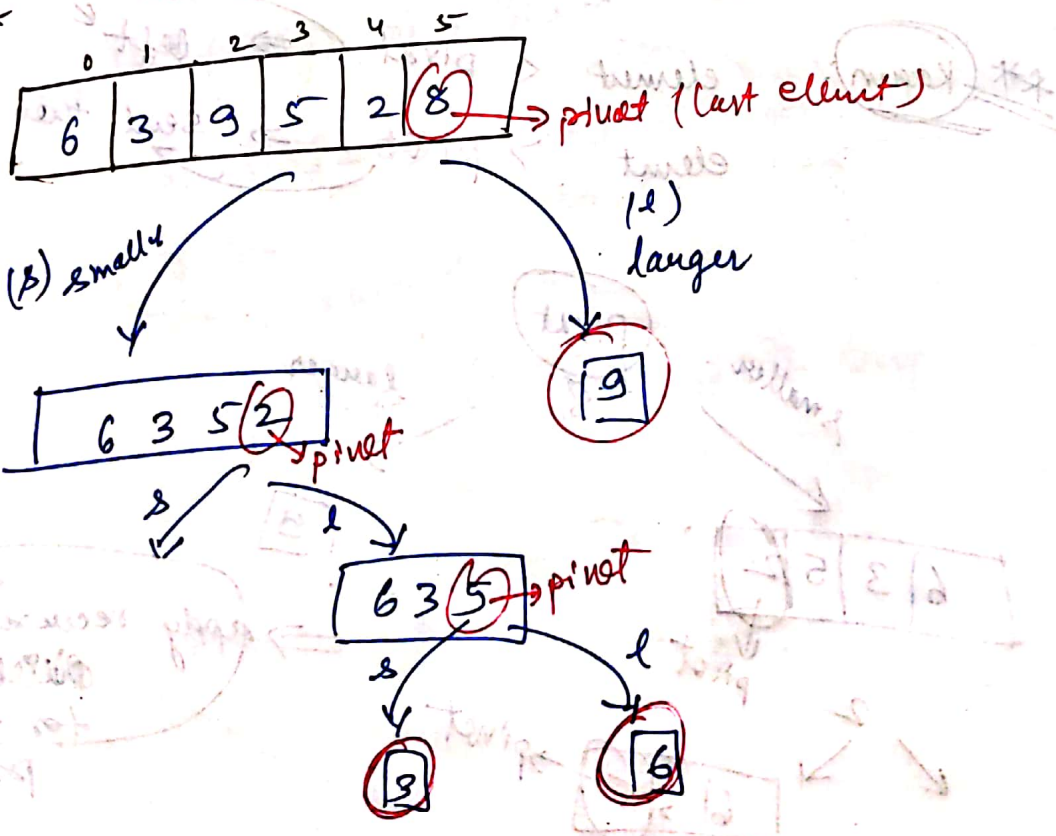


=> sorted Array



Sample → treat it as Tree

Note



⇒ Move from left to right in tree we get :

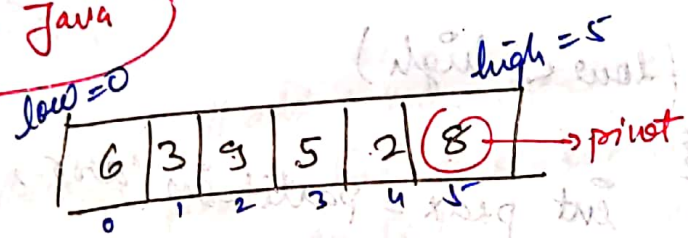
2, 3, 5, 6, 8, 9

⇒

0	1	2	3	4	5
2	3	5	6	8	9

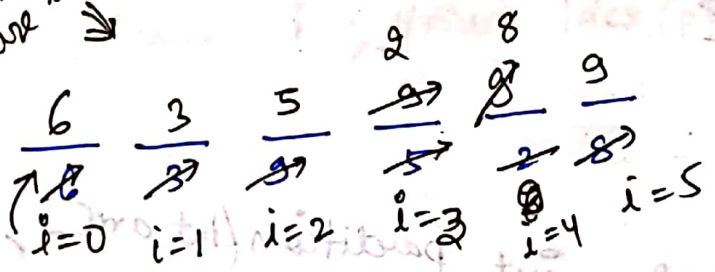
⇒ sorted array

Code in Java



i = -1 (no element less than pivot initially bzc pivot not selected)

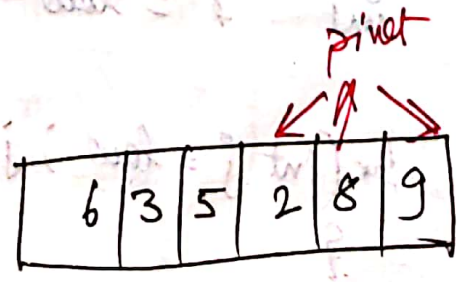
(Swap method kii use kara)

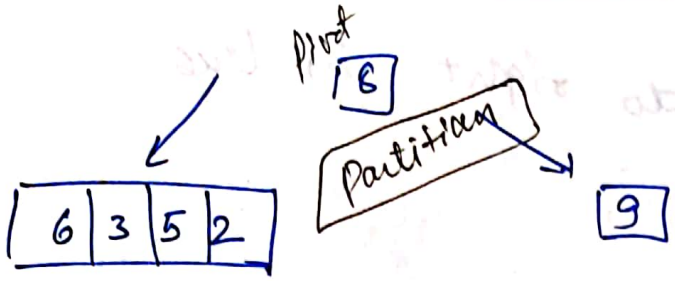


* Compare 6 with pivot, then (6 < pivot) only if follow karega then i will create space for '6'. (i++)

⇓

Now we get :





apply again same quick sort (recursive use karo)

Code

```
public static void quick(int arr[], int low, int high)
{
    if (low < high)
    {
        int pidx = partition(arr, low, high);
        quick(arr, low, pidx-1);   // less than pivot pat
        quick(arr, pidx+1, high);  // more than pivot pat
    }
}

public static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = low-1; //track for empty space

    for(int j = low; j< high ;j++)
    {
```

```
if(arr[j] < pivot) //element < pivot
{
    i++; //space create to store smaller element
    //swap → pehle jo us position par store element with abb jo small element mila.
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}//if
}//for
i++; //add 1 more space for pivot.
//swap pivot → b=c pivot end me hai abhi→to wapas sahi position pare lane ke liye
int temp = arr[i];
arr[i] = pivot;
arr[high] = temp;
return i; //pivot idex (position)
}//fn

Main()
{
    int arr[] = {6,3,9,5,2,8};
    int n = arr.length;
    quick(arr, 0, n-1); //
}
```

Time Complexity ⇒ Quicksort

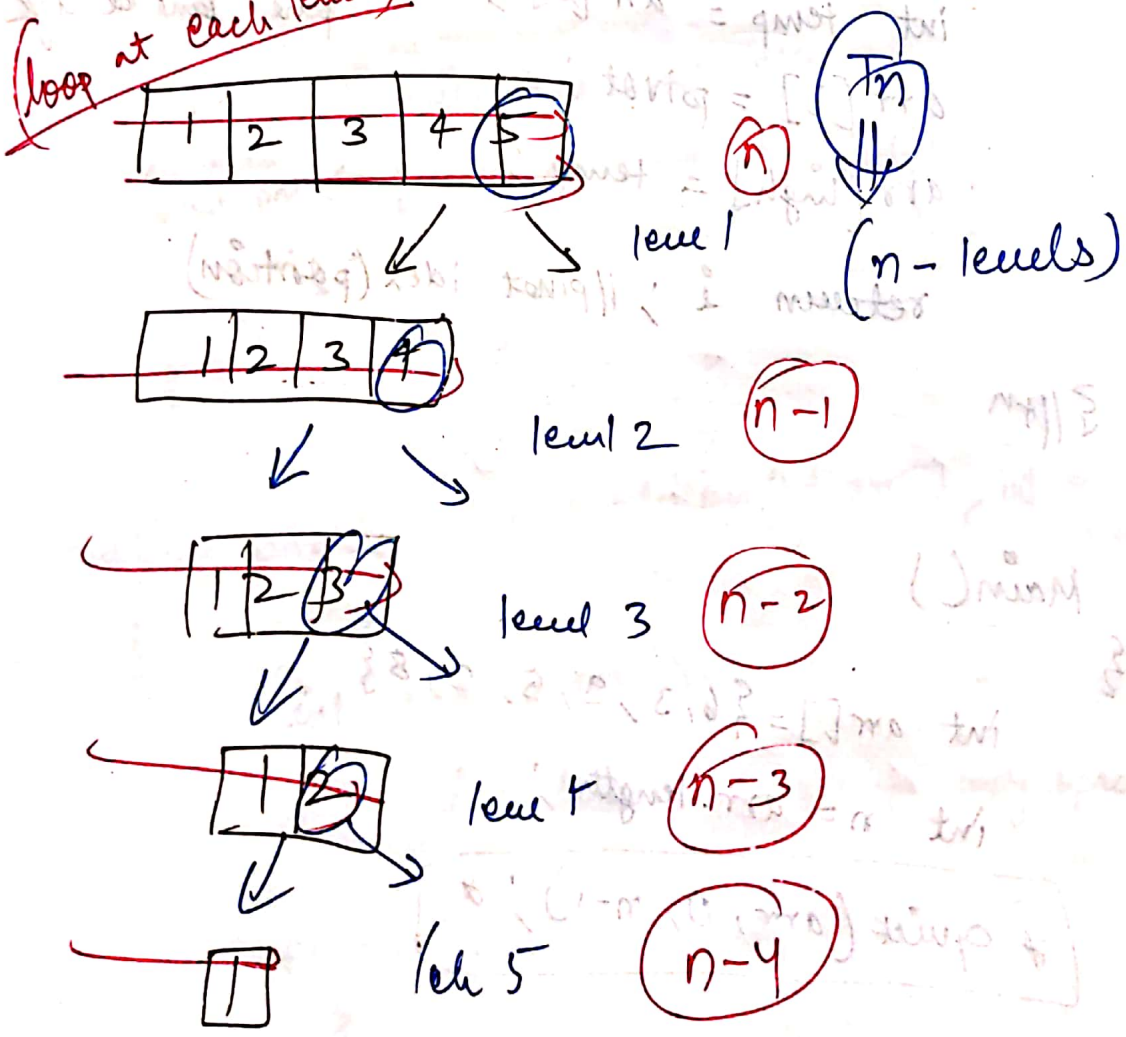
* **Worst Case** occurs when pivot is always the smallest or the largest element.

Worst Case : $T_n = O(n^2)$

Avg. Case : $T_n = (n \log n)$

→ when our array is completely sorted or ~~reversed~~ (either ascendig or descendig we sort)

(loop at each level)



⇒ Har level par loop chalane me

$T_n = O(n)$

⇒ $n + (n-1) + (n-2) + \cdots + 1$ (A.P)

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

$T_n = O(n^2)$ → $\therefore T_n = O(n^2)$

at worst case.

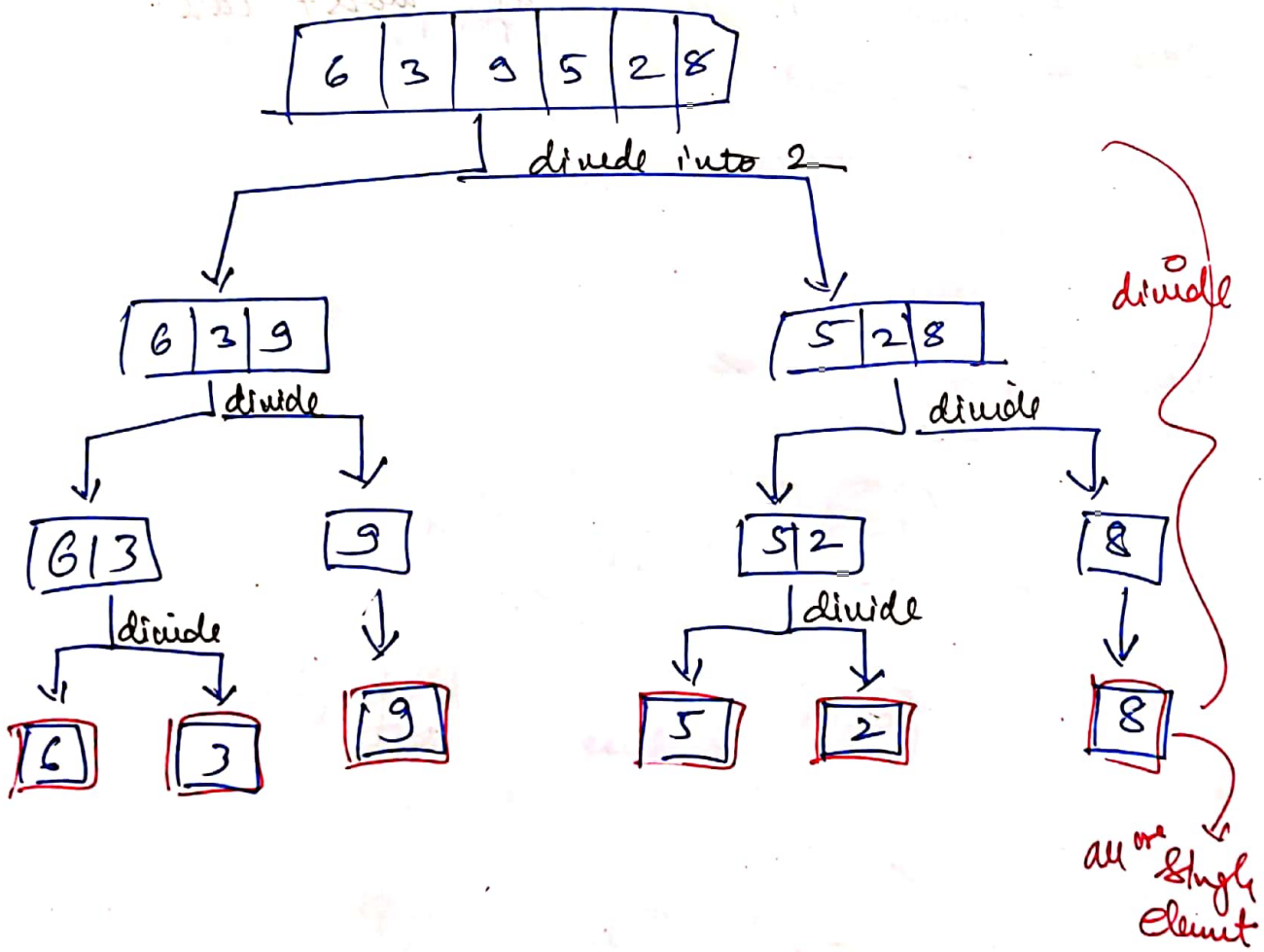
Lecture -20

Merge - Sort

Unsorted

6	3	9	5	2	8

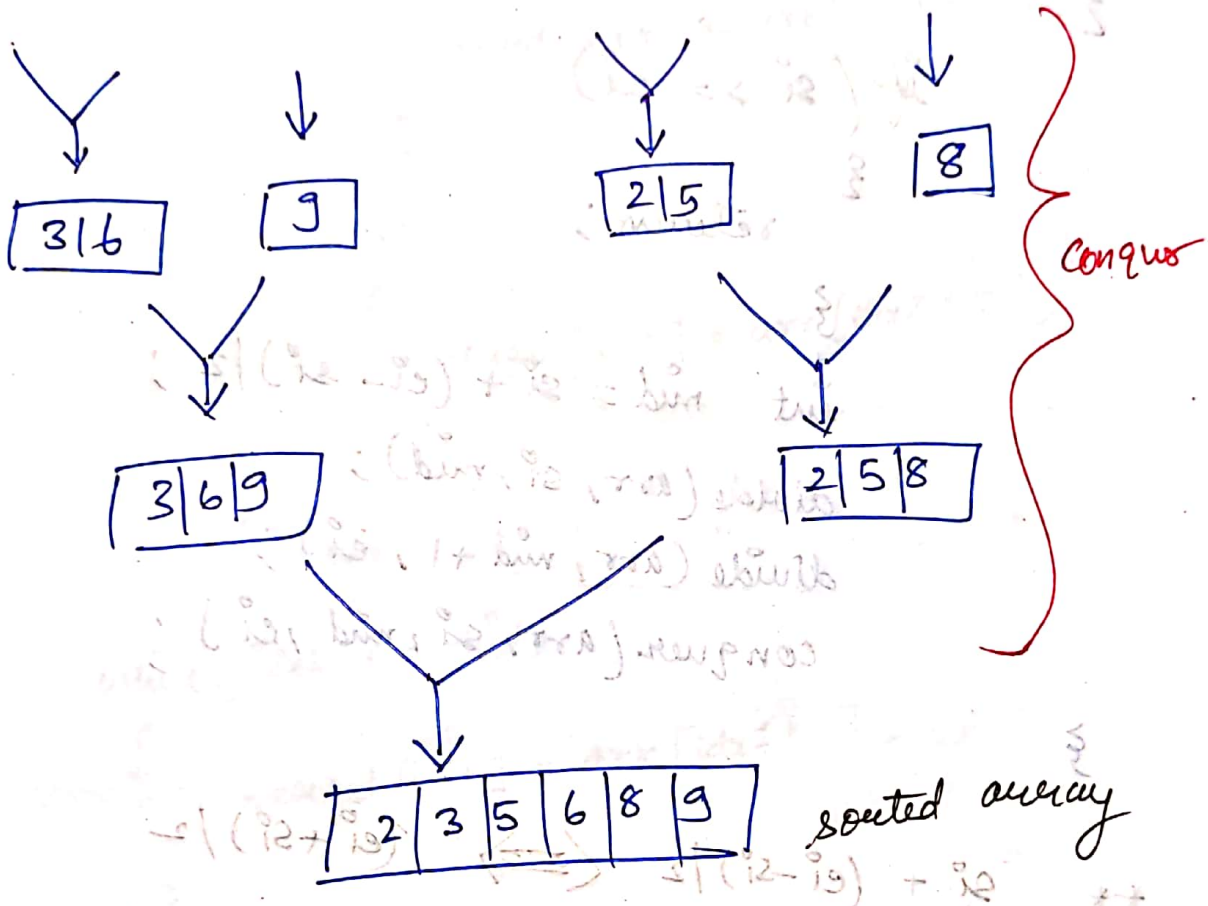
* Based on :
  Divide & Conquer Property

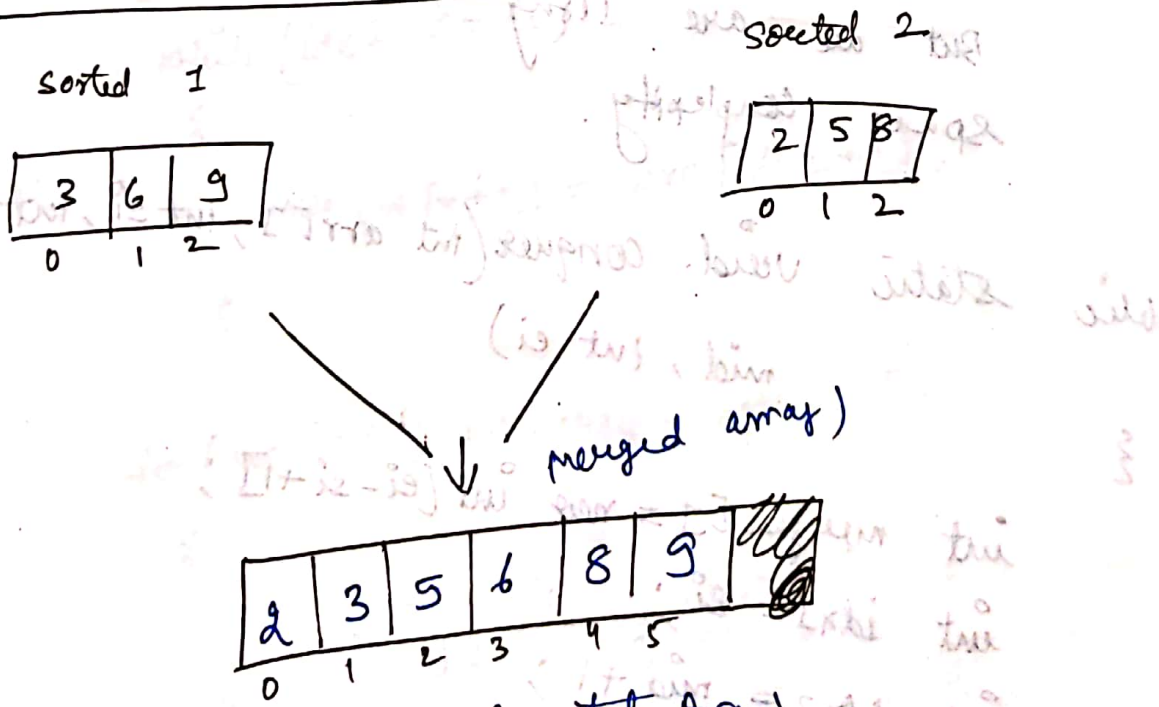


* Single element are already sorted.
* Now, Conquer these single elements and

while doing so make the comparison (i.e., sort them)

Now, extending prev. Tree structure :-





si = starting index
ei = ending index

Code

```
public static void divide(int arr[], int si, int ei)
{
    if (si >= ei)
    {
        return;
    }
    int mid = si + (ei - si)/2;
    divide(arr, si, mid);
    divide(arr, mid+1, ei);
    conquer(arr, si, mid, ei);
}
```

** $si + (ei - si)/2 \iff (ei + si)/2$

⇓

But we are using this to reduce space complexity.

```
public static void conquer(int arr[], int si, int
                           mid, int ei)
{
    int merged[] = new int[ei - si + 1];
    int idx1 = si;
    int idx2 = mid + 1;
    int x = 0;
```

```
while(idx1 <= mid && idx2 <= ei)
{
    if(arr[idx1] <= arr[idx2])
    {
        merged[x++] = arr[idx1++];
    }
    else
    {
        merged[x++] = arr[idx2++];
    }
} // while

while(idx1 <= mid)
{
    merged[x++] = arr[idx1++];
}

while(idx2 <= ei)
{
    merged[x++] = arr[idx2++];
}

for(int i=0; i<merged.length; i++)
{
    arr[j] = merged[i];
} // for

} // fn
```

```
main()
{
    int arr[] = {6, 3, 9, 5, 2, 8};
    int n = arr.length;
    divide(arr, 0, n-1);
}//main
```

$$T_n = O(n \log n)$$

* $\log_2 n$ (divide)
* n (conquer)

Lecture 15

Recursion - 1

Recursion

Prerequisites

i) iteration / loop
ii) function

* ① outer f^n → takes steps
* ② inner f^n → assume rest of work is done by it

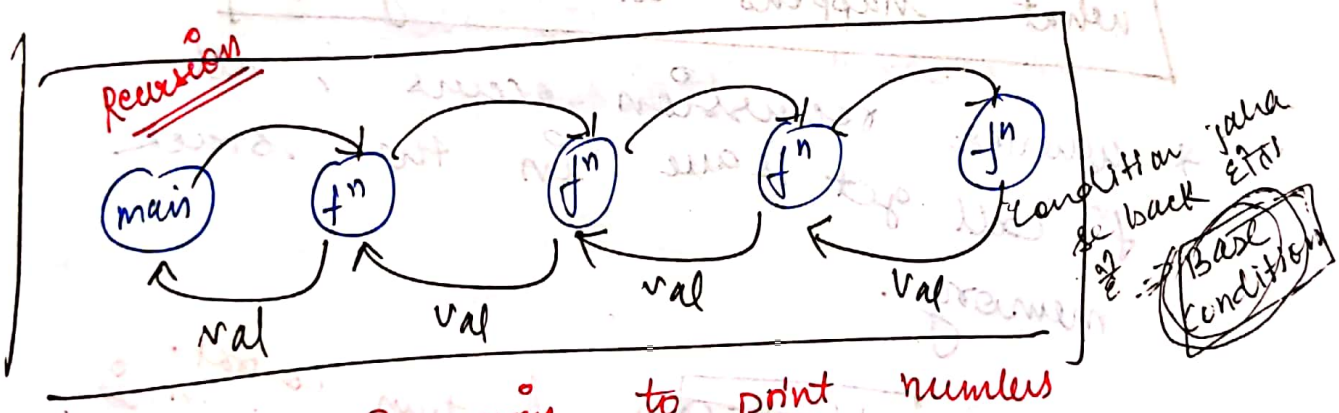
$f(x) = x^2$ → given

$f(f(x)) = f(x^2)$

Recursion

↓↓

function that calls itself

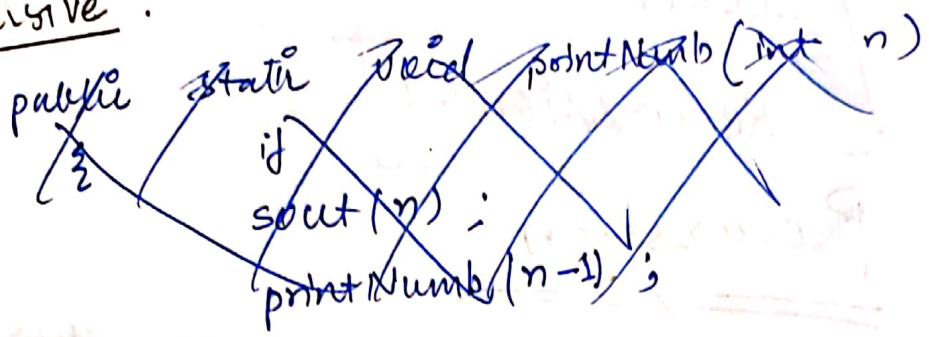


Ques: Use Recursion to print numbers from 5 to 1.

iteration :

```
for (i=5; i>0; i--)
{
    cout(i+" ");
}
```

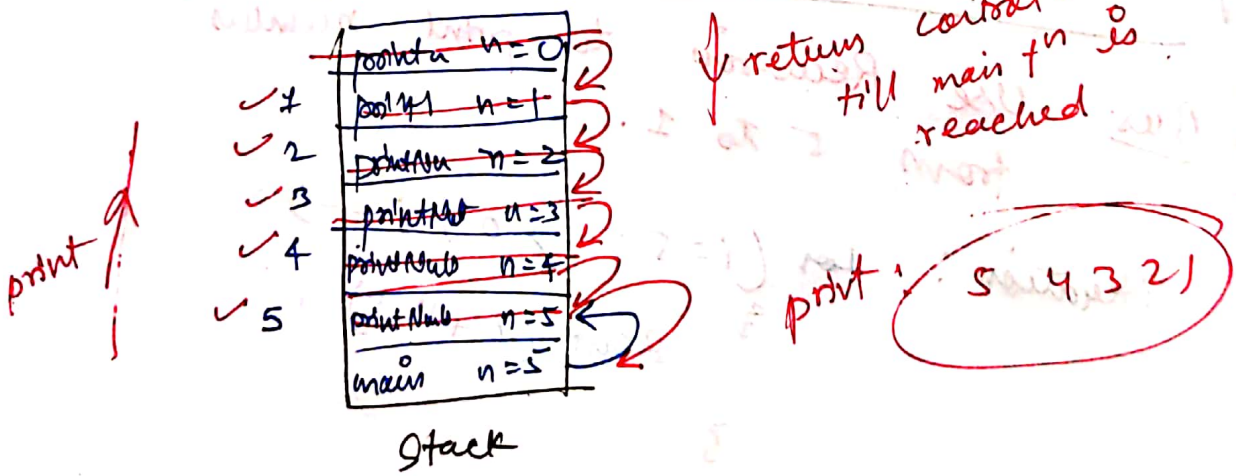
Recursive :




```
public static void printNumb (int n)
{
    if (n == 0)
        return ;          } Base condition
    sout (n) ;  // print
    printNumb (n-1) ;  //recursion
}
```

what happens in Memory ?

* whenever recursion occurs ; all fn call get save in the stack memory.



* Recursion uses more memory. Variables gets created again & again

in the memory.

* when base condition is not there, then after some time of execution when our memory gets full

// Stack overflow.

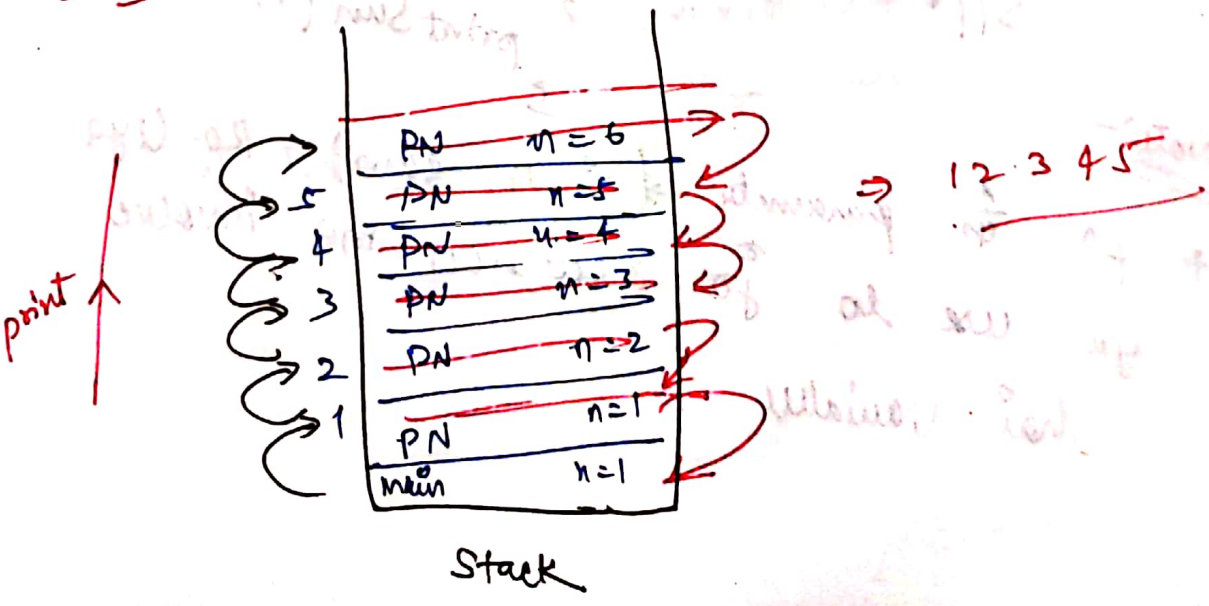
// (∞ recursion in case of n)

(Ques 2)

Print Numbers from 1 to 5

```
public static void printNumb(int n)        n = 1
{
    if (n == 6)
        return ;        } Base

    sout(n); // print
    printNumb(n+1); // recursion
}
```

Memory Space



Ques: Print Sum of n-natural No.

① main f^n ⇒ 1

② Base Condition ⇒ 'n' (sum)

③ work ⇒ Calculate sum

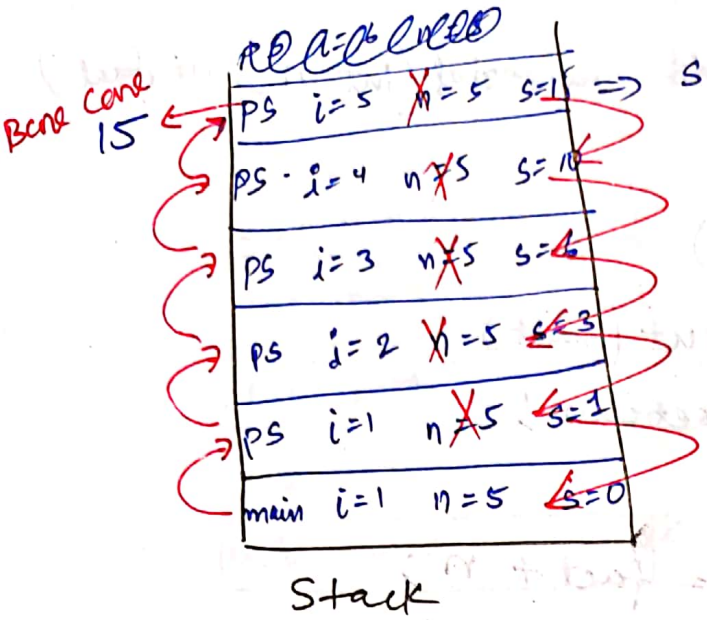
Code:

```
public static void printSum(int i, int n, int sum)
{
    if (i == n)
    {
        sum = sum + i;
        sout(sum);
        return;
    }
    sum = sum + i;   // sum calculate
    printSum(i+1, n, sum);  // recursion
}// fn

main()
{
    printSum(1, 5, 0);
}
```

base condition

Note:-

* f^n ke 1 parameter → (i, n, sum) ko liya ya use ko fo recursion me involve hai variable



Ques: Print Factorial of a Number 'n'

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

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

$$1! = 1$$

$$0! = 1$$

(1) Infor. ⇒ $n = 5$

(2) Kaam ⇒ i) $(n-1)!$

ii) $n \times (n-1)!$

↳ return

(3) Base ⇒ $(n == 1)$

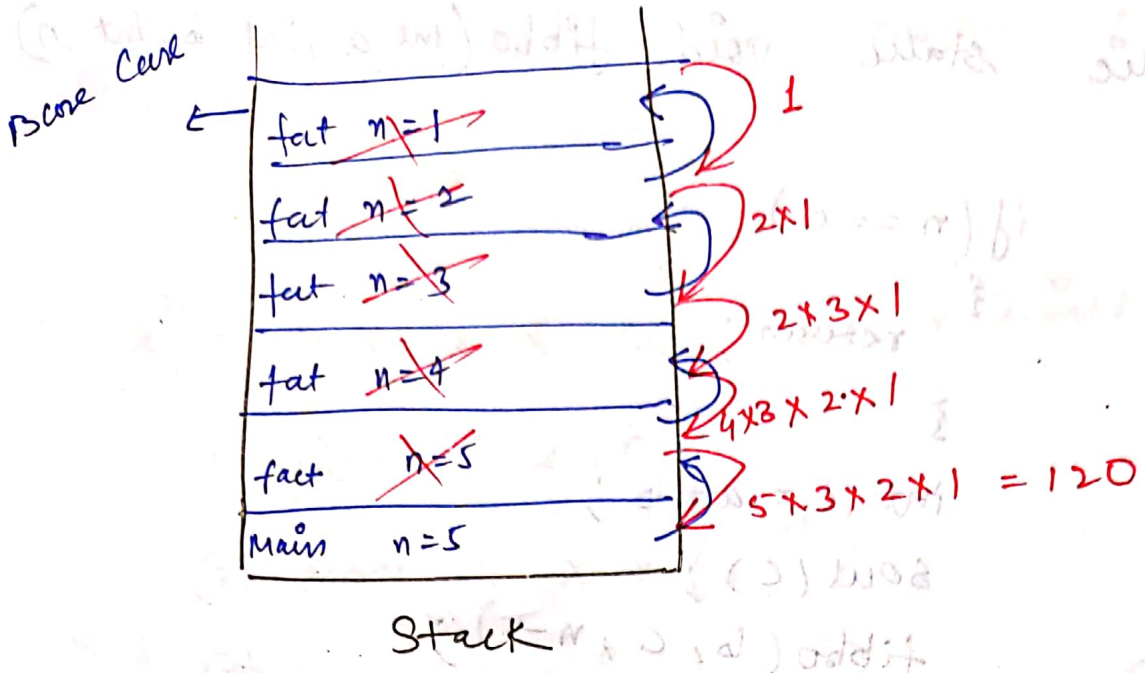
return 1

```
public static void factorial(int n, int fact)
{
    if(n == 0)
    {
        sout(fact);
        return;
    }
    fact = fact * n;
    factorial(n-1, fact);
}
main()
{
    factorial(5,1);
}
```

OR

```
public static int factorial(int n)
{
    if(n==1 || n==0)
    {
        return 1;
    }
    int fact_nm1 = factorial(n-1);
    int fact_n = n * fact_nm1;
    return fact_n;
}
```

```
Main()
{
int f = factorial(5);
sout(f);
}
```



Ques: Print the fibonacci sequence till n^{th} term:

Fibbonaci series: 0 1 1 2 3 5 . . . .

1st 2 no.

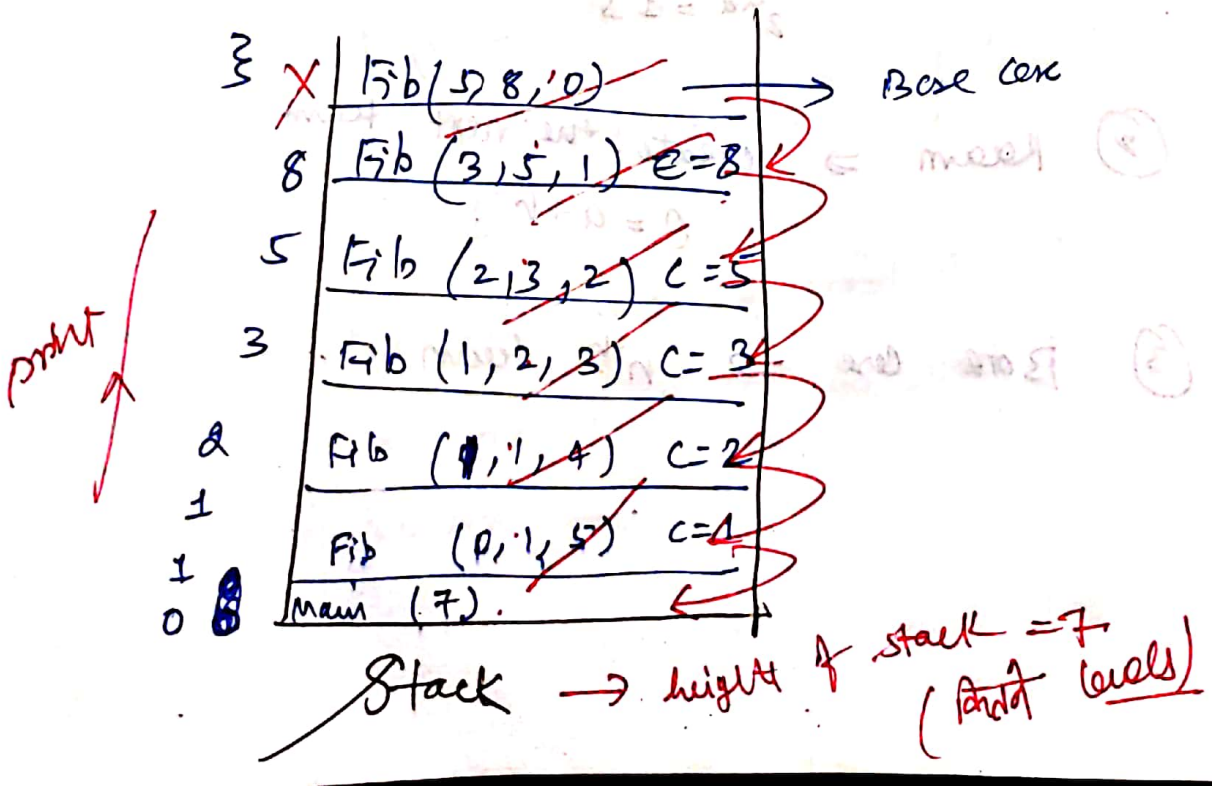
(1) given ⇒ 1st = 0, 2nd = 1

(2) Ream ⇒ create the next term

$c = a + b$

(3) Base case ⇒ n^{th} term

```
public static void fibbo (int a, int b, int n)
{
    if (n == 0)
    {
        return;
    }
    int c = a + b;
    sout (c);
    fibbo (b, c, n-1);
}
main ()
{
    int a = 0, b = 1;
    sout (a);
    sout (b);
    int n = 7;
    fibbo (a, b, n-2);
}
```



Ques) Print x^n (stack height = n)

* $(x)^n$ *

$x^n = x \times x \times x \times \ldots$ n times

① given : x, n

② work : $x * (x^{n-1})$

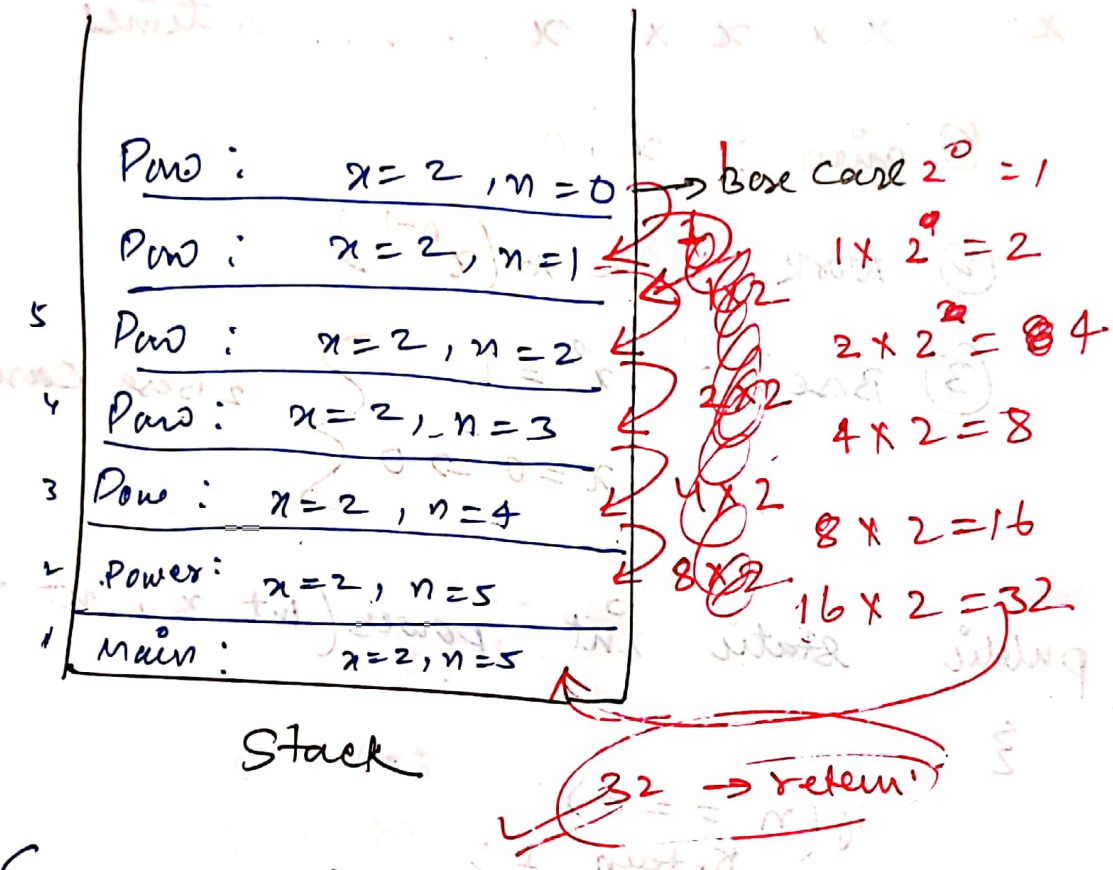
③ Base : $x^0 = 1$
$x = 0 \Rightarrow 0$
} 2 base case

```
public static int power(int x, int n)
{
    if(n == 0)
        return 1;

    if (x == 0)
        return 0;

    int xPownm1 = power(x, n-1);
    int xPow = x * xPownm1;
    return xPow;
}

main()
{
    int x=2, n=4;
    int ans = power(x, n);
} Sout(ans);
```

Ex:

$x = 2$, $n = 5$

$x^n = (2)^5 = \underline{32}$



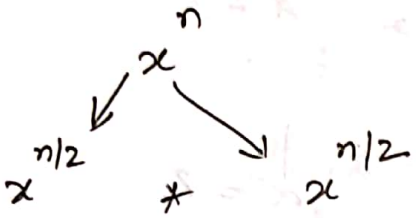
$\{ n = 5 \}$

$\{ level = 6 \}$

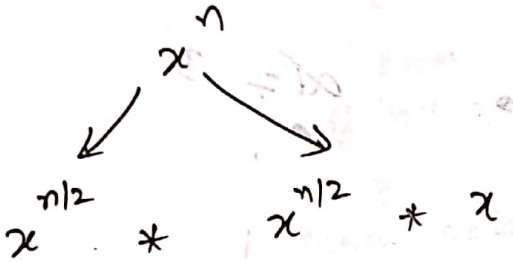
$T_n = O(n)$

Ques) Print $(x)^n$ [stack height = $\log n$]

stack ht = $\log n$



(n = even power)

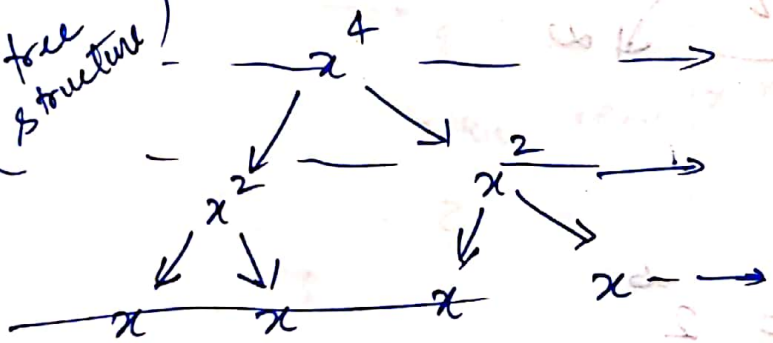


(n = odd power)

n = java integer

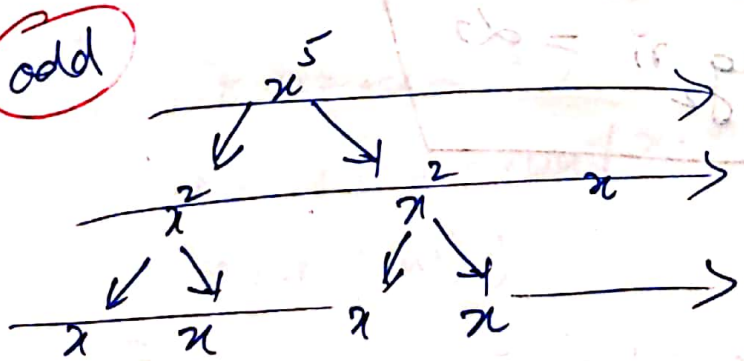
Even

(tree structure)

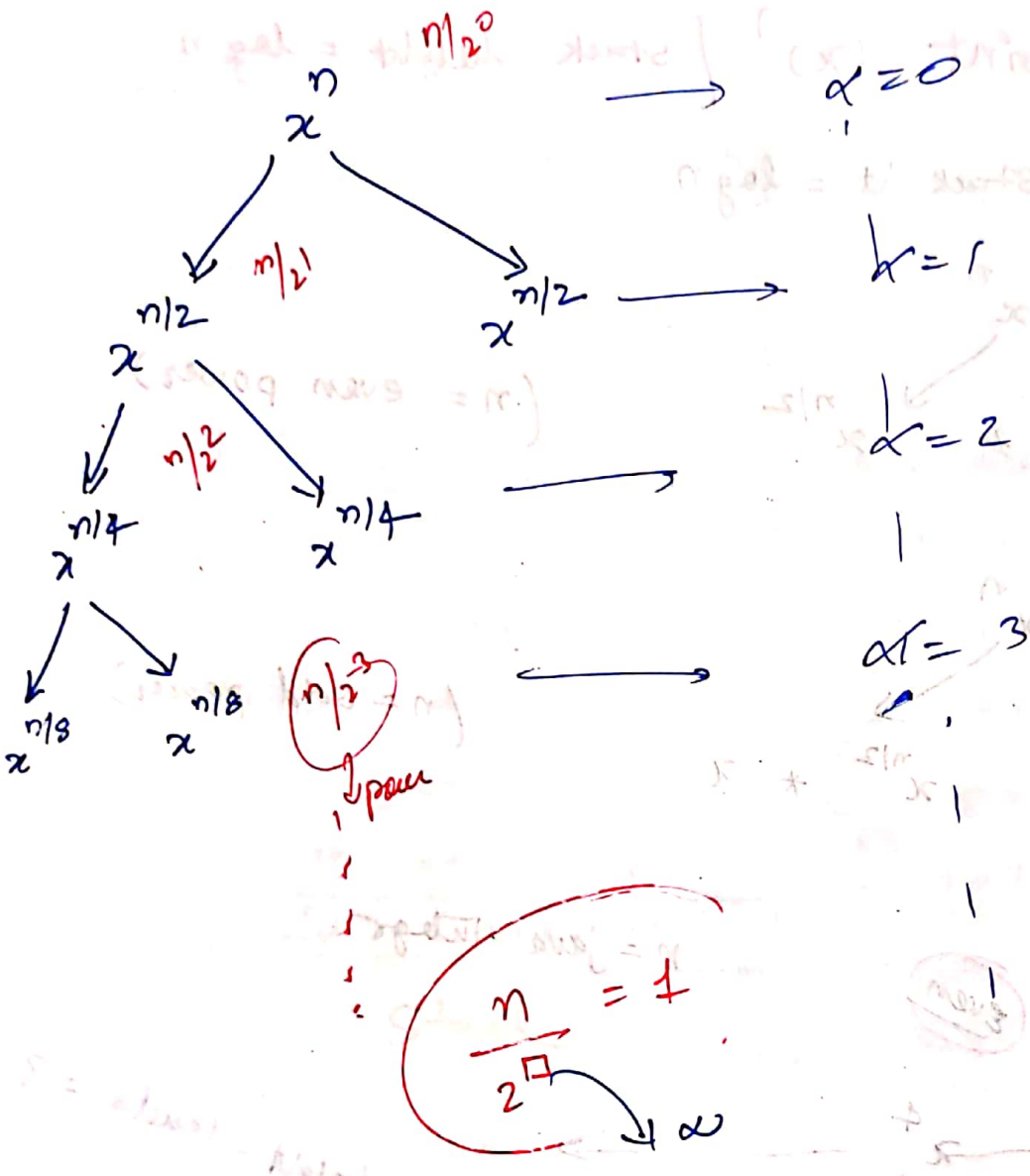


height = levels = 3

Odd



height = level = 3



$$\frac{n}{2^{\alpha}} = 1$$

$$n = 2^{\alpha}$$

$$\boxed{\log_2 n = \alpha}$$

```
public static int power(int x, int n)
{
    if(n == 0)
    {
        return 1;
    }
    if (x == 0)
    {
        return 0;
    }
    //Even
    if(n%2 == 0)
    {
    return power(x, n/2) * power(x, n/2);
    }
    //odd
    if(n%2 != 0)
    {
    return power(x, n/2) * power(x, n/2) * x;
    }
}

Main()
{
    int x=2, n=5;
    int ans = power(x, n);
    Sout(ans);
}
```

Base Case

$$T_n = O(\log n)$$

Lecture - 16

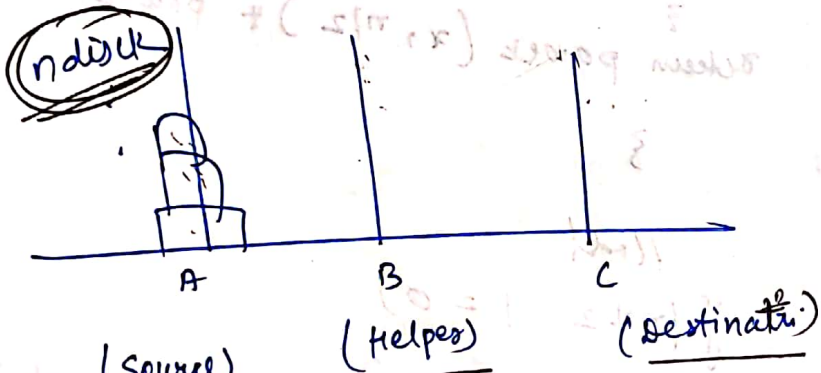
Recursion - 2

(Q1) Tower of Hanoi

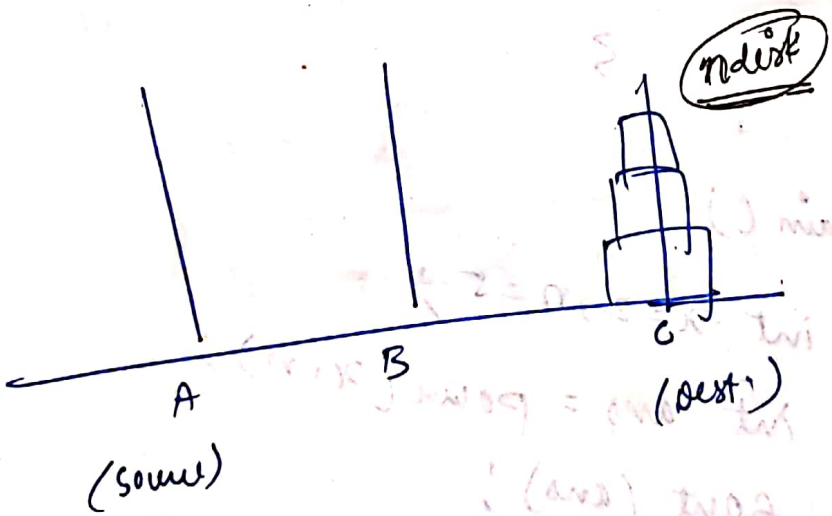
Rules:

* Only 1 disk transferred in 1 step
* Smaller disk are always kept on top of larger disks

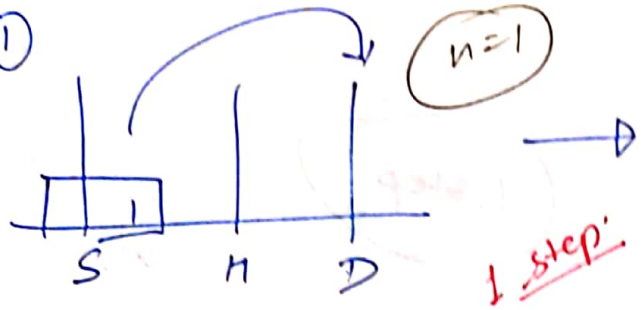
Start



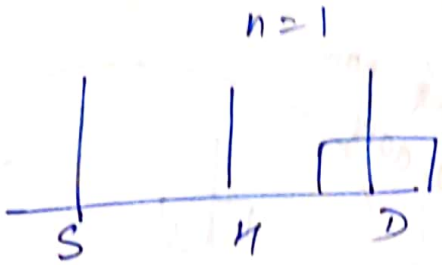
End



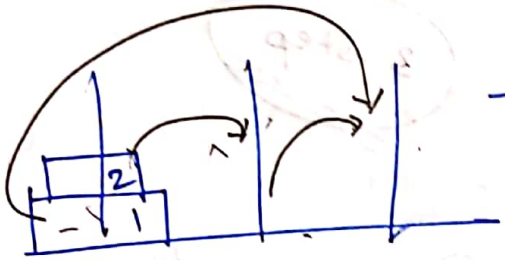
① n=1

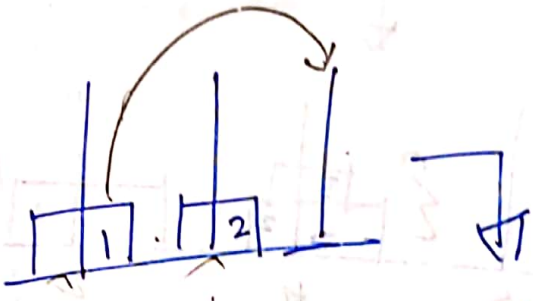


1 step.

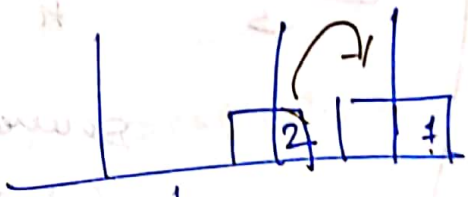


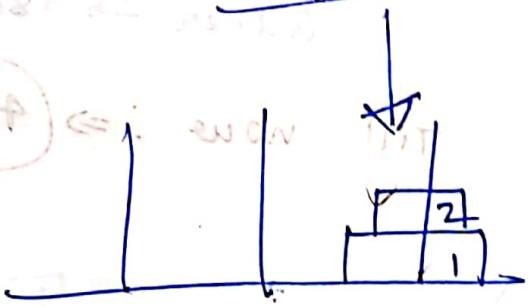
② n = 2



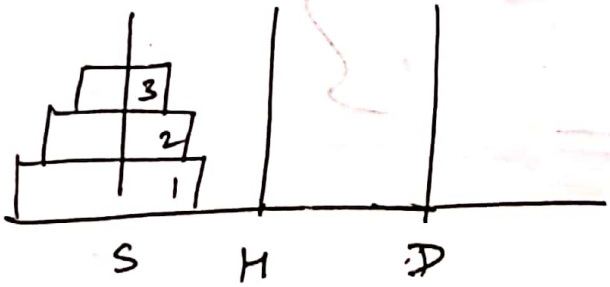


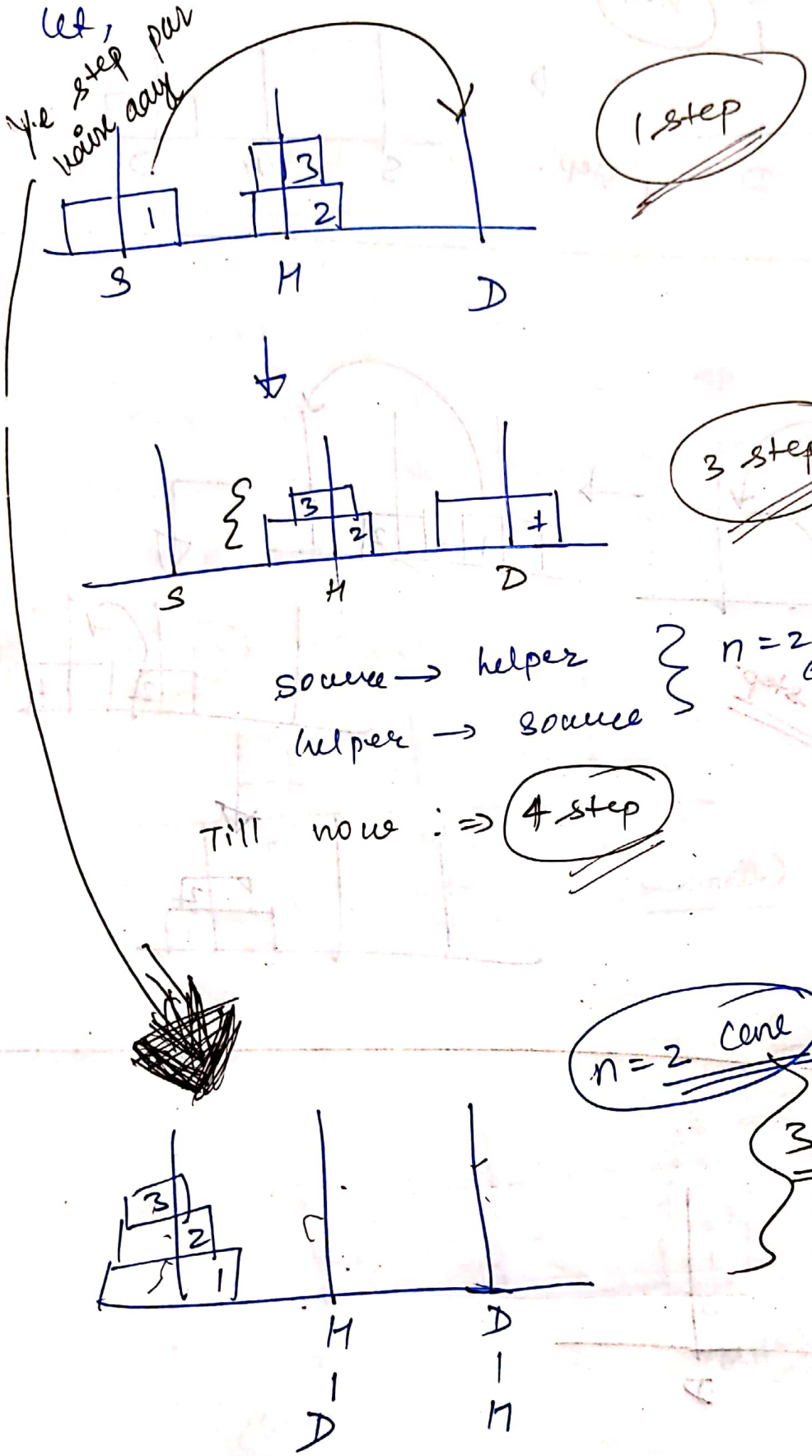
3 step.





③ n = 3

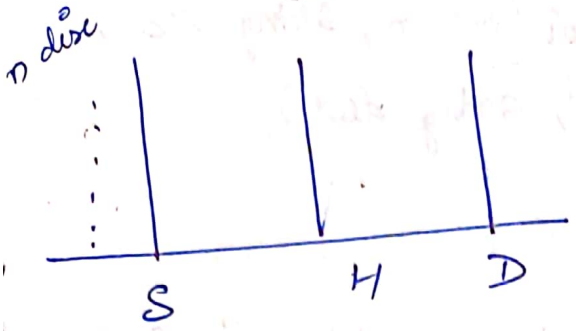




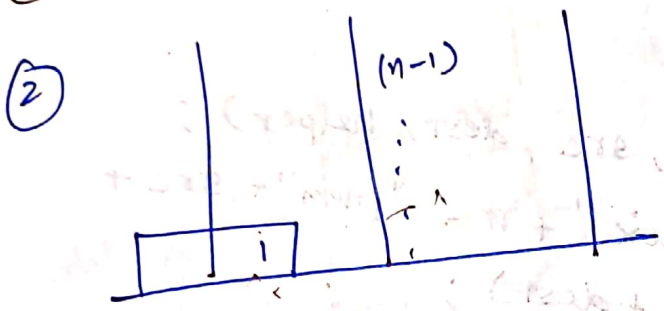
∴ Total step = (3) + (1) + (3)

= 7 step.

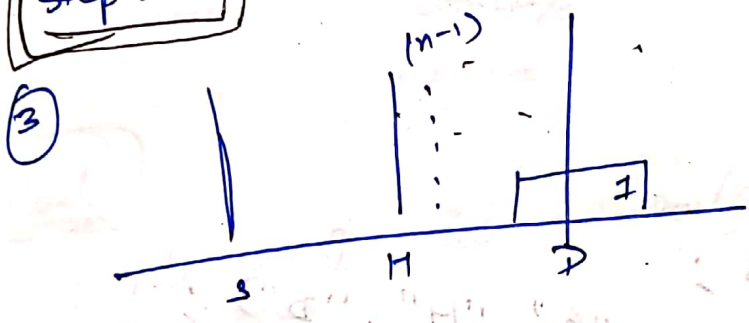
① For General Case



Step 1 : (n-1) disc ⇒ Helper tak pahuchana de



Step 2 : "1" disc → source to destination



(n-1) disc H → D

Using source as helper

Code

```
public static void towerHanoi (int n, String src,
                String helper, String dest)
{
    if (n == 1)
    {
    sout ("transfer disk " + n + "from " + src
        + " to " + dest);
    return;
    }

towerHanoi (n-1, src, dest, helper);
sout ("transfer disc " + n + "from" + src +
        "to" + dest);
towerHanoi (n-1, helper, src, dest);

}

main ()
{
    int n = 3;
towerHanoi (n, "S", "H", "D");
}
```

(#) Time Complexity $\Rightarrow T_n = O(2^n - 1) \approx O(2^n)$

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

(2 times (n-1) disc transferred + 1 step)

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

$T(n-2) = 2T(n-3) + 1$

$\vdots$

$T(1) = 1$

$n - 2 = 1$

$n - x = 1$

$x = n - 1$

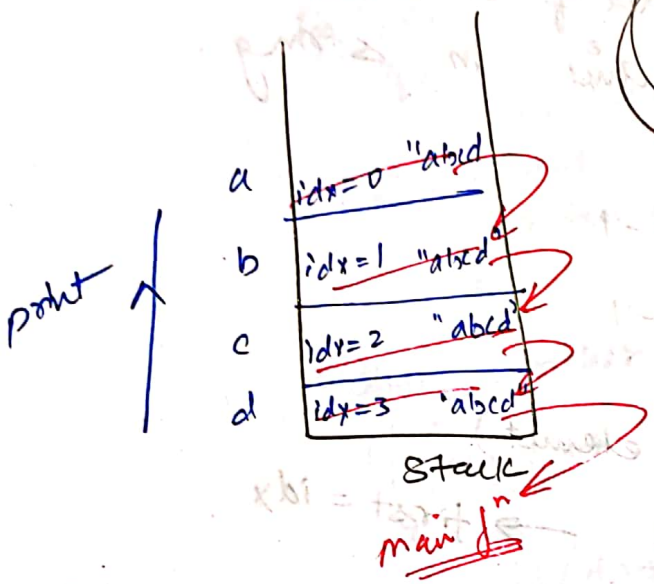
$2^{n-1} T(1) + 2^{n-2} + 2^{n-3} + \ldots$

$= -2^n$

(Q2) Print a String in Reverse

"abcd" ⟹ "dcba"

$T_n = O(n)$ — n = length of string.



idx = (str.length − 1)

Code

```
public static void printRev(String str, int idx)
{
    if (idx == 0)
    {
        sout(str.charAt(idx));
        return;
    }
    sout(str.charAt(idx));
    printRev(str, idx-1);
}
Main()
{
    String str="abcd";
    printRev(str, str.length()-1);
}
```

Ques3) Find the 1st & last occurance of an element in string.

"abaacdaefaah"

↑ string

let, element = a

then get index of a when it's 1st time & last time in string.

initialize : first = -1

last = -1

(currchar == element)

→ first == -1 → first = idx

→ else ⇒ last = idx

* value of 1st & last = remains find in stack only idx value in stack changes.

$T_n = O(n)$

↑ length of string

Code

```
public static int first = -1;
public static int last = -1;

public static void findOccur(String str, int idx,
                             char element)
{
    if(idx == str.length())
    {
        sout(first);
        sout(last);
        return;
    }
    char currchar = str.charAt(idx);
    if(currchar == element)
    {
        if(first == -1)
        {
            first = idx;
        }
        else
        {
            last = idx;
        }
    }
    findOccur(str, idx+1, element);
}
main()
{
    String str = "abaacdabad";
    findOccur(str, 0, 'a');
}
```

(Q4) Check if an Array is sorted (strictly increasing)

Arr[] = {1, 2, 3, 4, 5}

Code:-

```
public static boolean isSorted(int arr[], int idx)
{
    if(idx == arr.length-1)
    {
        return true;
    }
    if(arr[idx] < arr[idx+1])
    { // array is sorted till now
        return isSorted(arr, idx+1);
    }
    else
    {
        return false;
    }
}

main()
{
    int arr[] = {1, 3, 5};
    sout(isSorted(arr, 0));
}
```

CR.

```
{ inside f(n).
    if (idx == arr.length - 1)
    {
        return true;
    }
    if (arr[idx] >= arr[idx+1])
    { // array is unsorted
        return false;
    }
    return isSorted(arr, idx+1);
}
```

$T_n = O(n)$

↑ length of array

(Qs) Move all 'x' to the end of the string

I/p ⇒ "axbcxxd"

⇓

O/p ⇒ "abcdxxx"

idx ⟶ currChar (current character)

count ⟶ track no. of x

newString ⟶

code:

```
public static void moveAll(String str, int idx,
                           int count, String newString)
{
    // Base case
    if(idx == str.length())
    {
        sout(newString);
    }

    char currChar = str.charAt(idx);
    if(currChar == 'x')
    {
        count++;
        moveAll(str, idx+1, count, newString)
    }
    else
    {
        newString += currchar;
        moveAll(str, idx+1, count, newString);
    }
}
```

```
main()
{
    string str = "axbcxxd";
    moveAll(str, 0, 0, "");
}                          ↑ empty string
```

// Base Case of n-Recursion:

```
if(idx == str.length())
{
    for(i = 0; i < count; i++)
    {
        newString += 'x';
    }
    sout(newString);
    return;
}
```

$T_n = O(n + count)$ (count: n)

$= O(2n)$

$= O(n)$

$T_n = O(n)$

length of string

(Q6) Remove Duplicate in a String

I/p => str = "abbccda"

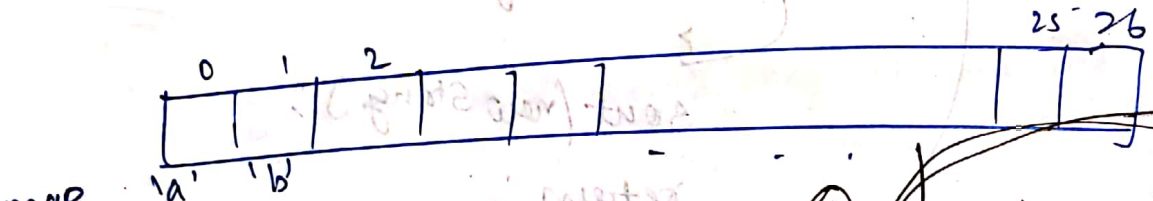
O/p => "abcd"

(i) * create array to track - keep of duplicate element

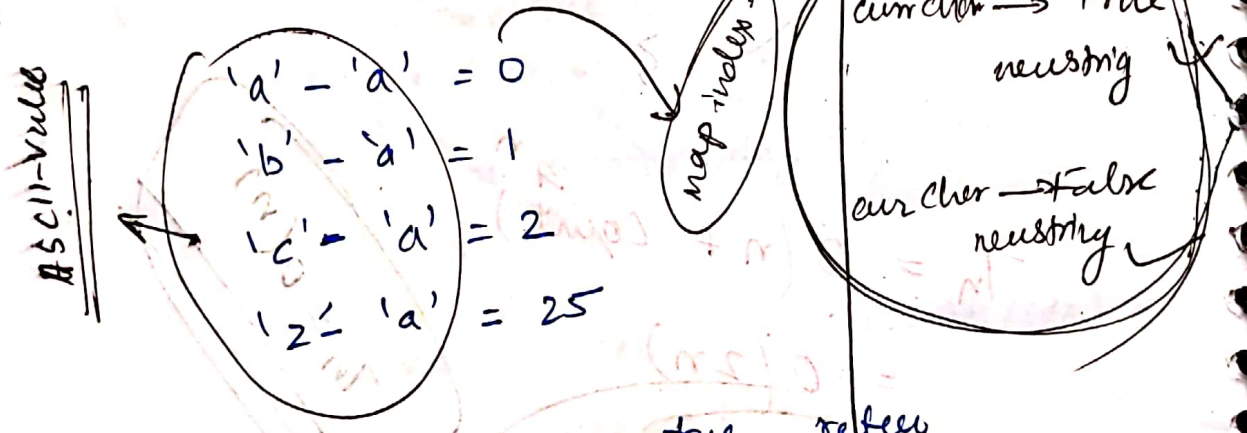
(ii) * size of array = 26

⇓

no of alphabet = 26



map



if character found ⇒ true refer

else false

curr-character = True

⇓

new string me add kardo kaunse

Cur character = false

⇓

add in new string.

// map[pos] = True

to keep track of whenever we character we have got.

Code:

```
public static boolean[] map = new boolean[26];

public static void removeDup(String str, int idx, String newString)
{
    //Base case
    if (idx == str.length())
    {
        sout(newString);
        return;
    }
    char currChar = str.charAt(idx);
    if (map[currchar - 'a'] == true)
    {
        removeDup(str, idx+1, newString);
    }
    else
    {
        newString += currChar;
        map[currChar - 'a'] = true;
        removeDuplicates(str, idx+1, newString);
    }
}
//1st
```

(characters - characters) gives integer value.

```
main()
{
    string str = "abbccda";
    removeDup(str, 0, "");
}
```

$T_n = O(n)$

n = length of string

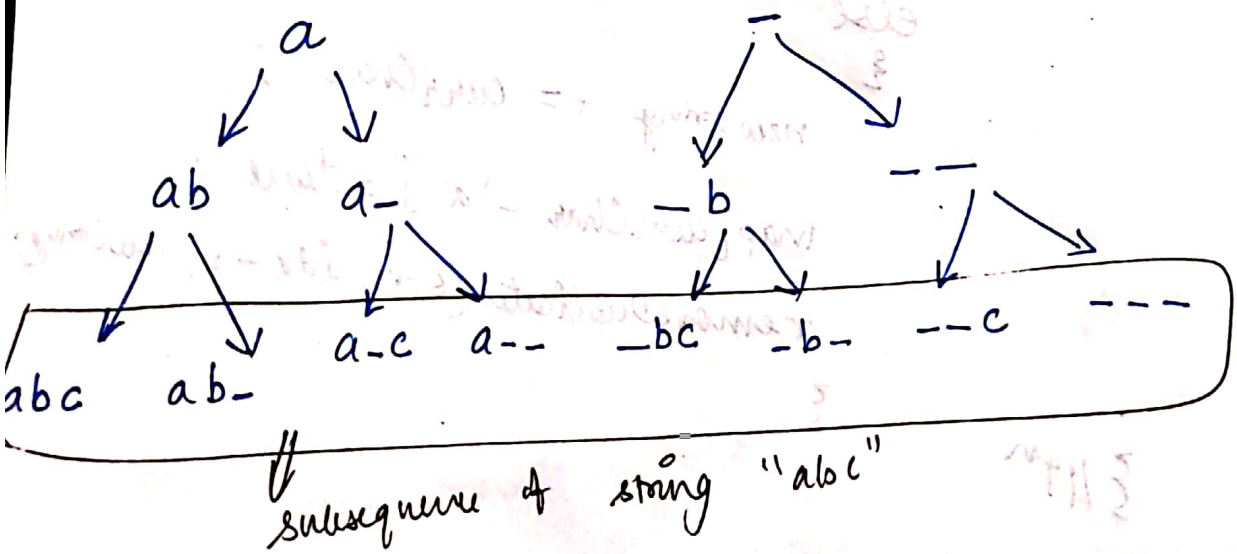
(QT) ## Important Ques:

* Print all the subsequences of a string

"abc"

* Subsequence :⇒

⇒ a sequence that can be derived from the given string by deleting zero or more elements without changing the order of the remaining elements



:- Subsequences we got :-

⇒ abc, ab, ac, a, bc, b, c, _ ⇒ In order wise

abc
ab
bc
ac
a
b
c
_

let, newString = "" (empty)

"abc"
idx

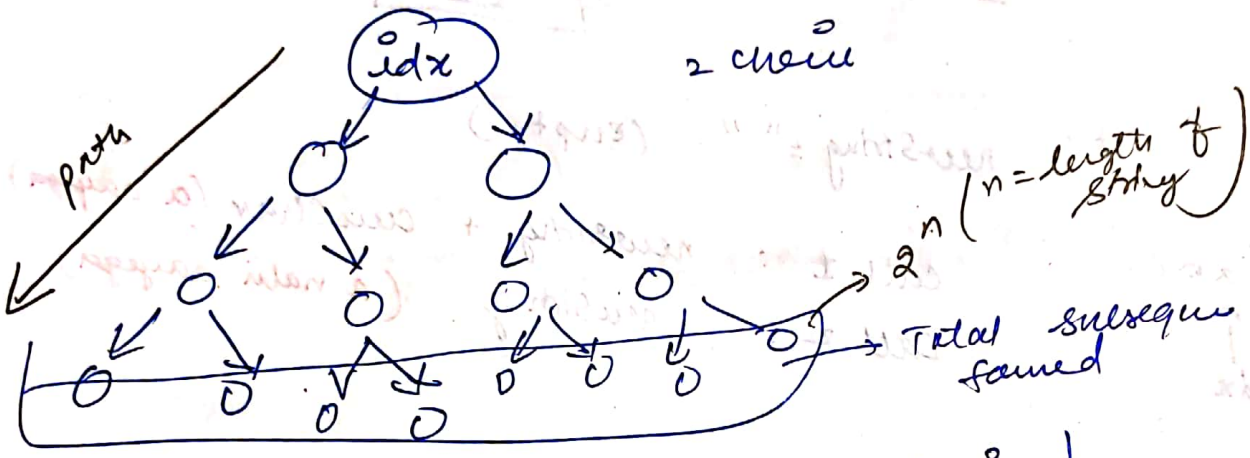
call 1 : newString + currChar (a aayega)
call 2 : newString (a nahi aayega)

Code

```
public static void subsequence(String str, int idx,
                               String newString)
{
    // Base case
    if (idx == str.length())
    {
        sout(newString);
        return;
    }

    Char currChar = str.charAt(idx);

    // to be (take it) → choose to be
    subsequence(str, idx+1, newString + currChar);

    // Not to be (not take it) → choose not to be
    subsequence(str, idx+1, newString);
}
```

```
Main()
{
    String str = "abc"
    subsequences (str, 0, "");
                          ↑ empty string
}
```

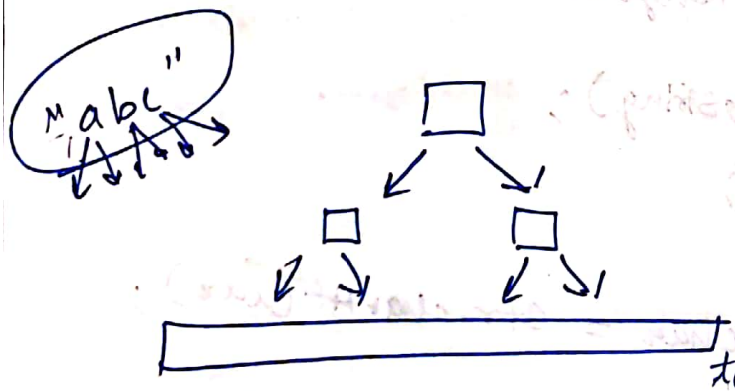
Time Complexity



(Har level par getting 2 choices)

"abc" → 3 character or 3 = length

∴ subseq no. = $2^3 = 8$



subs = 2^n

$= 2 \times 2 \times 2 = 2^3$

last level (node) = $2^n = 2 \times (2^{n-1})$

2nd last level (node) = $2^{n-1} = 2 \times (2^{n-2})$

$= 2 \times (2^{n-3})$

$= 2 \times (2^{n-4})$

⋮

⋮

1

} f^n call

Add the f^n calls

GP ⇒ Geometric progression

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

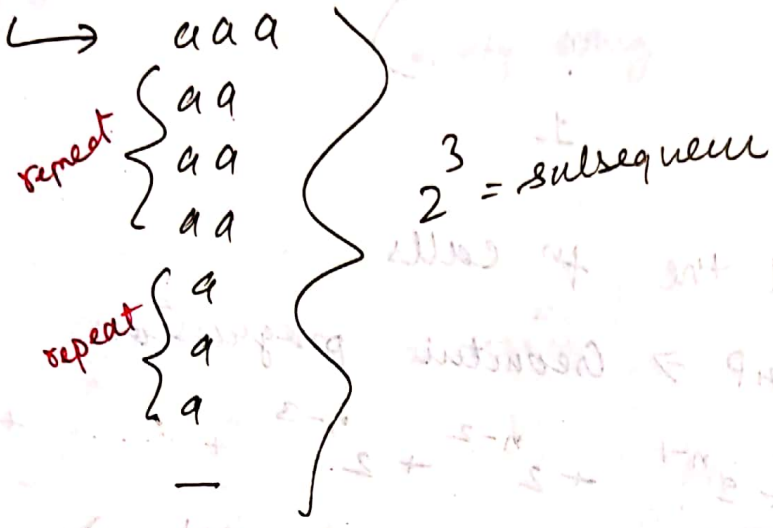
$= \frac{a(r^n - 1)}{(r-1)} = \frac{1(2^{n+1} - 1)}{(2-1)}$

$= (2^{n+1} - 1)$ → time complexity

$\therefore T_n = O(2^n)$ → in terms of asymptotic notation

(Q8) Print all the subsequence of a string unique.

* "aaa"



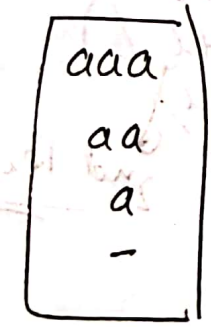
* Use: Special Data Structure ⇒ HashSet

Code

```
public static void subsequence(String str, int idx,
                               String newString, HashSet<String> set)
{
    //Base Case
    if(idx == str.length())
    {
        if(set.contains(newString))
        {
            return;
        }
        else
        {
            sout(newString);
            set.add(newString);
            return;
        }
    }
```

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

//to be
subsequences(str, idx+1, newString + currChar, set);

//not to be
subsequences(str, idx+1, newString, set);

}//fn

Main()
{
    String str = "aaa";
    HashSet<String> set = new HashSet<>();
    subsequences(str, 0, "", set);

}//main
```

Output :

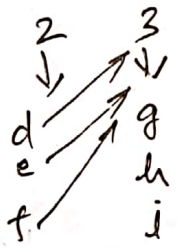


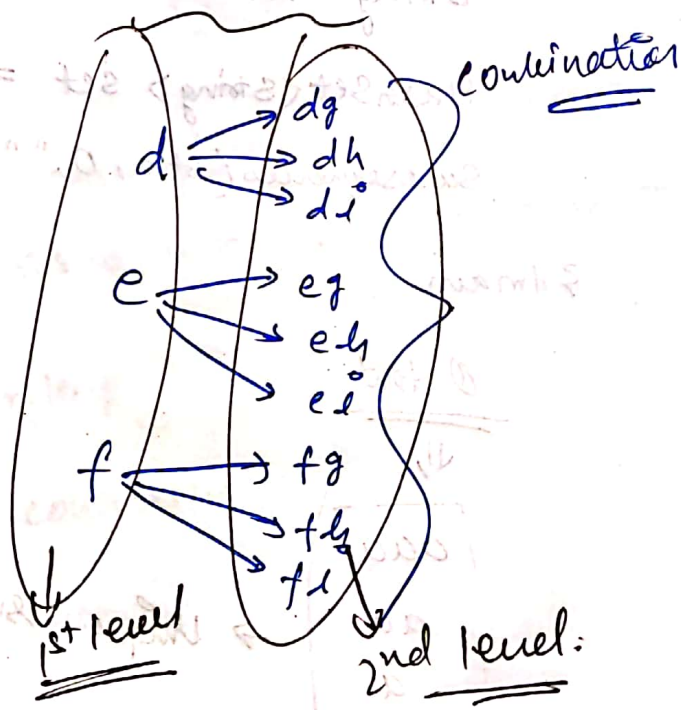
→ Unique subsequence

(Q9) Print Keypad Combination

0 → .
1 → abc
2 → def
3 → ghi
4 → jkl
5 → mno
6 → pqrs
7 → tu
8 → vwx
9 → yz

let say :
"23"
↓
find possible keypad combination of 23.






```
public static String[] Keypad
= {"", "abc", "def", "ghi", "jkl", "mno", "pqrs",
   "tu", "vwx", "yz"};
```

```
public static void printKey(String str, int idx,
                            String comb)
{
    //Base Case
    if (idx == str.length())
    {
        sout(comb);
        return;
    }
    char curChar = str.charAt(idx);
    String maping = keypad[curchar - '0'];
    for(i=0; i< maping.length(); i++)
    {
        printKey(str, idx+1, comb + maping.charAt(i));
    }
}//fn

main()
{
    String str = "23";
    printKey(str, 0, "");
}//main
```

comb → makes a new string

0 → zero

"" → empty string

$T_n = (4)^n$ → time complexity.

Lecture - 17

Recursion - 3

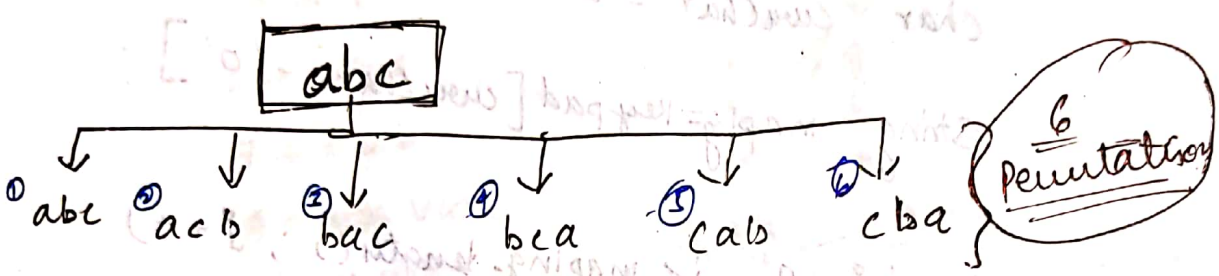
→ Adv. level

(Q1) Print all permutation of a string.

↓

all possible combination of letters

"abc"



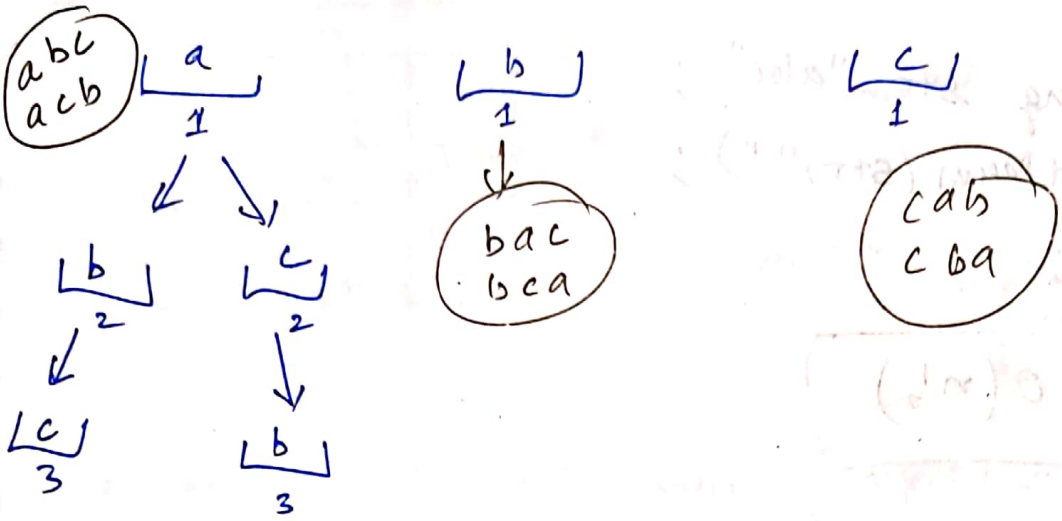
1	2	3
a	b	c
a	c	b
b	a	c
b	c	a
c	a	b
c	b	a

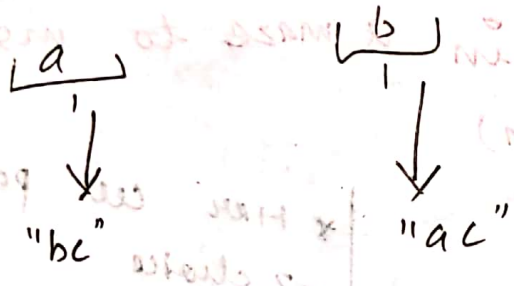
→ total No = 6

* $^3P_3 = 3! = 6$

$(^3P_3) = 6$

abc





Code

```
public static void printPerm (String str , String permutation)
{
    //base case
    if(str.length() == 0)
    {
        sout(permutation);
        return;
    }

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

        String newSt = str.substring(0,i) +
                       str.substring(i+1);

        printPerm(newSt , permutati + currChar);
    }
}
```

```
main()
{
    string str = "abc";
    printPerm(str, "");
} //main
```

$T_n = O(n!_0)$

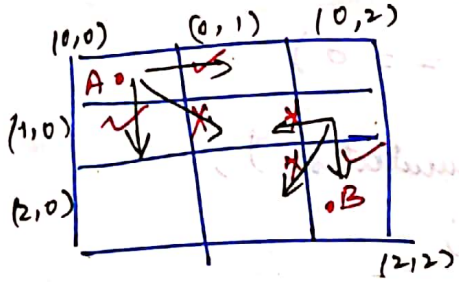
(Q2) Count total paths in a maze to move from (0,0) to (n, m)

$n = 3, m = 4$.

Eg:-

$n \times m$ → size (matrix)

i.e., $n = m = 3$



3X3

* Har cell par 2 choice except: corner cell : 1 choice (Boundry)

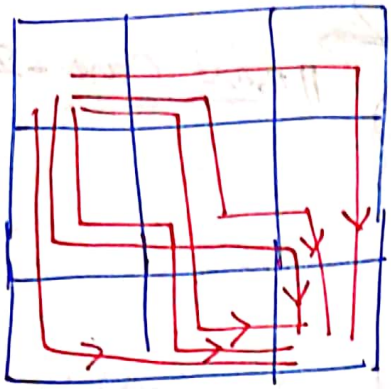
* go from A to B & travel ke total path batane hai

Condition:

(i) right move

(ii) diagonal & back not allowed

(i) downward move



* 6 path possible

total path = 6

moving allowed

* each cell → 2 choice :⇒ right / down

(i, j)

down → (i+1, j)

right → (i, j+1)

Base cond : (i = n-1) (j = m-1)

(i, j)

⇓

count(i+1, j) + count(i, j+1) } total path

Code

```
public static int countPath (int i, int j, int n, int m)
{
    //Base Case -1
    if( i==n || j==m)
    {
        return 0;
    }
```

```
if (i == n-1 && j == m-1)
{
    return 1;
}
```
// Base Case - 2

```
// move downward
int downPath = countPath(i+1, j, n, m);

// move right
int rightPath = countPath(i, j+1, n, m);

return downPath + rightPath;
}//fn

main()
{
    int n = 3, m = 3;
    int totalPath = countPath(0, 0, n, m);
    cout(totalPath);
}//main
```

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

(Q3) Place Tiles of Size $1 \times m$ ($1 \times m$) in a floor of Size : $n \times m$

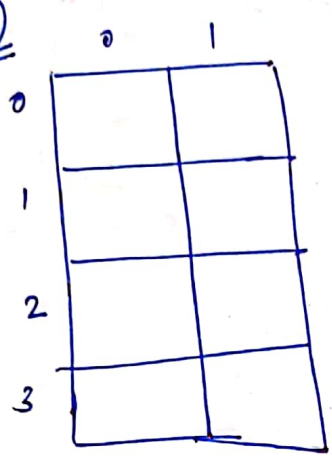
$n = 4$, $m = 2$

Size of Tile

(1×2)

Eg!-

floor



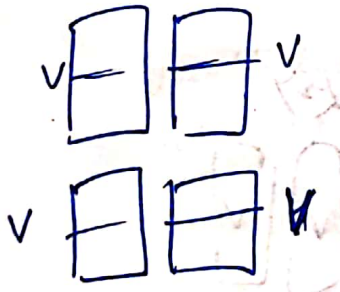
(4×2)

Sways

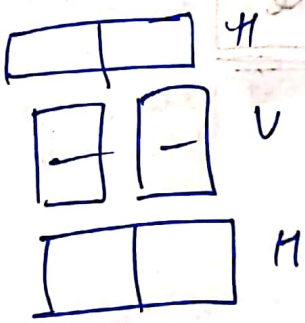
(i) place tiles horizontally

(ii)

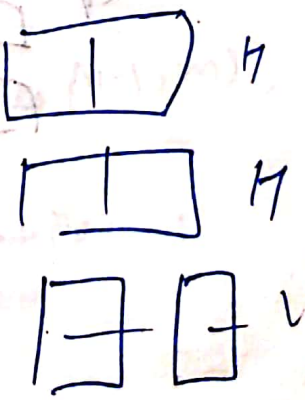
(iii)

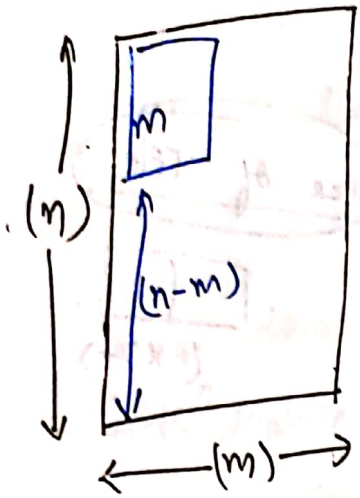


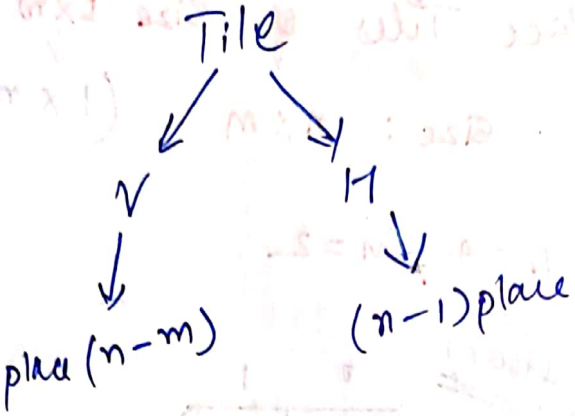
(iv)

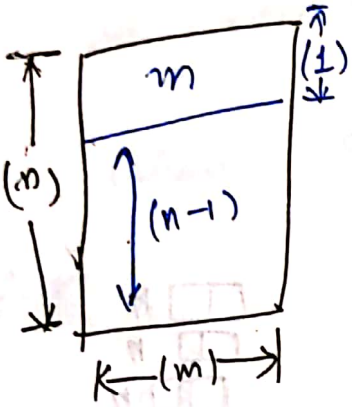


(v)





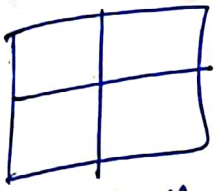


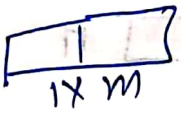


Base Case

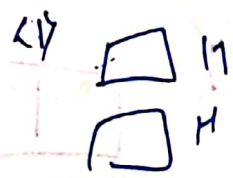
(i) $n = m$ = 2 way

$n = 2, m = 2$





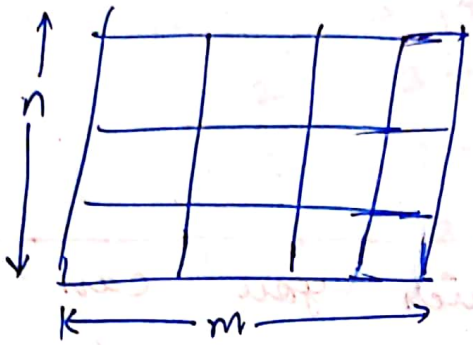
$m \times m$

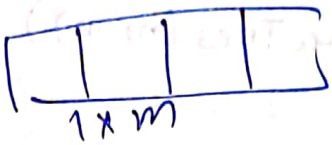


Code

(ii) $n < m$ ⟶ 1 way

$n = 3, m = 4$





only 1 way

⇩

* Horizontally place karo.

Code

```
public static int placeTiles(int n, int m)
{
    //Base case
    if (n == m) → either place vertical/horizontal
    {
        return 2; //only 2 way possible
    }
    if (n < m)
    {
        return 1; //only 1 way possible
    }

    //Vertically
    int vertPlace = placeTiles(n-m, m);

    //Horizontally
    int horPlace = placeTiles(n-1, m);

    return vertPlace + horPlace;

} //fn
```

```
main()
{
    int n = 3, m = 2;
    sout(placeTiles(n, m));
}//main
```

(Q4) find no of ways in which you can invite 'n' people to your party, single or in pairs.

n = 4

Eg:- when:

n = 1 ⇒ (1) way

n = 2 ⇒ single (1 2), pair (1-2) ⇒ (2) way

n = 3 ⇒
2 3 → (1)
1-2 3 → (2)
1-3 2 → (3)
1 2-3 → (4)
(4) way

n = 4 ⇒
1 2 3 4 →

1 ko akele

1 2 3 4
1 2-3 4
1 2 3-4

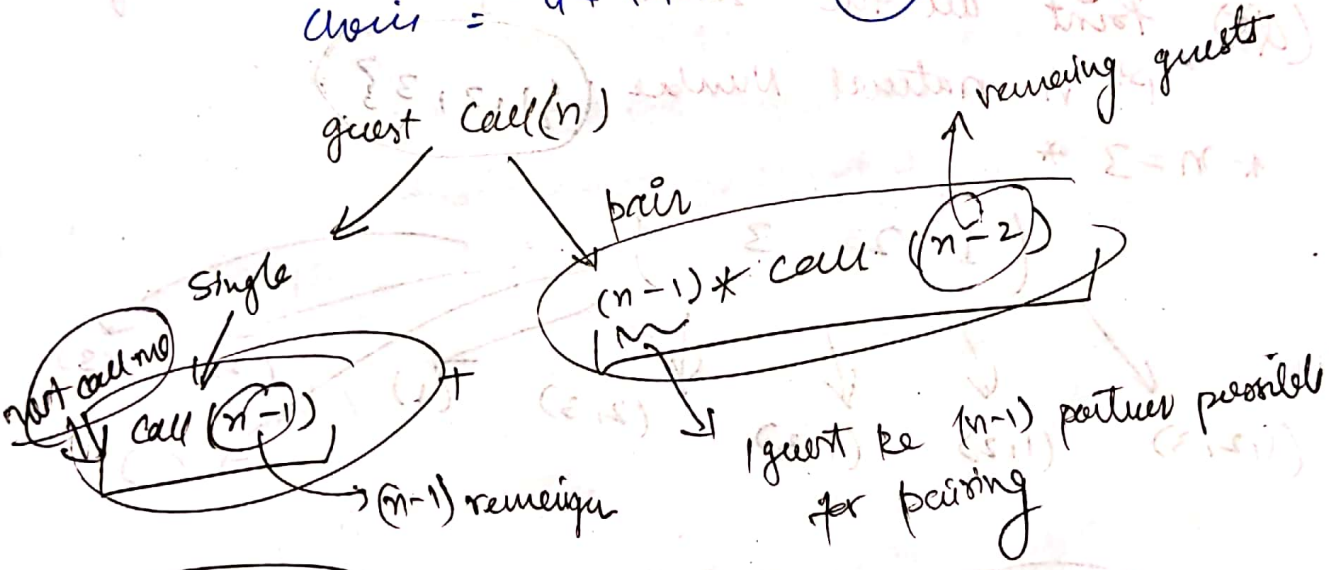
1 2-4 3 } 4 way

* 1 ko pair karo.

1-2 3 4
1-2 3-4
1-3 2-4 } 4 way
1-3 2 4

1-4 2 3 } 2 way
1-4 2-3

Choice = 4 + 4 + 2 = (10)



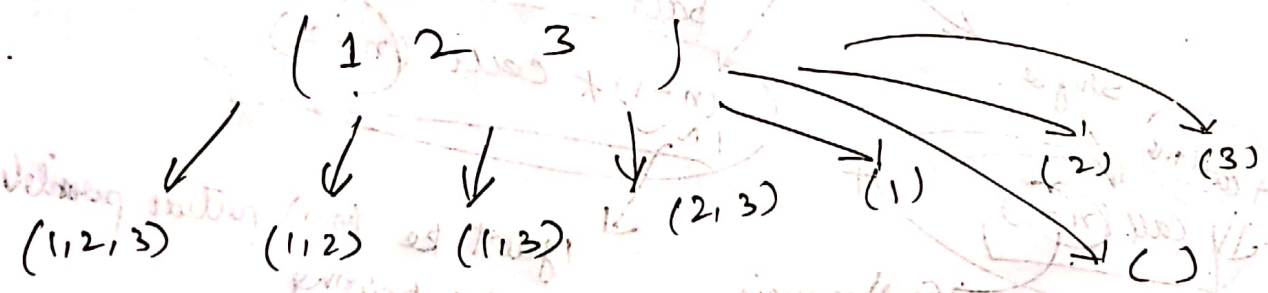
Coding

```
public static int callGuest(int n)
{
    //base case
    if (n <= 1)
    {
        return 1;
    }

    //single
    int ways = callGuest(n-1);
```

```
// pair
int ways2 = (n-1) * callCount(n-2);

return ways1 + ways2;

}//fn

main()
{
    int n = 4;
    sout(callCounts(n));

}//fn
```

(Q5) Print all the subset of a set of 1st n natural Number {1,2,3}

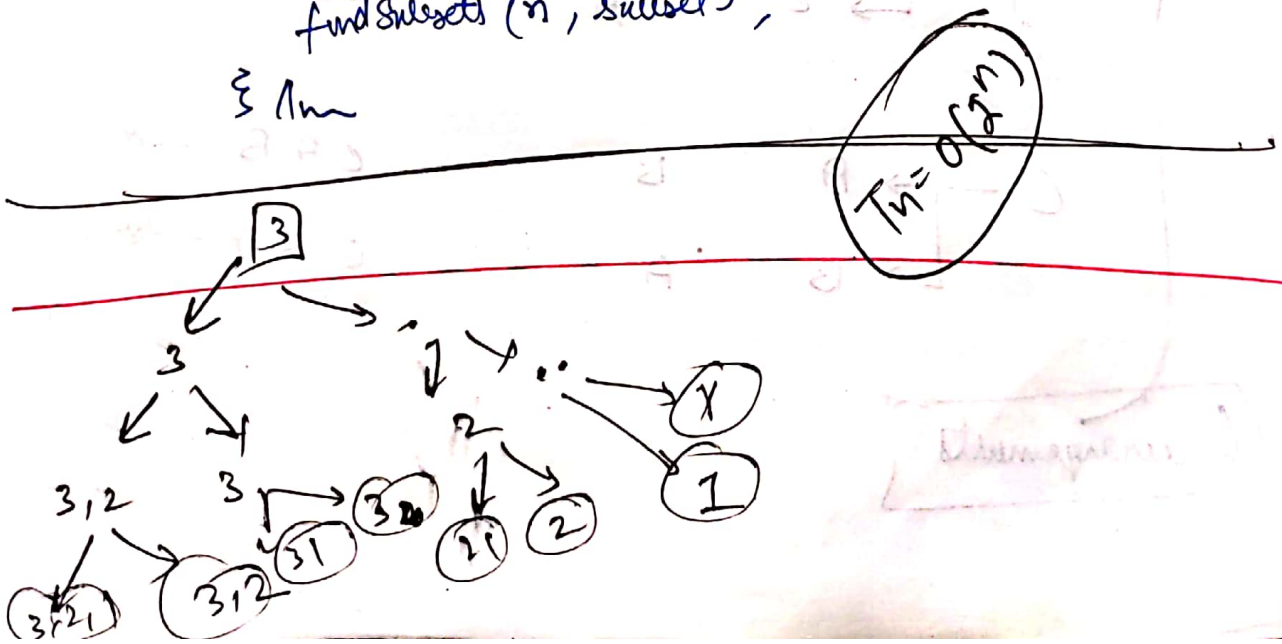
* n=3 *



Code

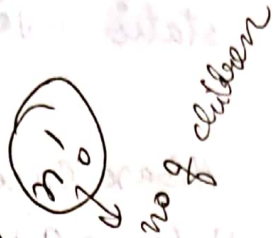
```
public static void printSubset(ArrayList<Integer> subset)
{
    for(i=0; i<subset.size(); i++)
    {
        sout(subset.get(i)+" ");
    }
    sout("\n");
}
```

```
public static void findSubset(int n,
                              ArrayList<Integer> subset)
{
    //Base case:
    if (n == 0)
    {
        printSubset(subset);
        return;
    }

    //add hoga
    subset.add(n);
    findSubsets(n-1, subset);

    // add nahi hoga
    subset.remove(subset.size() - 1);
    findSubsets(n-1, subset);

} //fn

main()
{
    int n = 3;
    ArrayList<Integer> subset = new ArrayList<>();
    findSubsets(n, subset);
} //fn
```

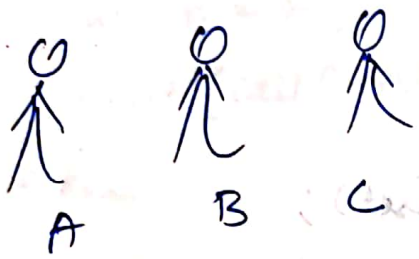


Lecture -18

Back-Tracking

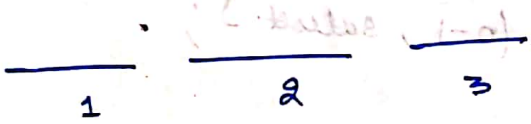


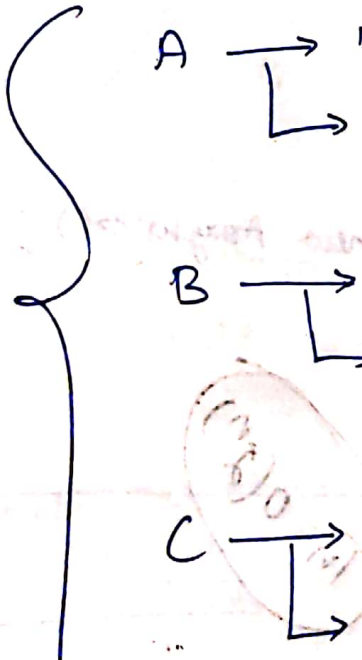
* Find all Possible solutions & use the one you want.



→ (permutation Problem)

* Arrange them in a single line *





ABC —(i)

ACB —(ii)

BAC —(iii)

BCA —(iv)

CAB —(v)

CBA —(vi)

6 arrangements

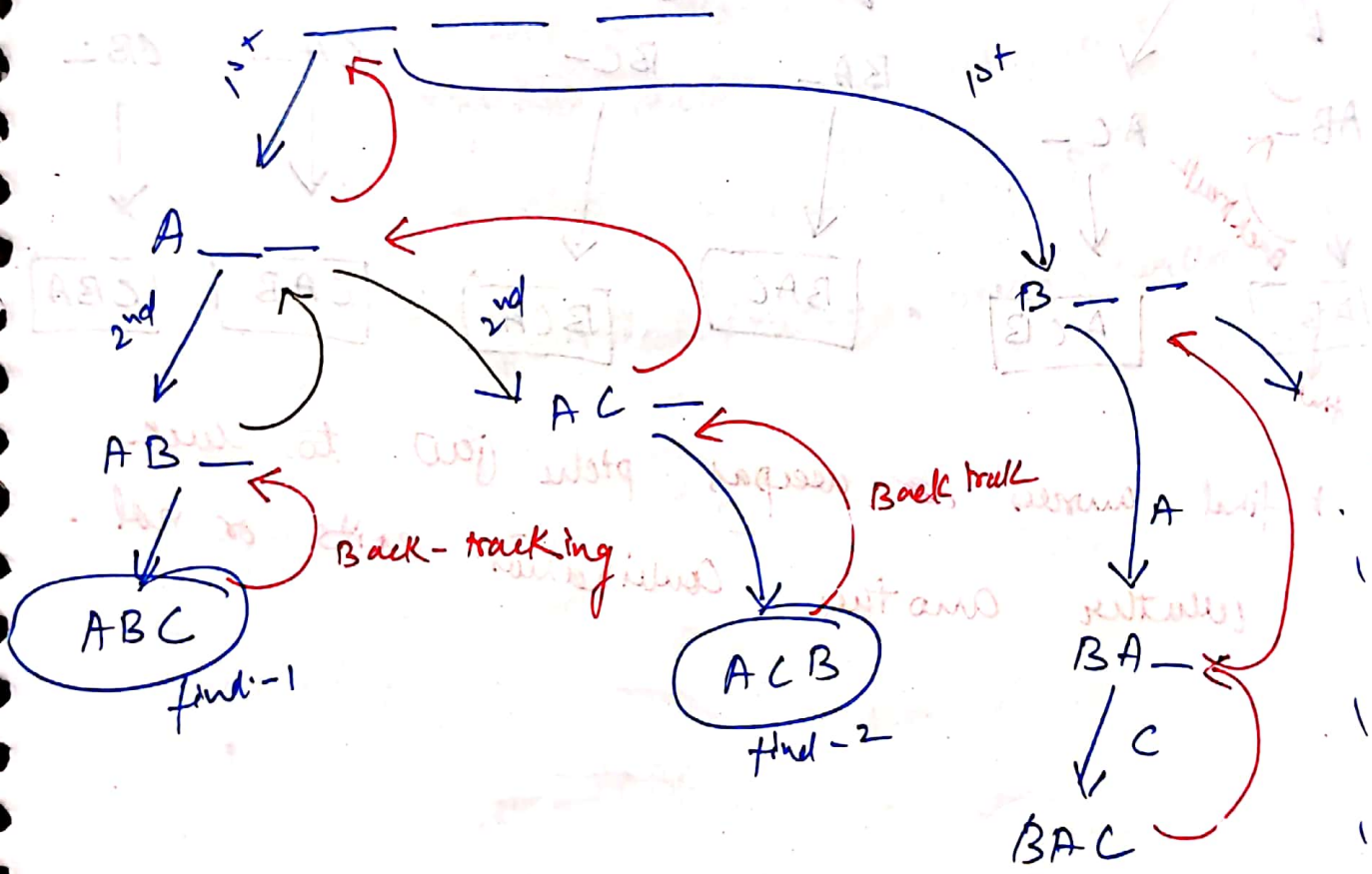
total permutation = n!

3 books → 3! = 6

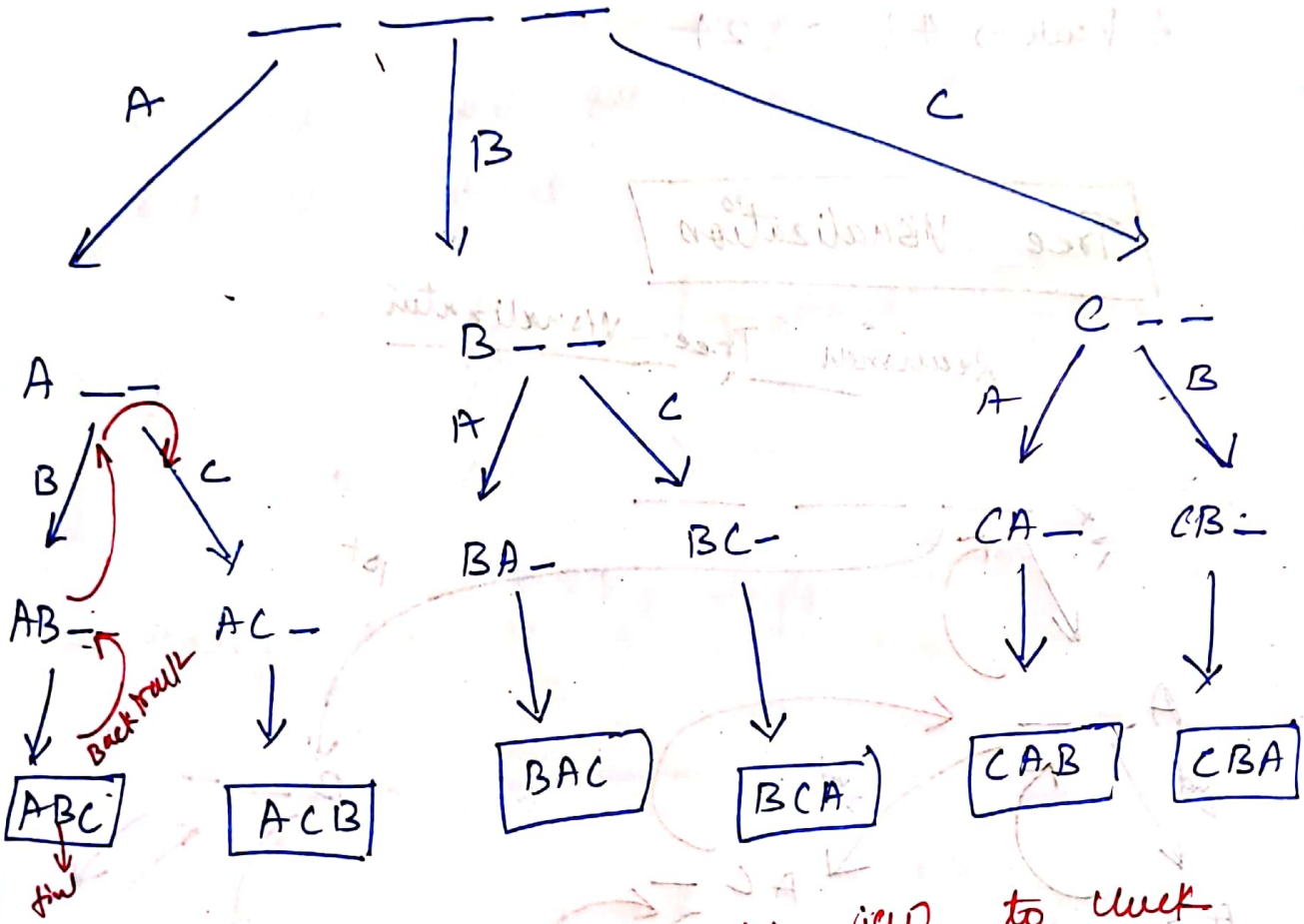
4 book → 4! = 24

Tree Visualization

Recursion Tree Visualization



Tree Visualization



* find answer se recursive piche jaao to check whether another combination exits or not.

Code

```
public static void printPerm(String str, String perm,
                             int idx)
{
    //Base - case
    if(str.length() == 0)
    {
        sout(perm);
        return;
    }

    for(i = 0; i < str.length(); i++)
    {
        char curChar = str.charAt(i);
        String newStr = str.substring(0, i)
                        +
                        str.substring(i+1);
        printPerm(newStr, perm + curChar, idx+1);
    }
}

//main
main()
{
    String str = "ABC";
    printPerm(str, "", 0);
} //main
```

Time Complexity

$T_n = O(n * n!)$

- n → steps
- $n!$ → total answer

N - Queens

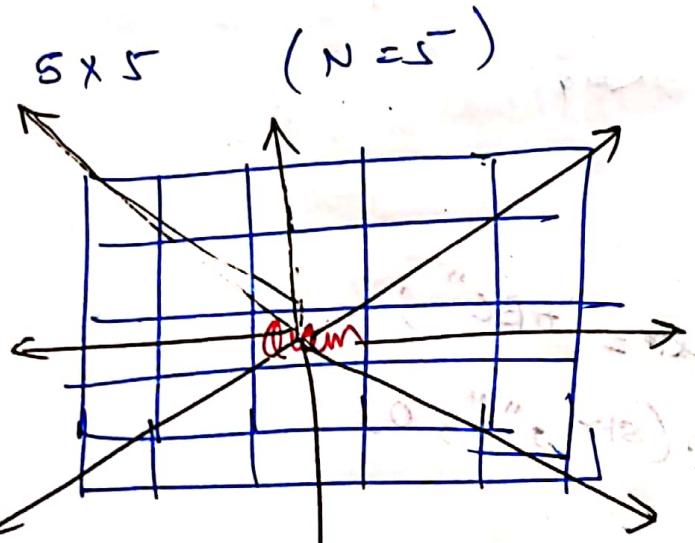
**

N * N : Chess Board

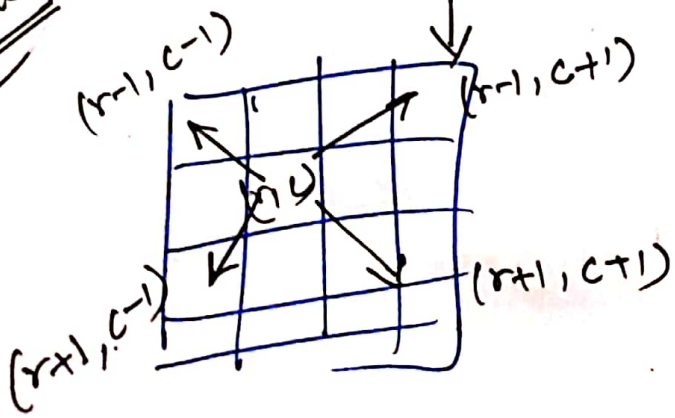
N : Queen

Print all sol^n where queens are safe

Eg:

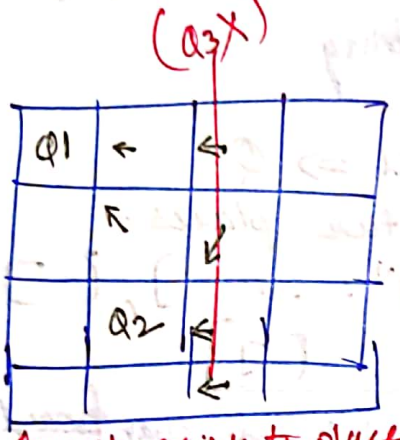


formula

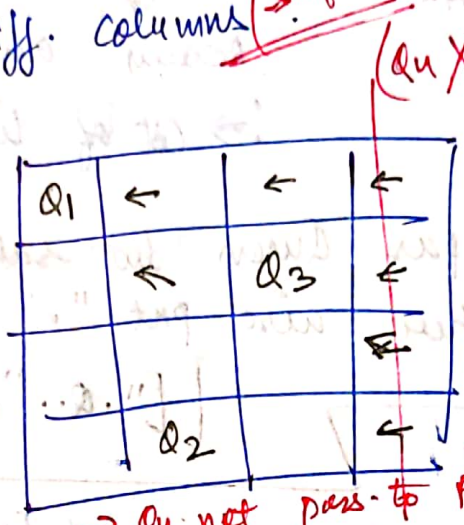


Q1, Q2, Q3, Q4 (Q = Queen)

Place them in diff. columns (safely)

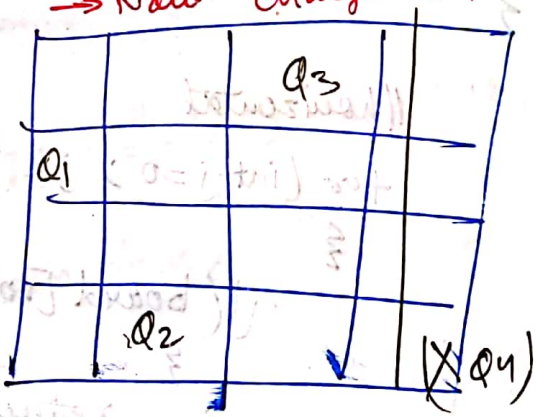


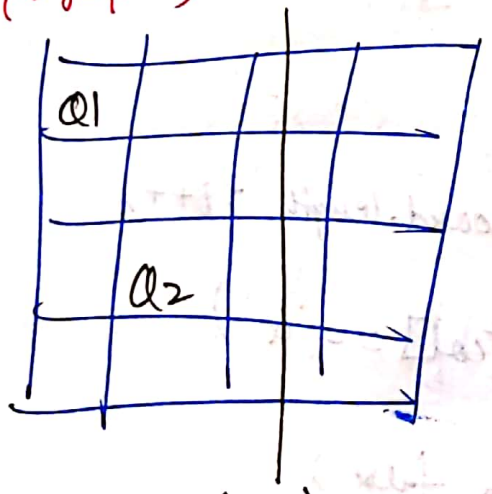
→ Q3 not possible to place
→ Back track & change pos of Q1 & Q2
→ Arrangement wrong

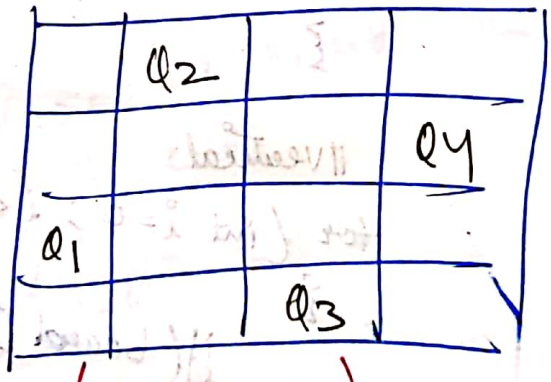


→ Q4 not pass. to place
→ Back track
→ Q2 will say I'm at last pos.
→ Now change Q1 position

Result → 1 (all Queen placed (safe pos.)







(Result → 2)

and so on & so forth

Code ⇒ N-Queen Puzzles

Return answer in the form of
:⇒ List of list of string

* jaha par Queen ko sakti hai ⇒ Q
other wise print "." in other places.

[["..Q.." "...." "Q" . . .], [ ]
. . [ ]]

Only Code

# public boolean isSafe(int row, int col, char board[][])
{

//horizontal
for (int j=0; j< board.length; j++)
{
if(board[row][j] == 'Q')
{
return false;
}
}

//vertical
for (int i=0; i< board.length; i++)
{
if(board[i][col] == 'Q')
{
return false;
}
}

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

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

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

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

return true ;

}//5th

# public void saveBoard (char [][] board ,
    List<List<String>> allBoards)

{
    String row = "" ;
    List<String> newBoard = new ArrayList<>() ;
    for (int i=0 ; i < board.length ; i++)
      {
          row = "" ;
          for (int j=0 ; j < board[0].length ; j++)
            {
               if(board [i][j] == 'Q')
                 {
                    row += 'Q' ;
                 }
```

N-Queen ... (continuation)

```
        else
        {
            row += ".";
        }
        newBoard.add(row);
    }
    allBoards.add(newBoard);
}//fn
```

* public void helper(char[][] board, int col,
  List<List<String>>)

```
{
    //Base case
    if(col == board.length)
    {
        SaveBoard(board, allBoard);
        return;
    }

    for(int row = 0; row < board.length;
                                row++)
    {
        if(isSafe(row, col, board))
        {
```

```
board[row][col] = 'Q';
helper(board, allBoard, col+1);
board[row][col] = '.';
}//for
}//fn
```

Lecture -21

Java Sudoku Solver

(#) A sudoku soln must satisfy all the following :-

(1) Each of digit 1-9 must occur exactly once in each row

(2) Each of the digit 1-9 must occurs exactly once in each column.

(3) Each of the digit 1-9 must occur exactly once in each of the 9 → (3x3) sub-boxes of the grid.

* The '.' character indicates empty cells.

* Sample Input :

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

9x9

board = [["5", "3", ".", ".", "5", ".", ".", ".", "."], ...]

Code

a) * Starting_Row = 3 * (row/3) (Row se kitne me hai)
   Starting_Column = 3 * (col/3)

b) * Starting Row = row - row % 3
   Starting Column = col - col % 3

isSafe
1. row
2. column
3. grid

Code

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

```
//row
for (int j=0 ; j< board.length ; j++)
{
    if (board[row][j] == (char)(number + '0'))
    {
        return false;
    }
}
```

```
//grid
for (int j=0 ; j< board.length ; j++)
{
    if (board[row][j] == (char)(number + '0'))
    {
        return false;
    }
}
```

```
//grid
int sr = 3 * (row/3) ; // starting row
int sc = 3 * (col/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;
        }
    }
}
```

return true ;

} //fn

* public boolean helper (char board[][], int row,
int col)

{

if (row == board.length)

{

return true ;

}

int nrow = 0 ;

int ncol = 0 ;

if (col == board.length - 1)

{

nrow = row + 1 ;

ncol = 0 ;

}

else

{

nrow = row ;

ncol = col + 1 ;

}

if (board [row] [col] != '.')

{

if (helper (board , nrow , ncol)

{

return true ;

} }

```
else
{
    //fill the places (empty)
    for(int i=1 ; i<=9 ; i++)
    {
        if(isSafe(board, row, col, i))
        {
            board[row][col] = (char)(i+'0');
            if(helper(board, nrow, ncol))
            {
                return true;
            }
            else
            {
                board[row][col] = '.';
            }
        }//if
    }//for
    return false;
}//th

main()
{
    helper(board, 0, 0);
}
```