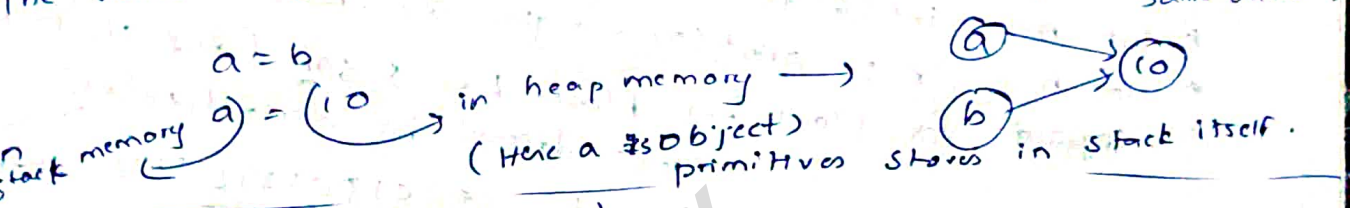
Three types of programming language

1. procedural : well-structured steps & procedures
2. functional :- pure functions i.e. never modify variables
3. Object oriented :- revolves around objects :, code + data = Objects

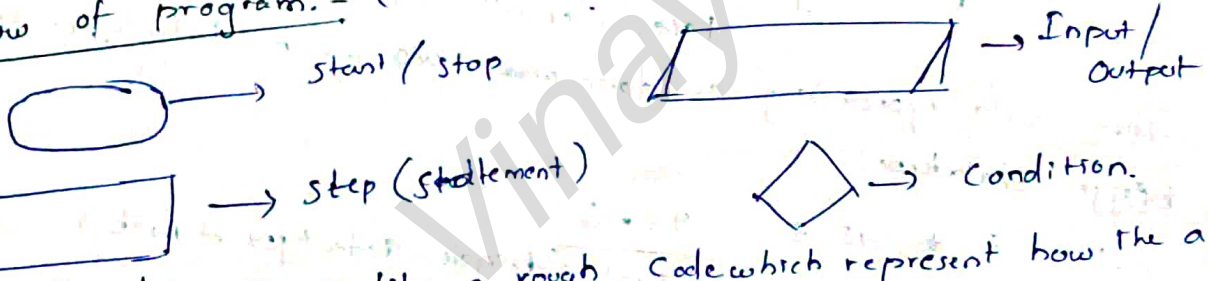
* Static Language :-	Dynamic Language :-
* type checking at Compile time.	type checking at runtime.
* Error shows at Compile time.	Errors might not show the program run.
* Control over the program	need not to declare of datatypes of variable. So, it saves the time.

memory management :

* Variable stores address of value in the stack memory.
* The value is stored in heap memory



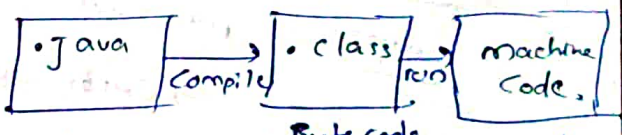
Flow of program :- (flow chart)



pseudo code :- It is like a rough code which represent how the algorithm of a program works. which don't need Syntax

Introduction to Java

* Java is not pure object oriented.
* Java is platform independence.
  - because Java gives the Byte code when we compile.
  - That Byte code can run in any JVM of different OS.



where as in other languages the languages directly convert into mc.
The JVM is platform dependent but, Java (Byte code) not dependent.

JDK → [Java Development kit] → JRE + Development Tools

* Compiler - javac
* JRE
* Development tools

(Java Runtime Environment) → JVM + Library classes

- Bases libraries
- JVM
- User Interface tools

Java virtual machine → JIT (Just in Time)

- loading — Read .class file. — create in heap
- linking — verifies the .class file — allocate memory
- Initialization → static variable Assign
  Contains stack & heap memory.

* there are wrapper class for primitive datatypes. → Integer, ...
* primitive data types are those data types which are not breakable
* Comments → // for single line. → select ctrl + shift + ? (shortcut)
  /* */ → multiple lines.

int a = 239_000_000
sprint(a) → given → 239000000 (ignores the _)
Here a is Identifier, 239_000_000 → literals.

For input :-
nextInt() → for Integer
next() → for 1st string of the line. → 1st word of line
nextLine() → for total line.
nextFnt() → for float

float → 8 digits

typecasting: Converting one type of data to another type of data.

* int n = int (69.3).
* int n = 'A'; print(n) → given 65 (ASCII value)
* for float → float k = 5.6f (bcz it taken double default for decimal values)

for long → long k = 5763563789L (bcz compile. take default as int)

The auto typecasting is done of two value datatypes to bigger datatype:

Ex:- 5 * 15.5 = 97.5
int * float = float (bigger data type of both)

Conditional statement:- if, else, else if.

Loops:- for, while do while.

== → checks whether the reference object (or value is same (or not)
.equals() → check whether they are equal (or not.

* Switch-case: Use break to all cases (not for default)

Switch-Case vs if-Else, which is efficient?

```
Switch (day)
  Case 1:
    x x x x
    break
```
} is also written as Case 1 → x x x x; (no need of break)

```
Case 1:
Case 2:
Case 3:
    Sop("x");
Case 4:
    Sop("y");
```
} → Case 1, 2, 3 → Sop('x');
Case 4 → Sop('y');
default is choice.

Nested Switch case:

```
Switch(var){
Case 1:
    Sop("x");
    break;
Case 2:
    Switch(m){
        Case L:
            Sop(y);
            break;
        Case M:
            Sop("z");
            break;
    }
    break;
}
```

```
Switch(var){
case 1 → Sop("x")
case 2 → {
    switch(m){
        case L → Sop("y");
        case M → Sop("z");
    }
}
```

(Same & Code of both)

Break: used to break the loop, not the conditional statement (if-else)

Return: return to the End of program.

Float must append an F or f to the Constant.

Strings are not implemented as arrays of characters in Java, because Java implements strings as objects.

$-15 \% 2 = -1$
$-15 \% -2 = 1$ } v.i

The Java provides various wrapper sub classes. They are:
Byte, Integer, Double, Short, Long, Float.

byte = 256 when we Convert Int to byte.

Sop(b) → 43 (which is 256 + 43)

```
int a = 299;
byte b = byte(a);
```

operands will atomatically promoted into int from byte.

Ex:- byte b = 100;
byte c = 100 * 2; → Invalid (which operand is want to converted int)

for taking character as input is:

```
char ch = sc.next().charAt(0);
```

Functions / methods (in Java):

A method is a block of code which only runs which it called

It is used for many time, which coded once.

Semicolon (;) most used at End of functions (;)

block Scope:-

If a variable is declared outside of block, we can use the variable in block.

If we declared variable in side of block, we cannot use the variable outside of block.

Shadowing:- Overlapping of variables it overrides the value.

Ex:-
```
int a = 10;
{
  a = 50;
}
sop(a) → 50
```

```
public class V {
(imp) static int n = 50;
psvm (String[] args) {
  n = 100;
}
```
Out:- 100.

Array is data structure use to store a collection of data.

```
int [] arr = new int [5]
```
- int [] → represent of Array data type.
- arr → Object
- size is important

array's arr defined in stack, actually memory allocation happen in heap which the objects are stored.

Internally in Java, memory allocation totally depends on Jvm, whether the data is stored in continuous or not (of addresses) because object are stored in heap memory which is not continuous.

default array stored with '0' from declaration.

Same as Array

String.

```
String [] arr = new String [4];
```

default String stores null values.

primitives (int, char etc) are stored in Stack, where all other objects are stored in heap memory.

Array is mutable, we can change objects.

2D array:-

```
int [][] arr = new int [size][ ]
```
- row → mandatory to give size
- col. → choices to give the size.

Jagged array:- The array contain different no. of element in different rows.

Arraylists:-

It is part of collection framework & presents in java.util package. It provides us with dynamic arrays in Java. It is slower than standard arrays. We can add any no. of elements (independent of size). Internally size is fixed but, when the data is filled by some amount it will create double the size of arraylist Intially. & Old Arraylists are deleted.

```
Arraylist <Integer> lis = new Arraylist<>();
```
↓ wrapper class method:- set(), add(), remove() etc

Arrays are mutable & strings are immutable.

Linear Search:-

It is a Seqential Search, by comparing the target with all elements in array.

Best case:- O(1) → constant (where the 1st element of array is target)

worst case:- O(n) → n is no. of elements in Array.

Binary Search:-

Searching the target by breaking the array at the mid. & Searches in pieces of array.

Best case:- O(1)

worst Case:- O(log n) (Efficient than the linear Search)

For linear Search the array is need not to be sort but for Binary Search, the array should be sorted.

Bubble Sort:- It is simple algorithm that works by repeatedly by swapping the adjacent elements. It is also known as Sinking Sort (or) Exchange Sort, inplace Sorting algorithms.

Ex:-

5	2	1	7

i = 0 { 2 5 1 7 ; 2 1 5 7 ; 2 1 5 7 } → The largest element is

i = 1 { 1 2 5 7 } → So, we need not check again of largest element

So, we use n − i − 1.
↳ no. of last element

Space Complexity :- O(1) // Constant.

Time Complexity :- O(n) → if array is Sorted
O(N²) → if array is in decreasing order.

Stable Sorting Algorithm :-
Here after Sorting the order is maintained

Ex:- ① 10 20 30 10 20

↓

(See the color)

10 10 20 20 30

Unstable Sorting Algorithm :-
order is not maintained & different

for Ex ① :- 10 10 20 20 30

unstable Sorting Algorithm is

Selection Sort :- Select an Element & put it on its Correct Index. we can Sort by Ascending. (Comparing with max Element (or) min Element)

Time Complexity :-
worst case :→ O(n²), best case → O(n²)

It is not stable Sorting Algorithm (unstable)

Better performs on Small lists.

Cyclic Sort :-
It is used when the Elements are in range of (0, N) (or) (1, N)

if the Elements are from 1 to N then we will use. index = value - 1

if the Elements are from 0 to N then we will use. index = value.

Worst case :- O(n)

String & StringBuilders:-

Stack memory: The reference variable stored in Stack.

heap memory: The value or object of the variable is stored.

String:- Strings are the Sequences of character.

String is class (So, It is starts with Capital letter)

String pool is a Seperate memory structure in heap memory which help to the duplicate value (not like to recreated same value or object twice).

Ex:- String v = "vinay"; } Here both the reference variable are reference of 1 object.
String t = "vinay";

which mates our program more optimed.

v ⟶ vinay (no recreation of object)
k ⟶

Strings are immutable for Security purpose.

Because if the 1 or more reference variables refer to same object. If one variable is change then all the variable should change. So, that's why String are immutable.

* == given true. when the two variables refers to same object (Comparing).

we can create different object of same value by creating the ↑ two different classes.

String a = new String("vinay"); which is looks like { a ⟶ vinay, b ⟶ vinay
String b = new String("vinay");

which here a == b given false. (these classes are stored out of pool)

* .equals( ) is used to check the value not the reference.

.charAt( ) is used for Index value.

we can use System.out.printf(" "); for printing values.

%d ⟶ int; %f ⟶ float %c ⟶ char. %s ⟶ string

%.5f ⟶ 5 digits after the decimal point.

%.o ⟶ octal %.e ⟶ Exponential floating point numbers

Operator Overloading:-

Sout('a'+'b') ⟶ ab

Sout('a'+13) ⟶ 110

Sout(char('a'+19) ⟶ s

Sout("vinay"+19)
↓
vinay19

Character is represented in Single quotes.
Strings represented in double quotes.
"+" operators is used for primitives & when anyone of these value is a string.

String are immutable. So, we cannot change the value. By using StringBuilder we can change the value.
StringBuilder is Seperate class
StringBuilder sb = new StringBuilder();

Methods:-
- toCharArray() → String to Array.
- length() → length of string.
- toLowerCase() → to convert lowercase.
- indexOf → gives Index value.
- strip → Spaces are removed

Recursion:-
The function stored in stack memory until the Execution of function Completed.

Time Complexity:- Functions that gives us the relationship about how to time will grow as the input grows..
Time Complexity is the time taken to run the algorithm It is dependent on the no. of statement (steps) not on machine.
Ex:- linear Search (O(N)) & Binary Search (O(log N))
log N is more better than N.
we will ignore the Constants & less dominating terms.

O(1) < O(log(N)) < O(N) < O(N log N) < O(n²).

The time Complexity is done in 3 Notation:

① Big-oh Notation:- (O()):
The Big-oh is upper bound which the worst case of the time Complexity.

$$\lim_{n \to \infty} \frac{F(n)}{g(n)} < \infty \qquad F(n) = O(g(n))$$

Here $F(n) \longrightarrow$ The Function of algorithm

$g(n) \longrightarrow$ the worst time Complexity of algorithm

Ex:- $6N^3 + 12N \longrightarrow F(n)$

$N^3 \longrightarrow g(n)$

$$\lim_{n \to \infty} \frac{6N^3 + 12N}{N^3} = 6 + \frac{12}{N^2} \quad \left(\frac{1}{\infty} = 0\right)$$

$= 6$ (Constant) which is less than ∞.

② Big Omega :- It is Opposite of Big-oh. which is lower bound. It is gives the best i.e. least Complexity.

$$\lim_{n \to \infty} \frac{F(n)}{g(n)} > 0$$

③ Theta notation:- It is Combination of both big Omega & Big-oh which given the both best & worst cases.

$$0 < \lim_{n \to \infty} \frac{F(n)}{g(n)} < \infty$$

If N is the time Complexity

$1 < \log N < N < N \log N < 2^N$

value of Big-oh — values of Big-omega.

The Big-oh ($O()$) is used over the Big-Ω

Little-oh Notation:- It is a loose upper bound.

$F = O(g)$ & $F < g$ (Strictly Slower)

Big oh:- $F \leq g$ (actual order of Complexity)

Little-oh:- $F < g$ (Rough value of Complexity)

which means degree of $g(n)$ is more than $F(n)$

$$\lim_{n \to \infty} \frac{F(N)}{g(N)} = 0$$

Little omega :-

It is loosely lower bound.

Big Ω :- $f \geq g$

Small ω :- $f > g$ (f is strictly greater than g)

$$\lim_{N \to \infty} \frac{F(n)}{g(N)} = \infty$$

The degree of $F(N)$ is greater than $g(N)$.

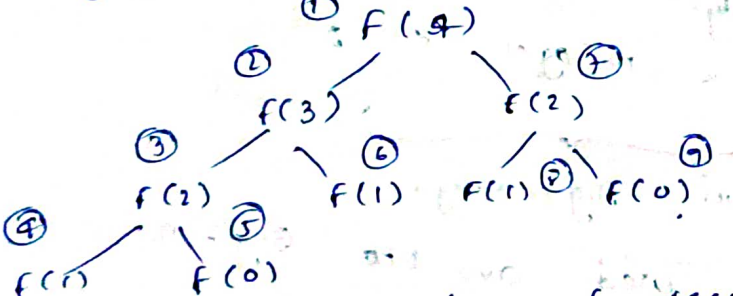
Space Complexity :- It is total space taken by the algorithm with respect to the input size. Space Complexity includes both Auxiliary space & Space used for input.

Auxiliary Space :- Extra space used by algorithm.

Insertion Sort & Heap Sort uses $O(1)$ auxiliary space. where other $O(n)$ almost.

Recursive Algorithms :- In case of Recursive Algorithm, the Space Complexity is not constant. Because for each function calls it stores in stack.

The Space Complexity of Recursive Algorithm is height of tree (Recursive tree)



Even f(2) is executed at ③ again f(2) at 7 we executed. They are not interlinked.

There are two types of recursions :-

① Linear ② Divide & Conquer.

1. Divide & Conquer :-

Form :-

$$T(x) = a_1 T(b_1 x + \xi_1(x)) + a_2 \xi T(b_2 x + \xi_2(x)) + \cdots + a_k T(b_k x + \xi_k(x)) + \underbrace{g(x)}_{\text{Constant}}$$

we can find the time Complexity of the Divide & Conquer recursion problem by Akra bazzi formula.

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(x)}{x^{p+1}}\, dx\right)$$

$$\sum_{i=1}^{k} a_i b_i^p = 1 \longrightarrow (1)$$

Ex:- Merge Sort Time Complexity

$$T(N) = 2T\left(\frac{N}{2}\right) + (N-1)$$

$a = 2, \; b = 1/2 \Rightarrow g(x) = (N-1)$

$p = 1$ by $2 \times \left(\frac{1}{2}\right)^p \rightarrow \left(\sum a_i b_i^p = 1\right)$ from Equation (1)

$$O\left(x^1 + x^1 \int_1^x \frac{x-1}{x^2}\, dx\right) = x + x \int_1^x \left(\frac{1}{x} - \frac{1}{x^2}\right) dx$$

$$= x + x\left[\log(x) + \frac{1}{x}\right]_1^x = x + x\left(\log x + \frac{1}{x} - 1\right)$$

$$= x + x\log x + 1 - x = O(\log x + 1)$$

ignore the 1 the time Complexity is $O(\log x)$

If $p <$ powers of $(g(x))$ then time Complexity is $O(g(x))$

Because If p is less than to powers of $(g(x))$ then It is Ignored because of Small value than the $g(x)$.

(2) Linear Recurrance :-

form:-

$$f(x) = \sum_{i=1}^{n} a_i f(x-i)$$

n is fixed, Order of recurrance.

Ex:- fibnoacci number.

The complexity of fibonacci number is $O(1.6180)^n$ (golden ratio).

Homogeneous Equation in which $g(x) = 0$ (const

Ex:- fibonacci no:-

we know that

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

$f(n) = x$ ⟶ assume

$$x^n = x^{n-1} + x^{n-2} \Rightarrow x^n + x^{n-1} + x^{n-2} = 0$$

$$x^2 - x - 1 = 0 \longrightarrow \text{roots}$$

$$x = \frac{1 \pm \sqrt{5}}{2}$$

$$f(n) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$ c_1, c_2 are Constants

we got two roots It means we already know 2 values

$f(0) = 0, \quad f(1) = 1$

$c_1 + c_2 = 0$

$c_1 = -c_2$ ⟶ (1)

$c_1 \left(\frac{1+\sqrt{5}}{2}\right) + c_2 \left(\frac{1-\sqrt{5}}{2}\right) = 1$

from (1)

$c_1 \left(\frac{1+\sqrt{5}}{2}\right) - c_1 \left(\frac{1-\sqrt{5}}{2}\right) = 1$

$c_1 = \frac{1}{\sqrt{5}}, \quad c_2 = \frac{-1}{\sqrt{5}}$

$$f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n$$

formula of n^{th} fibo no: (Ignoring Const)

$$f(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n \sim (1.6180)^n$$

Non - Homogeneous equation (which g(x) is existed)

* If g(x) is exponential guess of same type.

Ex:- $g(x) = 2^n + 3^n$ then $f(x) = a2^n + b(3)^n$

* If g(x) is polynomial guess of same degree.

Ex:- ① $g(x) = x^2 - 1$ then $ax^2 + bx + c$, ② $g(x) = x$ then $ax + b$

if g(x) is combination of poly & expo then.

Ex:- $g(x) = 2^n + n$ then $a(2^n) + (bx + c)$

then Step-1:- $g(x) = 0$ & solve for roots.

Step-2:- guess the f(x) by above three points

Step-3:- General step : $f(x) = c_1(g(x)\ \text{root}1) + c_2(g(x)\ \text{root}2)$

Step-2 which is known as particular solution.

Step-3 :- c_1 (step 1) + c_2 (step 2) (we can get c_2 value in step2)

Bit. manipulation (Bitwise Operator) :-

1. AND :- (&) only 1 & 1 = 1 Remaining all are 0

any number & 1 = same number.

2. OR :- (|) only 0 | 0 = 0 Remaining all are 1

3. XOR (^) :- Exclusive OR (if & only if)

$0 \wedge 0 = 0$ (same gives 0)

$1 \wedge 0 = 1$ (different gives 1)

$a \wedge 1 = \bar{a}$ (a's compliment)

$a \wedge 0 = a$

$a \wedge a = 0$

4. Complement (~) :- $\sim a = \bar{a}$

Octal number :- (0 to 7)

↳ 0, 1, 2, 3, 4, 5, 6, 7 (octal)

Hexadecimal :- 0 - 9 & A - F

A - 10, B - 11 ... F - 15

Shift Operators :-

left shift Operator (<<) :- It shifts all the bit towards the left side (one by one) & add Zero at right

10 << 1 ⟶ 1010 << 1 ⟶ 10100

when the left shift operator is used the value is doubled.

a << 1 = 2a a << b - a × 2^b

Right shift Operator :- (>>) :- It is opposite of left shift operator which the bit moves toward right side by adding Zero's at left.

10 >> 1 ⟶ 1010 >> 1 ⟶ 0101

when the Right Shift Operator is used the value is halfed

a >> 1 ⟶ a/2, a >> b ⟶ $a/2^b$

Q :- Find no is Even (or) odd using Bitwise Operators.

a & 1 == 1 ⟶ odd Else, Even

Because the least Significant bit is 1 it is odd.

Q :- Set the i^{th} bit.

Set means converting the bit into 1 from whether 0 to 1 (or) 1 to 1

Reset means Converting the bit into 0

The MSB (Most Significant bit) should be 0 for +ve number
1 for -ve number

for Negative number ⟶ 2's Complement

1's compliment of number & add 1 to get the -ve number of given number.

For no. of digits in base b. $\longrightarrow int\left(\log_b^n\right) + 1$

b → base, n → number

$\log_b^n \longrightarrow \frac{\log n}{\log b}$

The sum of n^{th} row, in pascal triangle is 2^{n-1}.

To check number n is power of 2 → n & (n-1) = 0
(& operator)

To find the a power b (a^b).

```
int ans = 1;
while (b > 0) {
    if ((power & 1) == 1) {
        ans = ans * a;
    }
    a = a * a;
    b = b >> 1;
}
print (ans);
```

For no of set bits in number

```
while (n > 0) {
    count++;
    n = (n & (n-1));
}
return count;
```

For reversing of array :-

```
for ( int i = 0; i < (arr.length-1)/2; i++ ) {
    int temp = a[i];
    a[i] = a[arr.length - i - 1];
    a[arr.length-i-1] = temp;
}
```

To flip the number (0, 1) which mean.
(or)
Invert (or) complicate.

0 → Flip → 1
1 → flip → 0

By doing XOR with 1
0 ^ 1 = 1
1 ^ 1 = 0

LCM (3, 7) = 21

LCM (2, 4) = 4

Lowest Common multiplier :- The highest number which can divide the both numbers.

Ex:- LCM (10, 90) = 90

LCM (of two prime numbers) = multiplication of two numbers

Highest Common factor (HCF) (or) Greatest Common factor (GCF):-

$$LCM(a, b) = \frac{a * b}{HCF(a, b)} \longrightarrow (1)$$

HCF (2, 18) = 2 (least no. which can divide the both numbers)

(12, 24) → 2.

HCF (two prime numbers) = 1.

Program :

```
gcd (int a, int b) {
    if (a == 0) {
        return b;
    } else {
        return gcd (b % a, a);
    }
}
```

for LCM by using formula (1) (above)

The Efficient code for prime numbers in range is Sieve of Eratosthenes

Recursion:-

function calling another function which the body & definition is Common. (parameters may different)

main function is the first function which enters the stack & last function which come out.

* when the function is completed, it is removed from stack.
* without base condition it will run infinity. time (stack overflow).
* Recursion helps to break the bigger problem into simple way
* we can convert the iteration into recursion & vice versa.

we know that 2 types recurrence Relation

1. Linear Recurrence 2. Divide & Conquer Recurrance.

1. Linear Recurrence is inefficient which function calls same function over & over. So, we use dynamic programming

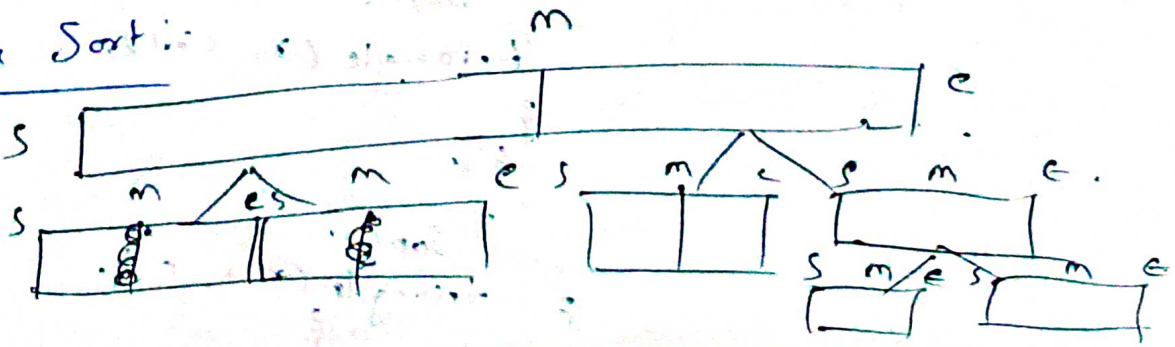
Return function:- The datatype of Return function call & the data type of the variable which is returning should be same.

Sum of digit using Recursion:-

```
F(N) {
if (N % 10 == 0) {
        return N
} else {
return F(N) = F(N / 10) + (N % 10);
}
}
```

Same we can find No. of digits. Reverse number.

Merge Sort:-



Time Complexity :- $N \log N$

merge Sort is just splitting & Sort 2 then joining (or) merging the array.

Quick Sort :- (unstable Sort)

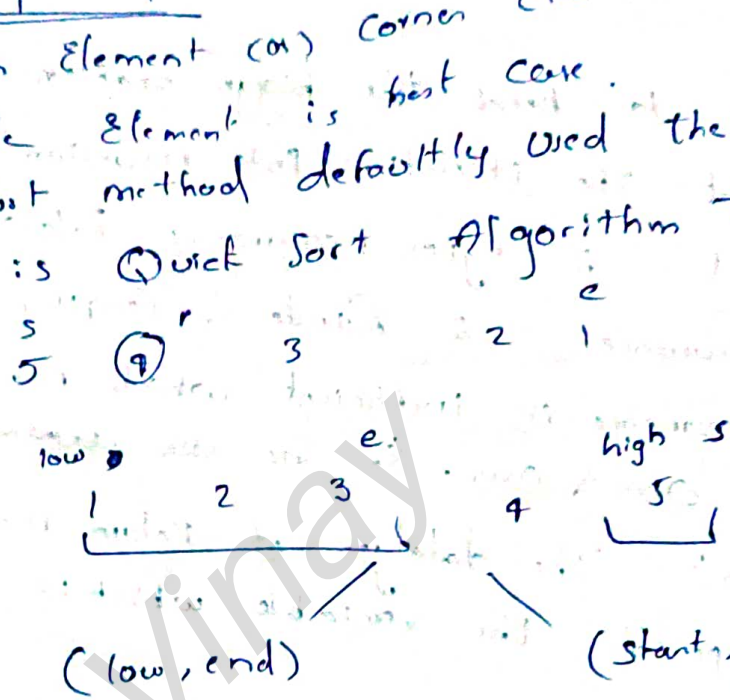
Choosing the pivot (Choosing of Elements)

How to pick pivot :-

Random Element (or) corner element (or) middle Element

middle Element is best case.

Java Sort method defaultly used the dual pivot which is Quick Sort Algorithm to Sort.



Hybrid Sorting Algorithm: (Tim Sort) :-

Merge Sort + Insertion Sort

works well with partially sorted data.

Pattern :-

Ex :-

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

```
static void triangle(int r, int c) {
    if (r == 0) {
        return ;
    }
    if (c < r) {
        Sop("* ");
        triangle(r, c+1);
    } else {
        Sop();
        triangle(r-1, c);
    }
}
```

for removing Certain Element from strings

Ex:- removing 'a' from string vinaya & returning the string.

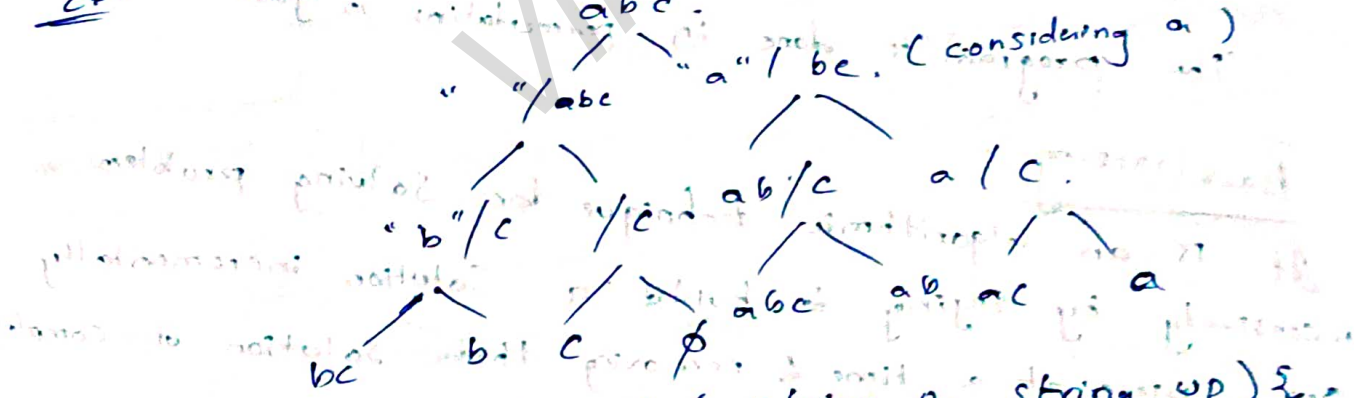
```
static void v(String p, String up){
    if(up.isEmpty()){
        sop(p);
        return
    }
    char ch = up.charAt(0);
    if(ch == 'a'){
        v(p, up.substring(1));
    }
    else{
        v(p+ch, up.substring(1));
    }
}
```

p = "" up = "vinaya".

⟶ (p = "vinay") output.

Subsets :-

Ex:- [abc] ⟶ subsets are :- [a], [b], [c], [ab], [bc], ∅, [ac], [abc].




```
static void sub(String p, String up){
    if(up.isEmpty()){
        sop(p);
        return;
    }
    char ch = up.charAt(0);
    sub(p+ch, up.substring(1));
    sub(p, up.substring(1));
}
```

Returning in ArrayList form:-

```
static ArrayList<String> Subseq(String p, String up) {
    if (up.isEmpty()) {
        ArrayList<String> a = new ArrayList<>();
        a.add(p);
        return a;
    }
    char ch = up.charAt(0);
    ArrayList<String> left = Subseq(p + ch, up.substring(1));
    ArrayList<String> right = Subseq(p, up.substring(1));
    left.addAll(right);
    return left;
}
```

Permutations:-

Permutations of vin = [vin ivn niv nvi vni nvi]

which n! (3!) = 6

The program is done in permutations in Java folder.

Backtracking:-

It is an algorithmic technique for solving problems recursively by trying to build a solution incrementally one piece at a time & removing those solution of completion of steps.

Maze problems are done by Backtracking & N-queens also.