→ Mnemonics – combination of high & low
ex of middle level – Assembly language
C – high & middle level because of Pointers Concept it is also a middle level (deals with address)
To interact with Computer we need language

→ low – binary
middle – C
high – python, C, Java

Compiler – execution fast (not given directly where the error occur
entire Program compile at once
Interpreter – given directly where the error will occur

→ (1) Documentation (comments)
(3) Import
(2) Package
(4) class
5) main

Syntax

Semantics :- meaning of the symbol (or) words

Data types :- specific type of data

Paradigm :- style of writing the Program
(or) way of writing the Program

Paradigm 2 types : (1) Imperative
(2) Declarative

(1) Imperative - What to do

all Java supports

- Procedural POPs
- Functional FOPs
- Object oriented OOPs

S-system

(2) Declarative - How to do

Optimise the Imperative Code

(1) logical

(2) Functional Programming

3) Database Programming < relational / Non-relational

higher order functions

Takes input as a function and also o/p as a function.

It is use in big data Technologies

ex: $f(g(h(x)))$

$y = h(x)$

$z = g(y)$

$f(z)$

→ lamda

(1) interfaces in Versions

(2) abstract

Database

- relations (Tables) ← Non-relational

Java is not fully object oriented because of 2 reasons

(1) Impelative - What to do

all Java suppalts { Procedural POPs, Functional FOPs, Object oriented OOPs } — s-system

(2) Declarative - How to do

optimise the Impelative Code

(1) logical

(2) Functional Programming

3) Database Programming < relational, Non-relational

higher order functions

Takes input as a function and also o/p as a function.

It is use in big data Technologies

ex: $f(g(h(x)))$

$y = h(x)$

$z = g(y)$

$f(z)$

→ lamda

(1) interfaces in Versions

(2) abstract

Database

rdbms (Tables) — Non-relational

Java is not fully object oriented because of 2 reasons

(1) we can't create objs for Primitive data types

(2) when using static keyword the memory is already reselved.
(no obj creation)

Static block: for entire block for a Valiable
it keeps memory

At compile time memory reselvation
we don't need object.)

we don't need an object to access.

we have wrapper class for creating objects for int, float, char

int → Integer() class
float → Float()
char → Character()

Create object by using wrapper class.

Integer il = new Integer(20);

→ Control statements

do while – atleast once execute
while – check condition first
for each – traverse theough directly with elements

for loop → by using index

Break, Continue → Jump statements

Functions

same task Performed multiple times with different inputs.

Control Statements — Control the flow of execution

Access modifiers

Public — access everywhere

Private — Same class

Protected — same directory

Default — Same Package

Private — not specified

Data Types

Primitive

int
float
char
long

Non Primitive

Derived
String
Array
Function

User Defined
User can create their own D.T
Class
Objects
enum
in c structures & Union also example

Basic building blocks of Programming language

(1) Source Code
(2) Algorith
(3) Flow chart

Wrapper classes in Java

A wrapper class in Java is a class whose object wraps or contains primitive data types. We can wrap a primitive value into a wrapper class object.

Need

(1) They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method.

(2) The classes in java.util package handles only objects and hence, wrapper classes help in this case also.

(3) DS in the collection framework, such as ArrayList & Vector store only objects.

→ wrapper classes allow primitive data types to be used as objects.

Primitive Data Type	Wrapper class
char	Character
byte	Byte
short	Short
int	Integer
long	Long

float	Float
double	Double
boolean	Boolean

1. Autoboxing:

The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For ex - Conversion of int to Integer, long to Long, double to Double, etc.

2. Unboxing

It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive types is known as Unboxing. For example:- Conversion of Integer to int, Long to long, Double to double, etc.

Predefined methods

toString(): Value of given integer type argument

ValueOf(): Value of given Primitive type

equals(): checks

CompareTo(); Compares

intValue(): return wrapped object's value as integer

double Value(): Returns the wrapper object's value

equals(): Determines if the supplied object

ValueOf method:

To create a wrapper object for a given primitive or String, There are 3 types of valueOf() methods:

A) Wrapper valueOf(String s): Every wrapper class except Character class contains a static valueOf() method to create wrapper class object for a given String

B) Wrapper valueOf(String s, int radix): Every Integral wrapper class Byte, Short, Integer, Long contains the valueOf() method to create a wrapper object for a given String with specified radix. The range of radix is 2 to 36.

C) Wrapper valueOf(primitive p): Every wrapper class including the Character class contains the following method to create a wrapper object for given primitive type

toString() Method

We can use the toString() method to convert the wrapper object or primitive to String. few forms of toString() method

(A) Public String toString(): Every wrapper class contains the following toString() method to convert wrapped object to String type

B. toString (Primitive P): Every wrapper class including the character class contains the following static toString() method to convert Primitive to string

C. toString (Primitive P, int radix);
Integer and Long classes contain the following toString() method to convert Primitive to specified radix String

Task-2

List of all types of Data Types & mention size of memory To store.

Data Types	Memory	Range
Boolean	1 bit	True or false
Byte	8 bit – 1 byte	-128 to 127
Char	16 bits – 2 bytes	
Short	16 bits – 2 bytes	-32,768 to 32,767
int	32 bits – 4 bytes	-2,147,483,648 to 2,147,483,64
long	64 bits – 8 bytes	-9223,372,036,854,775 808 to 9,223,372036 775,807
float	32 bits – 4 bytes	6 – 7 decimal digit
double	64 bits – 8 bytes	15 to 16

Task-3

Access Modified	Within Class	Within Package	Outside Package	outside sub class
Public	✓	✓	✓	✓
Private	✓	×	×	×
Protected	✓	✓	×	✓
Default	✓	✓	×	×

Task-4

Types of Recursions

Recursion:

The Process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called recursive function.

Two types of recursions

(1) Direct Recursion

(2) Indirect Recursion.

Direct Recursion:

These can be 4 types

Tail Recursion: If a recursive function calling itself and that recursive call is the last statement in the function. After that call the recursive function performs nothing.

Time complexity : O(n) may vary for other
Space complexity : O(n)

Head Recursion: If a recursive function calling itself and that recursive call is the first statement in the function. There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

Time Complexity : O(n)
Space Complexity : O(n)

Tree Recursion:

If a recursive function calling itself for one time then it's known as Linear Recursion. Otherwise if a recursive function calling itself for more than one time then it's known as Tree Recursion.

Time Complexity : $O(2^n)$
Space Complexity : O(n)

Nested Recursion: A recursive function will pass the parameter as a recursive call. That means "recursion inside recursion".

→ Q11 favourite Nomber (or) not
if it is devisible by 3 it's My fav
if it divisible by 5 it's your fav
if it both divisible by 3 & 5 its our fav
if it neither divisible by 3 & 5 its Not fav

```
n=9
if (n%3==0)                    if(n%3==0 && n%5==0)
    Ptintf("My favorite");          ↓ our fav
else if (n%5==0)&& n%3!=0          else My fav
    Printf("Your favarite");       else no fav
else if (n%3==0 && n%5==0)
    Ptintf("our favarite");
else
    Print("Not favarite");
```

we should not give that if Condition first Because it doesn't goes to next condition if it was true so, also we can write nested if Statement (blaklink)

```
n=9
if (n%3==0 && n%5==0) {
    printf ("our favolate");
}
else if (n%3==0) {
    printf ("my favorite");
}
else if (n%5==0) {
    printf ("your favorate");
}
else {
    printf ("not a favorate");
}
return 0;
}
```

also
if (n%3==0)
if (n%5==0)

our fav
Nested if (not &&) used.

→ 123/10 = 12
2145/10 = 214
245/100 = 2
4578/100 = 45

* if we want to remove n digits from last then divide with 10^n

% → remainder
/ → Quotient

* if 2%3 = means 3)2(can't the answer will be 2 if dividend is < divisor it gives dividend.

→

132 % 10 = 2

132 % 100 = 32

78647 % 100 = 47

78647 % 1000 = 647

* if we want to retrieve n digits from last then do % by 10^n

Q11 your given with 3 distinct numbers find which is greatest.

```
int a=15;
int b=42;
int c=9;
if (a>b && a>c) {
    printf ("a is big");
}
else if (b>c) {
    printf ("b is big");
} else {
    printf ("c is big");
}
return 0;
}
```

(or)

```
printf ("%d", a>b && a>c ? a : b>c ? b : c);
```

in java

Math.max(a,b,c);

Q11 Reverse of a number :-

```
int a=676745;
int r=0;
while(a!=0){
int d=a%10; (retrieve last digit)
r=r*10+d;
a = a/10; (eliminate last digits)
}
printf("%d",r);
return 0;
}
```

we don't know exact no. of iterations but know where to stop so use while loop.

a>0

↓ retrieved digit.

ans 547676

→ while for -ve if we give o/p -67674

-4767 6.

→ we don't don't know the range then Integer.MIN_VALUE (It gives very low range(last) integer value)

Integer.MAX_VALUE (It gives higest values)

→ Leetcode 7 "Make changes in below code"

r should be in range $[-2^{31}, 2^{31}-1]$ then

at last long r=0;

```
if (r < Integer.MIN_VALUE ||
        r > Integer.MAX_VALUE)
    {
        return 0;
    }
    else {
        return (int) r;
    }
}
```

→ Printing Positive (or) Negative (or) Zero, with out using if else statements.

Use switch statement.

```
num = 15;
switch (Integer.compare(num, 0)) {
    case 0 : SOP ("Zero"); break;
    case 1 : SOP ("Positive");
             break;
    case (-1): SOP ("Negative");
               break;
    Default : SOP ("Unexpected");
```

Compares here & give 0 (or) -1 (or) 1 & go to that case & executed

O/P Positive

→ Printing Week Days :- (To find weekday & weekend);

```
String day = "this";
 Switch (day) {
 Case "mon";
 System.out.Println("weekday");
   break;
 Case "tue" :
       S.O.P ("weekday"); break;
  case "wed" :
       S.o.P ("weekday");
   ;
  Case "sun":
       S.O.P("Week End !!!");
       break;
   Default :
       S.o.P("Not Expected");
 }
}
```

O/P : Weekday

→ Switch Expression (@) Arrow expression (→)

To reduce code, they introduced this concept Switch Expression. *(New update in java)

```
day = "Sun";
String result = switch (day){
 Case "mon" → "weekday";
 Case "tue" → "Tuesday";
 !
 Case "sun" → "WeekEnd!!"
```

To Print result ans stored

default → "unexpected";

} ; important (Because this is expresion)

S.O.P (result); → Printing result.

}

}

O/P Week End!!

This is introduced from 14th Version Onwards

java -version → 11 (in cmd)

→ To Reduce Above Code ------

P.S.V(S.arg[]) {

day = "sun"

String res = switch (day) {

case "mon", "tue", "wed", "thu", "fri" → "weekday";

case "sat", "sun" → "weekEnd";

default → "Unexpected";

};

S.O.P ("res");

}

H.w In Hackerrank Complete the course "Java Basic".

* Data structures and Algorithms *

Data - Collection of raw facts.

information: Process data, adding meaning to data.

Data types < Primitive
Non-primitive

Non-primitive → Lineal, Non Lineal

Lineal → Static, Dynamic

→ characteristic of algorithm
- well-defined inputs
- well-defined outputs
- clear & Unambiguous
- Finite-ness
- Feasible
- Language Independent

How to Analyze an algorithm
- Time
- Space
- Network Capability
- Power Consumption
- CPU Registers

→ different types of time complexity
1) Constant — O(1)
2) Logarithmic — O(log(n))

Linear – O(n) no. of steps increase w.r.t to input side

Log-Linear – O(n log(n))

Quadratic – O(n²) Polynomial – O(nᵏ)

Exponential – O(2ⁿ)

worse: O(n!)✓

→ Best case.
→ Average case.
→ Worst case.

→
```
for (i=0 ; i<n; i++){
        statement;
    } ---;
```
n steps

n = 10

i=0 stv✓
i=1 stv✓
⋮
i=10 st✗

So Time Complexity O(n)

→
```
for (i=0; i<n; i++) {  ⟶ n
    for (j=0; j<n; j++) {  ⟶ n×n
        ---- stm;
    } }
```
Time Complexity O(n²)

→
```
for (i=0; i<n; i++) {
    for (j=0; j<i; j++){
        Statement;
    } --;
}
```

i	j	statement
0	0	✗ 0<0
1	0	0<1✓
1	1	1<1✗
2	0	✓
2	1	2✓
2	2	✗

$i \rightarrow 2n+2$

$j \rightarrow \frac{n(n+1)}{2}(n)$ (in for loop) so $\times n$

$Stm \rightarrow \frac{n(n+1)}{2}$ ✗

$2n+2 + \frac{n^3+n^2}{2} + \frac{n^2+n}{2}$

Time complexity $O(n^3)$ //

i	j	#of times
0	0	0
	0 ✓	1
1	1 ✗	
	0 ✓	2
2	1 ✓	
	2 ✗	
	0 ✓	3
3	1 ✓	
	2 ✓	
	3 ✗	
k		k

Condition fails when $i=n$

$(i \approx k^2 = n)$

$K = \sqrt{n}$

$0+1+2+3 \cdots +K$

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

$= \frac{K^2}{2} + \frac{K}{2}$ Neglete

Time Complexity $O(\sqrt{n})$

⇒ P=0

```
for (i=1 ; p<=n ; i++) {
        P=P+i ;
    }
```

i	P	p<=n
1	0+1	0<=n
2	0+1+2	1<=(n)
3	0+1+2+3	3<=n
1	!	!
1	!	!
	0+1+2+---K	

$$P = \frac{K(K+1)}{2}$$

Condition fails when $P > n$

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

$$\frac{K^2+K}{2} > n$$

$$K^2 > n$$

$$K > \sqrt{n}$$

Time Complexity $O(\sqrt{n})$

→

```
for (i=1 ; i<n ; i=i*2) {
        Stmt ;
    }
```

i	i<n	i = i * 2
2^0 → 1	1<n	i = 1*2
2^1 → 2	2<n	i = 2*2
2^2 → 4	4<n	i = 4*2
2^3 → 8	8<n	i = 8*2
⋮		
2^k		

$i = 2^k$

The condition fails when

$i >= n$

$2^k >= n$ (applying log on b.s)

no. of steps taken $\leftarrow k >= \log_2 n$

Time complexity was $O(\log n)$

$\frac{\log n}{\log 2} = \log_2 n$

→ for (i=n ; i>=1 ; i = i/2) {

stat---

}

i = n	i >= 1	i = i/2
i = n	n >= 1	i = n/2
i = n/2	n/2 >= 1	i = n/4
i = n/4	n/4 >= 1	i = n/8
⋮		⋮
$i = n/2^k$		$i = \frac{n}{2^k}$

Condition fails when

$i < 1$

$\frac{n}{2^k} < 1$

$n < 2^k$

$\log n < \log 2^k$

$\boxed{k = \log_2 n}$

Time Complexity = $O(\log_2 n)$

→ for (i=0; i*i<n; i++){

stalc--

--i;

}

i

i>=n

k²>=n

$\boxed{k = \sqrt{n}}$

i=0	i*i<n	i++
0	0*0<n	1
1	1*1<n	2
2	2*2<n	3
3	3*3<n	4
⋮	⋮	
k	k^2	

Time Complexity $O(\sqrt{n})$

→ for (i=0; i<n; i++){ } O(n)

--st;

}

+

for (j=0; j<n; j++){ } O(n)

stm;

--j

} T.C = O(n)

→* if O(n) + O(n²) means max (O(n), O(n²))

Time Complexity was O(n²)

→ for(i=0; i<m; i=i+1){ ⟶ m

for(j=1; j<n; j=j*2){
stmt;
--i;
} } } $\log_2 n$ (m)

Time Complexity
$O(m \log n)$

i	i	j<n	j=j*2
0	1	1<n	j=1*2=2
1	2	2<n	j=2*2=4
2	3	4<n	j=4*2=8
3	4	8<n	j=8*2=16
4	5	16<n	j=16*2=32
5	6	32<n	j=32*2=64
6	7	64<n	j=64*2=128
7	8	128<n	j=128*2=256
8	9	256<n	j=256*2=512
9	10	512<n	j=512*2=1024
10	11	1024<n	j=1024*2=2048

$= 2m + 2 + \log_2 n (m)$

$= (m \log_2 n)$

→ P=0;
for(i=1; i<n; i=i*2){ } $\log_2 n$
P++;
}
for(j=1; j<P; j=j*2){ } $\log_2 P$
stmt;
}
--j;

(P)value is $\log_2 n$

i	i<n	P++	i=i*2
1	1<n	1	1=1*2
2	2<n	2	i=2*2
4	4<n	3	i=3*2
⋮	⋮	⋮	⋮
$i=2^k$	$2^k<n$	k	

$i >= n$ Condition fails

$2^k >= n$ for P, P = k

$\boxed{k = \log_2 n} \Rightarrow P = \frac{\log_2 n}{k}$ P, K times Came

for 2nd loop $\rightarrow \log_2 P \Rightarrow \log(\log_2 n)$ → same as 1st for loop

T.C = $\log(\log_2 n)$

$f(n) = \log_2 n + \log(\log_2 n)$

$f(n) = \log(\log_2 n)$

→ function fun(N) { Recursive function

if (N==0) {

Print ("Hello Vignan !!");

return;

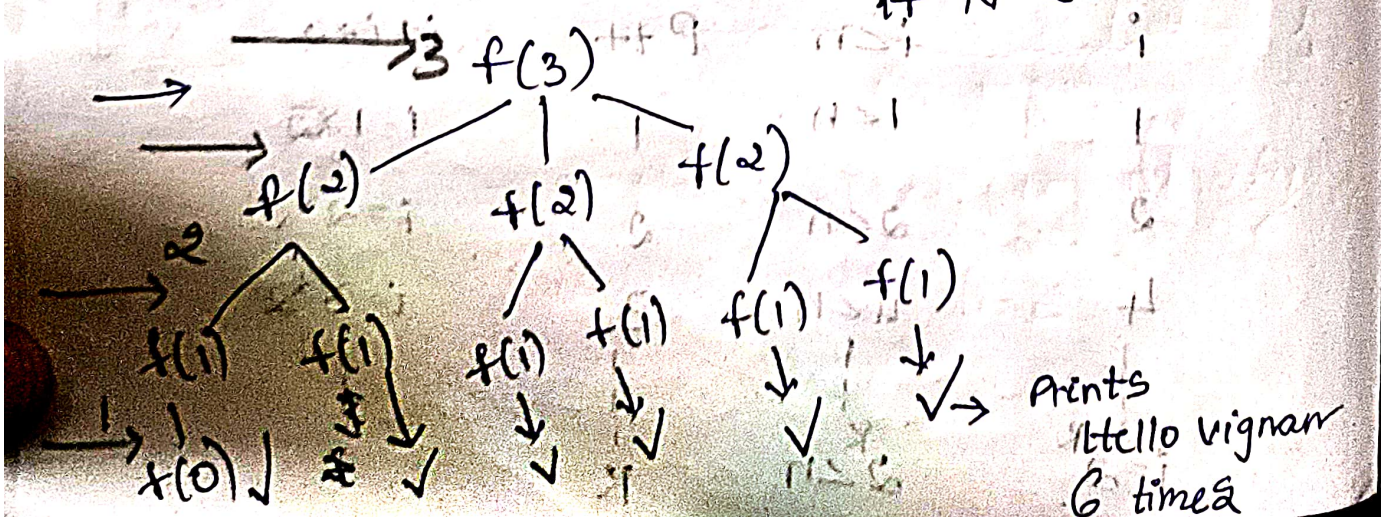
}

for (i=0; i<N; i++) {

fun(N-1);

} }

fun(3)



- $f(3) = 3 \times 2 \times 1$

when Recursive function given then represent as Recursive Tree.

if if condition it checks N==0 if not it goes to for loop, N-1, (3-1) ⇒ 2, f(2) calls 3 times, Next, if Condition goes N==0 (2==0) X so it goes to for loop (N-1)(2-1) = 1, 1 calls 2 times.

→ for (i=0; i<n; i++) → O(n)

for (i=0; i<n; i=i+2) → O(n)

for (i=n; i>1; i--) → O(n)

→ GCD algo

```
while (m!=n) {
    if(m>n) {
        m=m-n;
    }
    else {
        n=n-m;
    }
}
```

Time Complexity $O(m/n)$

m=16	n=2	16-2 = 14	$\frac{16}{2} = 8$
m!=n	m>n	14-2 = 12	
16!=n	16>n	12-2 = 10	if m=16, n=4
14!=n	14>n	10-2-8	$\frac{16}{4} = 4$
12!=n	12>n	8-2-6	(12, 8, 4)
10!=n	10>n	6-2-4	
8!=n	8>n	4-2-2	
6!=n	6>n		
4!=n	4>n		
2!=n	2>n X		

X → $\frac{m}{n}$ times the loop executes

Linear Recursion

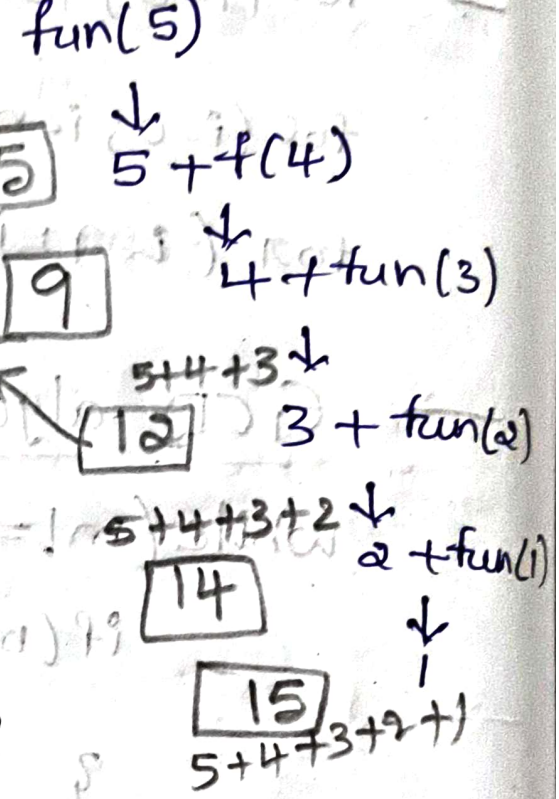
```
fun(n) {
  if (n==1)
    return n;
  return n+fun(n-1);
}
fun(5);
```

Linear Recursion

num 5

5 recursive calls

Stored in another memory location



This is linear space complexity

Space Complexity O(n)

Quadratic Space Complexity type

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}_{2\times4}$$

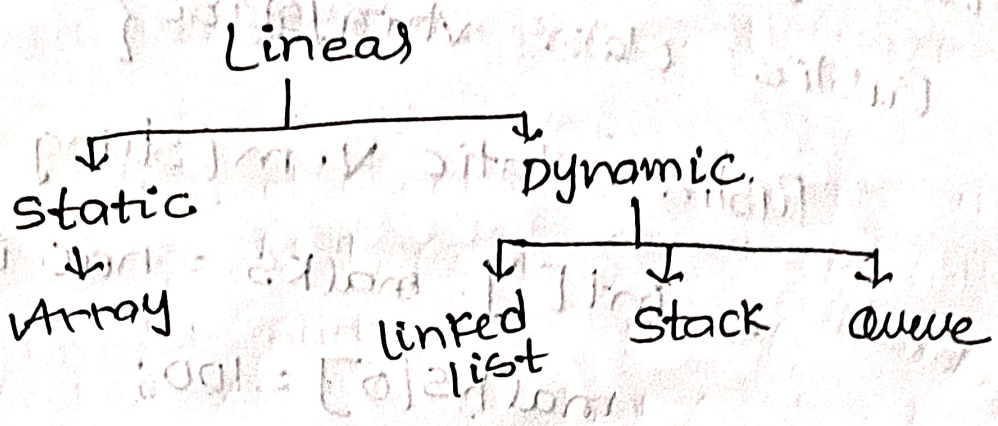
Transpose then

$$\begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}_{4\times2}$$

The Space is m×n

if m=n then space complexity is $O(n^2)$

Linear data structures:



→ C → Creation
A → Addition
R → Retrieval
V → Verification
U → Update
D → Delete
Should apply

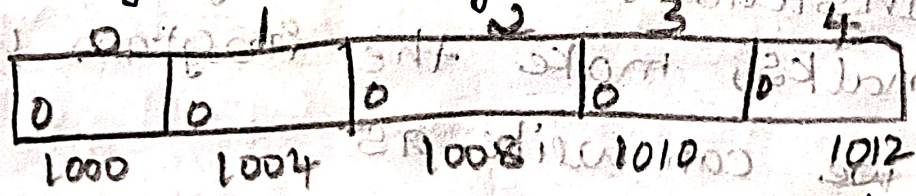
→ in Netbeans if we want main method then type psvm-(tab)

→ in eclipse type main (then directly give main method).

→ Array → 1) 1 Dimensional
2) Multi Dimensional

declaration of array:

→ datatype[] arrayname = new datatype[size];



initially with declaring it stores zero

→ Package pc;

Public class ArraysPrac {

Public static V. m(String args[]) {

declaration ← int[] malks = new int[5];

initialization { malks[0] = 100;

malk[1] = 98; Just given student malks

S.O.P (malk[0]);

S.O.P (malk[4]);

} }

→ It is not good way for taking each every student malks so,

for (int i=0; i< malks.length;i

{

S.O.P (malks[i] + " ");

} }

O/P 100 98 0 0 0

→ Instead of initializing every student malks, make the program user friendly

→ we can write as

int [] malks = new int[] {100, 98, 91, 85, 75};

declaring & initialising in one line

→ Read from user

```
Scanner sc = new Scanner(System.in);
int size = sc.nextInt();
for(i=0; i<size; i++) {
    a[i] = sc.nextInt();
```

→ In codechef 2 problems, max & Largest and second largest sum.

→ 2 sum Problem in Leetcode.

→ any 2 numbers sum is equal to target.

```
for (i=0; i<n; i++) {
    for (j = i+1; j<n; j++) {
        if (a[i]+a[j] == target)
        {
            return new int[] {i,j};
        }
    }
}
return new int[]{};
```

↓ if not pairs exist then return null.

H.W → In leetcode max product of two elements in array

→ solved 26th Problem in leetcode [Remove duplicats]

→ Q11 80 – in leetcode – Remove Duplicate Array II

→ Homework Problems – 9

→ 11 – Q11 in Leetcode

{1,2,3}
{4,5}
{6}

No. of cols are not equal in each row.
Jagged array

declaration :- int[][] arr=new int[3][];

{1, 2, 3}
{ 4,5,1}
{ 6,5,1}

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

Initialization

int [][]arr = new int[][]{ {1,2}, {3,4}};

Printing: (row wise)

```
for (int[] row : arr){
    S.O.P (Arrays.tostring(row));
}
```

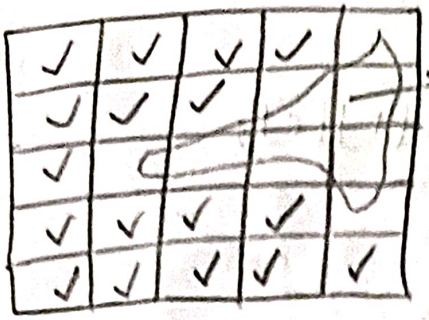
O/P [1,2]
[3,4]

Initialization in Jagged array

```
int [][] arr2 = new int[3][];
arr2[0] = new int[]{1,2,3,4,5};
arr2[1] = new int[]{6,7,8};
arr2[2] = new int[]{9};
```

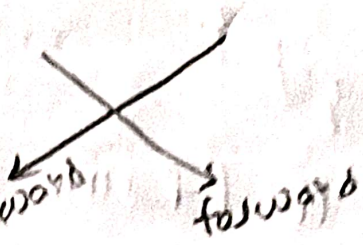
Printing row wise Jagged array

```
for (int[] row : arr2){
    S.O.P(Arrays.toString(row));
}
```



→ This memory wasted

so we go with jagged array



→ Supper Matrix: forward diagonal sum > backward diagonal sum

→ supper Duper Matrix: forward diagonal sum = backward diagonal sum

→ Duper Matrix forward diagonal sum < backward diagonal sum

→ Code for above senario:

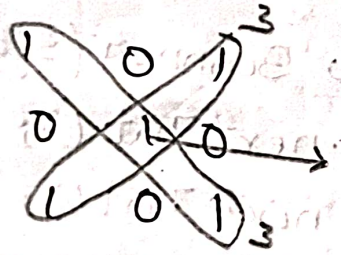
```
→ Scanner sc = new Scanner(System.in);
  int rows = sc.nextInt();
  int cols = sc.nextInt();
  int [][] arr = new int[rows][cols];
  for(int r=0; r<rows; r++){
      for(int c=0; c<cols; c++){
          arr[r][c] = sc.nextInt();
  }}
  int fsum=0; bsum=0;
  for(int r=0; r<rows; r++){
      for(int c=0; c<cols; c++){
          if(r==c){
              fsum+=arr[r][c];
          }
```

```
if (r+c == rows-1){
        bsum += arr[r][c];
}}}

if (fsum == bsum){
        S.O.P ("SuperDuper");
}
else if (fsum > bsum){
        S.O.P ("Super");
}
else
        S.O.P ("Duper");
```

O/P

3
3



SuperDuper

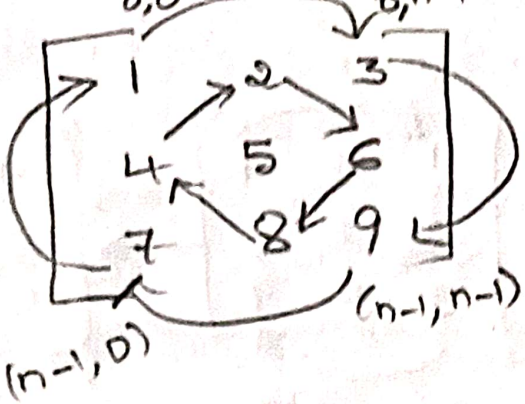
To include that 1 also we take if instead of else if.

→ With in single loop get forward & backward sum;

```
int fsum=0, bsum=0
for (r=0; r<rows/3; r++){
        fsum += arr[r][r];
        bsum += arr[r][rows-1-r];
}
```

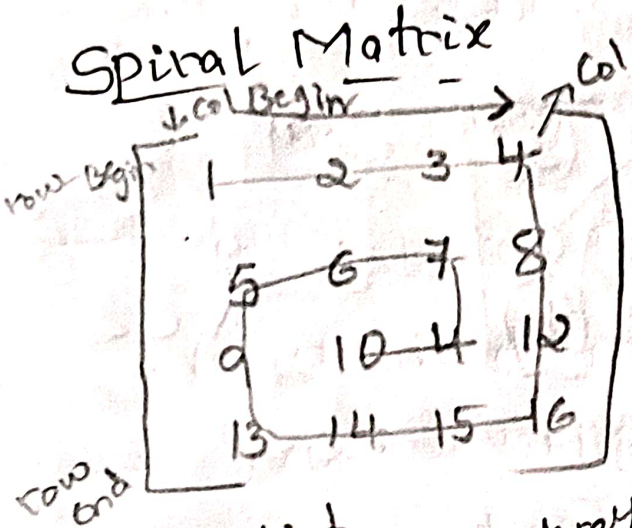
if Conditions

48



rotate image

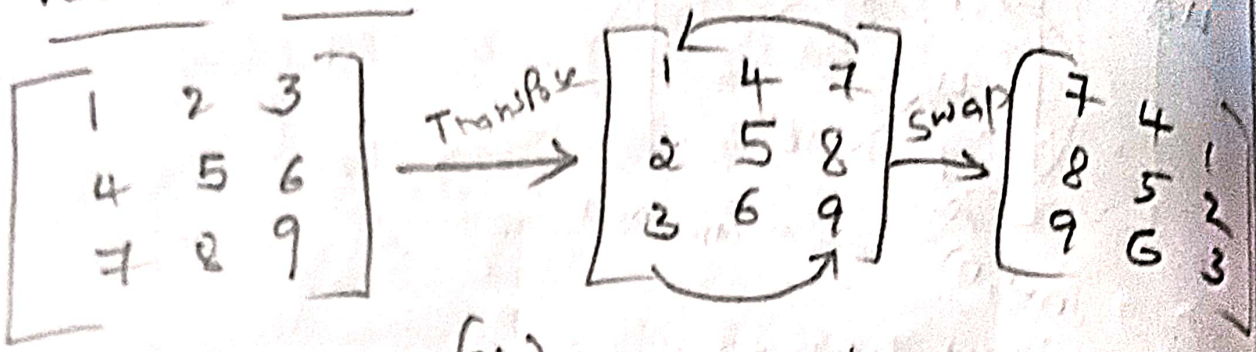
Spiral Matrix



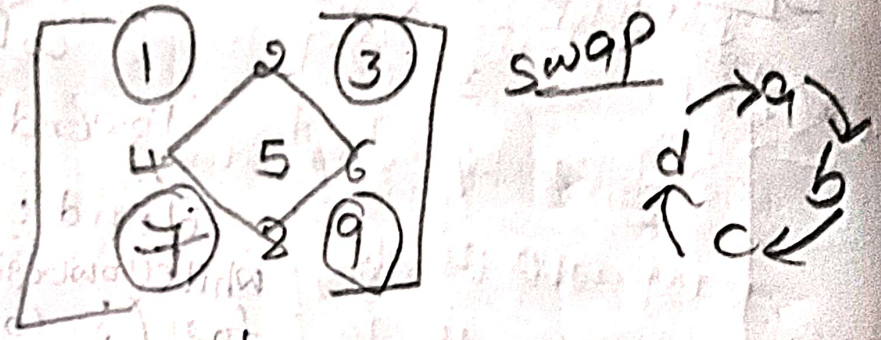

```
rowBegin = 0,
colBegin = 0
rowEnd = Mat.length-1
colEnd = Mat[0].length-1
while(rowBegin <= rowEnd && colBeg <= colEnd)
for (j = colBegin; j <= colEnd; j++) {
    ans.add(Mat[rowBegin][j]);
}
rowBegin++;
for (i = rowBegin; i <= rowEnd; i++) {
    Mat[i][colEnd];
}
colEnd--;
if (rowBegin <= rowEnd) {
    for (j = colEnd; j >= colBegin; j--) {
        Mat[rowEnd][j];
    }
    rowEnd--;
}
if (colBegin <= colEnd) {
    for (i = rowEnd; i >= rowBegin; i--)
        Mat[i][colBegin];
    colBegin++;
}
```

Take arraylist
ArrayList<Integer> ans = new ArrayList<>();

48/ Rotate image :-



(or)



→ Collections :-

Interfaces Provided are

Collections
- Collection Interface
  - list interface
  - set interface
  - Vector interface
- Map interface

Classify all the collection based on implementation style

we have 4 styles

→ Array Based implementation

→ Node Based implementation

→ Tree Based Implementation

→ Hash Based implementation

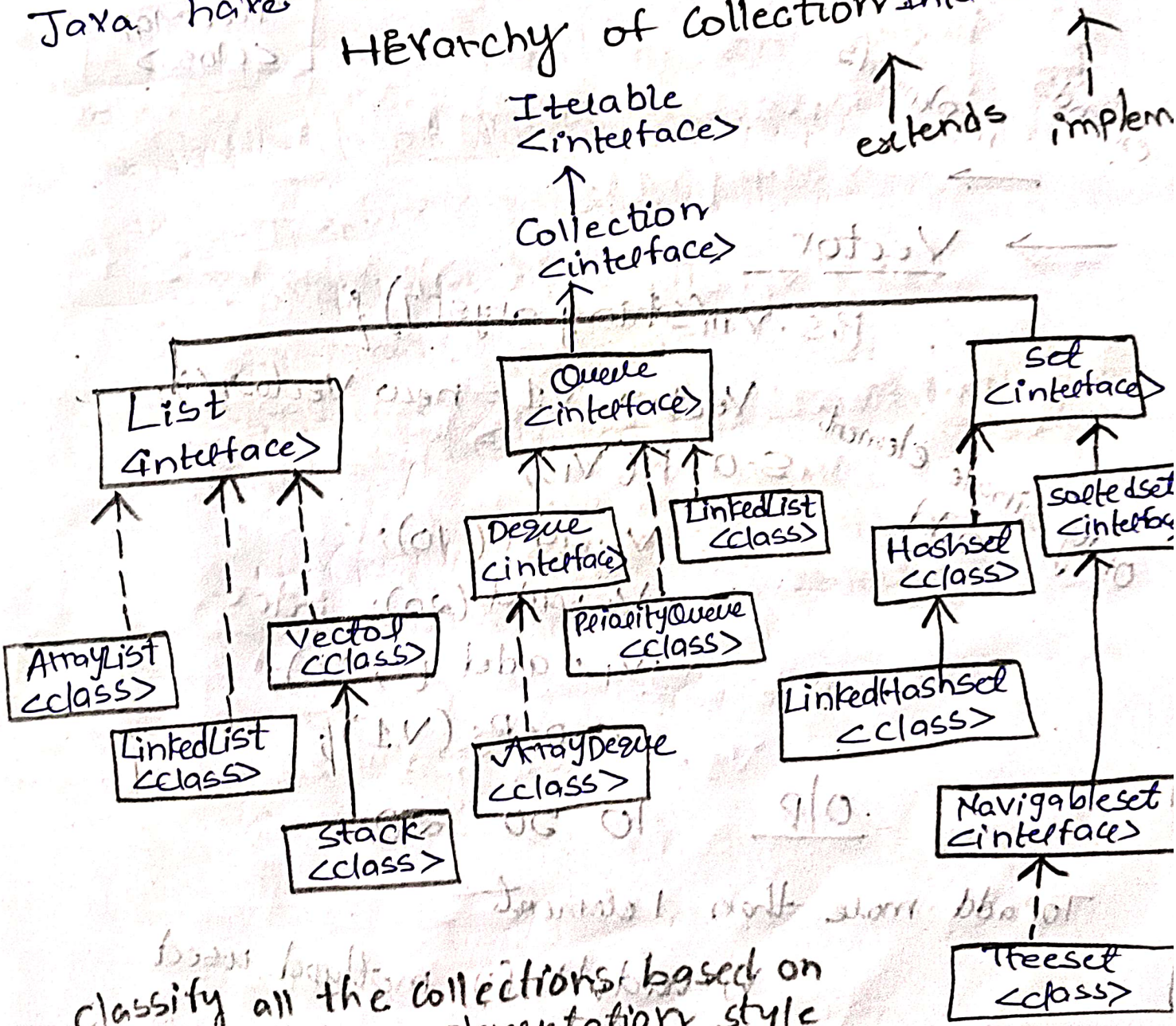
From 1.5 Version we have Collections & draw of Vector leads to Collections

Arraylist
Linkedlist
Vector
Stack
ArrayQueue
PriorityQueue
Treeset
Hashset
Collections
Linked Hashset

HashTable
HashMap
TreeMap
LinkedHashMap

Java have initially Vector class.

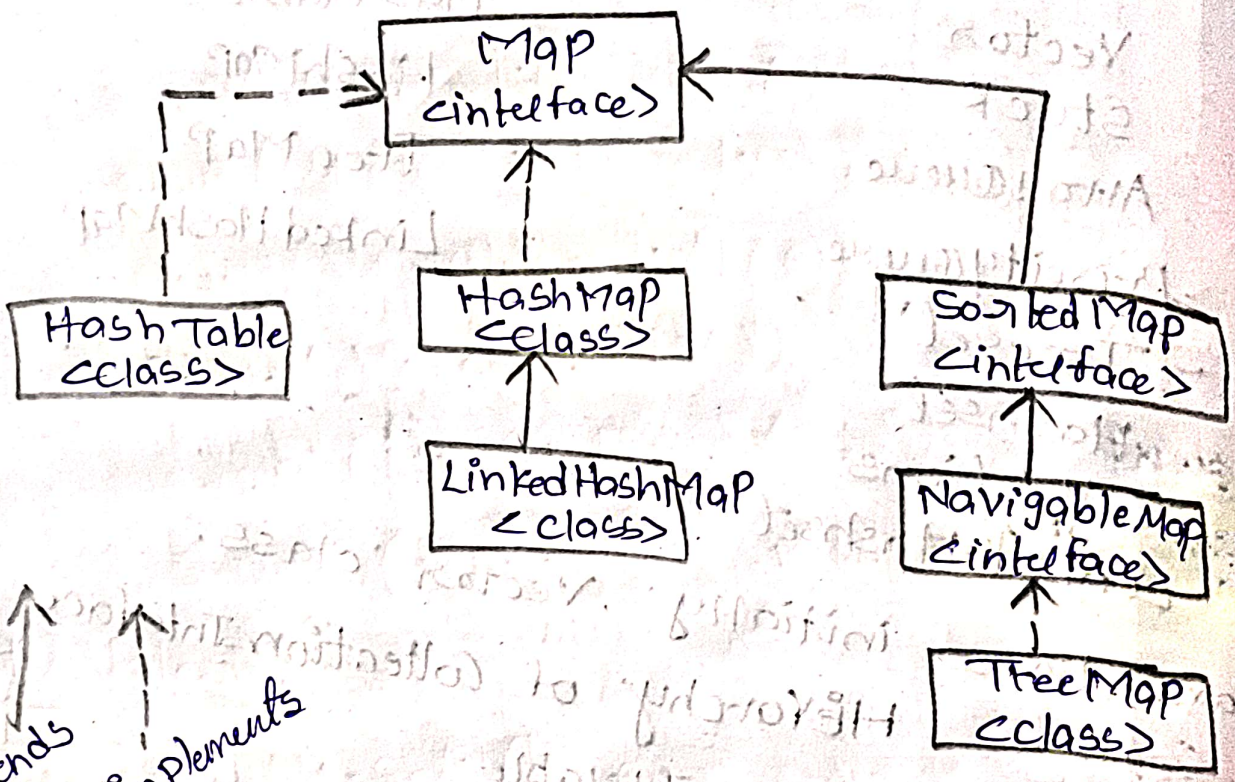
Hierarchy of Collection Interface



Classify all the collections based on their implementation style

a) method name | Example | significance

Hierarchy of Map Interface



→ Vector

```
P.S.V.M(String args[]){
    Vector V1 = new Vector();
    S.O.P(V1);
    V1.add(10);
    V1.add(20);
    V1.add(1,50);
    S.O.P.(V1);
```

only single element add

index

O/P 10 50 20

To add more than 1 element

V1.addAll() method used

Collection Vedal example

adding V_2 to V_1

```
Vector1 V2 = new Vector1 ();
V2.add (100)
V2.add (200)
V2.add (300)
S.O.P (V2);
V1.addAll (V2);
S.O.P(V1);
```

→ added at last

O/P :- 10 50 20 100 200 300

if V_1.addAll (3, V_2) then

↓ index

O/P 10 50 20 100 200 300 ----

Clone - copy

Contains

ContainsAll

elementAt (int i)

equals

get

indexOf → first occurence

getClass

↓ class name of Collection

* Retrival of Values from Vector

→ LastIndexOf ( )

remove ( ) → throw error in not found element

splitelator ( )

ListItelator ( )

SubList (int i, int i1) → (index 1-3 means it gives elements blw them.

toString ( ) → convert to string

add ( ) → given index, object.

clear ( )

addElement ( )

CopyInto ( )

forEach

Retrival

v1.get(4) → index

```
S.O.P(V1.firstElement());
S.O.P(V1.lastElement());
S.O.P(V1.elementAt(5)); → index
S.O.P(V1.elements());
S.O.P(V1.contoins(200)); → True
S.O.P(V1.containsAll(V2)); → True
```

Updation:

```
V1.set(0, 500); index 0 updated with 500
S.O.P(V1);
```

→ `S.O.P(V1.subList(1,5));` b/w 1-5 it gives

→ Deletion:

```
V1.remove(5); → throw error if not found
          index
V1.removeElementAt(2);
S.O.P(V1);
$ V1.removeAll(V2) → V2 elements removed from V1
V1.clear(); All elements removed
```

default capacity is 10.

if want to change capacity then

Vector v1 = new Vector(5);

10	20	30	40	50

if trying to add 6th element

then the capacity update by double to present capacity (ie) previous (10)

Imp Points to remember

For Vector

1) What is the default capacity? — 10 (capacity when elements are not added)

2) What is the initial capacity? — 10

3) Does it allow duplicate elements? — Yes

4) Does it allow Null values? Yes

5) Does it maintain the insertion order? Yes

6) Does it maintain the sorted order? No
Before adding only it finds position to add

7) Does it offer the random access of elements? Yes

8) Is it synchronised? Yes

9) What is it good at? memory dynamically increase

→ the memory doubled when we adding to 1 element remaining Wasted so, we introduced Array list to overcome remaining all same (Vector & array list) except memory.

↑ memory is efficiently used Array list

Array List :-

No Generics

```
ArrayList marks = new ArrayList();
marks.add(97);
marks.add(86);
marks.add("Vikky");
```

o/p [97, 86, Vikky]

In place of Marks they enter name when we try to add, it not possible... error.

So, Generics ↓

restrict the use of other type

* With Generics :-

* `ArrayList<Integer> marks1 = new ArrayList<Integer>();`

o/p [97, 86]

`marks1.add(1)` (2) waste of time

if we want to remove at rightside we can

`ArrayList<Integer> marks1 = new ArrayList<>();`

Now if we try to add string then we can't give giving error like incompatible type

2 Dimensional Collection:

Collection with in Collection

```
ArrayList<ArrayList<Integer>> scores = new
        ArrayList<ArrayList<Integer>>();
scores.add(marks);  → before er
scores.add(marks1); → before er
S.O.P(scores);
```

O/P [[97, 86, Vikky] [1, 2]]

↓ This is the draw back.

→ Total Program:

```
ArrayList<Integer> marks = new ArrayList<>();
marks.add(89);
marks.add(90);
marks.add(99);
ArrayList<Integer> ids = new ArrayList();
ids.add(189);
ids.add(176);
ids.add(193);
ArrayList<ArrayList<Integer>> s = new
        ArrayList<ArrayList<Integer>>();
s.add(marks);
s.add(ids);
S.O.P(s);
```

Customized Generics – Creating our own Generics

```
class Student {
    String name;
    int id;
    char grade;
//Constructor
    Student (String name, int id, char grade) {
        this.name = name;
        this.id = id;
        this.grade = grade;
    }
}
class Practise {
    P.S.V.M (String args[]) {
        ArrayList<Student> students = new ArrayList();
        Student s1 = new student ("alpha", 10, 'A');
        Student s2 = new student ("Beta", 20, 'B');
        Students.add (s1);
        Students.add(s2);
        for (int i=0; i<2; i++) {
            S.O.P (Students.get(i).name);
        }
    }
}
ArrayList <ArrayList <String>> names = new
                ArrayList();
```

(or)

```
ArrayList<Vector<String>> names = new
                ArrayList();
names.add(v2);
names.add(v3);
```

Collection interface:-

The Collection interface is the interface which is implemented by all classes in the collection framework. It declares the methods that every collection will have

List interface:

List interface is the child interface of Collection interface. It inhibits a list data structure in which we can store the ordered collections of objects. It has duplicate values

- List interface is implemented by the classes Arraylist, Linkedlist, Vector & stack.

```
List <data-type> list2 = new ArrayList();
List<data-type> list1 = new LinkedList();
List<data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

- There are various methods in list interface that can be used to insert, & delete, access the elements from the list.

Queue interface:

* Queue interface maintains the first in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. These are various classes like PriorityQueue, & ArrayDeque which implements the queue interface

Queue <String> q1 = new PriorityQueue();

Queue <String> q2 = new ArrayDeque();

Set Interface:

Set Interface in Java is present in java.util Package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. Set is implemented by HashSet, LinkedHashSet & TreeSet

```
Set<data-type> s1 = new HashSet<data type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

Map Interface:

Map interface in Java is a part of Java Collections Framework. It represents a mapping b/w a set of keys & their corresponding values. No duplicate keys, each key can map to at most to at most on value.

```
Map <T, T> hm = new HashMap<>();
Map<T, T> tm = new TreeMap<>();
```

Method name	Example	Significance
add (E e)	List.add ("Apple")	Adds an element to a Collection (List, set, Queue)
remove (Obj O)	List.remove("Apple")	Remove the 1st occurrence of specified element
contains(Obj)	set.contains("Apple")	Checks if the collection contains the specified element
size()	List.size()	Returns the no. of elements
isEmpty()	queue.isEmpty()	Checks if the collection is empty
clear()	list.clear()	Removes all elements from the collection
get (int index)	List.get(0)	Retrieves the element at the specified index in a list
set(int index, E.e)	List.set (0, "Apple")	Replaces the element at specified index in list
indexof (Obj O)	List.indexof("Apple")	Returns index of 1st occurence
toArray()	Obj [] arr = List.toArray()	Converts the collection into an array
Push (E.e)	stack.Push("Apple")	Adds an element to the top of a stack
POP()	Stack.POP()	removes the top element.

→ Class Practise {

P.S.V.m(String args[]) {

ArrayList<Session> s = new ArrayList<>();

Session h1 = new Session(1, "DAA", "A");

Session h2 = new Session(2, "CC", "M");

Session h3 = new Session(3, "TOC", "B");

s.add(h1);

s.add(h2);

s.add(h3);

for (int i=0; i<3; i++){

S.O.P (s.get(i).subName + "→" + s.get(i).facName);

}

}

}

Class Session {

int hour;

String subName;

String facName;

Session (int hour, String subName, String facName){

this.hour = hour;

this.subName = subName;

this.facName = facName;

}}

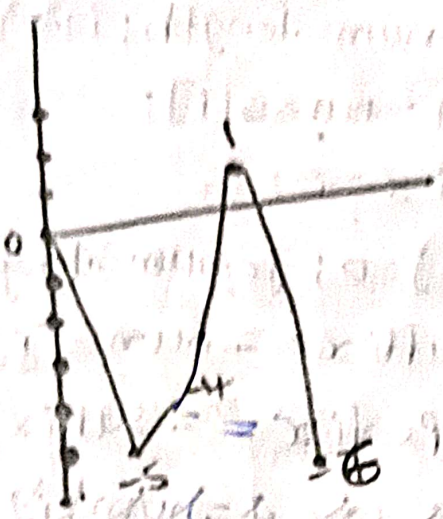
O/P

DAA → A

CC → M

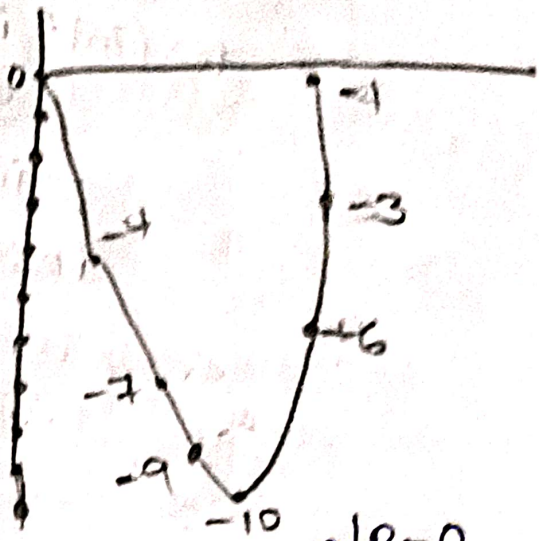
1732

[-5, 1, 5, 0, -7]



o/p 1

[-4, -3, -2, -1, 4, 3, 2]



o/p - 0

max = 0
p = 0
i = 0, 0<5 ✓
p = -5
max = 0
i = 1, 1<5 ✓
c = -5+1
= -4
max = 0
i = 2, 2<5
c = -4+5
= 1
max = 1
i = 3, 3<5 ✓
p = 1+0
= 1
i = 4, 4<5 ✓
c = 1+(-7)
= -6
max = 1 ✓

```
MaxAlt = 0
Prefixsum = 0
for (i=0 ; i< gain.length ; i++){
    Prefixsum += gain[i]
    MaxAlt = Prefixsum > maxAlt ? Prefixsum
                                : MaxA
                                ↘ Math.Max(-,-)
}
return MaxAlt;
}
```

724

[1, 7, 3, 6, 5, 6]
Pivot
11 ← ← 11 →

total sum = 28

[4, 7, 3, 6, 5, 6]

28 - 1 = 27 ⇒ 17 ≠ 27
27 - 7 = 20 → 7 ≠ 20
20 - 3 = 17 →
17 - 6 = 11

Suffix = 28 - 1 = 27
Prefix = 0 + 1 = 1+7

Suffix = 27 - 7
= 20
20 - 3 = 17

Prefix = 8 + 3 = 11
Suffix = 17 - 6 = 11

```
int suffix = 0
for (int i = 0; i < nums.length; i++) {
    suffix += nums[i];
}
int Prefix = 0;
for (int j = 0; j < nums.length; j++) {
    suffix -= nums[j];
    if (Prefix == suffix) {
        Prefix += nums[j];
        return j;
    }
}
return -1;
```

Tracing [1, 7, 3, 6, 5, 6]

s = 0, i = 0, 0 < 6 ✓
s += 1+0 ⇒ 1
i = 1, 1 < 6 ✓
s = 1+7 = 8
⋮
sum all nums = 28
P = 0
i = 0, i < 6 ✓
s = 28 − 1 ⇒ 27
27 == 0 ✗

Prefix += 1
i = 1, 1 < 6
s = 27 − 7 = 20
20 == 1
P = 1+7 = 8
j = 2, 2 < 6 ✓
s = 20 − 3
= 17
17 == 8 ✗
P = 8+3 = 11

i = 3, 3 < 6
s = 17 − 6
= 11
11 == 11 ✓
return 3.

→ Sum of array except Self.

```
s = 0
for (int i = 0; i < num.length; i++) {
    s += nums[i];
}
for (int i = 0; i < num.length; i++) {
    nums[i] = s - nums[i];
}
```

238: Product of array except itself

```
class Solution {
Public int[] Product (int [] nums) {
   int n = nums.length;
int [] result = new int[n];
    result[0] = 1;
   for (int i=1; i<n ; i++) {
    result[i] = result[i-1] * nums[i-1];
}
int suffix = 1
   for (i=n-1 ; i>=0; i--) {
     result[i] = result[i] * suffix;
        suffix *= nums[i];
   } return result;
}}
```

[1 2 3 4]

[1 1 2 6]

[24,12,8,6]

1×4 = 4

4×3 = 12

(OR)

```
int [] Prefix = new int[n];
int [] suffix = new int[n];
    prefix [0] = 1;
  for (int i=1; i<n; i++) {
     Prefix [i] = Prefix [i-1] * nums[i-1];
}
  suffix [n-1] = 1;
  for (int i=n-2; i>=0; i--) {
    suffix [i] = suffix [i+1] * nums[i+1];
}
for (int i=0; i<n; i++) {
 nums [i] = suffix [i] * Prefix [i];
} return nums;
}
```

→ Strings :-

String literal

String Pool
heap memory

```
String s1 = "Vignan";
String s2 = "Vignan";
S.O.P (s1)
```

Here content same for s1 & s2 and then it doesn't allocate different memory. It allocate same memory for both.

→ Check s1==s2 equal o/p: True
checks address location equal or not
if change one Variable means it effect other so Strings are immutable

→ Using string class

```
String s3 = new String ("Guntur");
String s4 = new String ("Guntur");
```

These 2 Contents stores in different byt new means new memory allocated

S.O.P (s3==s4) o/p false.

Methods of String

Check	Position	Manipulate	char
✓ Contains()	✓ Length()	trim()	charAt() ✓
✓ equals()	✓ indexOf()	toUpperCase()	
equalsIgnoreCase()	lastIndexOf()	toLowerCase()	
startswith() ✓	CompareTo()	Concat() ✓	
endswith() ✓	CompareToIgnoreCase()	replace() ✓	
		substring() ✓	

```
P.S.V.M(String args[]){
    String s = new String ("Vignan University");
    S.O.P (s.Contains ("vignan")); O/P True
    S.O.P (s.Contains ("V")); O/P True
    S.O.P (s.Contains ("Gnanvi")); O/P false

    String s2 = new String ("Vignan");
    String s3 = new String ("Vignan");
    S.O.P (s2.equals(s3)); O/P - True
    S.O.P (s1.equalsIgnoreCase(s3)) True
```

S_1 = "Vignan"
S_2 = "vignan" equal (ignore caps) True

Java 1.3 Version

```
    S.O.P (s2.startswith ("Vi")); True
    S.O.P (s2.startswith ("Gnan")); False
    S.O.P (s2.Endswith ("gnan")); True
    S.O.P (s2.endswith ("gnan")); False
    String s4 = new String ("vignan");
    S.O.P (s1.length()); ——— 6
    S.O.P (s1.indexOf('n')) ⟶ 3 (first occurance)
    S.O.P (s1.lastIndexOf('n')); (last occurence)
    S.O.P (s2.CompareTo(s3)); ⟶ 0 (equal s2 & s3)
    S.O.P (s2.CompareTo(s4)); ⟶ -32 Ascii value
    S.O.P (s2.CompareToIgnoreCase(s4))
                ⟶ 0
                V-V ignore
```

s2 = Vignan;

String s4 = new String(" Hyd ");

s.o.p(s4.trim()); o/p Hyd (without spaces)

s.o.p(s4.toUppelCase()); HYD

s.o.p(s4.toLowerCase()); hyd

s.o.p(s1.concat(s4)); Vignan hyd

s.o.p(s2.replace('n','m')); Vigmam

s.o.p(s2.substring(2)); gnan (from 2)

s.o.p(s2.substring(2,5)); gna (2-5)

s.o.p(s2.charAt(5)); n

Useful Method

ValueOf()

isEmpty() ""

isBlank() "" space

Split()

toCharArray()

isEmpty "" (no space)

isBlank means "" (space) gives True or false

S = "Vignan University guntur";

s.split(" ") → splits where spaces is there

o/p ["Vignan", "University", "guntur"]

if S = "vignan"; → converted to chars

ch [] = {'v', 'i', 'g', 'n', 'a', 'n'};

→

```
Char[] chars = s1.toCharArray();
for (char c : chars)
    s.o.p(c+" ");
```

O/P
v i g n a n

→

```
String s5 = new String("vignan University guntur");
String[] words = s5.split(" ");
for (String word : words) {
    s.o.p(word);
}
```

O/P
vignan
University
guntur

(Mutable) if we change it remains same

StringBuffer & StringBuilder

append()
insert()
setCharAt()
delete()

deleteCharAt()
getChars()
replace()
reverse()

```
StringBuffer sb1 = new StringBuffer("vignan");
sb1.append("University");
```
O/P
vignan University

```
sb1.append("Guntur", 0, 3);
```
Vignan gun
only gun append at last

```
sb1.insert(6, "Hyderabad");
S.O.P(sb1);
```

o/p vignan Hyderabad University Gun

→ `sb1.insert(6, "Hyderabad", 0, 6);`

`S.O.P(sb1);` vignan Hyder University gun

`sb1.setCharAt(3, 'm');` ←

`S.O.P(sb1)`

o/p vignan Hyder University gun

`sb1.delete(0,10);`

↳ 0-9 (not print)

o/p er University gun

→ String Buffer & String builder difference

String Buffer → synchonised, Easier

String builder → Asynchronized

String Programs

Read char by char

Print
- ASCII
- UpperCase
- LowerCase
- Digit
- Remove Spaces

Count
- UpperCase
- LowerCase
- Digit
- Vowels
- Any char

New string
- Reverse
- Palindrome
- only Digits
- only chars
- Reverse case
- Encoding

Read wold by wold

Plint
- Eachwold
- starting with specific char
- Wold length
- find wold
- longest

Count
- countwolds
- Count specific wolds
- wolds length
- starting with specific char

New string
- Capital
- swap wold
- Delete wold
- Edit wold

→

```
class Main{
    p.s.vm (String args[]) {
        Sconner sc = new Scanner(System.in);
        String s = sc.next;
        char c = sc.next().charAt(0);
        So.P(s.startsWith(Character.toString
                                    (c)));
```

O/P
hello
h
true

→

```
String str = sc.nextLine();
char[] chars = str.toCharArray();
```

→ converting string to char array

```
l = 0;
r = str.length()-1
while (l<r) {
    char temp = char[l];
    char[l++] = char[r];
    char[k++] = temp;
    char[r--] = temp;
```

swapping

y t i s r e v i n U
U n i V e r s i t y
↑ r ↑ ↑ l ↑ r

when overlap stop

Madam → no swap so Pointing to same Position

```
Print (Arrays.toStrings(chars));
S.O.P (toString(c));
```

→ Palindrome

str = "civic"
↑ ↑

```
l = 0
r = str.length()-1;
while (l<r) {
```

```
if (str.charAt(l) != str.charAt(r)) {
        return false;
        l++;
        r--;
    }
    return true;
}
```

checking by using 2 Pointers approach until middle of the chars same then Palindrome

→ A-Z 65-90

a-z 97-122

0-9 48-57

'\0' 0

H.W

"m, a: d# a@m, Palindrome?

→ Count specific words

```
Class Main {
    P.s.v.m (String[] args) {
        Scanner sc = new Scanner (System.in);
        String s = sc.nextLine();
        String c = sc.nextLine();
        int count = 0;
        String[] w = s.split("\\s+");
        for (String word : w) {
            if (word.equalsIgnoreCase(c)) {
                count++;
            }
        }
        s.o.p(count);
    }
}
```

O/P

hello vignan hello

hello

2

Encoding a string changing the string

```
Public class Stringpro {
    Static String encode(String str, int k)
    {   String encodedstring = "";
        for (char c : str.toCharArray()) {
            encodedString += (char)(c+k);
        }
        return encodedstring;
    }
    P.s.v.m(String[] args) {
        S.o.P(encode("ABC", 7));
```

O/P
HIJ

→ Valid Palindrome (125)

```
int l=0, r=str.length()-1;
while(l<r) {
    while (!character.isLetterOrDigit
                (str.charAt(l))
        l++;
    while(!character.isLetterOrDigit
        l<r &&      (str.charAt(r))
        r--;
if (character.toLowerCase(str.charAt(l)) !=
    Character.toLowercase(str.charAt(r)))
        return false;
        l++;
        r--;
    }
    return true;
}}
```

387: First unique character in a string

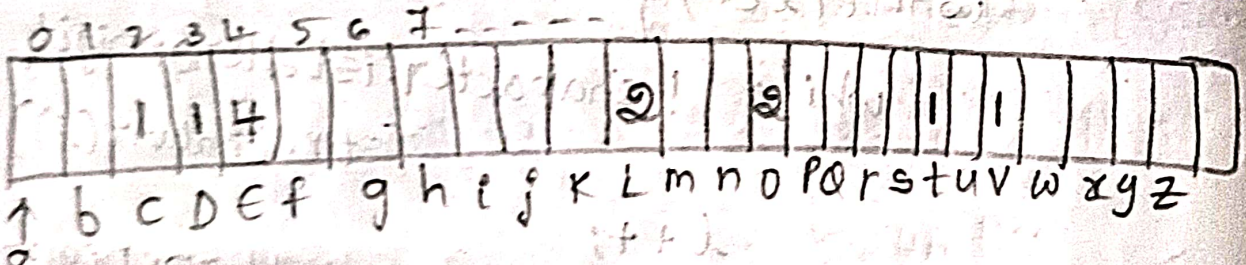
→
```
int n=s.length();
for(int i=0; i<n; i++){
    int freq=0;
    for(int j=0; j<n; j++){
        if(s.charAt(i)==s.charAt(j)){
            freq++;
        }}
    if(freq==1){
        return i;
    }}
return -1;
}}
```

But Time Complexity $O(n^2)$

Time limit Exceeded

Reduce Time:

a-z → 26 letters, so size 26 constant



ex: "lovelectcode"

Take another array & take the frequency of every character in the array

for 'a' position 'a'-'a' = 0

b index → 'b'-a = 1 ← (index)

Corresponding letter - 'a' → we get index of corresponding letter

we have to check first occurence of letter having freq 1.

```
int a[] = new int[26];
for(char ch: s.toCharArray()) {
    a[ch - 'a']++;
}
for(int i=0; i<s.length(); i++) {
    if (a[s.charAt(i) - 'a'] == 1) {
        return i;
    }
}
return -1;
}
```

Monday

```
Package memory;
Public class Student {
    int orgID = 1234;
    String orgName = "Vignan Universe";
    int noOfDepts = 4;
    double netwealth = 500.00;
    Static String chairman = "Dr. Lara";
Public static Void main(String[] args) {
        int SID = 420;
        String sName = "chitti";
        String sDept = "CSE";
    Student S1 = new Student();
    S.O.P(S1.getNetWealth());
        S1.getOrgName();
  Student S2 = new Student();
        S2.getDetails(SID);
  }
    Public double getNetWealth() {
        return netwealth;
    }
  Public String getOrgName() {
        return orgName;
    }
 Public Static Void getDetails(int SID
    {
 }
```

ROM =16MB

Heap Area

Objects
String Literals
Arrays
object = Instance Variables + Methods

String Pool
String literals

x201	Vignan University
x202	Dr. Lavu
x203	Chitti
x204	CSE

S1

orgId : 123
orgName x201
no.of Dept = 4
NetWealth = 500
Chairman = x202
getNetWorth
getName

S2

getDetails()

RAM

IMB

Stack Area

Local Variables
Reference Variables

Stack Frames(T1)

getOrgName()
getNetWorth()

t.bird()

SID	420
SName	Chitti
SDept	CSE

PermGen / MetaSpace
Class structure
Static Variables
Static Methods
Static blocks

Method Area

chairman
getDetails()

```
Package memory;
Public class MemoryManagement {
        Void run(int i) {
            System.out.println(i);
            i++;
            run(i);
        }
        Public static void main(String[] args){
// new MemoryManagement().run(i);
        }}
            int size=100;
        for(int i=0; i<Integer.MAX_VALUE;
                                    i++){
            S.o.p(i);
            int[] arr =new int[size];
            size *=2;
        }}
```

1 Program

upto hell i program

Tuesday

```
class Student {
    String name;
    int age;
    int id;
    Public Student(String name, int age,
                                    int id) {
        this.name =name;
        this.age =age;
        this.id = id;
    }
```

```
Public String tpString() {
return "Student {" + "name =" + name + '\'' +
                age =" + age + ", id =" + id
                + '}';
}
}
Public class main {
  Public static String student_info(Students)
  {
    return s.toString();
  }
  Public.s.v (str[]args) {
    Student, Student = new Student ("Alice",
                                    20, 123);
    String info = student_info (student);
    s.o.p(info);
}
}
```

updation:

```
Package recursion;
Public Class Functionex {
    Static Student updateInfo(Student s){
        s.cgpa = 9.9f;
        s.address = "Bihar, India";
        return s;
    }
Public static Void main (String[] args){
    Student s1 = new Student();
    S.O.P("Original Information");
    S.O.P(s1.sID);
    S.O.P (s1.sName);
    S.O.P (s1.cgpa);
    S.O.P (s1.address);
    updateInfo(s1);
    S.O.P("Updated information");
    S.O.P (s1.sID);
    S.O.P (s1.sName);
    S.O.P (s1.cgpa);
    S.O.P (s1.address);
}
}
```

Just add all & Just subtract that num from Total Sum

*↑ Sum of elements of array except itself.

Just update

```
import java.util.ArrayList;
Public Class Function {
Static ArrayList<Integer> updateArray(ArrayList
    <Integer> arr) {
```

```
int sum=0
for(int i=0; i<arr.size(); i++){
    sum += arr.get(i);
}
for(int i=0; i<arr.size(); i++){
    arr.set(i, sum-all.get(i));
}
return all;
}
}
```

↓ subtract

```
P.S.V (String args[]){
ArrayList<Integer> nums = new ArrayList(
    nums.add(1);
    nums.add(2);
    nums.add(4);
    nums.add(3);
    nums.add(5);
    S.O.P(nums);
S.O.P(updateArray(nums));
}}
```

Complex task → divided into simple task → all simple task having same work then use → recursion

1) Base Condition → (finite (where to stop))
2) Recursive call

Recursive (or) Recursive

Calling same function itself

```
Package recursion
  Public class RecursionExample {
     static void PrintNum1() {
          S.O.P (1);
     }
     Static void PrintNum2() {
          S.O.P(2);
     }
    Static void PrintNum3() {
          S.O.P(3);
     }
    P.S.V.m(){
        PrintNum1();
        PrintNum2();
        PrintNum3();
    }
```

} calling functions

(More) So, if i call first function, the first will call second, second — third ——— then

```
Public class RecursionExample {
     Static void PrintNum1() {
          S.O.P(1);
          PrintNum2();
     }
     static void PrintNum2() {
          S.O.P(2);
          PrintNum3();
     }
```

and so on ....

Here Print & calling happening. all are following (same task) Pythonhubs.com

→ Static Void PrintNum(int n){
S.O.P(n);
PrintNum(n+1); ← here it should Print all numbers so, do increment
}
Static Void PrintNum5(){
S.O.P(5);
}
P.S.V.m(St ar[]){
PrintNum(1);
}}

So just Put n+1

→ Give base Condition to stop infinite loop

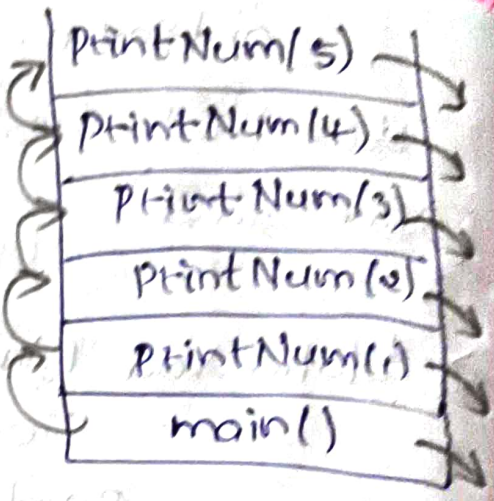
Public Class RecursionExample {
Static Void PrintNum(int n){
if (n==5){
S.O.P(n);
return;
}
S.O.P(n);
PrintNum(n+1);
}
Static Void PrintNum5(){
S.O.P(5);
}
P.S.V.m(Stl arg){
PrintNum(1);

Base Condition

→ Recursive Call

op

1
2
3
4
5



→ Fibanacci

0 1 2 3 4 5 6 ...

0, 1, 1, 2, 3, 5, 8 - - - - - - - -

I need nth fibanocci

$$fib(N) = fib(N-1) + fib(N-2)$$

reccurence relation

$fib(3) = fib(2) + fib(1)$

$fib(2) = fib(1) + fib(0)$

→ $fib(1) = fib(0) + fib(-1)$

we have no -1 so this is the base condition, we stop here

$fib(0) = fib(-1) + fib(-2)$ ✗

→

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

fib(4) → call

fib(4) ✓

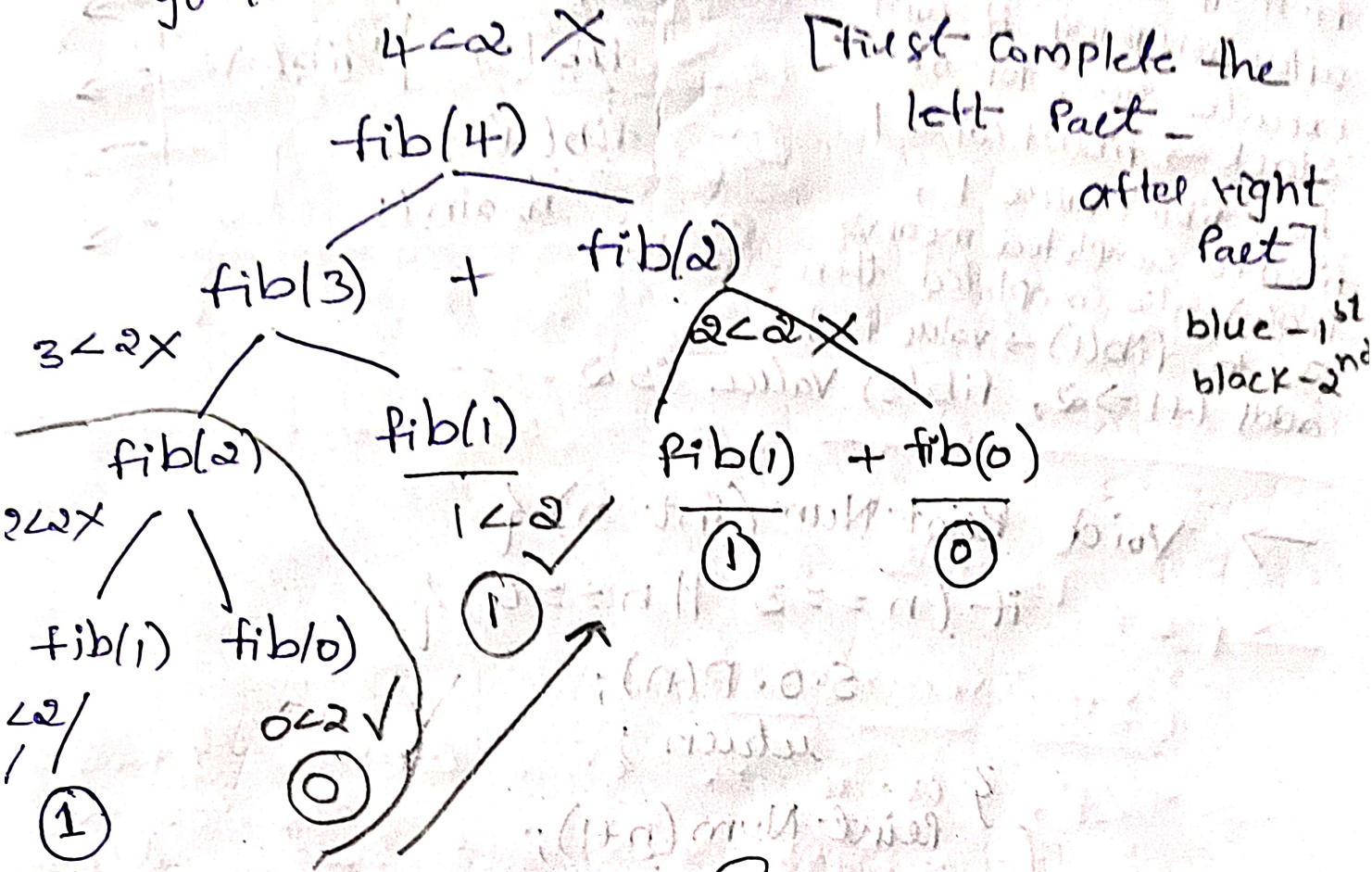
go to function

4<2 ✗

[First complete the left part_ after right part]

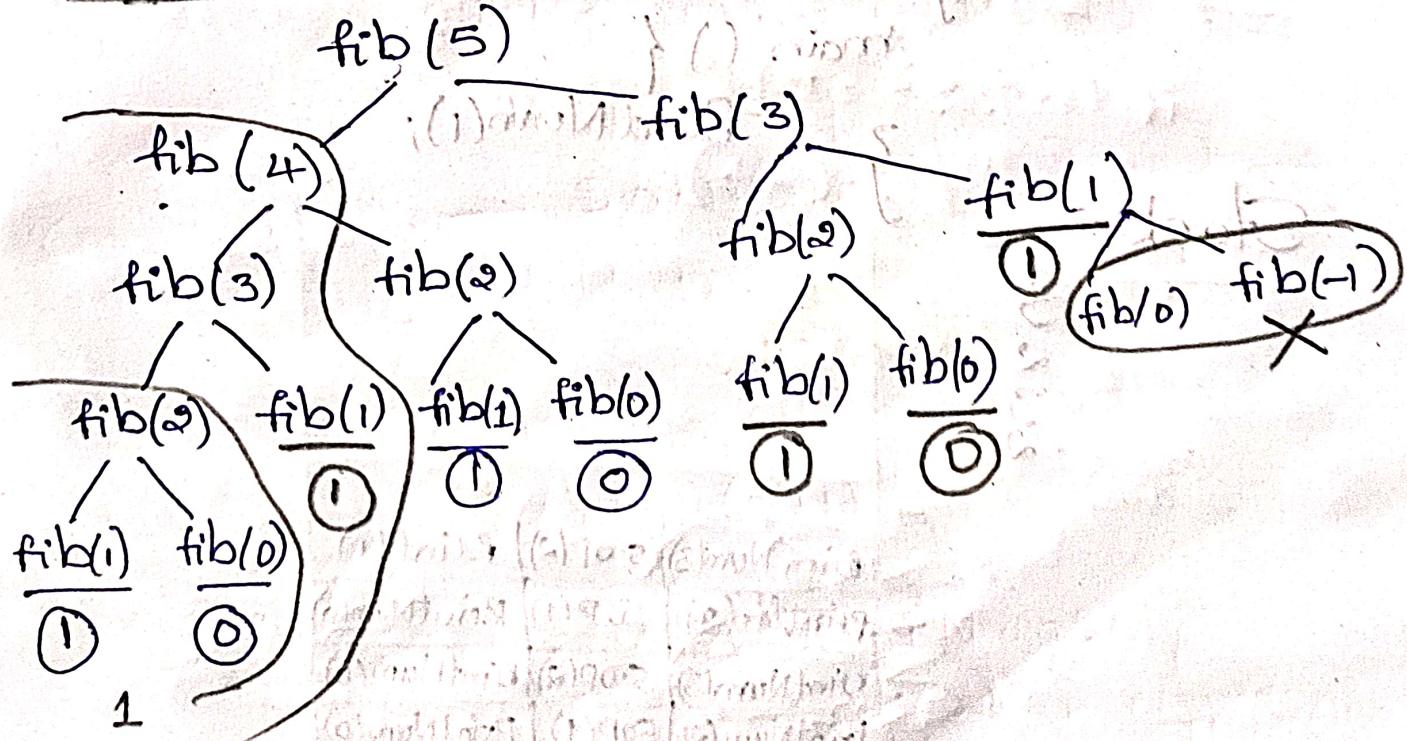
blue - 1st

black - 2nd

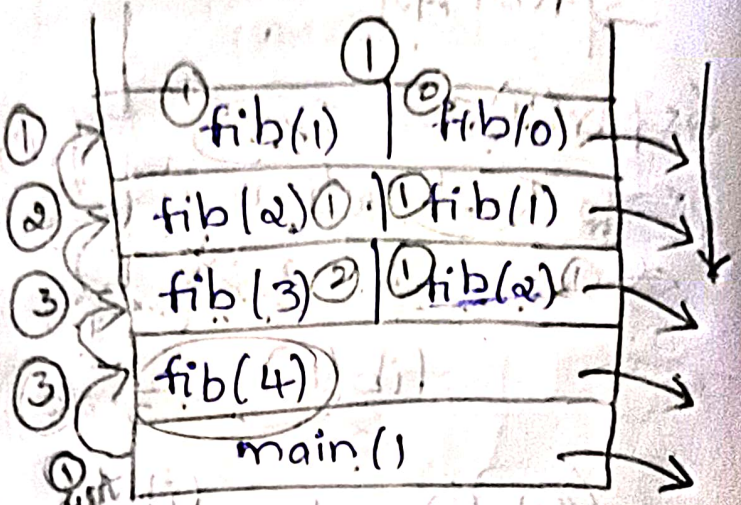


The Recursive Tree

→ fib(5)



Stack



first it will asign memory after asigning it will execute from top of stack → fib(1)+(0) ⇒ 1 here fib(2) value 1, fib(1)+0 Completed means fib(2) also Completed then go to fib(1) → value is 1 (add) 1+1 ⇒ 2, fib(3) value is 2 ----

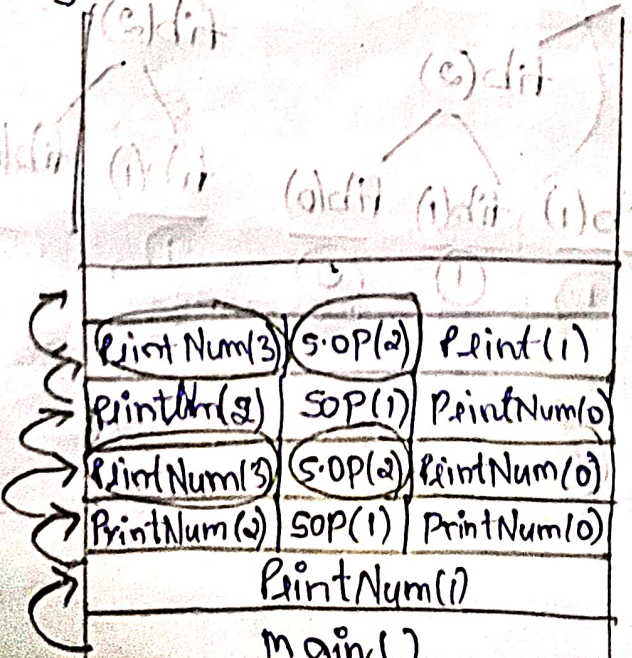
---

→
```
Void PrintNum(int n){
    if(n==3 || n==0){
        S.O.P(n);
        return;
    }
    PrintNum(n+1);
    {
        S.O.P.(n);
        PrintNum(n-1);
    }
}
main(){
    PrintNum(1);
}
```

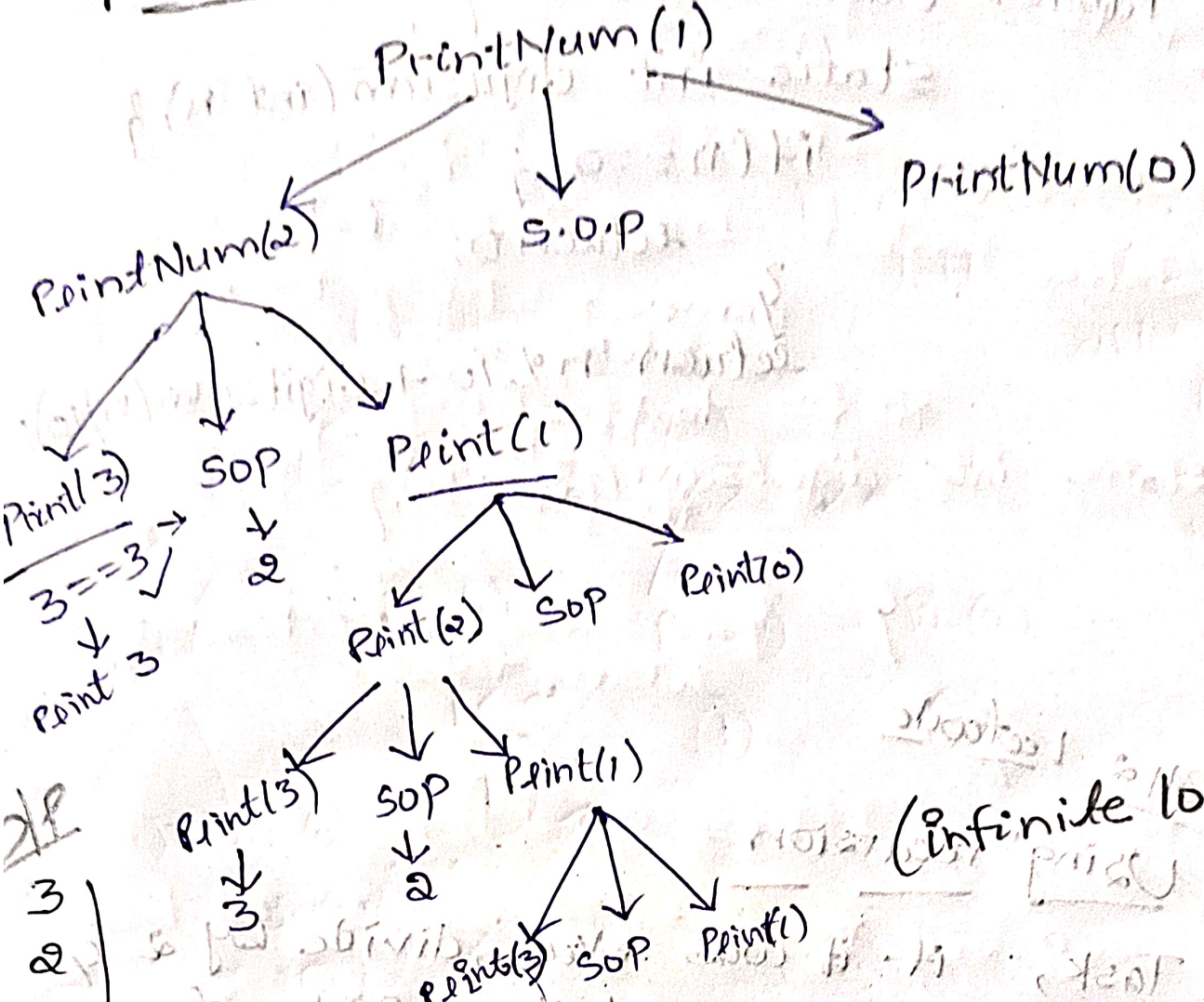
3 statement 2 recursive calls

Stack:

3
2
3
2



Recursive Rec



→ 1 4 2 3 5

T.C = log n

S.P → mole

Sum(N) = 5 + sum(1423)

3 + sum(142)

2 + sum(14)

4 + sum(1)

1 + 0

Sum(N) = N%10 + sum(N/10)

Here the base Condition is, if we reach '0' it had added all.

→ Recursive relation :

```
Public class Recursive {
    static int digitsum(int n) {
        if(n==0) {
            return 0;
        }
        return n%10 + digitsum(n/10);
    }
```

```
static int digitsum(int n) {

}
```

→ 1342 Leetcode

Using recursion

→ Task, if it was even, divide by 2 & Count, if odd subtract 1 & count. Here some operation & incrementing

→ Base Condition n==0

```
if (n%2==0)
    (n/2, c++)
else
    (n-1, c+1)
```

```
Countsteps(n) {
    helper(n, 0);
}

helper(n, cnt) {
```

```
if (n==0)
    return cnt;
if (n%2==0)
    return helper(n/2, cnt+1);
return helper(n-1, cnt+1);
}
```

271
326
342

→ static boolean isPowerof(int n) {

```
if (n==0)
    return false;
else if (n==1)
    return true;
else if (n%2!=0)
    return false;
return isPowerof(n/2);
}
```

→ static double isPower(double x, int n) {

int k = n>0 ? n : -1*n;

(if it is not a +ve convert to +ve)

← double

```
double res=0;
for (int i=1; i<len; i++) {
    res *= x;
}
return res;
```

→ res = n>0 ? res : 1/res;

```
}
p.s.v.m() {
    S.O.P isPower(3, 2)
```

↓

if (3, -2)

→ 1823 LeetCode
Convert to recursion

Linear Approach :-

```
→ Public class Linearsearch {
    static boolean Linearsearch(String [] names, String target) {
        steps = 0
        for (String name : names) {
            steps += 1;
            if (name == target) {
                return true;
            }
        }
        → S.O.P("iters" : steps);
        return false;
    }
    P. S.V.m (String args[]) {
        String[] names = {"Priya", "Harshita", "Nihari", "Nandu", "Minakshi", "Rujitha"};
        String target = "Sreeleela"
        S.O.P(Linearsearch(names, target));
    }
}
```

O/P
iters : 6
false

```
Static int Linearsearch (String [] names, string target) {
    for (int i=0; i<nums.length; i++) {
        if (nums[i] == target) {
            return i;
        }
    }
    return -1;
```

→ No. of occurences

```
static ArrayList<Integer> findOccurences(String[]
                                names, String target){
    ArrayList<Integer> res = new ArrayList();
        for (int i=0; i<names.length; i++)
            if (names[i] == target)
                res.add(i);
    return res;
}
```