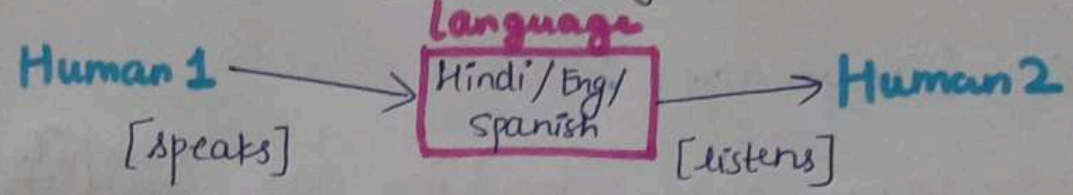
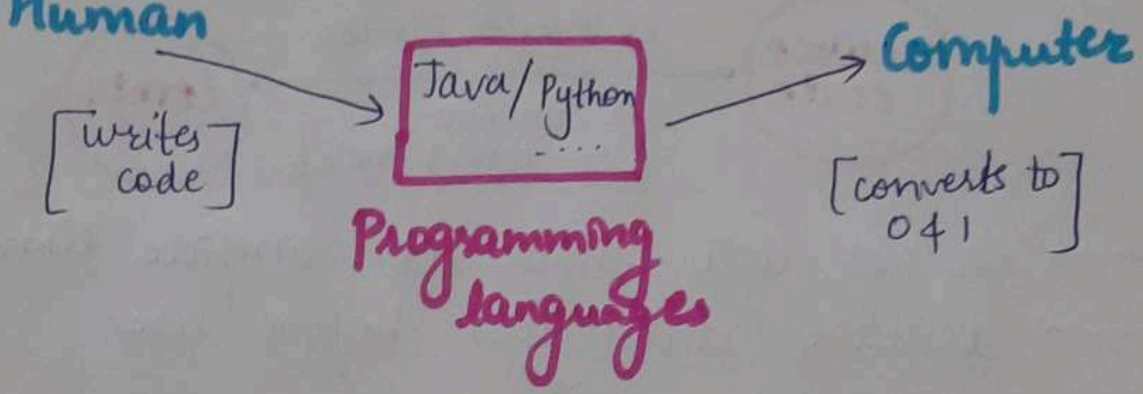
2|8|21

Introduction to Programming Language

- Computers at very minute level only understands zeros & one's (0's & 1's)
- What is programming language?





- Types of Programming Languages :

Procedural :

→ series of well-structured steps & procedures to compose a program

→ contains a systematic order of statements functions and commands to complete a task.

Functional :

→ Writing a program only in pure functions i.e; never modify variables but only create new ones as an output

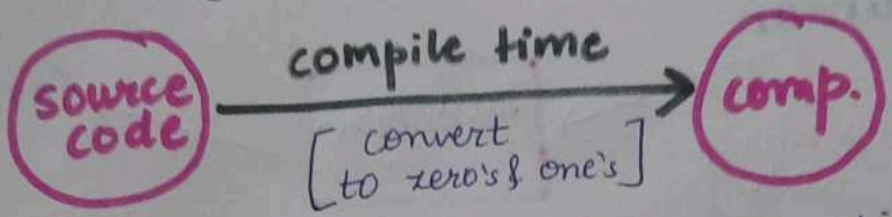
→ Used in a situation where we have to perform lots of different operations on the same set of data like ML.

Object Oriented:

→ Revolves around objects
→ code + data = objects
→ developed to make it easier to develop, debug, reuse & maintain.

• Static Languages:

→ Perform type checking at ~~runtime~~ compile time.



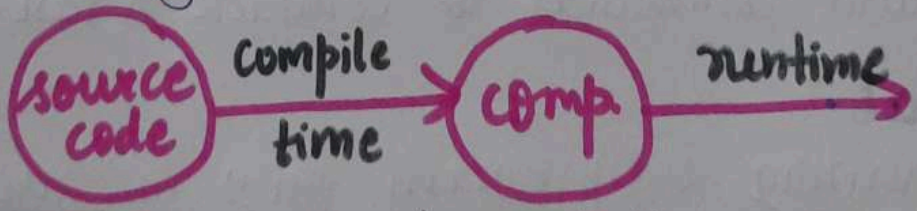
→ errors will show at compile time
→ declare datatypes before use

```
int a = 10
```

→ More control over the program.

• Dynamic Languages:

→ Performs type checking at runtime



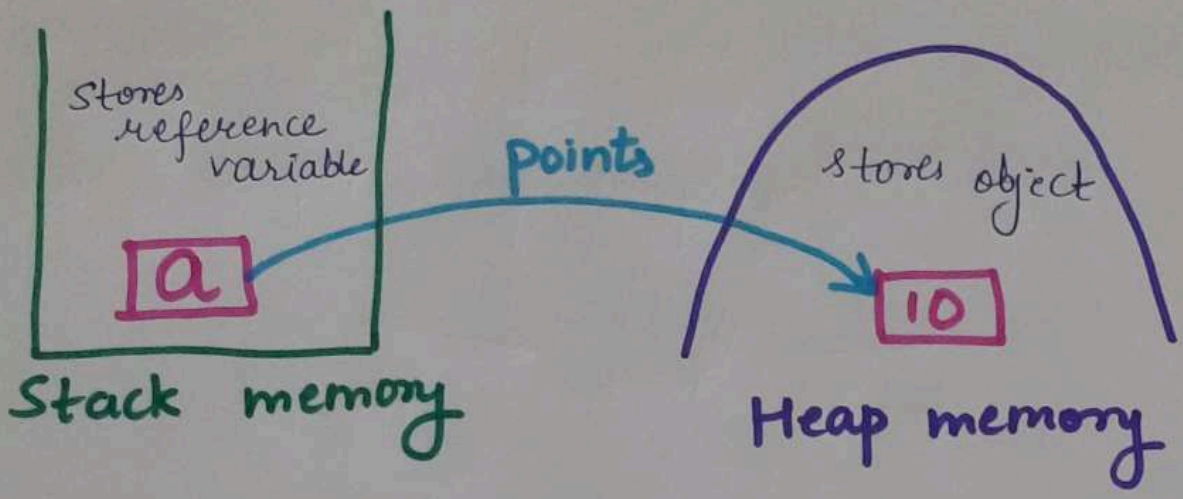
→ error might not show till programs run
→ no need to declare datatype of variables

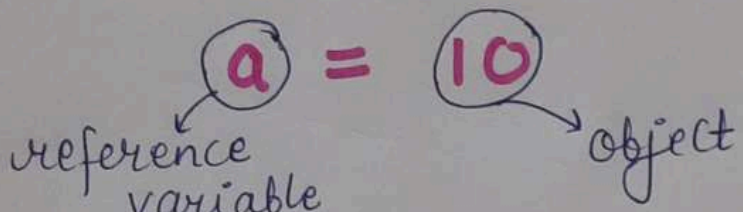
```
a = 10
```

[language by itself figures out data type]

→ Saves time in writing code but might give error at runtime.

→ Memory Management :

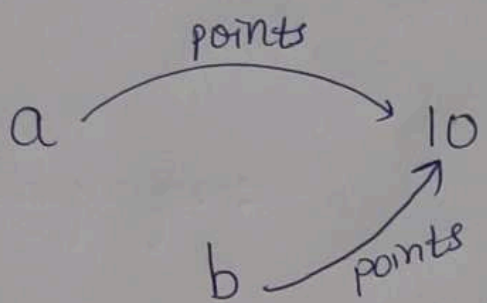




Now suppose,

a = 10

b = a



→ More than one reference variable can point towards one object.

→ If any of the reference variable changes the object then it is changed for all reference variable ~~poi~~ that points towards same object.

Now intially,

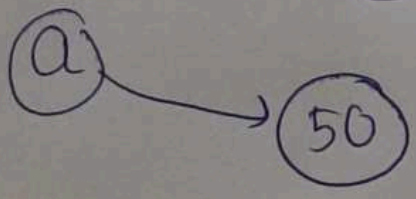
a = 10

Object with no reference variab

10 [Removed by garbage collector]

then,

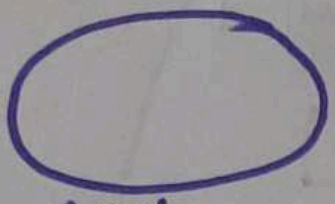
a = 50



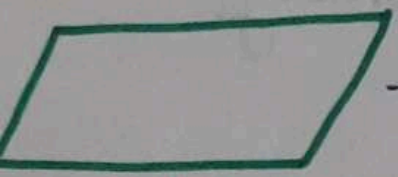
2/8/21

Flow of program

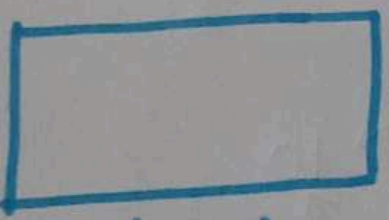
* Flow Chart Symbols :



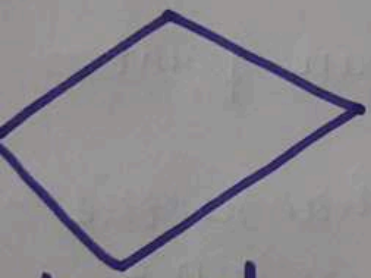
oval → Start/Stop → Represents start or end point of program.



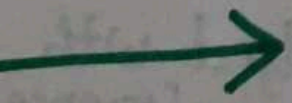
parallelogram → Input/Output → Represents the input & output



rectangle → Processing → Represents a process like addition, subtraction et

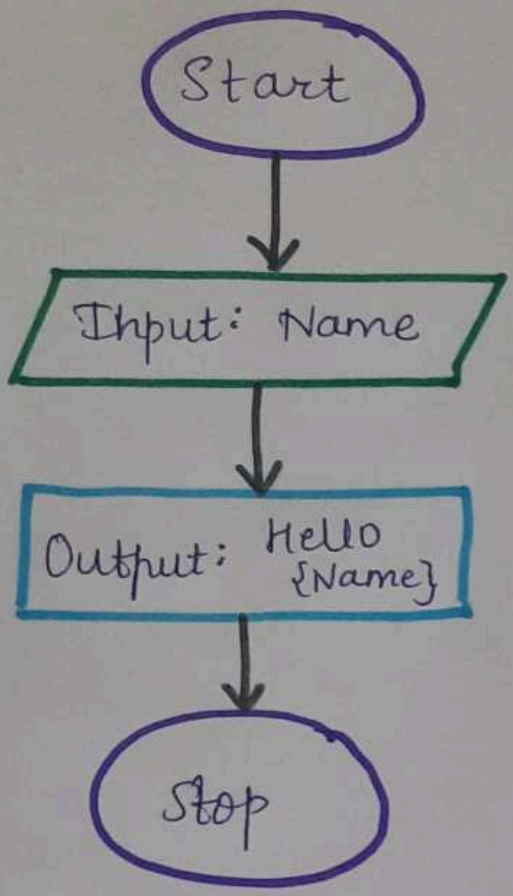


diamond → Condition → Represents for conditional statement



→ Flow direction of program → A line connector which shows what is the flow of program.

eg: Take a name & output Hello Name :



* Pseudo Code :

It is just a way to write steps which is human readable format. [It is not a code].

It is mainly meant for human reading not for machine reading.

eg: Take above example to take a name & output Hello name :

Step 1 : → Start

Step2 : → Take Input from user [name = Input ("enter na

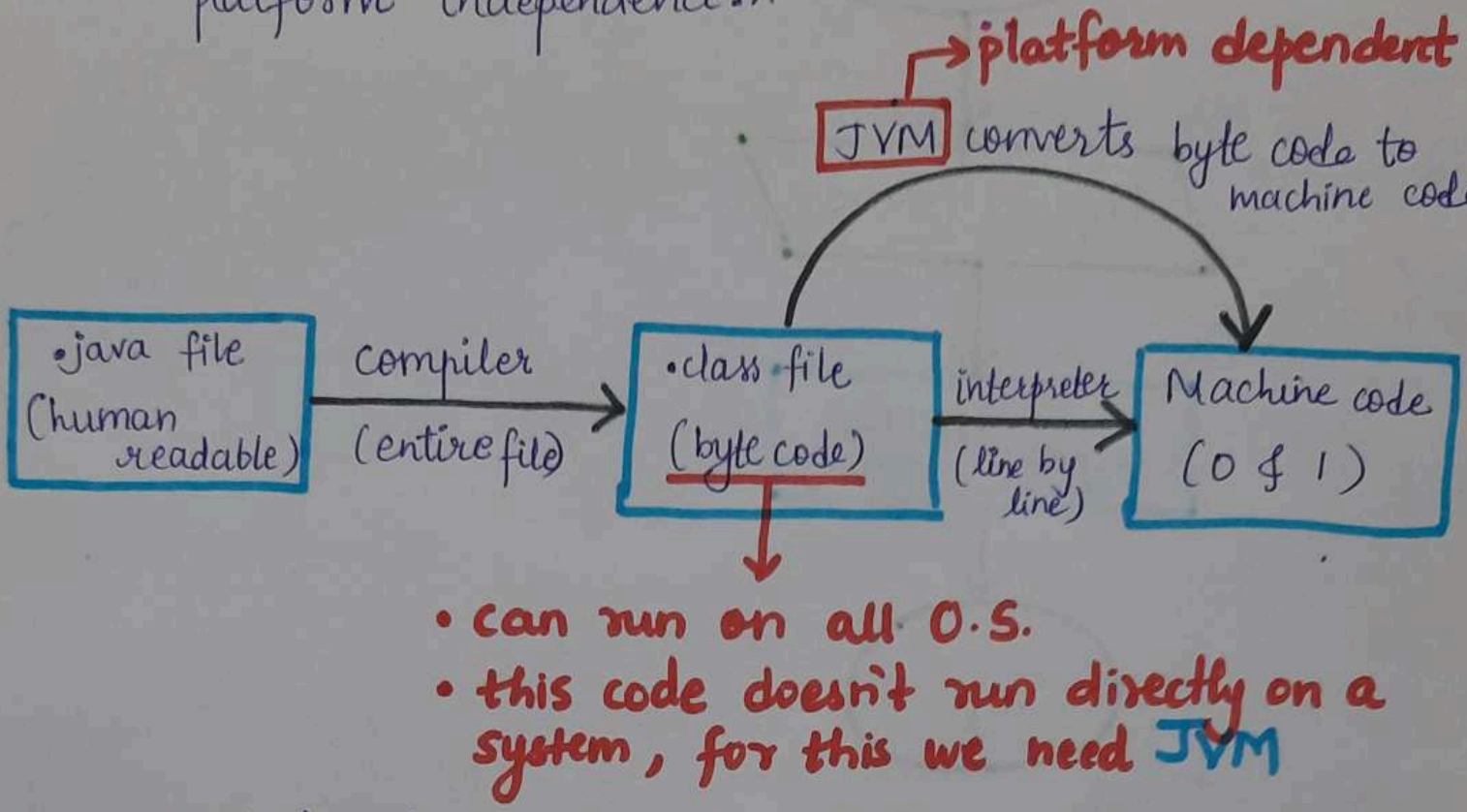
Step3 : → Hello {Name} [output]

Step4 : → Stop

3/8/21

Introduction to Java ♥

★ How Java code executes and more information about platform Independence...



- can run on all O.S.
- this code doesn't run directly on a system, for this we need JVM

★ Therefore, Java is platform independent ★

⇒ We can provide this byte code to any system means we can compile the java code on any system.

⇒ But JVM is platform dependent means for every O.S. the executable file that we get, it has step by step set of instruction dependent on platform.

✯ JDK vs JRE vs JVM vs JIT

JDK [Java Development Kit]

↳ provides environment to develop & run Java program

JRE [Java Runtime Environment]

↳ provides environment to only run the program

- → deployment technologies
- → user interface toolkit
- → integration libraries
- → base libraries

etc.

JVM [Java Virtual Machine]

JIT [Just-in-time]

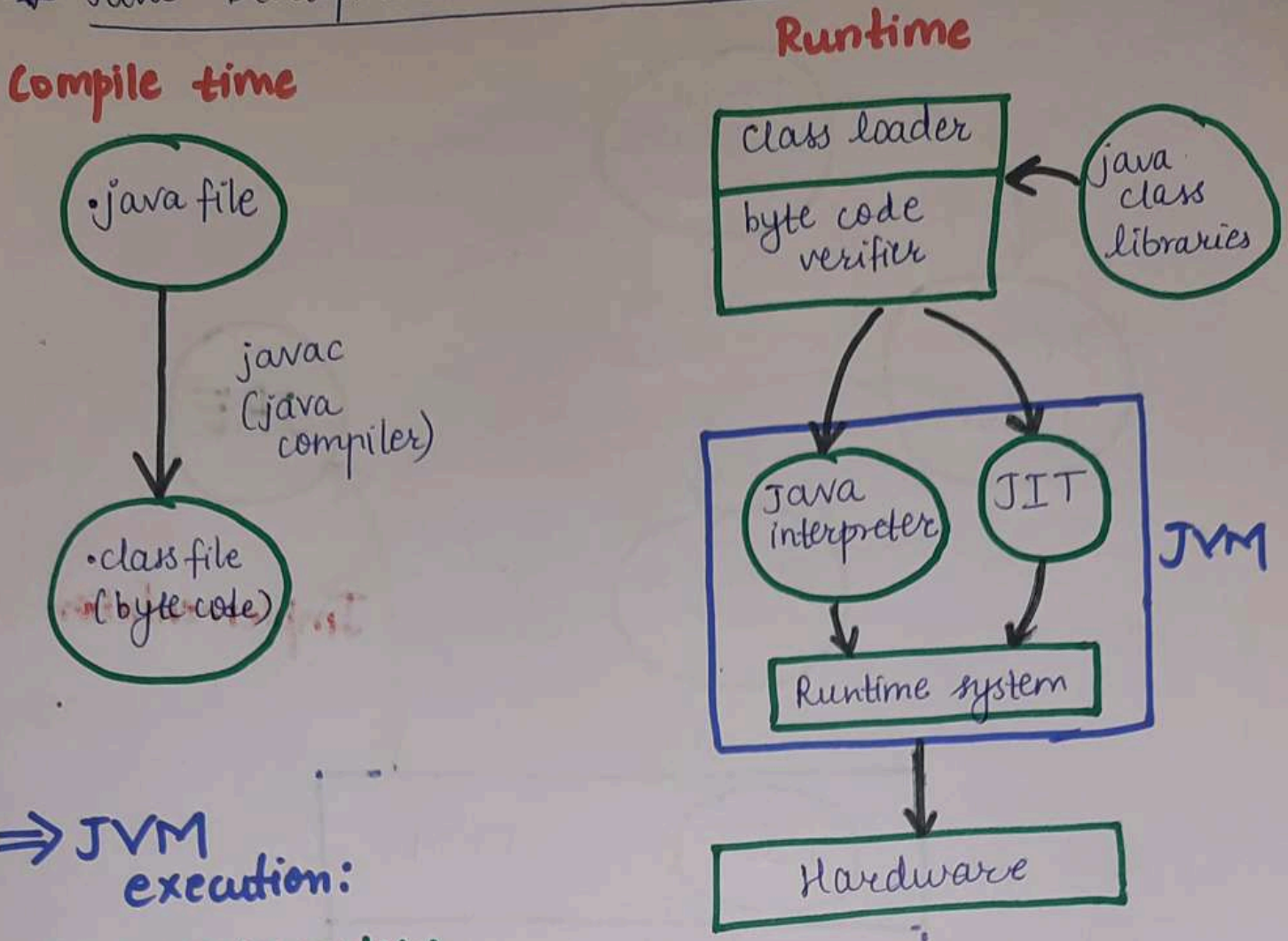
- → Java Interpreter
- → Garbage collector

etc.

- → development tools
- → javac → Java compiler
- → archiver → jar
- → docs generator ↳ javadoc
- → interpreter/loader

etc.

✱ Java Development and Runtime Environment



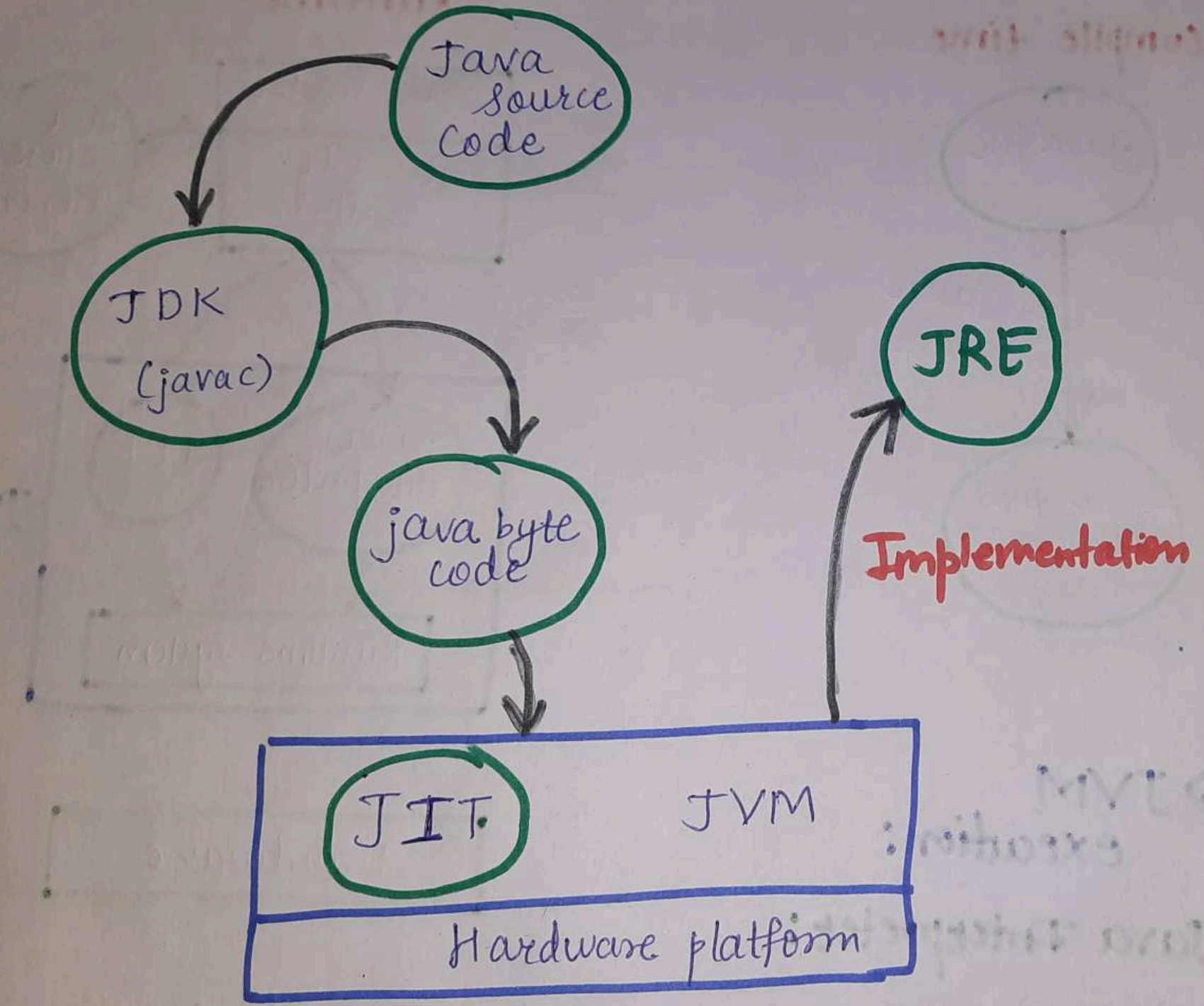
⇒ JVM execution:

- **Java Interpreter:**
  - → line by line execution
  - → when one method is called many times, it will interpret again & again
- **JIT:**
  - → methods that are repeated, JIT provides direct machine code so re-interpretation is not required
  - → makes execution faster
- **Garbage Collector**

✱ Class loader:

- **Loading**
  - → reads byte code file & generates binary data
  - → an object of this class is created in heap
- **Linking**
  - → JVM verifies .class file
  - → allocates memory for class variables & default values
  - → replace symbolic references from the type with direct references
- **Initialization**
  - → all static variable are assigned with their values defined in the code & static block.

Summary:


Java Source Code
JDK (javac)
java byte code
JRE
Implementation
JIT
JVM
Hardware platform

3/8/21

First Java Program - Input/Output, Debugging & Datatypes

File name: **Demo.java** Class Name: **Demo**

Its good practice to use initial character as capital (you can use small also)

public → this keyword means, it is used so that we can access the class from anywhere.

functions → Collection of code, that we can use again & again. Functions are also known as methods.

void → The void keyword specifies that a method should not have a return value.

String[] args → means an array of sequence of characters ("strings") that are passed to the main function.

array

- After compiling, .class file is always saved in current location where you are in.
- If you want to change the location, use **-d** (destinato option while compiling and specify the path.

javac -d <path> Demo.java

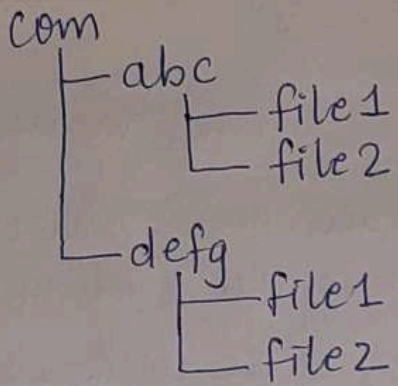
- **echo $PATH** → every command looks for this location before executing.

[environment variables]

- class name & file name should be same, but if we don't want to make class name as file name then it should not be public.

for eg → **class Divide**

- package com.abc OR package com.defg



- `System.out.println("Hello");` → this means print the output on Standard output stream (here, terminal)
  - `System` → class
  - `out` → var
  - `println` → fn/method

println → adds new line
print → does not add new line.

- `Scanner input = new Scanner(System.in);`
  - `Scanner` → class that allows us to take input
  - `new` → creating object
  - `System.in` → take input from standard input (here, keyboard)

Primitive → means any data type that cannot be broke further.
integer, character etc. are primitive datatype.

⇒
```
int rollno = 64;  → 4 bytes
char letter = 'r';
float marks = 98.67f;  → 4 bytes
double largeDecimalNumbers = 456789.12345;  → 8 bytes
long largeIntegers = 12345678l0L;  → 8 bytes
boolean check = true;
```

- String is written in double quotes whereas while specifying char we write it in single quotes.
- All decimal values that we use are by default of double datatype, therefore if we want to store in float we have to use "f", same for int & long.

```
float marks = 7.2f
(by default) double largeDecimalNumbers = 456789101.12345

int rollno = 64; (by default)
long LargeInteger = 1234567891011L;
```

- Integer → Wrapper class → provides additional functionalities
  - converts primitive datatype to object.
- Comment → the lines that we comment are ignored by Java and will not be executed.
  Comment in Java → //

→ `int a = 10` → literal

a → identifier

- Literals : Java literals are syntactic representations of boolean, character, numeric or string data.
  here, 10 is an integer literal.
- Identifiers : Identifiers are the names of variables, methods, classes, packages & interfaces.

- **int a = 234_000_000;**
  ↳ the value of a will be 234000000, underscore will be ignored.
- 564.12345678 $\xrightarrow{\text{rounds off}}$ 564.12345
  If we give float very big, than it rounds off the value which gives floating point error.

⟹ Type Casting & Type Conversion:

• Widening or Automatic Type Conversion:

→ Two datatypes are automatically converted.
→ This happens when we assign value of smaller datatype to bigger datatype & two datatype must be compatible.

byte → short → int → long → float → double

eg→ `int i = 100;` → 100
`long l = i;` → 100
`float f = l;` → 100.0

• Narrowing or Explicit Conversion:

→ This happens we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

double → float → long → int → short → byte

eg→ `double d = 100.04;` → 100.04
`long l = (long) d;` → 100
`int i = (int) l;` → 100

• Automatic Type Promotion in Expressions :

→ While evaluating expressions, the intermediate value may exceed the range of operands & hence the expression value will be promoted.

→ Some conditions of type promotion are:

1. Java automatically promotes each byte, short, char to int when evaluating an expression
2. Long, float or double the whole expression is promoted to long, ~~whole~~ float or double.

eg: After solving expression :

```
(f * b) + (i / c) - (d * s);
```

we get → float + int − double = double

converted to biggest one

• Explicit type casting in expressions :

→ If we want to store large value into small data type

eg:
```
byte b = 50;
b = (byte)(b * 2);
```
→ type casting int to byte.

• If-else syntax in Java

```
if (condition) {
  // block of code
} else {
  // block of code
}
```

• For loop syntax

```
for (statement1 ; statement2 ; statement3) {
  // code block
}
```

CONDITIONAL AND LOOPS

Condition:- It provide check for the statement.

1. If-else statement → Used to check the condition, it checks the Boolean condition True or False.
   Syntax :-

```
if (boolean expression True or false){
    //Body
} else{
    // Do this
}
```

Example:-

```
public class IfElse {
    public static void main(String[] args) {
        int salary = 25400;
        if (salary> 10000) {
            salary = salary + 2000;
        }
        else {
            salary = salary + 1000;
        }

        System.out.println(salary);

    }
}
```

Output :- 27400

2. Multiple if-else statement

→ It executes one condition from multiple statements.

Syntax :-

```
if (condition 1){
    // code to be executed if condition 1 is true
} else if (condition 2) {
    // code to be executed if condition 2 is true
} else if (condition 3){
    // code to be executed if condition 3 is true
} else {
    // code to be executed if all conditions are false
}
```

Example :-

```
public class MultipleIfElse {
    public static void main(String[] args) {
        int salary = 25400;
        if (salary<= 10000) {
            salary +=1000;
        }
        else if (salary <= 20000) {
            salary += 2000;
        }
        else {
            salary += 3000;
        }
        System.out.println(salary);

    }
}
```

Output :- 28400

Loop → Loops are used to iterate a part of program several times.

1. for loop :- It is generally used when we know how many times loop will iterate.

   Syntax :-

```
for (initialization; condition; increment/decrement){
    // body
}
```

Example:- print numbers from 1 to 5

```
public class forloop {
    public static void main(String[] args) {
        for (int num=1;num<=5;num+=1){
            System.out.println(num);
        }
    }
}
```

Output :- 1
2
3
4
5

Example 2 :- print numbers from 1 to n

```
import java.util.Scanner;
public class forloop {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        for (int num=1;num<=n;num+=1){
            System.out.print(num + " ");
        }
    }
}
```

Input : 6

Output :- 1 2 3 4 5 6

2. While Loop :- It is used when we don't know how many time the loop will iterate.

   Syntax :-

```
while (condition){
    // code to be executed
    // increment/decrement
}
```

   Example :-

```
public class whileloop {
     public static void main(String[] args) {
         int num = 1;
         while (num <=5){
             System.out.println(num);
             num += 1;
         }
     }
}
```

   Output :- 1
   2
   3
   4
   5

3. do while loop :- It is used when we want to execute our statement at least one time.
   → It is called exit control loop because it checks the condition after execution of statement.
   Syntax :-

```
do{
    // code to be executed
    // update statement -> increment/decrement
}while (condition);
```

Example :-

```
public class doWhileloop {
    public static void main(String[] args) {
       int n = 1;
       do{
           System.out.println(n);
           n++;
       } while(n<=5);
    }
}
```

Output :- 1
2
3
4
5

While Loop	Do while loop
→ used when no. of iteration is not fixed	→ used when we want to execute the statement at least ones
→ Entry controlled loop	→ Exit controlled loop
→ no semicolon required at the end of while (condition)	→ semicolon is required at the end of while (condition)

■ Program to find largest of three numbers.

"Take 3 integer input from keyboard, Find the largest numbers among them ".

Approach -1 :-

```
import java.util.Scanner;
public class LrgestOfThree {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int c = in.nextInt();

        int max = a;
        if(b>max){
            max = b;
        }
        if (c > max){
            max = c;
        }
        System.out.println(max);
    }
}
```

Approach – 2:-

```
import java.util.Scanner;
public class LrgestOfThree {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int c = in.nextInt();

        int max = 0;
        if(a > b){
            max = a;
        } else {
```

```
            max = b;
        }
        if (c > max){
            max = c;
        }
        System.out.println(max);
    }
}
```

Approach 3 :-

Using Math.max :- Math is a class present in java.lang package and max is a function present in it which takes two number as an argument and return maximum out of them.

```
import java.util.Scanner;
public class LrgestOfThree {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int c = in.nextInt();

        int max = Math.max(c,Math.max(a,b));
        System.out.println(max);
    }
}
```

Input :- 3 6 5

Output :- 6

■ Alphabet case check

" Take an input character from keyboard and check weather it is Upper case alphabet or lower case alphabet"

```
import java.util.Scanner;
public class AlphabetCaseCheck {
    public static void main(String[] args) {
        Scanner in = new Scanner (System.in);
        char ch = in.next().trim().charAt(0);
        if (ch > 'a' && ch <= 'z'){
            System.out.println("Lowercase");
        }
        else {
            System.out.println("Uppercase");
        }
    }
}
```

Input :- a

Output :- Lowercase

Input :- Z

Output :- Uppercase

■ Fibonacci Numbers :- a series of numbers in which each number (*Fibonacci number*) is the sum of the two preceding numbers.

Ex :- 0,1,1,2,3,5,8,13...

→ **Find the nth Fibonacci number.**

" Given three input a, b, n a is the starting number of Fibonacci series and b is the next number after a, n is an number to find the nth Fibonacci number"

```
import java.util.Scanner;
public class FibonacciNumbers{
    public static void main(String[] args) {
        Scanner in = new Scanner (System.in);
        int n = in.nextInt();
        int a = in.nextInt();
        int b = in.nextInt();
        int count = 2;

        while(count <= n){
            int temp = b;
            b = b+a;
            a = temp;
            count++;
        }
        System.out.println(b);
    }
}
```

Input :- 0 1 7

Output :- 8.

- **Counting occurrence :-**

  " Input two numbers, find that hoe many times second number digit is present in first number"

  Ex :- first number = 14458

  Second number = 4

  Output = 2, because 4 is present 2 times in first number.

```
import java.util.Scanner;

public class CountingOccurence {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int count = 0;
        int Fn = in.nextInt();
        int Sn = in.nextInt();
        while (Fn>0){
            int rem = Fn % 10;
            if (rem == Sn){
                count++;
            }
            Fn = Fn/10;
        }
        System.out.println(count);
    }
}
```

Input :- 45535 5

Output :- 3

■ Reverse a number

" A number I input from the keyboard and Show the output as Reverse of that number "

Example :- Input :- 12345

Output :- 54321

```java
import java.util.Scanner;
public class ReverseANumber {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num = in.nextInt();
        int ans = 0;
        while(num > 0){
            int rem = num % 10;
            num /= 10;
            ans = ans * 10 + rem;
        }
        System.out.println(ans);
    }
}
```

Input :- 458792

Output :- 297854

Calculator Program

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // Take input from user till user does not press X or x
        int ans = 0;
        while (true) {
            // take the operator as input
            System.out.print("Enter the operator: ");
            char op = in.next().trim().charAt(0);

            if (op == '+' || op == '-
' || op == '*' || op == '/' || op == '%') {
                // input two numbers
                System.out.print("Enter two numbers: ");
                int num1 = in.nextInt();
                int num2 = in.nextInt();

                if (op == '+') {
                    ans = num1 + num2;
                }
                if (op == '-') {
                    ans = num1 - num2;
                }
                if (op == '*') {
                    ans = num1 * num2;
                }
                if (op == '/') {
                    if (num2 != 0) {
                        ans = num1 / num2;
                    }
                }
                if (op == '%') {
                    ans = num1 % num2;
                }
            } else if (op == 'x' || op == 'X') {
                break;
            } else {
                System.out.println("Invalid operation!!");
            }
            System.out.println(ans);
        }
    }
}
```

Input and output:-

```
Enter the operator: +
Enter two numbers: 86  94
180
Enter the operator: -
Enter two numbers: 75
12
63
Enter the operator: *
Enter two numbers: 12   3
36
Enter the operator: /
Enter two numbers: 70    5
14
Enter the operator: x
```

6/8/21 If-else conditions
Loops → while & for & do-while

7/8/21 Switch Statements + Nested case in Java.

• Switch Statements :

```
switch (expression) {
    case one:
        // code block
        break;
    case two:
        // code block
        break;
    default:
        // code block
}
```

break; → terminate the sequence

default: → default will execute when none of above does.
→ if default is not at end put break after it.

→ if break is not used then it will continue with other cases.

→ duplicate cases not allowed.

eg:
```
case one:
    // code block
    break;
case one:
    // code block
    break;
```

✗ not allowed.

New Syntax:

```
switch (expression) {
        case one -> //do this ;
        case two -> // do this;
        default -> // do this;
}
```

* `x.equals("word")` → here equals only checks value not reference.

`x == "word"` → here it checks reference

- **Nested Switch Case:**

```
switch (expression) {
        case one:
            // code block
            break;
        case two:
            switch (expression) {
                case one:
                    // code block
                    break;
                case two:
                    // code block
                    break;
                default;
                    // code block
            }
            break;
        default:
            // code block
}
```

8/8/21 Functions/Methods in JAVA

Functions/Methods (in java):

- A method is a block of code which only runs when it is called.
- To reuse code: define the code once, & use it many times.

Syntax:

```
public class Main {
    static void myMethod() {
        // code
    }
}
```

this method myMethod() does not have a return value.

name of method

```
public class Main {
    access_modifier return_type method() {
        // code
        return statement;
    }
}
```

f" ends here

- method() → calling the function.

  method → name of function

- **return_type :-**

A return statement causes the program control to transfer back to the caller of a method.
A return type may be premitive type like int, char, or void type (returns nothing).

⇒ there are a few important things to understand about returning the values:

- The type of data returned by a method must be compatible with the return type specified by the method.
  eg: if return type of some method is boolean, we cannot return an integer.
- The variable recieving the value returned by a method must also be compatible with the return type specified for the method.

⇒ Pass by value:

eg 1:

```
main() {
    name = "a";
    greet(name);
}
static void greet(naam) {
    print(naam)
}
```

object/value

name → @
naam → @

Creating copy of value of name

i.e., passing value of the reference.

eg 2:

```
psvm() {
    name = "a";
    change(name);
    print(name);
}
change(naam){
    naam = "b";
}
```

Creating copy

name → @
naam → @

name → @
naam → b

Since it is created inside fn it will not change original one.

not changing original object, just creating new object.

✱ Points to be noted:

1→• primitive data type like int, short, char, byte etc.
↳ just pass value

2→• object & reference:
↳ passing value of reference variable.

eg-1:

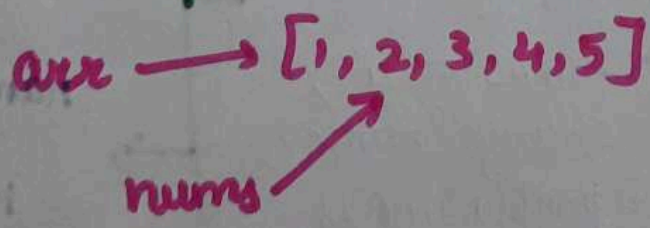
```
psvm() {
    a = 10;
    b = 20;
    swap(a, b);
}
swap(num1, num2) {
    temp = num1;
    num1 = num2;
    num2 = temp;
}
```

a → 10
b → 20
] but not here

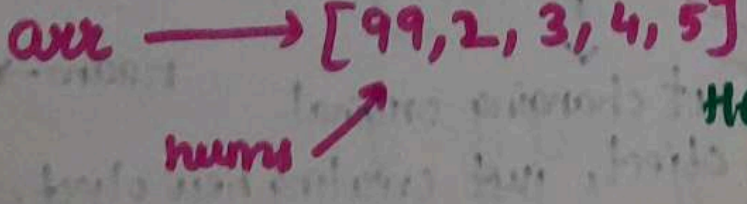
temp → 10
num1 → 20
num2 → 10
] at fn scope level they are swapped.

Here, they just passes the value....

eg-2:



nums[0] = 99 [now, the value of 0th position in nums will change which also changes value of arr[0]]



Here, passing value of reference variable

✱ Scopes:

- **function scope:**
  variables declared inside a method / function scope (means inside method) can't be accessed outside the method.

eg:-

```
psvm () {
}
all () {
    int x;
}
```

✗ ← can't be accessed outside

- **block scope:**

```
psvm ( ) {
    int a = 10;
    int b = 20;
    {
        int a = 5;   ✗
        a = 100;     ✓
        int c = 20;
    }
    c = 10;          ✗
    int c = 15;      ✓
    a = 50;          ✓
}
```

variables initialized outside the block can be updated inside the box.

variables initialized inside the block cannot be updated outside the box but can be reinitialized outside the block.

variables like "a" here, is declared outside the block, updated inside the block and can also be updated outside the block.

- **loop scope:**
  variables declared inside loop are having loop scope

⇒ Shadowing:

Shadowing in Java is the practice of using variables in overlapping scopes with the same name where the variable in low-level scope overrides the variable of high-level scope. Here the variable at high-level scope is shadowed by low-level scope variable.

eg:-
```
public class Shadowing {
    static int x = 90;
    psvm ( ) {.
        System.out.println (x);      // 90
        x = 50;                      // here high-level scope is shadowed by low-level scope
        System.out.println (x);      // 50
    }
}
```

⇒ Variable Arguments:

. Variable Arguments is used to take a variable number of arguments. A method that takes a variable number of arguments is a varargs method.

Syntax:

```
static void fun (int ...a) {
    // method body
}
```

Here, parameters would be array of type int [ ]

⇒ Method/ Function Overloading:

Function Overloading happens when two functions have same name.

eg → 1)
```
fun ( ) {
    //code
}

fun ( ) {
    //code
}
```
× function overloading

2)
```
fun ( int a ) {
    //code
}

fun ( int a, int b ) {
    //code
}
```
This is allowed having different arguments with same method name.

⇒ At compile time, it decides which fn to run.

⇒ Armstrong number:

Suppose there is number → 153

$153 \longrightarrow (1)^3+(5)^3+(3)^3 = 1+125+27$

$= 153$

10/8/21 Introduction to Arrays & Arraylist in Java

Why do we need Arrays?

⇒ It was simple when we had to store just five integer numbers and now let's assume we have to store 5000 integer numbers. Is it possible to use 5000 variable? NO

To handle these situations, in almost all programming language we have a concept called **Array**.

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

⇒ Syntax of an Array:

```
datatype [ ] variable_name = new datatype[size];
```

eg: we want to store roll numbers:

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

store 5 roll numbers

OR

```
int[ ] rollnos = {51, 82, 13, 15, 16};
```

represent the type of data stored in array.

All the type of data in array should be same!

⇒ Internal working of array:

```
int[ ] rollnos;   //declaration of array
```

↳ rollnos are getting defined in stack

```
rollnos = new int[5];   //initialisation
```

↳ actual memory allocation happens here

Here, object is being created in heap memory.

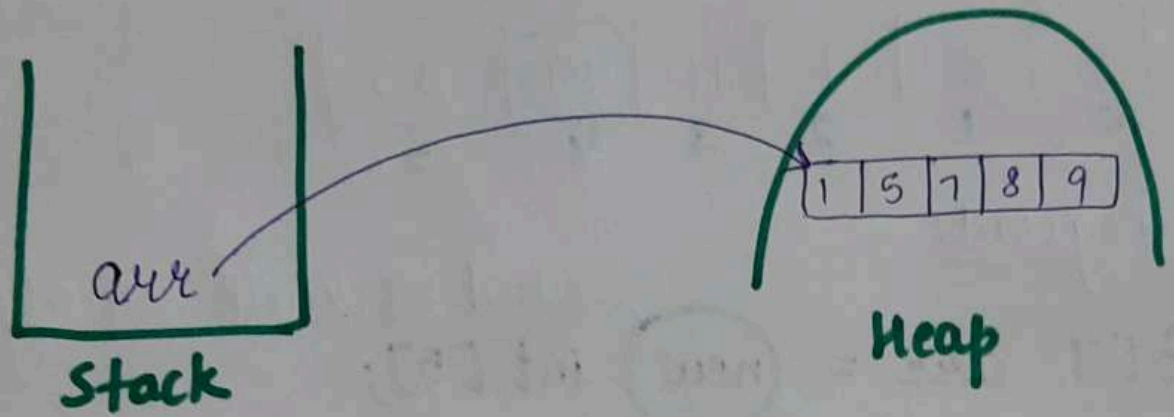
declaration of array → compile time

initialisation → run time

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

- `int []` : datatype
- `arr` : ref var
- `new int[5]` : creating object in heap memory.

⇒ This above concept is known as Dynamic memory allocation which means at runtime OR execution time memory is allocated.



⇒ Internal Representation of Array :

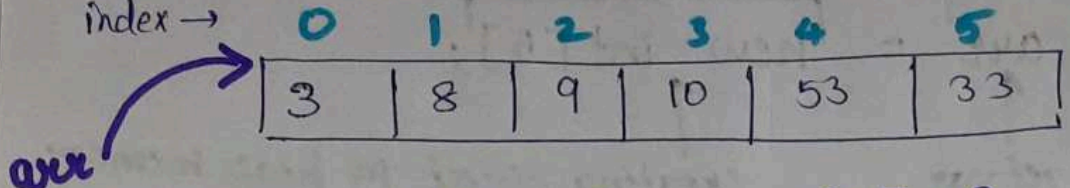
- Internally in Java, memory allocation totally depends on JVM whether it be continuous or not !

Reason 1 : Objects are stored in heap memory.

Reason 2 : In JLS (Java Language Specification) it is mentioned that heap objects are not continuous

Reason 3 : Dynamic memory allocation. Hence, array objects in Java may not be continous (depends on JVM)

⇒ Index of an array:

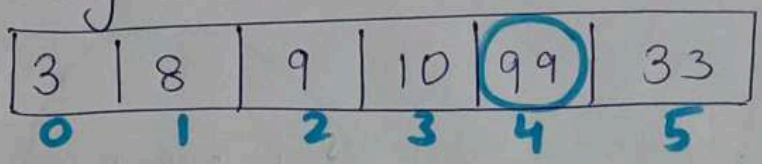


arr[0] = 3
arr[1] = 8
arr[2] = 9
arr[3] = 10
arr[4] = 53
arr[5] = 33

Suppose to change the value of certain index:

arr[4] = 99

New array will be:



⇒ new keyword:

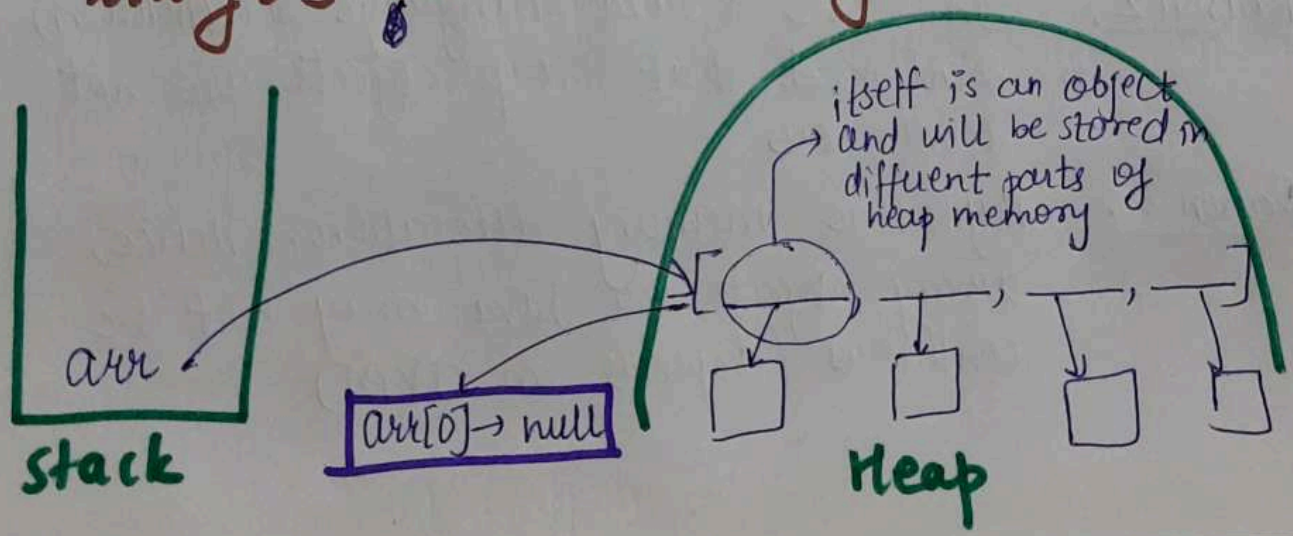
int[] arr = new int[5];

new → used to create an object

it will create an object in heap memory of array size 5.

⇒ If we don't provide values in the array, internally by default it stores [0, 0, 0, 0, 0] for above size of array.

String[] arr = new String[4];



* primitives (int, char etc) are stored in stack.
* All other objects are stored in heap memory.

⟹ Arrays.toString(array) → internally uses for loop and gives the output in proper format.

* In an array, since we can change the objects, hence they are <u>mutable</u>.
* strings are immutable.

⟹ <u>2 D Array:</u>

3 (columns)

1	2	3
4	5	6
7	8	9

3 (rows)

⟹ `int[][] arr = new int[size][];`

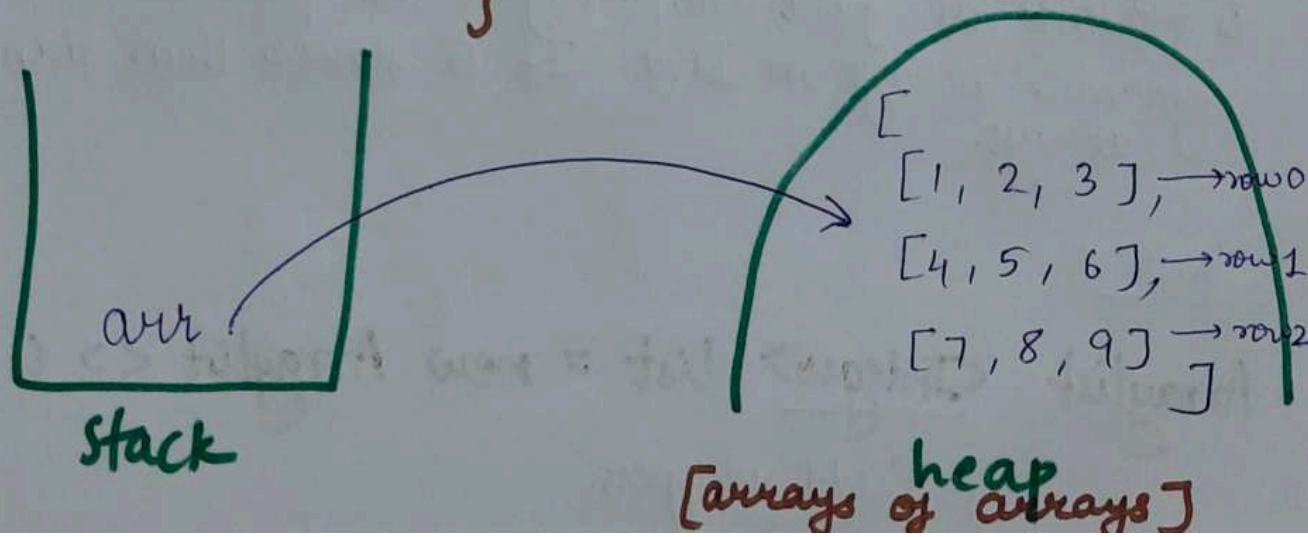
row ↓ column ↓

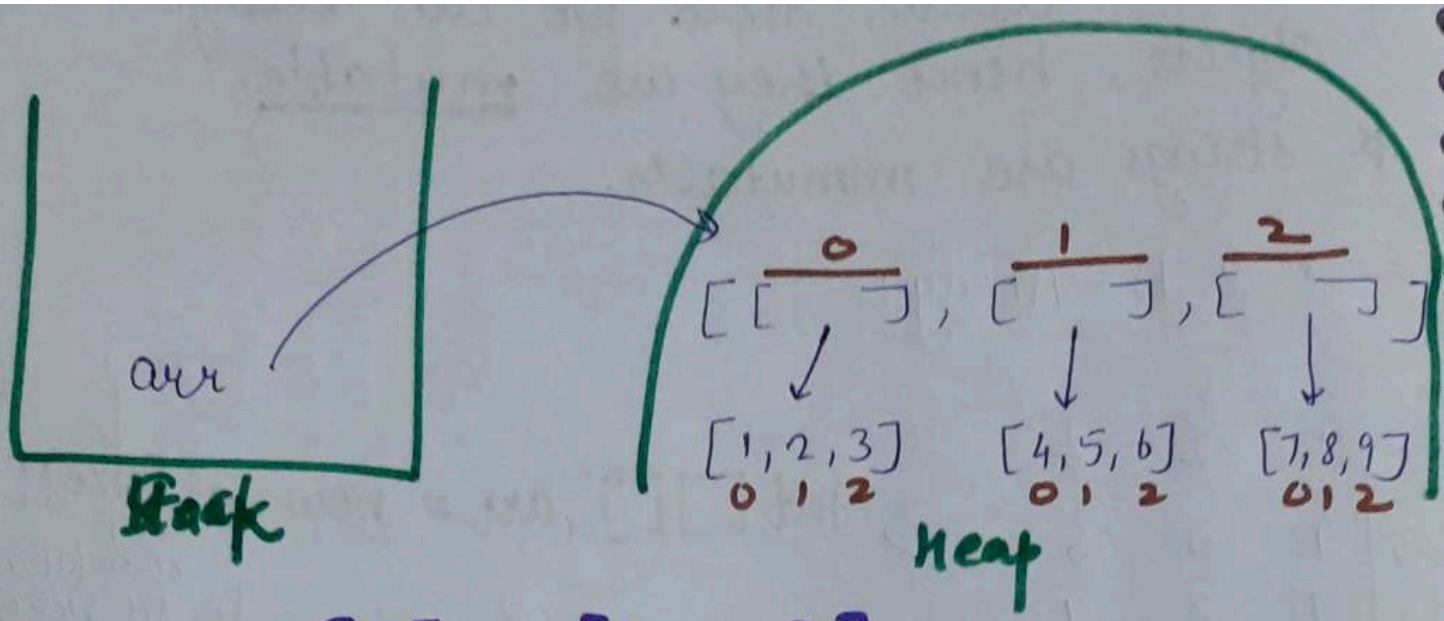
size → mandatory to give size of row

[] → not mandatory

OR

```
int[][] arr = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
              }
```





arr[0] = [1, 2, 3]
arr[0][2] = 3

⇒ Arraylists:

Arraylist is a part of collection framework and is present in java.util.package. It provides us with dynamic arrays in Java. It is slower than standard arrays.

Syntax:

```
Arraylist <Integer> list = new Arraylist <> ( );
```

↳ add wrappers.

⇒ Internal Working of Arraylist :

- size is fixed internally
- Suppose arraylist gets filled by some amount
  a) It will make an arraylist of say double the size of arraylist initially.
  b) Old elements are copied in the new arraylist.
  c) Old ones are deleted.

13/8/21

Linear Search in Java

Searching: It is a process of finding a given value position in a list of values.

Linear / Sequential Search:

- It is basic & simple search algorithm.
- In sequential search, we compare the target value with all the other elements given in the list.

eg:- arr = [18 (0), 12 (1), 19 (2), 77 (3), 29 (4), 50 (5)] (unsorted array)

target = 77 start

In above example, the target value is compared with all the elements in array in sequential / linear way.

Time Complexity:

→ **Best Case:** O(1) → constant

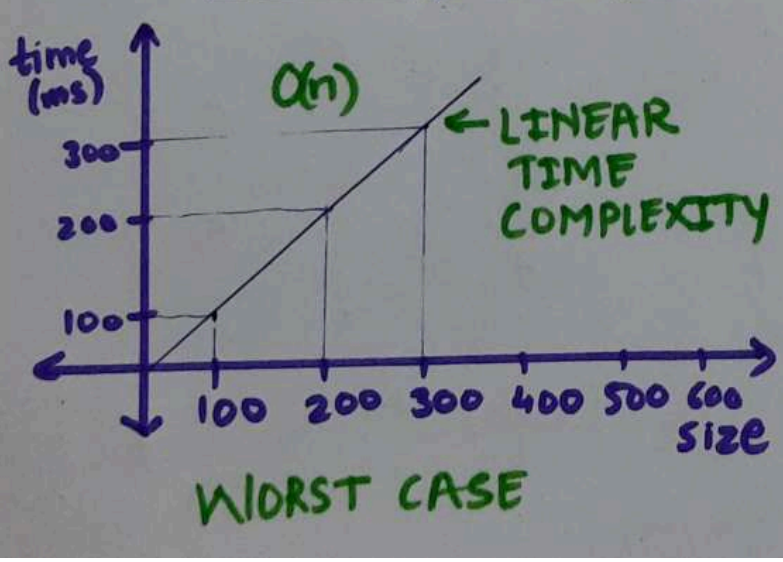
⇒ How many checks will the loop make in best case i.e; the element will be found at 0th index i.e; only one comparison will be made for best case.

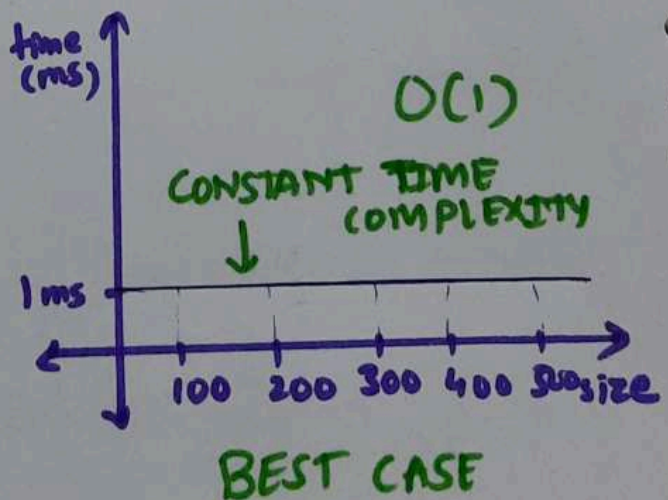
→ **Worst Case:** O(n)

⇒ Worst case, here it will go through every element and then it says element not found.

⇒

size of array	No. of comparisons	time (ms)
100	100	100 ms
200	200	200 ms
n	n	





15/8/21

Binary Search

Algorithm:

sorted array

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

ascending order

target (36)

1. Find the middle element
2. Check :
   if target > middle ⇒ search in right
   else ⇒ search in left
3. if target == middle ⇒ we found element.

Example:

In the above array,

target = 36

arr = [2(0), 4(1), 6(2), 9(3), 11(4), 12(5), 14(6), 20(7), 36(8), 48(9)]

1st → find middle element

mid = $\frac{start + end}{2}$ = $\frac{0+9}{2}$ = 4 [the element at index 4 is the middle element]

2nd → let's check:

Is target > middle ⇒ 36 > 11 ⇒ yes! ⇒ Check in right side

Now, arr = [2, 4, 6, 9, 11, 12(5), 14(6), 20(7), 36(8), 48(9)]

mid = $\frac{5+9}{2}$ = 7 [the element at index 7 is the middle element]

Let's check:

Is target > middle ⇒ 36 > 20 ⇒ yes! ⇒ check in right side

i.e., arr = [2, 4, 6, 9, 11, 12, 14, 20, 36(8), 48(9)]

mid = $\frac{8+9}{2}$ = 8 [the element at index 8 is the middle element]

Here, target == middle ⇒ 36 == 36 ⇒ we found Element at index 8

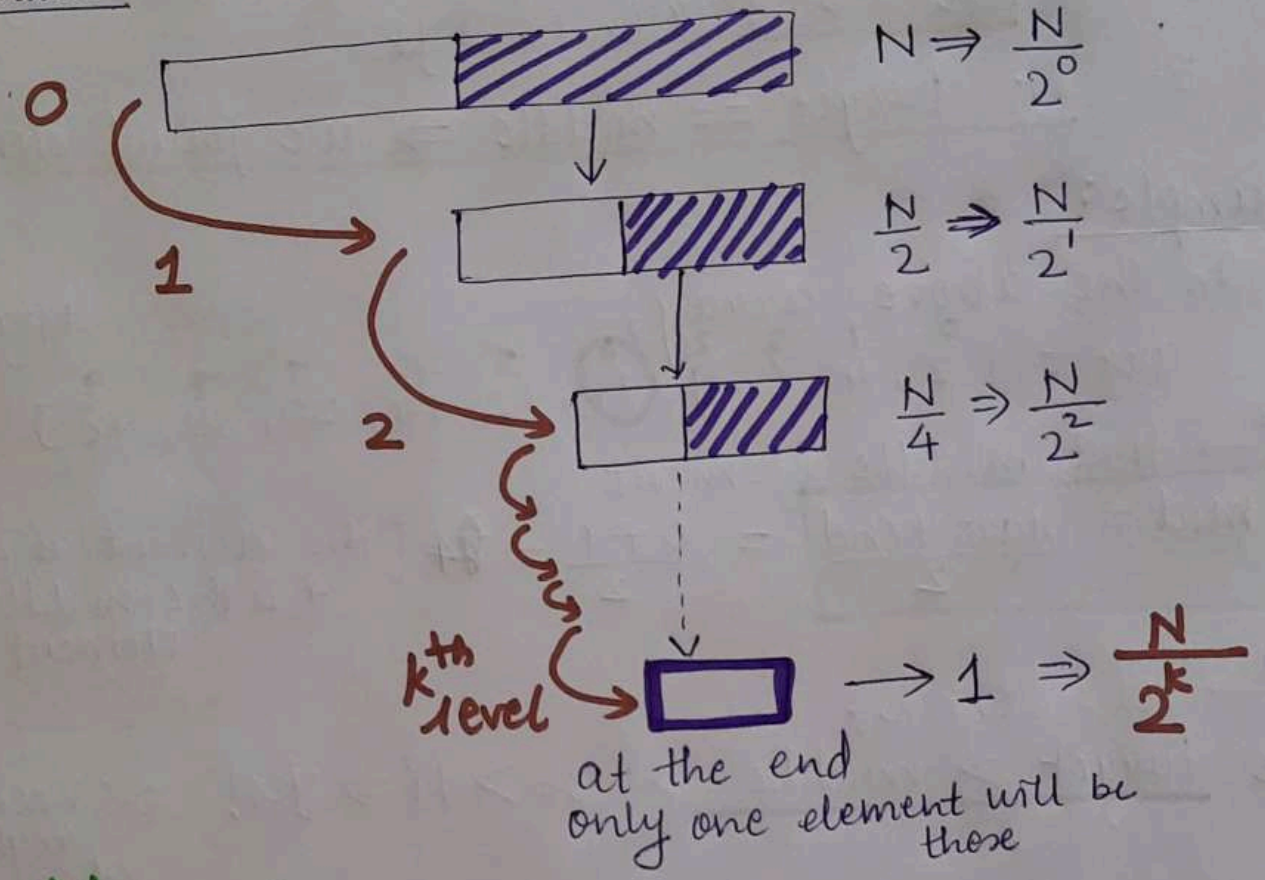
Here, the space in which we are searching is getting divided into two spaces.

⇒ Time Complexity:

Best case: $O(1)$

Worst Case: $O(\log n)$

Explanation: Find the maximum number of comparisons



$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$\log(N) = \log(2^k)$$

$$\log(N) = k \log 2$$

$$k = \frac{\log N}{\log 2}$$

$$\boxed{k = \log_2 N}$$

k → total number of comparison in worst case

N → size of array

⇒ Order agnostic Binary Search

Let's say if we don't know that the array is sorted in ascending or descending order.

arr = [90, 75, 18, 12, 6, 4, 3, 1]

target = 75

Here, target > middle ⇒ search in left

~~for descending order~~

Here, start > end] → descending order.

90 > 1

when start < end → Ascending order

Cyclic Sort :-

When given numbers from range 1 to N → use cyclic sort

Eg:

0	1	2	3	4
3	5	2	1	4

Q. What is cyclic sort and how it works ?

3, 5, 2, 1, 4 → { Here the no.'s are jumbled. but are from 1 to 5 }

let say N = 5

$ When the array is sorted in that case all the numbers are going to be at their correct ~~index~~ indices $

0	1	2	3	4
1	2	3	4	5

so, after Sorting = 1, 2, 3, 4, 5 - { Here after sorting index will become value - 1 }

Index = value - 1 - { using this we gonna sort the array }

Why ?

Ans Because index starts from 0

CHEK - SWAP - MOVE

Q. might be :- you have given the array find the missing & number ?

or you've given the no. from 1 to n find the duplicate no?

* worst case

Eg: 3, 5, 2, 1, 4 (0 1 2 3 4)

(1) - § Check is 3 at the correct index if not do 3-1 = 2 (index = value - 1) the swap with correct index §

∴ 2, 5, 3, 1, 4

- § after swapping we know that 3 is at correct position now but we do not know whether the other number that came at the position of 3 is correct or not so, check again §

(2) § If 2 is at correct position no it's not the again swap it, because it should be at index no 1. but it's at no. 0. so swap it with index 1 §

2, 5, 3, 1, 4

(3) 5, 2, 3, 1, 4

- § same theory as above §

(4) 4, 2, 3, 1, 5

- § same §

(5) 1, 2, 3, 4, 5

§ Now, we'll check if 1 is at the correct position, if it is then move forward. and so on § hence, it'll be our ans.

* We know that every unique item is only getting swapped once.

* Here we are not incrementing i when we are swapping so that might result in more than 'n' iterations of the loop.

* Worst Case = N-1 (swap)
(N-1) + N
= (2N-1)
∴ O(n) linear.

```
public class CyclicSort {
    public static void main (String[] args) {
        int[] arr = {3, 5, 2, 1, 4};
        sort(arr);
        System.out.println(Arrays.toString(arr
    }

    static void sort (int[] arr) {
        int i = 0;
        while (i < arr.length) {
            int correct = arr[i] - 1;
            if (arr[i] != arr[correct]) {
                swap(arr, i, correct);
            } else {
                i++;
            }
        }
    }

    static void swap (int[] arr, int first, int secon
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
}
```

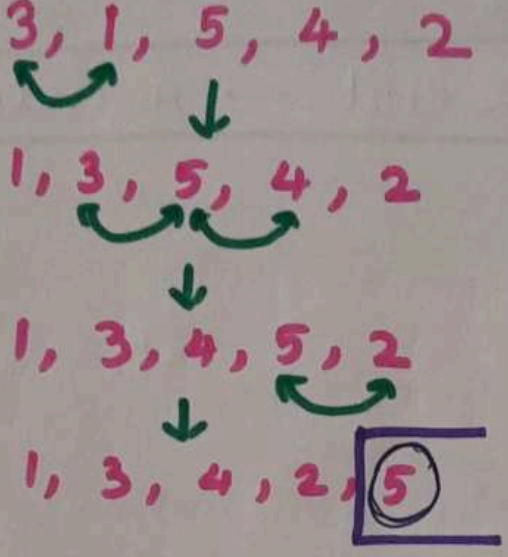
Bubble Sort

Sorting: It is a process of arranging items systematically.

Bubble Sort: It is the simplest algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

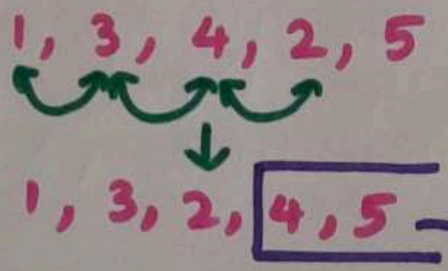
Example:

First pass:



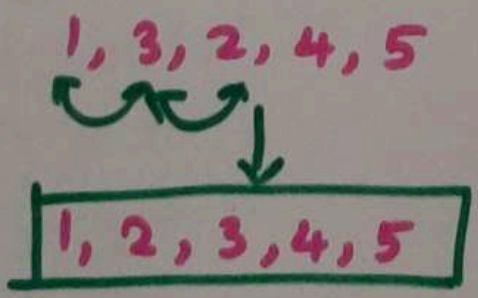
with first pass through the entire array, the largest element (here, 5) came to the end

Second pass:



with second pass second largest element comes at second from last index.

Third pass:

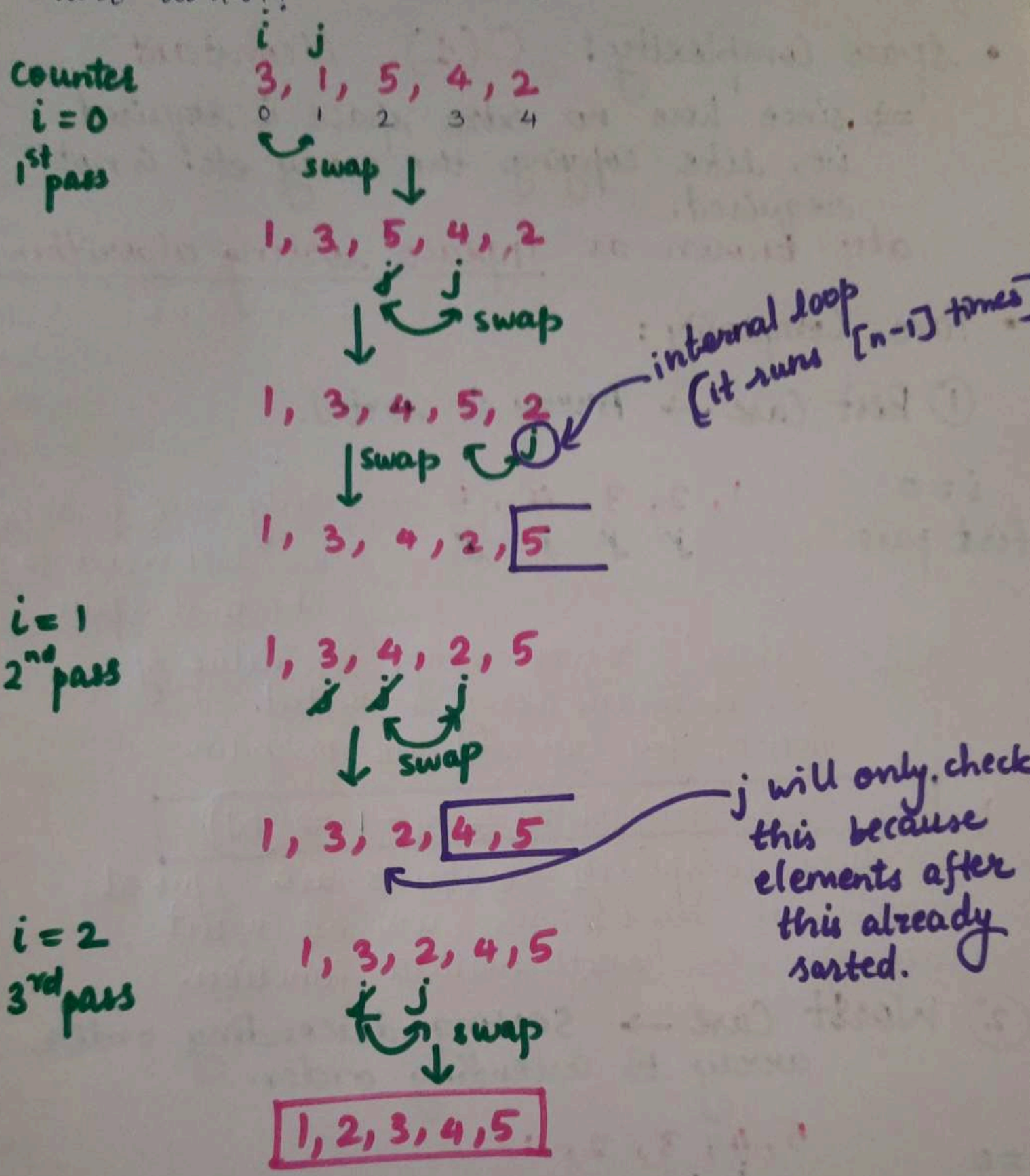


sorted!!

don't need to compare again & again since it is already sorted.

~~Sorth~~ Bubble Sort is also known as Sinking Sort or Exchange Sort.

Let's understand more deeply how actually bubble sort works!!



Complexity :

- Space Complexity: O(1) // constant
  ⇒ since here no extra space is required. i.e., like copying the array etc. is not required.
  also known as inplace sorting algorithm

- Time Complexity :

① Best Case → Array is sorted.

i = 0 first pass

1, 2, 3, 4, 5 → only once it ran we don't need to check it again.

NOTE: when j never swaps for value of i, it means array is sorted. Hence, you can end the program.

☆ Best Case comparisons = N − 1 ⇒ (N)

since in time complexity constants are ignored, we don't want exact time, we just want relationship i.e., mathematical function.

② Worst Case → Sorting descending order array to ascending order.

i = 0 first pass

5, 4, 3, 2, 1

4, 5, 3, 2, 1

4, 3, 5, 2, 1

4, 3, 2, 5, 1

4, 3, 2, 1, 5 ⟹ (N − 1) swaps

i = 1
second pass

4, 3, 2, 1, 5 ← already sorted
j
run loop only for this.

3, 4, 2, 1, 5
j

3, 2, 4, 1, 5
j

3, 2, 4, 1, 5 ⇒ (N-2) swaps

i = 2
third pass

3, 2, 1, 4, 5
j

2, 3, 1, 4, 5
j

2, 1, 3, 4, 5 ⇒ (N-3) swaps

i = 3
fourth pass

2, 1, 3, 4, 5
j

1, 2, 3, 4, 5 ⇒ (N-4) swaps

total comparisons = $N-1+N-2+N-3+N-4$

$= 4N - (1+2+3+4)$

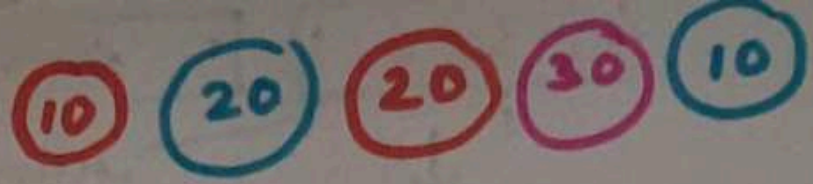
$= 4N - \left[\frac{N \times (N+1)}{2}\right]$

$= 4N - \frac{N^2+N}{2}$

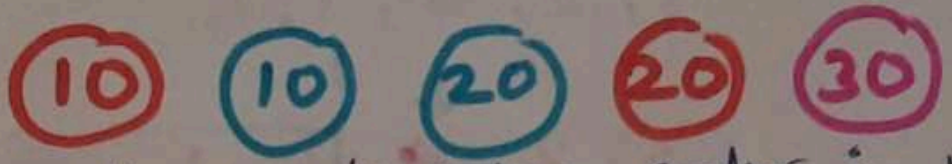
$= O\left(\frac{7N - N^2}{2}\right)$

total comparisons = $O(N^2)$ → In time complexity, constant & less dominating terms are ignored.

Stable Sorting Algo :

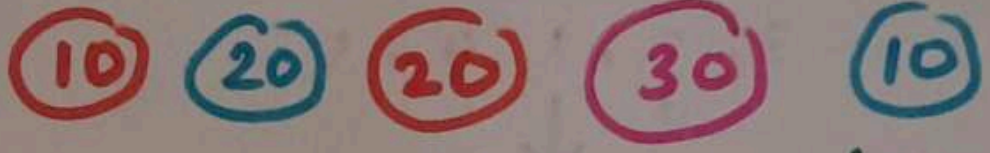


↓ after sorting



⇒ Here after sorting the order is maintained.
⇒ Order is same when value is same

Unstable Sorting Algo :



↓ after sorting

Insertion Sort:

Array ⟹ 5, 3, 4, 1, 2 [partially sorting the array]

For every index, put that index element at the correct index of LHS.

i.e., 1st pass → i = 0 → this will be sorted

5, 3, 4, 1, 2

↓

3, 5, 4, 1, 2

2nd pass → i = 1 → this will be sorted

3, 5, 4, 1, 2

↓

3, 4, 5, 1, 2

3rd pass → i = 2 → this will be sorted

3, 4, 5, 1, 2

↓

1, 3, 4, 5, 2

4th pass → i = 3 → this will be sorted.

1, 3, 4, 5, 2

↓

1, 2, 3, 4, 5 → array is sorted !!

* To understand what every 'i' is doing:

Outer loop:

5, 3, 4, 1, 2
0 1 2 3 4

	i	j
sort array till index 1 — pass (1)	0	
sort array till index 2 — pass (2)	1	
sort array till index 3 — pass (3)	2	
sort array till index 4 — pass (4)	3	

i.e., 'i' will run from 0 to (n-2)

* WORKING:

i<(n-2)	j>0
0	1
1	2

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

3<5 → swap

3, 5, 4, 1, 2

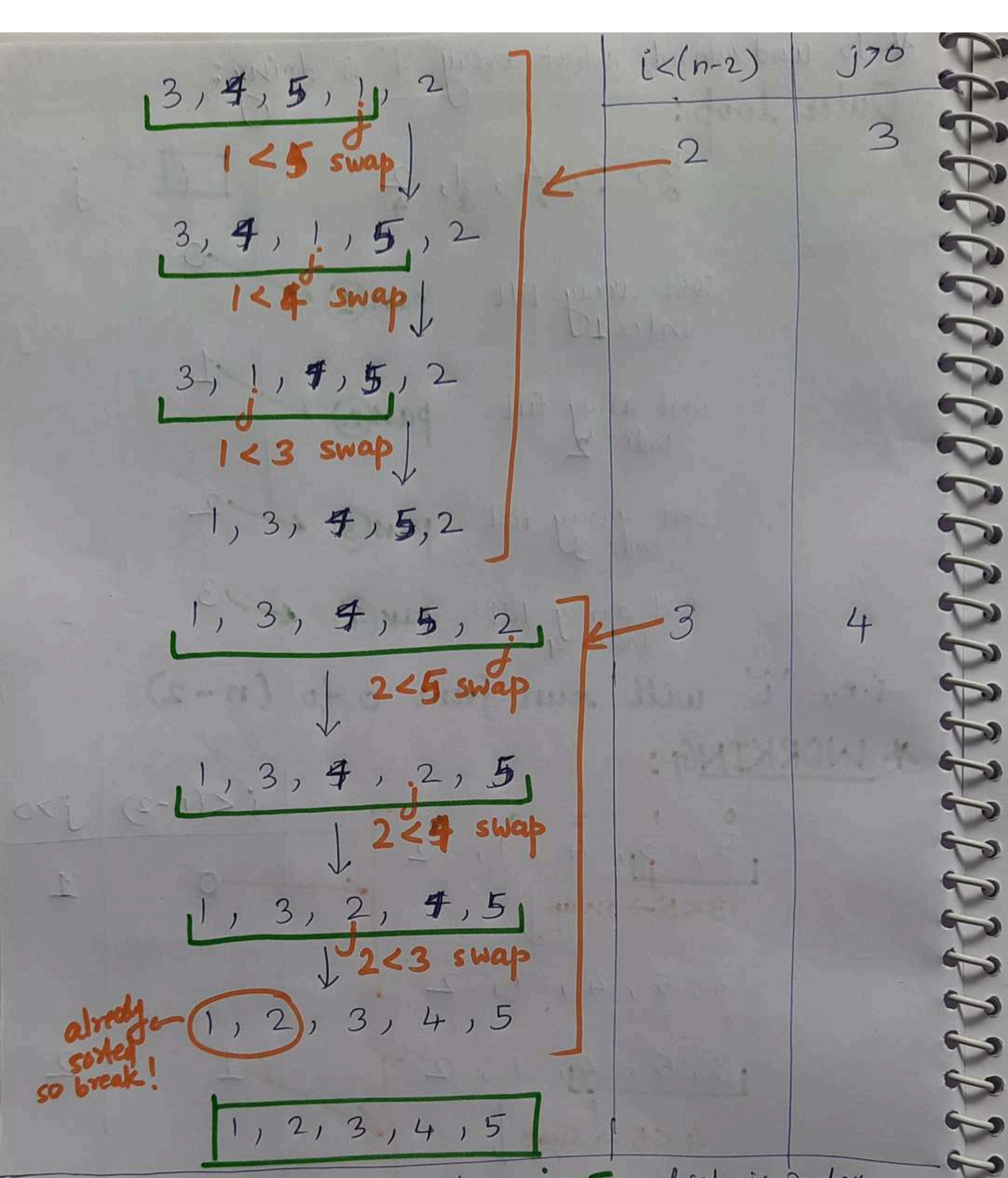
3, 5, 4, 1, 2

4<5 → swap

3, 4, 5, 1, 2

[now, since 3 < 4 already sorted]

when element j is not smaller than element (j-1) break the loop because the previous (LHS) side array is already sorted.



Now, if we take $i=4$ then $j=5$ which is index out of bound.

therefore, we take $\boxed{i<(n-2)}$ where n is length of array.

Time Complexity :

① Worst Case [descending sorted] $\Rightarrow O(n^2)$

② Best Case [already sorted] $\Rightarrow O(n)$

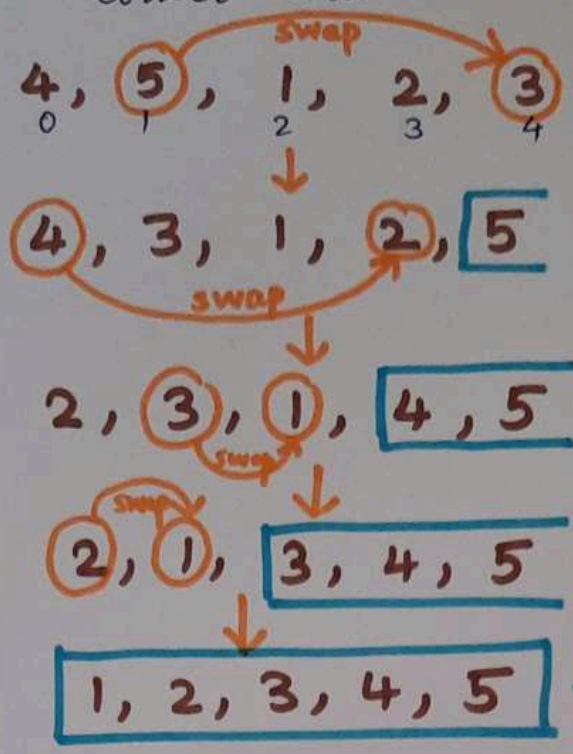
Why to use insertion sort ?

* **Adaptive** : steps get reduced if array is sorted [i.e, no. of swaps are reduced as compared to bubble sort]
* **Stable Sorting Algorithm**
* **Used for smaller values of n** : works good when array is partially sorted.
  - it takes part in hybrid sorting algorithms

Selection Sort

Selection Sort : Select an element & put it on its correct index.

Example:

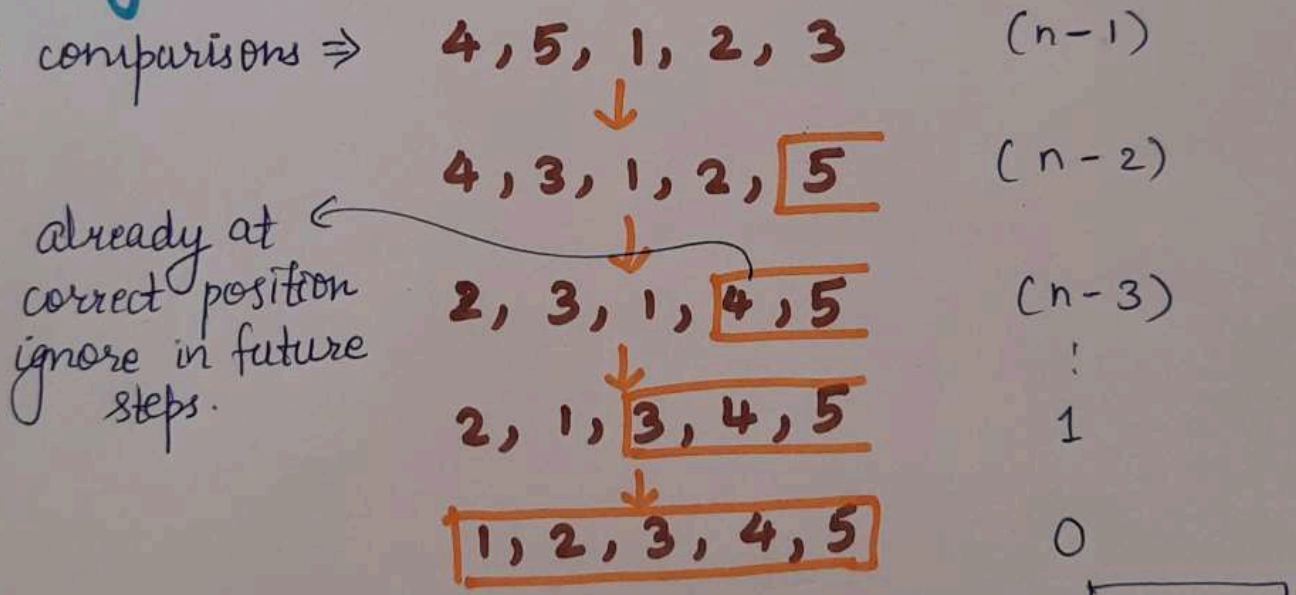


[get the greatest element of the array & put it on its correct index]

→ Here we selected maximum element & put it on right index, we can also do vice versa i.e, select minimum element and put it on right index !!

Complexity :

Total comparisons ⇒



$= 0 + 1 + 2 + 3 + \ldots\ldots (n-1) = \frac{n(n-1)}{2} = \boxed{\frac{n^2 - n}{2}}$

↑ neglect le dominating constant te

Worst Case → $O(n^2)$

Best Case → $O(n^2)$

⇒ It is not stable sorting algorithm.

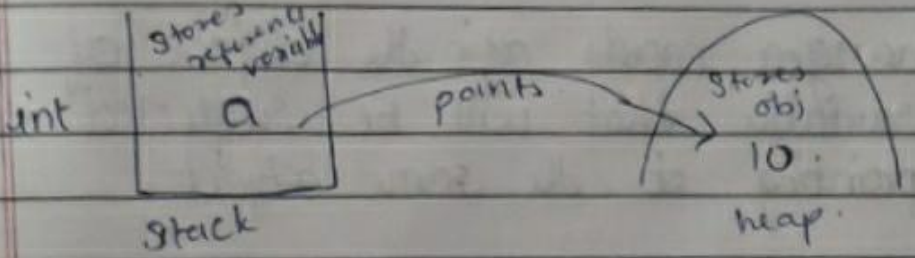
⇒ It performs well on small lists.

Strings & Stringbuilder ✍

① Stack memory :- When we declare a variable
eg :- int a = 10.
so, here the reference variable is stored in stack memory.

② Heap memory :- Reference variable stored in stack memory is pointing to the object of that variable are stored in heap memory.

int a = 10
s h



③ Memory allocations :- Memory allocations specifies the memory address to a program.

- There are two types of memory
  a) Stack memory & b) Heap memory

a) Static memory allocations :- It performs type checking at compile time.
- Here, errors will show at compile time.
- Declare datatype before you use it.
- More control & Runtime errors are reduced.
- Here you've to write a little bit more code. but you've more control over the type of data.

4d) Dynamic memory allocations:- Performs type checking at runtimes.

- Here, errors might not shown at till the program runs.
- No need to del declare a datatype of a variable
- It saves time in writing code but might give error at runtime.

5) Garbage collections:-

- More than one reference variable can points to the same object.
- If any changes made in the object of an reference variable that will be reflected to all others pointing to the same object.
- If there is an object without reference variable then the object will be disposed by "Garbage collection"
- So that's how garbage collection works.

Q What are strings? Strings will be in double quote " "

Ans:

- Strings are pile or a sequence of characters for Eg "kunal".
- It's a datatype & it is a non-primitive. so, a non-primitive datatype.
- Syntax :

```
String name = "Deepa Chaurasiya";
System.out.println(name);
```

↑ Everything that starts with a capital letter is a class.

- It is the most commonly used class in the JAVA's class library.
- Every string that you create it's actually an object of type String.

** Concepts :

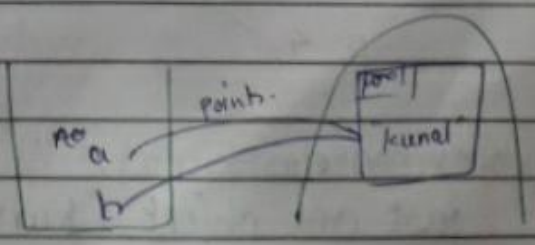
① String pool : It is a seperate memory structure inside the heap.

Q Why seperate pool? Why not just putting it out in the heap normally? like of every other object?

Ans: All the similar values of strings are not like recreated in the pool.

Eg:

String a = "kunal"
String b = "kunal"



so it's gonna be like it already exits in the pool no need to create it again pt b to the kunal.

Use case :- It makes our program more optimised.

Note :-

- If you try to change this object via this reference variable it will it not change for b.

Q Why ?

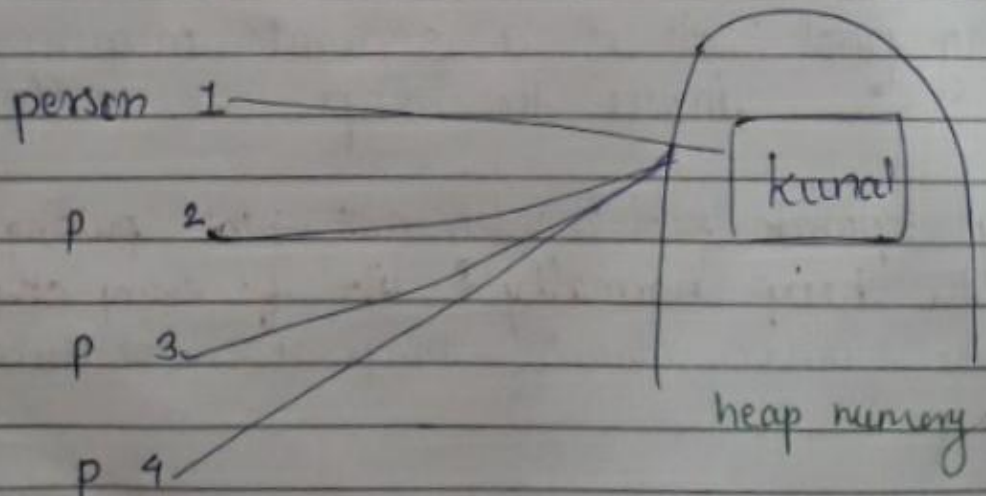
Ans :- Immutability.

- Strings are immutable in java, we can't change the object or modify object.
- If you wanna rename to "kunal" then you've to creat a new object for that.
- Strings are immutable for security purpose.

***

Why Strings are immutable ?

Ans :-



all person's named as kunal. ∴ All of them will be having just one object "kunal".

- Suppose one person decides to change their name if it was not immutable, if a person 1 decided to change their name to karan, so, that will change his name to karan
- If it was allowed to modify this then all person 1, 2, 3, 4 ~~will~~ name will be karan in the database.
- Therefore, for the security reasons strings are immutable.

* Comparisons of Strings.

① == method : comparator

Eg ① a ⟶ "kunal" } In this case == will give false.
b ⟶ "kunal"

② a ⟶ "kunal" } in this case == will be give true
b ⟶

- Comparator actully checks for value and reference variable if the reference variable is pointing to the same object

② How to creat different objects of same values.

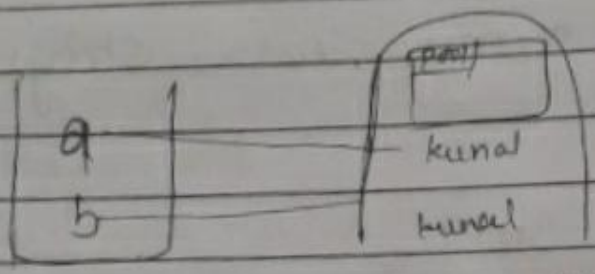
Ans:-

Syn Suppose, take an i/p as a string.
& by using new keyword we can create a new obj. →

String a = new String ("kunal")

String b = new String ("kunal")

// creating these values outside the pool but in heap because it is object so it will be in heap only.



a==b // false.

even though the values are same the but these two a & b are not pointing to the same object in that case it will give false.

- When you only need to check value, use .equals method or function.

so,

```
String name1 = new String ("kunal");
String name2 = new String ("kunal");

System.out.println(name1.equals(name2));
                                o/p :- true
```

Here does not care whether the reference variable are pointing to same object or not it just cares about the value.

* class is a name group of properties and funct^{n}.

* We can't do :

```
System.out.println(name1[0]);
```

- But in arrays we can do this.

- Though it's sort of collection of characters but we cannot do ↑ do this.

• So, we have to use a method called charAt().

```
System.out.println(name1.charAt(0));   o/p = K.
```

out : variable of type printstream.

printstream : class : it has method in it called println

* function overloading ! or method overloading.

Ans: Function overloading happens when two functions have same name.

eg:

```
fun () {
    // code
}
fun () {
    // code
}
```

} ✗ function overloading.

```
// fun (int a) {
    // code
}

fun (int a, int b) {
    // code
}
```

} It is allowed having different arguments with same function/ method name.

- At compile time, it decides which fish function to run.

* Pretty printing :-

```
float a = 453.1234f;
System.out.printf("Formatted number is %.2f", a);
```

Place holder

printf :- Formatted string

name declared

- Here % means a place holder & till how many decimal value do we want. for eg :- 2. % .2f (f because it's float)
- It rounds off all as well.

* for π

```
System.out.printf(Math.PI);
```

* for String

~~System.out.printf~~

System.out.printf("Hello my name is %s and I am %s.", "kunal", "student");

o/p :- Hello my name is kunal and I am student.

So, the order in which you have place the placeholders, int that order only you've to put the variable

- Some common formate specifiers :-

(1) %c :- Character
(2) %d :- Decimal number (base 10)
(3) %e :- Exponential floating-point number
(4) %f :- floating-point number
(5) %i :- Integer (base 10)
(6) %o :- Octal number (base 8)
(7) %s :- String
(8) %u :- Unsigned decimal (integer) number
(9) %x :- Hexadecimal number (base 16)
(10) %t :- Date/time
(11) %n :- Newline.

Operator overloading:

* Operater + ("Overloaaded for String type")

① sout ('a' + 'b'); o/p = 195

(character)

↓

Here the operater is converting this into its ~~character value~~ integer value. and then adding the "unicode or Ascii value"

② sout ("a" + "b"); o/p = ab

It concatenate the strings.

③ sout ('a' + 3); o/p = 100

④ Sout ((Char) ('a' + 3)); o/p = d

converting the 100 into a character (Explicit casting).

Note:- 1) When you're doing the addition with characters it converts into its value into a number then it use that number to solve the problem.

2) But with string it's not doing that, it actually taking the string value

⑤ Sout ("a" + 1); o/p = a1

// integer will be converted to integer that will call toString()

Note:- 1) When an integer is concatenated ("added") with a string it is converted to its wrapper class integer

2) This is same as: "a" + "1".

⑥ Sout ("kunal" + new Arraylist <>());

o/p :- kunal []

Initially arraylist is empty. then it's converted into kunal

So, we know it is will be like an object of type integer hence, it's calling the toString. which is returning a normal brackets. since it empty it will return a empty array.

⑦ Sout (new Integer (56) + new Arraylist <>());

error.

expression

Operator '+' cannot be applied to integer and array list.

* Operator '+' in Java is only defined for primitives and when any one of these values is a string

* Operator '+' you can only use with primitives and you can only use this with all the complex objects as well.

But, the only condition is at least one of these object should be of type String.

Eg: Sout (new Integer(56) + "" + new Arraylist <>());

this will work

complex obj

complex obj

obj of type string

Here the entire result will be of string type. o/p: 56[]

So, On string objects the plus operator is being overloaded, because it concatenate more than one strings.

- In java operater overloading is not supported for some software engineering but consideration But, in C++ it is supported.
- So, ~~you basically~~ + operater, you can basically modify what the plus '+' operater is doing in C++ & also in python
- you can also make it act like as a multiplication or subtring subtraction, you can add complex data type at as well.
- But this results in poor code, that is why in Java it is not supported.
- Java has only given us operator overloading exception for strings, but you cannot do it on your own if you want to merge two complex objects of your own type like array, hashmap it will not allow ever for the modification.
- It's only operater that is intentionally overloaded in java to support string concatenation or string joining.

- new Integer() : In future it's going to be removed.

(8) sout ("a" + 'a'); o/p : aa if one one of the data type is string ans will be String

String performance

loop

① Series : " " . . .

② Series : "" + "a" : "a"

③ series : "a" + 'b' : "ab"

④ series : "ab" + 'c' : "abc"

⑤. Here the new object is being created everytime, it is not changing the original object because the Strings are immutable. Therefore this one one copying the one old one and the appending it one with the new one.

- So much of waste of memory because of these and it we can't be have having any reference val variable. So time complexity : $O(n^2)$ worst.

- Solution :- wouldn't it be great if there was sort of datatype that will allows us to modify it value because strings fails to allow this. So, the answer is :- Stringbuilder

* The Here in Stringbuilder one object is made and the changes are done in that object only & the reference is also the same and it will not changed

* Stringbuilder is a seperate class.

* Methods that string provide (Some).

1) .toCharArray() :- It converts char into array of strings characters

Eg:- string name = "kunal kushwaha";
sout (Arrays.toString(name.toCharArray()));
o/p : [k, u, n, a, l, , , K, u, s, h, w, a, h, a]

PS Even that space will be counted.

2) .length() :- It will gi provide you the length.

3) .toLowerCase() :- It will convor into lowercase.
It's not actully going to convert the origin object because of immutability.

4) .indexOf :- It will give the index value.

5) .lastIndexOf :- It will give the last index value.

6) .strip() :- White spaces are removed.

7) .split() :- add a regex first :- After adding regex it will split ~~on it~~ over here

.Split(regex :)
↑
blueed

Code of plr palindrom string. Time On

```
public class Palindromstring{
    Public static void main(String[] args){
        String str = "abcba";
        System.out.println(isPalin(str));
    }

    Static boolean isPalin(String str){
        if (str == null || str.length() == 0){
            return true;
        }
        str = str.toLowerCase();
        for(int i=0; i<=str.length()/2; i++){
            char start = str.charAt(i);
            char end = str.charAt(str.length()
                       -1-i);
            if(strat != end)
                return flase;
            }
        }
        return ftrue;
    }
}
```

this will be having null.

If this is = 0, that mean for empty str

here length is a method

you can also put null as false completely depends upon interviewer but the ans will be same

(Str.length-1) = last index every time we will be doing -i b, Suppose i=2, then end index should be, end index -2 Hence ...

Recursion :- The most important.

Note:- Functions / Method & Memory Management knowledge is must.

① Functions / Method :- A Functions / Method is a collection of code that you can use again and again.

② Memory Management :- there are two types of memory. "Stack & Heap".

① Stack memory :- When we declare a variable

eg:- int a = 10;
stack heap.

the reference variable are stored in stack.

② Heap memory :- Variable which is pointing to the object of that variable are stored in Heap. are el called Heap memory.

Q ~~What is recursion~~? (In detail).

Ans:-

1) Function calling another function.

2) All of the fuctions will have one thing in common.
the body and the definition of these function

Eg:- Taking one parameter and doing something. just the name is different.

* Internal working of (recursion code): call stack + Debugging.

// code.

```
public static void main(String[] args){
        print1(1);
}
static void print1(int n){
    System.out.println(n);
    print2(n2);
}
static void print2(int n){
    System.out.println(n);
    print3(3);
}
static void print3(int n){
    System.out.println(n);
    print4(4);
}
static void print4(int n){
    System.out.println(n);
    print5(5);
}
static void print5(int n){
    System.out.println(n);
}
```

Now all the functions are same (print & call)

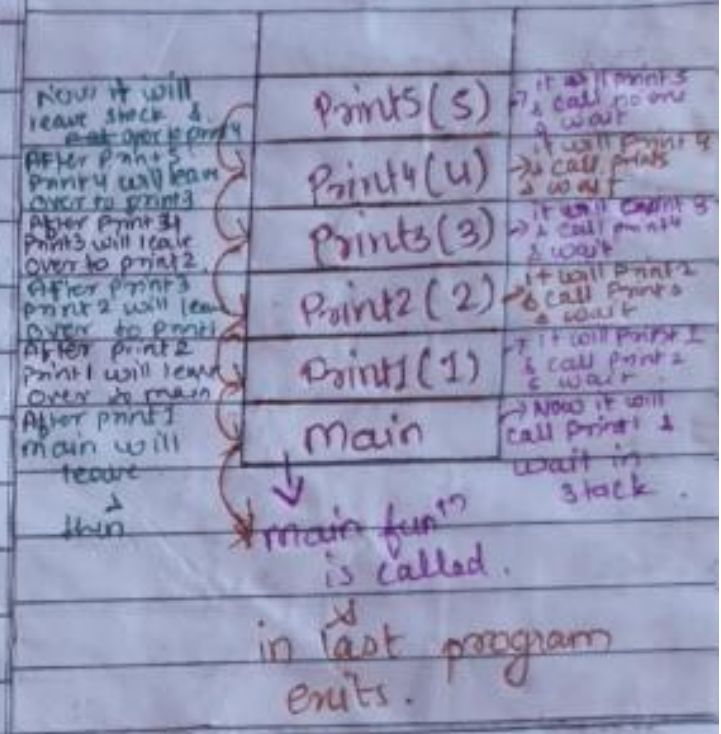


fig: flow of a code.

func^n body changes here because it is not calling anything, only print is there

it includes extra questions like

1) How function calls work in programming language?

Steps:-

1) First it's calling main function "all the functions call that happen in a programming language they go into the stack memory".

2) a) While the function is not finished executing it will remain in stack memory.

Q. Which is the first function that is called in programming language 'like JAVA, C, C++ etc'

Ans. main function!!!

b) Main function ~~it~~ is the first function that will go in stack and the last function that will come out of the stack.

c) When a function is staying inside the stack it basically means that function call is currently going on.

3) SO, the main function is called & then main function itself calls the print1 function. (given).
SO, print1 function is called & main function is currently in progress,
which say 'Hey' print1 please give me the answer that I've asked you to do.
So, after print1 function finished execution it will end, but it will rest in stack memory
So, the print1 will get called first & then it will also go in stack memory. & it is going to have a primitive. (primitives are also stored in a

stack memory) so, in the stack memory only there will be a variable primitive one, because we have passed that.
Now the print pun function is called so, it'll print 1 (one).

Q. What does print function do when it is called?
Ans It prints whatever the value 'w' is passed.

4) Now, the print1 is going to call print2. So, now the print1 will say to print2 that "main" actully wants us to print all the numbers from 1, 2, 3, 4, 5 I've printed 1, print2 could you, please print 2. & and asks the other functions to print the rest of the numbers! So, now print2 will take care for the rest of the numbers & it will print 2. Now, print2 will be inside ~~in~~ the stack

5) Now, print2 is like "hey" print3 I'm not finished executing print1, ~~print2~~ main are waiting on me to print 2, 3, 4, 5.
As I've printed 2, print3 can you please make sure that no. 3, 4, 5 are also get printed.
So, while you do that, I'll wait inside the stack memory.
So, the print3 will be like "fine", you wait, I'll take care of the no. 3, & 4, 5. Now, print3 function is called it will print 3 & go inside stack memory. but befor it will call print4.
"Every function is in the stack ~~meror~~ memory is so currently ongoing".

6) As print3 called print4 function it will perform same as above and then call another function and wait in stack

7) Now, print4 will call print5, so 4 is printed and the funtn is obviously in the stack memory which is saying, prints we have printed 1, 2, 3, 4 and of all of us are waiting in the stack could you please the last number?
So, print5 will print last number and it'll also gonna say that "do you want me to call print6 or something else or am I the last one" print4 will be like No you're the last one do not call anyone else. So, print5 will be called & it will go in the stack memory.
Because any function that is currently running will go in stack memory, so print5 is in stack memory & it'll print 5.
So all 1, 2, 3, 4, 5 has been printed.

8) Now, print5 is going to be like My work is done and I don't need to call anyone else. So my function call will be over

Note: When a function finishes executing it is removed from the stack and the flow of program is restored to where that function was called.

Eg: int a = b + c;
↓
it is calling the sum function & the sum function will return a value & it will return a value from where it was called.

9) So, print5 has finished executing so, from where will be our program executing right now?
- from where this funct" was called.
Because the function has finished executing so it will come out from where it was called. & the the print5 will be removed from the stack. So now it will come outside & prints is going to finish the executing its going to say to print4 hey I'm finished executing and I don't see that you're doing anything else. So you too can also finish executing.

10) Now, print4 is going to be like fine if print5 did it's work and nothing new needs to be done so, I'll also finish executing and I'll also leave the stack memory

11) Not, print 4 will like print 3 I'm done with my work and the rest is upto you. print 4 will leave the stack. So, now the program flow is with print 3

12) print 3 will do the same thing. and leave the stack print 2 will also do the same & leave.
print1 will be like I'm also done & leave. so it will go from where it was called "main" fundion.

13) Now, main function will be like I'm also done & leave. main is the last funct" that is removed from the stack & then the program will be over.

O/P :- 1, 2, 3, 4, 5

Recursion.

// eg: code o/p: 1 2 3 4 5

Q. Write a function that takes in a number and prints it. (1st five no).

```
public class NumberExample {
    public static void main(String[] args) {
        printnum(1);
    }

    static void printnum(int n) {
        // base condition.
        if(n == 5) {
            System.out.println(5);
            return;
        }
        // funct° body ~~here changes because~~ it is not calling anything
        // body: System.out.println(n);
        printnum(n+1); // recursive call. (a tail recursion)
    }
}
```

it is a base condition check

this is the last statement that is executed in this recursion fun°. ∴ it is a tail recursion.

Note: 1) Without check (a base condition) the function is being called then it will run infinitely. i.e stackoverflow if

2) If you are calling a function again and again, you can treat it as a seperate call in the stack.

3) Each call, you can treat it as a seperate call. function in the stack

4) Here, In this code the same function is being called again & again But, every function call is taking the memory seperately.

Notes:-

1) While a function call is not finished executing it will stay in the stack memory.

Q. What is Recursion?

Ans:-

Recursion means a function that calls itself.

Q. What is a base condition? (in recursion)

Ans:-

1) It is a condition where our recursion will stop making new calls.
2) It is a simple if condition.
3) It needs to be returned.

Q. What is stack overflow / error?

Ans:-

1) Suppose there was no base condition, we will get error.
2) It means that function calls will keep happening.
3) Stack will be keep getting filled again and again.
4) We know that, Every call takes a memory even though it's the same function or different one doesn't matter.
5) If you're calling a function more than one times simultaneously. So, again and again every function call will take some memory in stack.
6) So, if there is no base condition it will keep going on and one time will come where

memory exceeds of a computer's limit.
7) Hence, this is going to gives or throw an error.
8) That error is termed as Stack overflow error.

Q. Why recursion?

Ans:
1) It helps us in solving bigger / complex problems in a simple way.

2) You can convert recursion solutions into iteration, and vice-versa.

Q What is iteration?

Ans: 1) It basically means not using any function calls
2) It means loops → for loops.

3) Recursion takes a space aswell.
So, sapce complexity: not constant because each function call is taking some memory.
∴ recursive calls.

4) It help us in breaking down the bigger ~~prob~~ problem into smaller problems.

Since, these are recursion calls or function call linked to one another. is visualizing recursion.

* Visualization Recursion :- IMPORTANT

? print a no from 1 to 5.

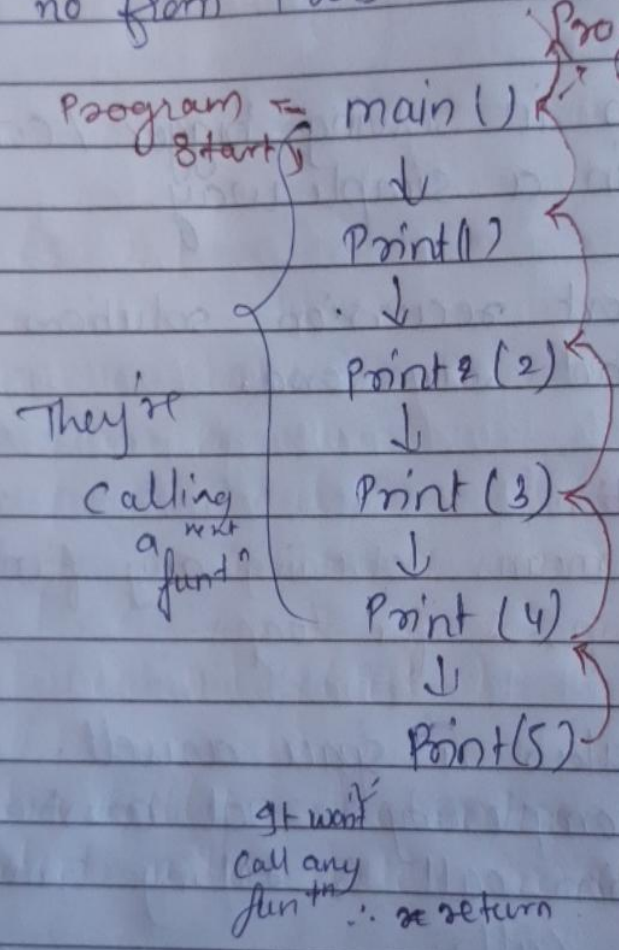


fig:- Recursion Tree / Recursive tree.

Q find n^{th} fibonacci number using recursion.

→

Q How to identify whether a prog problem can be solved in recursion or not?

Ans:-

① Practice

② Try to see if there's a smaller version of the problem that we can solve.

"Check if you can break the problem in to smaller problem"

Formula :- $fibo(N) = fibo(N-1) + fibo(N-2)$.

in abbr :- $fn = fn-1 + fn-2$ $(\because f_1 = f_2 = 1)$.

Now let's see the working of fibonacci no. using formula :-

Q. find F_3 ?

Now as we know

$fn = fn-1 + fn-2$ - (formula)

$f_3 = f_{3-1} + f_{3-2}$

$= f_2 + f_1$ } (given ↑)

$= 1 + 1$

$= 2$

$\therefore \boxed{f_3 = 2}$.

Q2) $f_5 = ?$

$f_n = f_{n-1} + f_{n-2}$ - (formula)

$f_5 = f_{5-1} + f_{5-2}$

$= f_4 + f_3$

$= 3 + 2$

$= 5$

$\therefore \boxed{f_5 = 5}$

So, the fibonacci series :-

f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	...
0	1	2	3	4	5	6	7	...

0, 1, 1, 2, 3, 5, 8, 13, ...

So basically, the entire problem is divided into two smaller f_n problem.

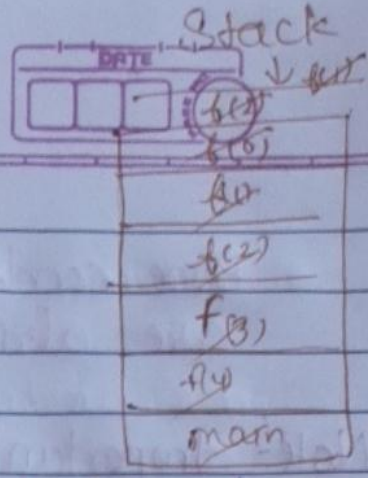
$f_n = f_{n-1} + f_{n-2}$

Here, Hence you can apply recursion.

- Now the divided part itself can be broken down into two more smaller problems

$f_{n-1} = f_{n-2} + f_{n-3}$

Hence it will keep going on....



Q Find the fifth fibo no.

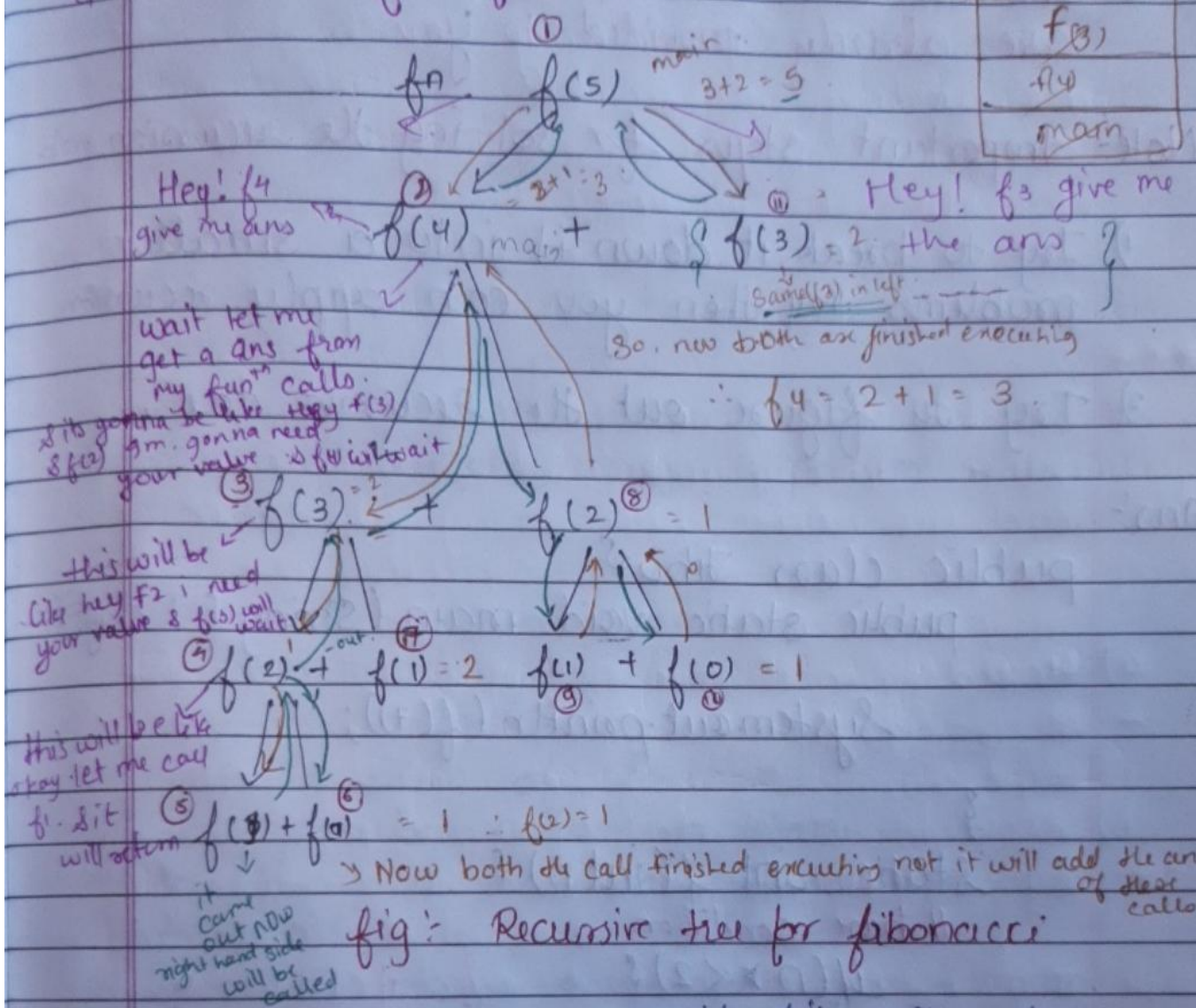


fig: Recursive tree for fibonacci

1) So, this is how you identify the solution can be solved via recursion ed or not. you try to see, if you can break it down into a smaller problem.

2) When you write the recursion in a formula it is known as recurrence relation.

3) The base condition is represented by answer we already have.
Eg: In this case we know that f(0) = 0, f(1) = 1. this is base condition.

Base conditions are those whose answers are already provided to you

Note:- Important steps for solving the recursion problem

1) Try to break it down it into a smaller problems they then you can apply recursion

2) Try to figure out the recursive tree.

Ans:-

```
public class Fibo{
    public static void main(String[] args){
        System.out.println(f(7));
    }
    static int f(int n){
    // base condition
        if(n<2){          // if n=0 : 0
            return n;     // if n=1 : 1
        }

        return f(n-1) + f(n-2);
    }
}
```

this is not the last statement in this function call because f(n-1) gives the ans & f(n-2) gives the ans the f(n-1) + f(n-2) will be the last statement.

Note:

∴ this is not a tail recursion because it will call f(n-1) then (f(n-2) the it will do the addition of it. so this extra step of adding & return are not tail recursion

Note: So when you have the last statement in the function call it is known as tail recursion.

Note: Q. How to understand and approach a problem?

Ans:-

1) Identify if you can break down a problem into a smaller problem.

2) Form the recurrence relation of write if needed.

3) Draw the recursive tree.

4) about the tree
   a) See the flow of function -
   b) How they're getting in stack
   c) Identify and focus on left tree calls & right tree calls.
   c) Draw the trees & pointer again & again using pen & paper.
   d) Use a def debuger to see the flow.

5) See how the values are returned at each step & what type of values are returned. (int, string, etc)
   See where the function call will come out of. & in the end you will come out of the main function.

* Binary search using recursion :-
because in Binary search we ever are dividing the problem into half, so it a problem is divided into sub-problems. so definitely we can apply Binary search in recursion

***

i) So, there are two key areas of focus when you're starting out with with recursion programs and you wanna have strong foundation for recursion so, the two key areas are :-

a) How the funct^n are getting called ? The tree ? as in fig.

b) Variables and datatypes and point a).
Eg :- of fibo :- there were three types of variable one is at the argument / param one is being returned and one is somethin you may be having in the body

Which variable type to use in which place and what to return is very important and if some one knows this then the entire recursion is a easy for them.

Q How do figure out what do pass and what do create and what do returned?

* Working with variables :-

a) Arguments what ever you put in the arguments it's going to go in the next fun^n call.

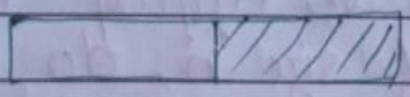
b) return type (simple) .. whatever you wanna return that'll be it

c) Body of the funct^n.
So variables which are specific to that fun^n call that will go inside of the body.

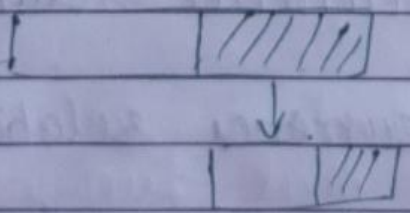
** Binary search with recursion

Q So what is binary search?

Ans Suppose you're given an array like this,



Now it will be like okay let compare the middle element with the target element.



So on & so forth.

At every steps it's doing two things

1) Comparing : It is of single step, so it takes constant amount of time. O(1)
You just need to check whether a no. is greater than or equal to or less than the no. middle no. and it does not depend on the size of an array ∴ O(1)

2) Dividing into two half

Suppose we're taking a no. to find a no. in a fun^{n}. N = size of array

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

- $F(n)$ → functh for binary search
- $O(1)$ → comparison in content time
- $F(\frac{N}{2})$ → now, I search in the array of size n/2 (Dividing array in to two parts)

This is the recurrence relation.

- If you want to apply binary search on the array of size n, do a step that takes constant amount of time. & search in the half of the array.

Note Types of recurrence relation

① linear recurrance relation Eg:- fibo (+ or -)

② Divide & conquer recurrence relation Eg:- binary search.

Here the search space is reduced by a factor in above case $N/2$.

So, divide the answer into something via a factor.

∴ our search space is getting much faster in Divide and conquer recurrences.

(Dividing a number by 2 is definitely much more faster than subtracting it by 1 or 2. ∴ Divide & conquer is much more efficient)

Q Why linear recurrence relation is inefficient?

Ans:-

If in the recursion calls two or more recursion calls are doing the same work (fig:- Recursive tree for fibonacci)
Don't compute it again and again.

Q So, How can we solve this problem?

Ans Dynamic programming.

Tip: DO NOT OVERTHINK

Now, In binary search what variables do we have i.e start, end & mid.
So, from this which will go in the body of the fun" and which will go in the arguments

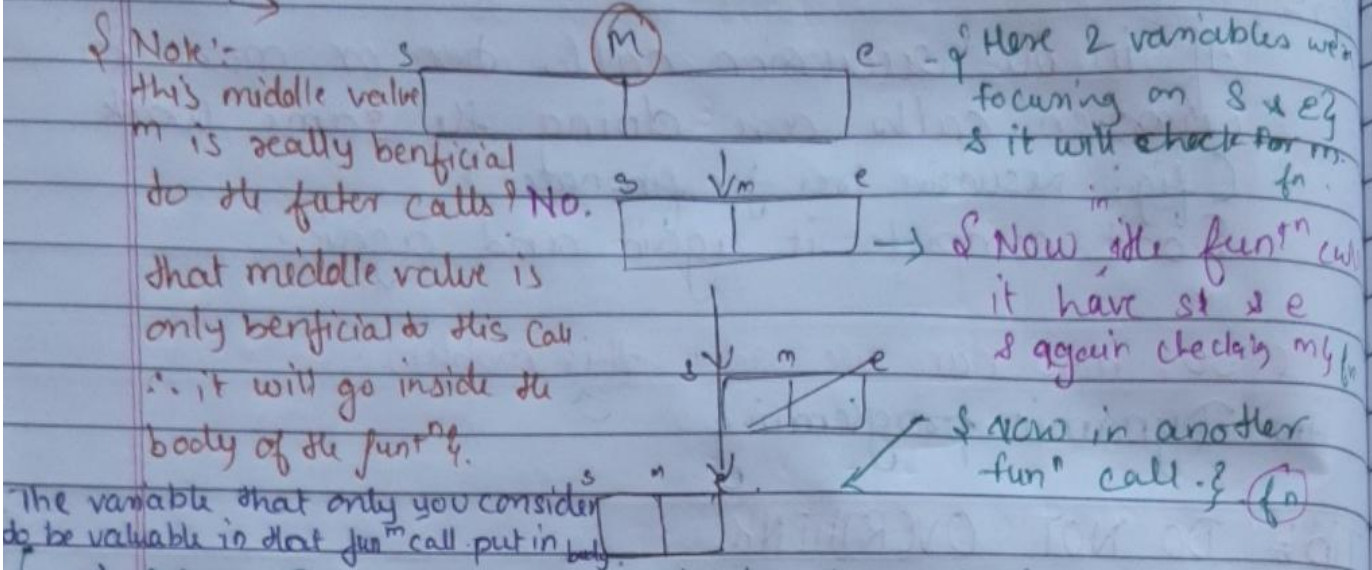
1) So, we are by trying to reduce the size of an array and what all the variables.
2) Are the variables that determine whether the size of the array is reduced:-
3) Start and end.

a) Whatever you put in the arguments/parameter. it's gonna go in the next func" call.

b) So, variables which specify to the fun" call will go inside the body of the fun".

* Continuation of where to take which variables

→



a) Now, Can you see that these 's' & 'e' variable these are actually being passed in the funtn call. that is why this will go in the arguments and it is very imperient because :- we'll be passing these values when we call fn in funtn call in the argument

* in short :- if there are a few variables that you need to pass in the future funtn calls then put it inside the agument.

*b) The variable that on you consider to be valuable in that funtn call only. that you don't need to pass inside the future recursion calls then put it inside the body of that function.

c) Make sure to return the result of a function call of the returned type. because if you don't return it, in the end it will be called after all the sub problems are solved and in the end it will be calling the main funtn

if original arr is not returned the main arr will also be not returned so, your answer will not be returned so, d Return the whatever ans you're getting

code:

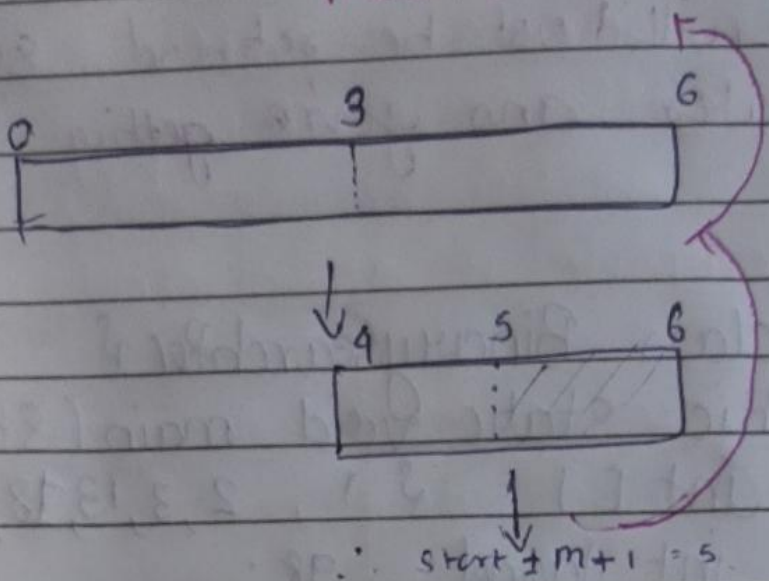
```
public class BinarySearchRec {
    public static void main(String[] args) {
        int [] = { 1, 2, 3, 13, 18, 98, 165};
        int target = 98;
        System.out.println(search(arr, target, 0, arr.length-1));
    }

    static int search (int[] arr, int target, int s, int e){

        if(s > e){
            return -1;
        }
        int m = s + (e-s)/2;

        if(arr[m] == target){
            return m;
        }
// whenever your calling a recursion call make sure returning it.
        if( target < arr[m]){                    if there is a return type.
            return search(arr, target, start, mid-1);
        }
        return search(arr, target, m+1, e);
    }
}
```

it means my element i.e. is in left hand side.

target = 98

0	1	2	3	4	5	6
1,	2,	3,	13,	18,	98,	165

main() - // ans.



① check

found the ans return 5

Where it is going to return from ? where it was called

* If there is a return condition, make sure you're returning the type which is same as the return type.

* make sure your returning whatever the subrecursion calls are giving.
Subrecursion calls is happening make sure you return it.

Q How to actually solve to get complexity?

Ans Few Few 3 ways :-

1) Plug & Chug

OR

2) Master's theorem.

OR.

3) Akra Bazzai (1996) :- ~~Solve~~ ~~any~~

** Akra Bazzi :-

formula : $T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(u)}{u^{p+1}} du\right)$

Words :- Time complexity = $T(x)$

$g(u)$:- this g funt" is basically a time complexity it-self. & we know that in time complexity like constants & more dominating terms are ignored.
we know that this going to be the simplest form of the function because all the less dominating terms and other things will be removed.

P :-

$$a_1 b_1^{\,P} + a_2 b_2^{\,P} + \cdots = 1$$

i.e $\sum_{i=1}^{k} a_i + b_i^{\,P} = 1$ //

$$T(N) = T\left(N/2\right) + C$$

Q In constants why do we write $O(1)$?

Ans Because we don't really care about constants for eg :- we can write $O(1)$ or $O(2k+1)$ all of these are constant, ∴ we can ignore the constants

∴ Basically $O((2k+1) \times 1)$

∴ Anything multiplied by anything, you can ~~ing~~ ignore the ~~constant~~ constants.

So, Hence anything in constants can be written as $O(1)$.

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

$a_1 = 2$

$b_1 = \frac{1}{2}$ $\qquad g(x) = xN-1$

$$\therefore \quad 2 \times \left(\frac{1}{2}\right)^P = 1$$

$$\therefore \quad P = 1.$$

Once you've found the P then substitute it into the Akra-Bazzi formula :-

$$T(x) = \Theta\left(x^{p} + x^{p}\int_{1}^{x} \frac{u-1}{u^2}\,du\right)$$

$$= \Theta\left(x + x\int_{1}^{x} \frac{1}{u} - \frac{1}{u^2}\,du\right)$$

$$= \Theta\left(x + x\left[\int_{1}^{x} \frac{du}{u} - \int_{1}^{x} \frac{du}{u^2}\right]\right) \quad \int u^{-2} = \frac{u^{-1}}{-1} = -\frac{1}{u}$$

$$= \Theta\left(x + x\left[(\log u) + \left[\frac{1}{u}\right]\right]\right)$$

Put ↑.

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

$$= \Theta\left(x + x\log x + \frac{1}{\log x} - x\right)$$

$$= \Theta(x\log x + 1)$$

$\therefore \Theta(x \log x)$. // Time complexity.

So, for array of size N :-
Merge sort complexity $= O(N \log N)$.

2) $T(N) = 2T\left(\frac{N}{2}\right) + \frac{8}{9} T\left(\frac{3}{4}N\right) + N^2$

$\therefore \quad 2 \times \left(\frac{1}{2}\right)^P + \frac{8^2}{9_3} \times \frac{3^1}{4_1}^P = 1$

$1 + \frac{2}{3}$ so, put $P = 1$

$2^1 \times \frac{1}{4_2} + \frac{8^1}{9} \times \frac{69}{16_2} = 1$

$\therefore \frac{1}{2} + \frac{1}{2} = 1$

$\therefore P = 2$ //.

Subsidute.

$$T(x) = \Theta\left(x^2 + x^2 \int_1^x \frac{xy^2}{3y^3}\, dy\right)$$

$= \Theta(x^2 + x^2 \log x)$ ✓ ignore the less dominating term ✓

$= \Theta(x^2 \log x)$

If you're unable do find value of P:

Q. $T(n) = 3T\left(\frac{n}{3}\right) + 4T\left(\frac{n}{4}\right) + n^2$

① $P = 1$ Case 1

$= 3 \times \left(\frac{1}{3}\right)^1 + 4 \times \left(\frac{1}{4}\right)^1 = 1$

$\therefore 1 + 1 = 1$

$2 \neq 1 \quad \because 2 > 1$ This means what? I need do increase the denominator

② $P = 2$ Case 2

$3 \times \frac{1}{9} + 4 \times \frac{1}{16} = 1$

$\frac{1}{3} + \frac{1}{4} = \frac{7}{12} < 1$

Hence, P is less than 2.

So, P actually lies in range 1 & 2

Note:

When P < power of $(g(n))$ than you your ans = $g(n)$.

Here, $g(n) = n^2$

$p < 2$ {i.e power of $g(n)$}

Hence, ans = $O(n^2)$.

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{u^2}{u^{p+1}} \, du \right)$$

$$= \Theta\left(x^p + x^p \int_1^x u^{1-p} \, du \right)$$

$$= \Theta\left(x^p + x^2 \right)$$

$\because p < 2$

$\therefore$ x^p will become less dominating term x^2 Hence, ignore.

$\therefore \Theta(x^2)$ //

Hence proved.

** Big Omega Notation : def:
Opposite of Big-Oh notation

Suppose than an algo has complexity of

$\Omega(N^3)$

Q what does it means in simple term?

Ans: 1) This means that it will take atleast N^3 time complexity.
2) So, this means it the lower bound.
3) It will take atleast N^3, it can also take N^4, $N^3 \log n$ or $N^3 * 2^n$ etc
But, it will never be lesser than N^3
4) Minimum N^3 time complexity will be required.

** Matho:

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

Note: But we actually care about Big-Oh notation
Why?
Ans: We always look at the worst case

* Big-Oh Notation : Def^{n} :

Suppose, that an algo has complexity of

$O(N^3)$

① What does it mean in simple words?

Ans :) So, in simple language it means that, this is the upper bound.

2) Meaning, CN^3 : Size of the ~~arrog~~ array will grow as the input grow in an N^3 fashion (eg: Binary & Linear search). But, what does this Big-Oh saying. It's saying that the complexity "the graph & the relationship" it can't exceed N^3.

3) For Eg : your algo that you've written may be solved in a constant time, it may be solved in $O(N)$ time, or $O(\log N)$ or $O(N^2)$ times etc. But it will never be solved or exceed the time complexity, relationship, the graph, the function value, it will never exceed more than N^3.
It will never be like $O(N^4)$ or $O(N^3 \log N)$ etc.

* Maths :

$$f(n) = O(g(n)) \quad \text{-- (1)}$$

$$\lim_{n \to \infty} = \frac{f(n)}{g(n)} < \infty \quad \text{-- (2)}$$

(2) It is actually some finite value.

As we said the "Always look for worst case complexity" & "Always look at complexity for large / ∞ data".

So, Here we're applying that.

$$\lim_{n \to \infty} = \frac{f(n)}{g(n)} < \infty$$

Eg:

$$O(N^3) \sim O(6N^3 + 3N + 5)$$

$g(n)$ $f(n)$

$$= \lim_{n \to \infty} \frac{6N^3 + 3N + 5}{N^3}$$ - ∴ when the value of n reaches ∞ (closed to limit) ∴

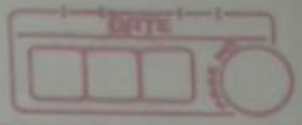
$$= \lim_{n \to \infty} 6 + \frac{3}{N^2} + \frac{5}{N^3}$$

$$\therefore 6 + \frac{3}{\infty} + \frac{5}{\infty}$$ - ∴ anything ÷ by ∞ = 0 ∴

$$= 6 + 0 + 0$$

$$= 6 < \infty$$

Hence, proved! It is a finite value because this is showing an upper bound.



Note: Our algorithm will never exceed the complexity of this. It can be better than this like it can also be solved in less complexity, but at any case it will never exceed.

* Types of Recursions :-

1) Linear ; Eg :- fibonacci no. $f(n) = f(n-1) + f(n-2)$

2) Divide & Conquer ; Eg :- binary search $f(n) = f(N/2) + a_1$

Note :-
You can represent recursion in the form of any question.

* Divide and Conquer Recurrences :-

This is actually same as $f(n) = f(N/2) + O(1)$

Form :-

$$T(x) = a_1 T(b_1 x + \varepsilon_1(x)) + a_2 T(b_2 x + \varepsilon_2(x)) + \cdots + a_k T(b_k x + \varepsilon_k(x)) + g(x)$$

for $x \geq x_0$ → same constant

If we relate this with $f(n) = f(N/2) + a_1$ we can say in binary search

Eg :- $T(N) = T(N/2) + C$ → constant.

$\therefore a_1 = 1$
$b_1 = 1/2$ $\quad g(N) = \Theta C$
$\varepsilon_1(x) = 0$

So, both are same & this is the form of divide & conquer recurrences

Other relationship can be something like this

$$T(N) = \underbrace{9}_{a_1}\, T\left(\underbrace{\frac{N}{3}}_{b_1}\right) + \underbrace{\frac{4}{3}}_{a_2}\, T\left(\underbrace{\frac{5}{6}N}_{b_2}\right) + \underbrace{4N^3}_{g(n)}$$

((a)) ((b)).

Q. What is the $g(n)$?

Ans: It means that

let say ((a)) ((b)) ↑

When you get the answer from ((a)) + what you're doing with that answer it takes how much time!

((b)) basically means the ((a)) recursion call is over then what amount of time complexity is required to do actually something with those recursion calls that are over right now. Extra time require at that step!

So, Extra time require at that step is equal to checking whether a number is greater greater than or equal to or less than middle that takes constant amount of time so, that's why we have constant over here.

** Solving Linear Recurrence :- (Homogenous eqn)

Eg: $F(N) = F(N-1) + F(N-2)$

form :-

$$f(x) = a_1 f(x-1) + a_2 f(x-2) + a_3 f(x-3) + \dots + a_n f(x-n)$$

$$\therefore f(x) = \sum_{i=1}^{n} a_i f(x-i)$$ for a_i's, n is find.

n is the order of recurrence.

Solution :- for ~~filter~~ fibonacci no.

$$f(n) = f(n-1) + f(n-2) \quad -①$$

Steps :-

1) Put $f(n) = \alpha^n$ for some constant α.

$$\therefore \alpha^n = \alpha^{n-1} + \alpha^{n-2}$$

$$\alpha^n - \alpha^{n-1} - \alpha^{n-2} = 0$$

$\div$ by α^{n-2}

$$\alpha^2 - \alpha - 1 = 0 \quad -(a)$$

$\left\{ \frac{1}{\alpha^{-1}} = \alpha \right.$

$\therefore \frac{\alpha^n}{\alpha^{n-2}}$

$\frac{\alpha^n \times \alpha^2}{\alpha^n}$

This eqn is also known as characteristics of recurrence

2) take roots of (a) by using quadratic eqn.

①

$$\alpha^2 - \alpha - 1 = 0$$

formula : $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$$\therefore \alpha = \frac{1 \pm \sqrt{5}}{2}$$

$$\alpha_1 = \frac{1+\sqrt{5}}{2} \qquad \alpha_2 = \frac{1-\sqrt{5}}{2}$$

② If α_1 & α_2 have 2 roots, you can write the eqn in such a way

$f(n) = c_1 \alpha_1^n + c_2 \alpha_2^n$ is a sol^n for fibonacci.

it will equal to
$= f(n-1) + f(n-2)$

Note:

Not just for this any equation you have the number of roots, take that many no. of constants & x that many no. of roots with the power of n, add all that will equal to the Q.
So, for any constant C_1 & C_2

$$f(n) = C_1\left(\frac{1+\sqrt{5}}{2}\right)^n + C_2\left(\frac{1-\sqrt{5}}{2}\right)^n \quad \text{—②}$$

Note:

So, if you had three roots, you will write something like $C_1\alpha_1^n + C_2\alpha_2^n + C_3\alpha_3^n$.

③ Fact

No of roots that you have = no of ans you have already

So, here we have 2 roots α_1 & α_2.

Hence, we should have 2 ans already

We know that

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

Note:-

When you have n. no of roots or any number of roots you will have that many amount of answer.

$$f(0) = 0 = C_1\alpha_1^0 + C_2\alpha_2^0$$
$$= C_1 + C_2$$

$$\therefore C_1 = -C_2 \quad \text{—③}$$

for $f(1) = 1 = C_1\left(\frac{1+\sqrt{5}}{2}\right) + C_2\left(\frac{1-\sqrt{5}}{2}\right)$

from ③

$$1 = C_1\left(\frac{1+\sqrt{5}}{2}\right) - C_1\left(\frac{1-\sqrt{5}}{2}\right)$$

$$C_1 = \frac{1}{\sqrt{5}} \quad \& \quad C_2 = -\frac{1}{\sqrt{5}}$$

put in ②

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

- formula for n^{th} fibonacci no.

Q. How do we get the time complexity from this?

Ans:

① $f(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$ from the formula of fibonnaci number.

② As n increases or as $n \to \infty$ "$\left(\frac{1-\sqrt{5}}{2}\right)^n$" will be close to 0.

and as we know about time complexity that we should ignore the less dominating term
Hence, ~~igore~~ ignore the "$\left(\frac{1-\sqrt{5}}{2}\right)^n$".

③ Time complexity = $O\left(\frac{1+\sqrt{5}}{2}\right)^n$. for n^{th} fibonacci no.

Note:
Time complexity of fibo = $T(N) = O(1.6180)^n$

④ This is also known as golden ratio in mathematics.

⑤ So, this was the reason why our program was hanging for even small no. because exponential time complexity is very bad.

Code :-

```
public class FiboFormula {
    public static void main (String[] args){
        System.out.println ( formula (50));
    }

    static int formula (int n){
        return (int) ((Math.pow(((1+Math.sqrt(5))/2
                ,n)-Math.pow((1-Math.sqrt(5))/2
                n))/Math.sqrt(5));
    }

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

Q When you'll get equal no. of roots

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

I put $f(n) = \alpha^n$

$\therefore \quad \alpha^n = 2\alpha^{n-1} + \alpha^{n-2}$

$$\frac{\alpha^n - 2\alpha^{n-1} - \alpha^{n-2}}{\alpha^{n-2}} = 0$$

both LHS & RHS

$= \alpha^2 - 2\alpha + 1 = 0$

$\therefore \alpha = 1$ ~~and~~ on both. (double root)

So, In such case what will happen is that if these root is repeated twice. so, let's say in the general case:-

* general case :-

If α is repeated 'r' times then, α^n, $n\alpha^n$, $n^2\alpha^n$ ... $n^{r-1}\alpha^n$ all are solutions to the recurrence.

Note:-

So, we know that if the number of roots are 2, so we know that there should be two roots but, we're getting only 1 root So, we can take extra roots from here because these are also a sol^n. we just multiply n, as we know that n divides n.

Hence, we can take two roots one is 1 and the other can be $n \cdot \alpha^n$.

as we know that $\alpha = 1$

$\therefore \quad 1, n$

putting in formula:

$$f(n) = c_1 (1)^n + c_2 n \alpha^n$$

$$= c_1 + c_2 n$$

let's say the ans given us is

$f(0) = 0$ & $f(1) = 1$

$\therefore f(0) = 0 = c_1$.

$\therefore c_1 = 0 \quad -(1)$

$\therefore f(1) = 1 = c_1 + c_2$

$\therefore c_2 = 1 \quad -(2)$

So, if we put eq (1) & (2) together,

$f(n) = n$.

$\therefore$ Time complexity of the above relation is $O(n)$.

* Homogenous linear recurrence :-

Q What does how homogenous means ?

Ans:- The form of a recurrance relationship where we do not have a particular other function like g(x).
So, there is no seperat function that is why it is known as homogenous.

Eg: Solving Linear Recurrance.

** Non - Homogenous Linear recurrences :-

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + a_3 f(n-3) + \ldots + a_d f(n-d) + \underbrace{g(x)}_{\neq 0}$$

So, when this extra function is present then it is known as Non homogenous linear recurrences.

Q How to solve ?

Steps:-

1) Replace g(n) by 0 & solve usually.

Eg:- $f(n) = 4f(n-1) + 3^n \quad , f(0) = 1$

$\therefore f(n) = 4f(n-1) + 0.$

$\alpha^n = 4\alpha^{n-1}$

$\therefore \alpha^4 - 4\alpha^{n-1} = 0$

$$\alpha - 4 = 0$$

$$\therefore \alpha = 4$$

∴ Homogeneous soln :-

$$f(n) = C_1 4 \alpha^n$$

$$f(n) = C_1 4^n \quad - (A)$$

$$\therefore f(n) = C_1 4^n + 3^n$$

2) Take g(n) on one side and find particular soln.

$$\therefore f(n) - 4f(n-1) = 3^n \quad - (1)$$

g(n) - as per the step

Q What is particula solution?

Ans: We ~~knee~~ need to do guess something that is similar to g(n), this is known as particular soln.

Eg: If $g(n) = n^2$, then guess a polynomial of degree 2.

Kunal's guess :-

$$f(n) = C\, 3^n \quad - (2)$$

put in eqn (1)

$$\therefore C3^n - 4C3^{n-1} = 3^n$$

$\therefore C = -3$ — (a)

$\therefore$ the particular sol^n $\therefore$ put (a) in (2)

$$\therefore f(n) = -3 \times 3^n$$

$$\therefore f(n) = -3^{n+1} \quad - (B)$$

3) Add both the sol^n together

$$f(n) = C_1 4^n + (-3^{n+1})$$

$f(1) = 1$ — {we know}

$$C_1 4 - 3^2 = 1$$

$$\therefore C_1 = \frac{5}{2} \text{ //}$$

put the value of C_1 * in Orignal eq^n.

$$f(n) = \frac{5}{2} 4^n - 3^{n+1} \text{ //}$$

Short:

1) Replace $g(n)$ by 0 & solve usually.
2) Take $g(n)$ on one side & find particular sol^n
3) Add both the sol^n together

Abbreviation :-

1) First, put $g(n) = 0$ & take the normal sol^n then guess the particular sol^n and get the answer for that.

2) After that put $g(n)$ on one side, guess the particular solution. put that sol^n in the eq^n where you $g(n)$ one one side you'll get the value of 'c' and then you can put it in the orignal eq^n so, got your particular sol^n.

3) So, doing this we can get 'c' & then find answer of a particular sol^n.
Then just add the (1) eq^n with the particular eq^n and put it the ans ("which is already provided") use it then you'll get your orignal answer.

Q How do we guess a particular solution?

Ans:-

1) If $g(x)$ is exponential, guess of the same type.

Eg: $g(n) = 2^n + 3^n$

guess: $f(n) = a2^n + b3^n$

So, this is your particular solution.

2) But if it is polynomial, $g(n)$, in that case guess of same degree.

① Eg:- $g(n) = n^2 - 1$

then guess should be of same degree. Here it is 2 so, 2.

guess:- $an^2 + bn + c = f(n)$

② Eg:- $g(n) = 2^n + n$

guess:- $f(n) = a2^n + (bn + c)$

Note:-

1) If it's exponential just multiply with the constant.
2) OR, if it's a polynomial take eq^n of that degree.
So, that's how you guess the particular sol^n.

3) let say you guessed, Eg:- $g(n) = a2^n$ & it fails, then try $\underline{(an + b)2^n}$, if this also fails increase the degree.

$\therefore (a^2n + bn + c)2^n$

Keep trying.

Eg: $f(n) = 2f(n-1) + 2^n$. $f(0) = 1$

① put $n\,2^n = 0$

$\therefore f(n) = 2f(n-1)$

put $f(n) = \alpha^n$

$\therefore \alpha^n - 2\alpha^{n-1} = 0$

$\alpha = 2$ //

② guess P.S

$g(n) = 2^n$

guess: $f(n) = a2^n$

put it in main eqn

$a2^n = 2a2^{n-1} + 2^n$

$a = a + 1$ ✗ wrong

Hence, guess another one from our rules because this is not working

So, $f(n) = (an+b)2^n$

$(an+b)2^n = 2(a(n-1)+b)2^{n-1} + 2^n$

$an + b = an - a + b + 1$

$\therefore a = 1$

discard b.

$f(n) = n 2^n$

our particular soln

3) General answer :-

$f(n) = c_1 2^n + n 2^n$ - ∫ some of both the eqns.

$f(0) = 1 = c_1 + 0$

$c_1 = 1$

$\therefore f(n) = 2^n + n 2^n$ //

Complexity = $O(n 2^n)$

* Little - Oh Notation : o(n)

- As we know that the Big - Oh notation was giving the upper bound this Little - oh notation also gonna give the upper bound only.

But, what is the distinct? This is not strict upper bound

It is a loose upper bound

Big - Oh	Little - oh.
1) if $f = O(g)$	$f = o(g)$
it means that the growth of f is no faster than g	it means that it is more like a smaller then g
$\therefore f \leq g$	$\therefore f < g$
	It is strictly slower than g.
	It is more stronger statement

Maths

If f is strictly slower than g, then you can say neumerater is slower than denominator, it should give us 0

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

$$f = N^2 \qquad g = N^3$$

$$\lim_{n \to \infty} \frac{N^2}{N^3}$$

$$= \lim_{n \to \infty} \infty \frac{1}{N} = 0.$$

* Little omega :-

As we know big omega was giving the lower bound. This little omega will also going to give lower bound. But it will not be tight. So, it's gonna be like loosely lower bound.

Eg:- ~~$f = \omega(g)$~~

$$f \lim_{N \to \infty} \frac{N^3}{N^2} = \lim_{N \to \infty} \quad N = \infty$$

Big Ω

$f = \Omega(g)$

this means

$f \geq g$

lower bound.

\$ It is giving a lower bound. It can increase more than that \$

little ω

$f = \omega(g)$

this means

$f > g$ (strictly greater) difference.

Maths:

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

Q When do use this?

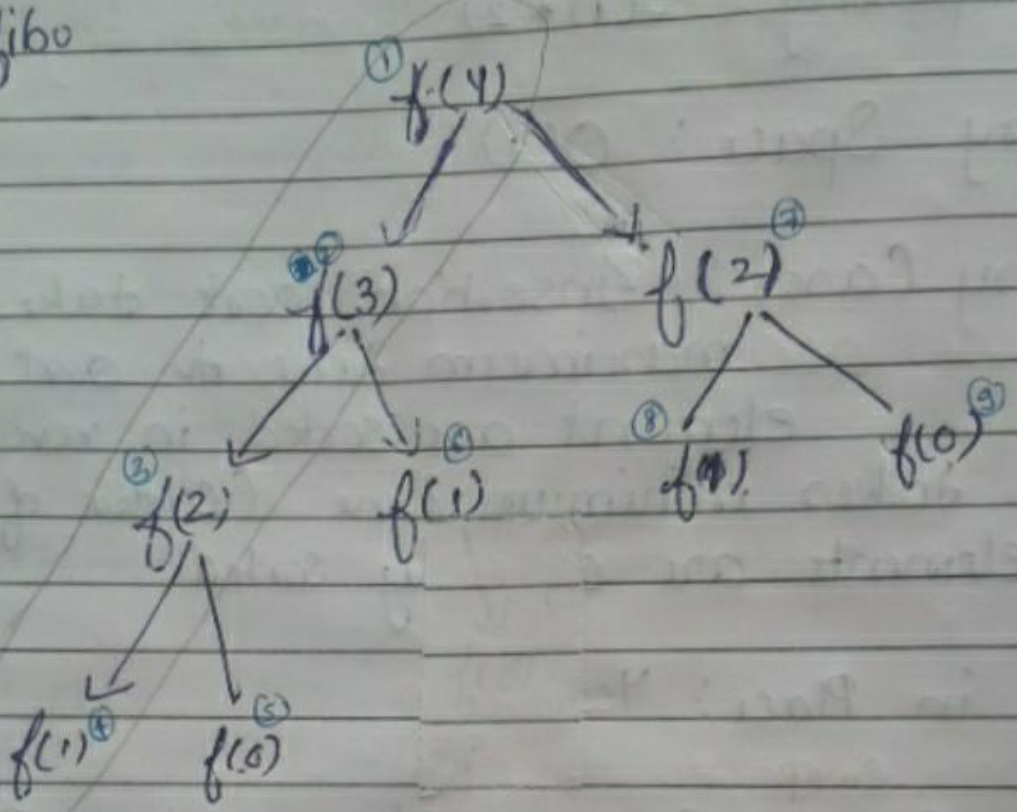
Ans

When you want something like strictly greater or strictly less than etc.

Note: In reality we use Big-Oh notations only.

* Recursive & algorithms

• fibo



O(n)

Q what is the space complexity ?

Ans:-

1) Sop Space complexity is not going to be constant in Recursive program because

a) In Recursion we know that functions calls are stored in stack.

b) So, those function call actually takes some memory in stack

2) Therefore Recursive program do not have constant space in complexity

3) So, the space complexity is actually the height of the tree

"" At any particular point of time NO two funct^n call at perform at the same level of recursion will be in the stack at the same time.

Points:- Space complexity of recursive programs:-

1) When we talked about the flow of recursive program do you think that at any particular level

eg:- ③ ↙ ↘ ⑥ ⑧ ↙ ↘ ⑨
f(2) & f(1) f(1) f(0)

is there going to be a possibility for more than one function calls in this level to be in the stack at the same time?

2) Can we say that f(2) & f(0) are present in stack at the same time? NO.
Because this f(0) not even executed until f(2) finishes.

3) Please refer the fig:-
So, this 7, 8, 9 will for eg, will only execute when the ans of 3 is given.

Trick:-
Only calls that are interlinked with each other will be in the stack & at the same time.
because the previous one will be waiting for the next one to execute, next one will be wait for the next one ..... and these should be interlinked together at the same time.

from the fig:-

① for eg. when the function call no. 7 will be executing fun^{n} call no 2, 3, 4, 5, 6 would have already been finished.

② So, we suppose we take function call no 7. then 2, 3, 4, 5, 6 would have been already executed. Why?
Because those are not linked with function call no. 7

③ At one particular level there won't be any more than one calls that are in the stack at the same time.

④ So, what is the maximum space that it'll be taking?
Ans: We know that the longest st chain starts from the root till the leaf.
So, the longest chain will be the answer.

∴ Space complexity = height of the tree or path

∴ On

⑤ So, what is the auxiliary space complexity required when we're calculating Nth fibonacci number?
Ans O(n)
Because these all will be in the stack at the same time. so, the maximum amt of space req^{d} at the time.

* Space complexity or Auxilary Space :-

1) Auxilary space :- It is the extra space or temporary space taken by an algorithm.

∴ Space complexity = input space + Auxilary space.

Suppose :-
Take an ip of array ~~size~~ size N & do something with it.
So the sap: Space complexity will be the input you're taking from the size N + extra space the algo is using
This is known as Space complexity.

- So, in the binary search the space complexity was constant. So, that means auxilary space was constant it was not taking any extra space.
- Taking 3 variable i.e Start, End & Mid.
- If the array is of size 100 or more than that. Every single time it's only going to take 3 that variable i.e start, end & mid.
Hence, constant

Q

```
for( i = 1 ; i ≤ N ; ' ){
    for( j = 1 ; j ≤ k ; j++) {
        // some operations that
           takes time at t
    }
    i = i+k
}
```

Here,

1) inner loop is running k times & for every time it's running it's taking t amount of time.

∴ $O(kt)$ time.

2) If inner loop is running once, once, so it's taking t amount of time. So, here it's actully running k times. Hence kt.

3) Ans :- $O(kt *$ times outer loop is running)

conditions :- here i will starts from 1, & the loop will break when i is ≤ N & i is incrementing with k

4) So let's say, $i_{last} = 1, 1+k, 1+2k, 1+3k \ldots 1+xk$
So, if the x^{th} value is 'xk' that means is & satisfy the condions. Hence x should be ≤ N

'1+xk is the value & x is the no. of times its running

$\therefore 1 + xk \leq N$

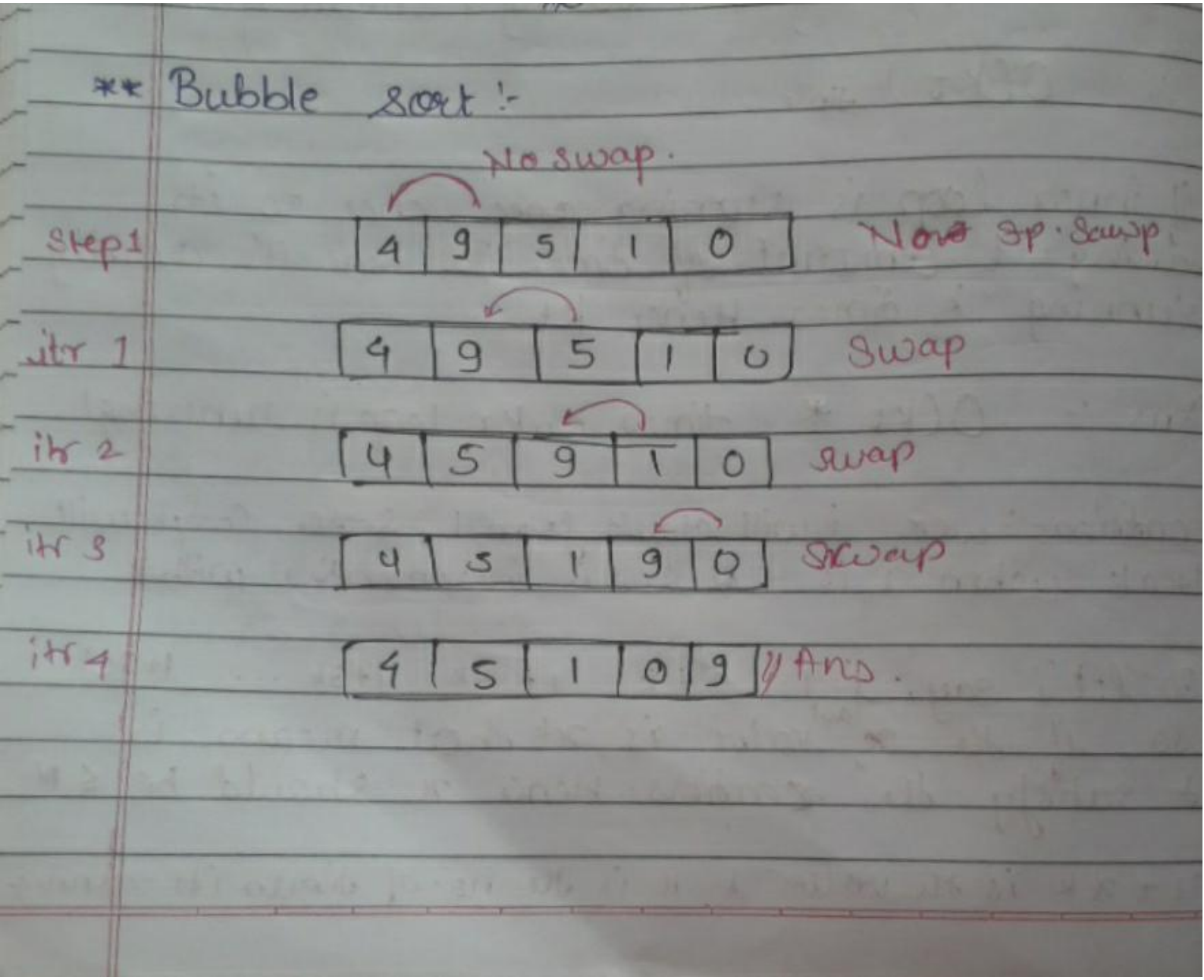
$xk \leq N-1$

$\therefore x = \frac{N-k}{k}$ — x = no of times the outer loop is running

Complexity : $O\left(kt * \frac{(N-1)}{k}\right)$ — const are removed

$\therefore = O(N * t)$

** Bubble sort :-


No swap.
step1
4 9 5 1 0
No sp. swap
itr 1
4 9 5 1 0
swap
itr 2
4 5 9 1 0
swap
itr 3
4 5 1 9 0
swap
itr 4
4 5 1 0 9
// Ans.

* Worst & Average case time complexity :-
O(n*n)
Worst case occurs when array is reverse sorted.

* Best case time complexity :-
O(n)
Best case occurs when array is already sorted

* Auxilary space
O(1)

* Boundary cases :-
Bubble sort takes minimum time (order of n) when elements are already sorted.

* Sorting in place :- Yes

* Stable :- Yes

** Selection Sort :

Worst complexity : n^2

Average complexity : n^2

Best complexity : n^2

Space complexity : 1

Method : Selection

Stable : No

Note: The good thing about selection sort is it never makes more than O(n) swaps & can be useful when memory write is a costly operation.

Eg:- (max) 4 , 5 , 1 , 2 , 3 — swap 5 & 3

4 , 3 , 1 , 2 , 5 — swap 4 & 2

2 , 3 , 1 , 4 , 5 — swap 3 & 1

2 , 1 , 3 , 4 , 5 — swap 2 & 1

1 , 2 , 3 , 4 , 5 ✓

** Insertion sort :-

Time complexity :- O(n * 2)

Auxiliary Space :- O(1)

Boundary Cases :- Insertion sort takes a maximum time to sort if element are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Sorting in Place :- Yes.

Eg :- 5 , 3 , 4 , 1 , 2

(Swap)

3 , 5 , 4 , 1 , 2

(sort & swap)

3 , 4 , 5 , 1 , 2

(sort)

1 , 3 , 4 , 5 , 2

(sort)

1 , 2 , 3 , 4 , 5 #.

Space & Time Complexity

1) What is time complexity?

Ans:

Functions that gives us the relationship about how the time will grow as the input grows

'OR'

As the input grows, times grows is known as time complexity

Now as an ~~Enpamp~~ Example :-

We have two computer

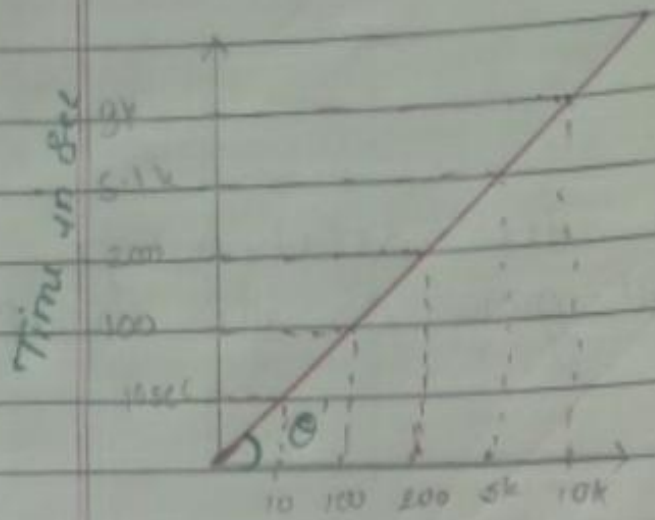
* We run an algo we have :-

	Old computer	M1 macbook (very fast)
data :-	1,000,000 elements in arr.	1000,000 elements in arr.
algo :-	linear search for target that doesn't exist in the array	Linear search -11-
Time taken	10 sec	1 sec.

Q. Which machine has a better time complexity in between these?

Ans :- Both of the machine have the same time complexity.

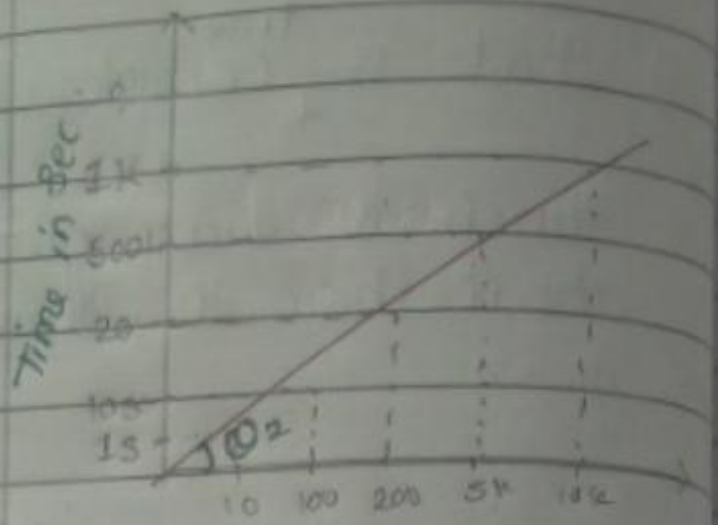
Time complexity != time taken.



Size of an array.

Eg:- old machine

1) Straight line.



Size of an array.

My Mack book.

1) steeper line with less slope.

Even though the time taken is different but the relationship between the size and the time is same "linear".

Over here, time is growing linearly as the size is growing. In both the case though values are different.

Q Why? & Why this relationship is important?

Ans:-

Linear search grows linearly. (N)
Binary search grows with logN

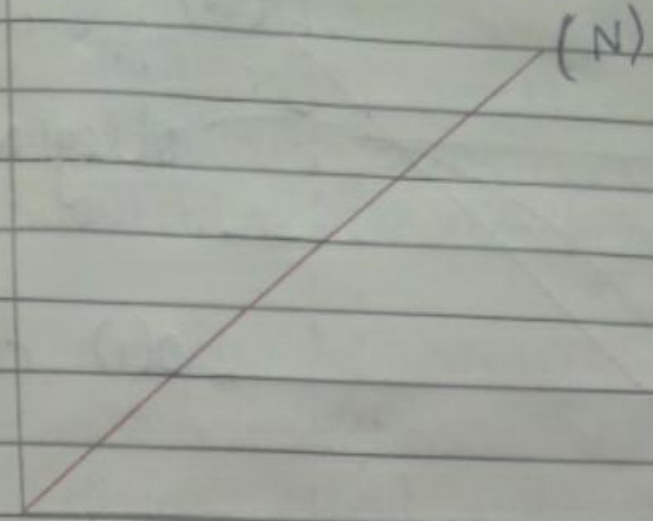


fig: Graph for linear search

Because the time complexity is linear

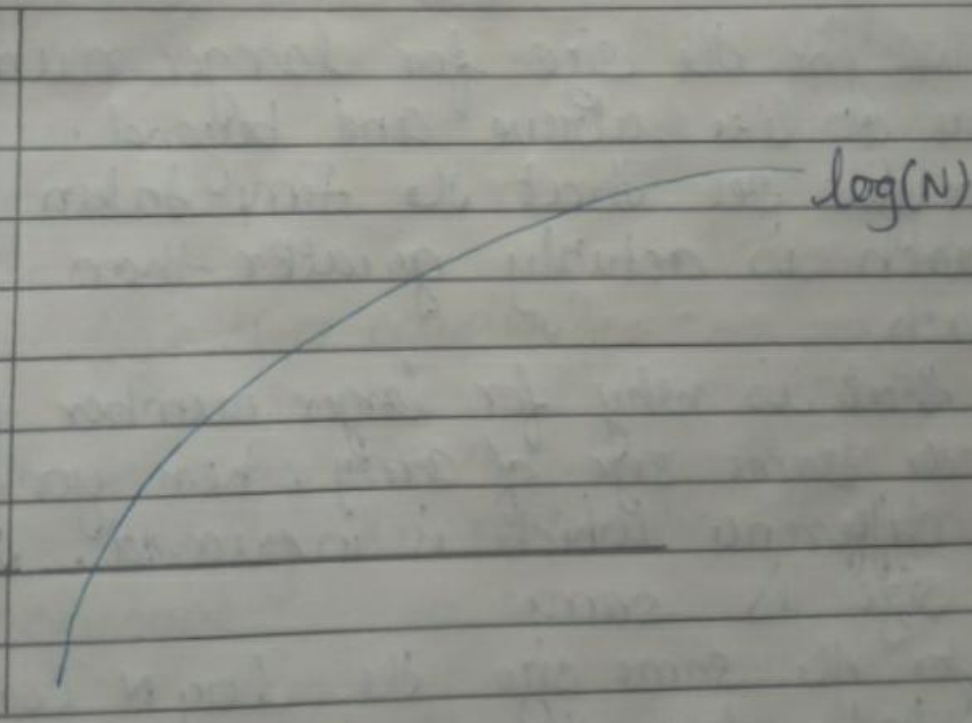
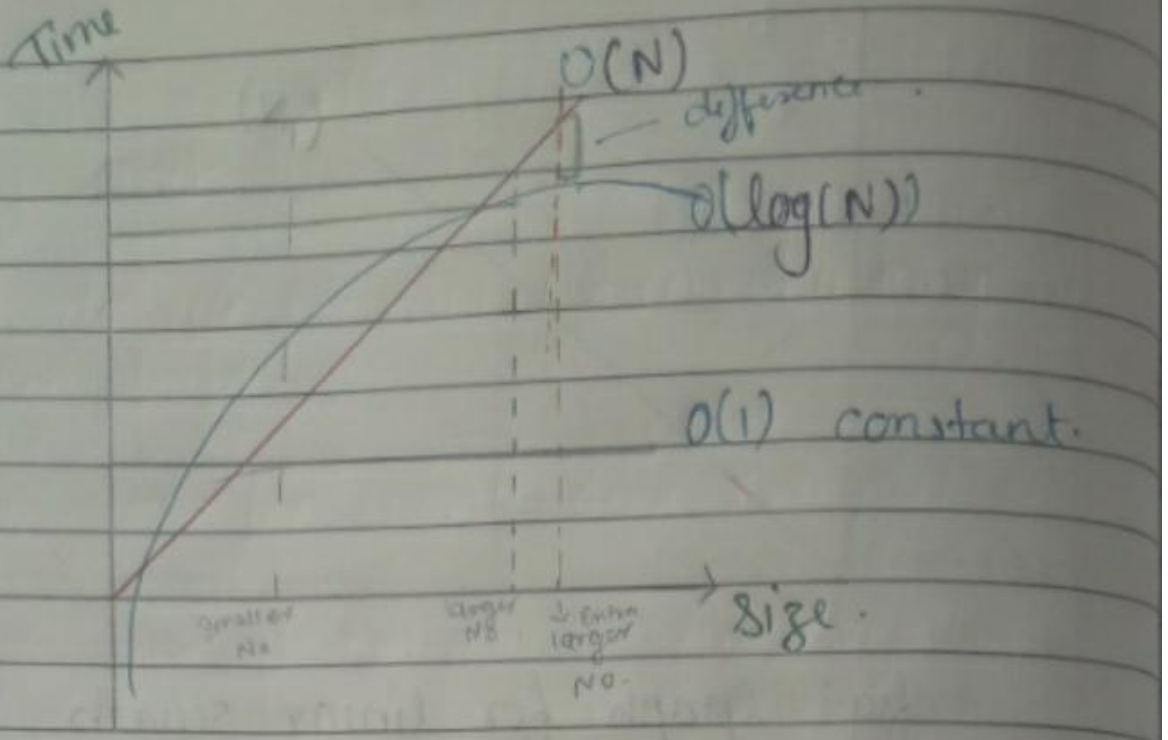


fig: Graph for Binary search.

This is growing 'log N' times.



Q Now let's Notice. Why does it matter?

Ans:

1) If we fix the size for larger numbers it may go like above and beyond.
2) Can you see that the time taken by linear search is actually greater than the binary search.
3) So, that is why for larger number. (fig :↑ graph) for the same size of array. here you can see the difference which is increased. though the the size is same
4) So, for the same size the log N complexity will take less time.
5) And, the linear complexity will take more time.

c) for smaller no. logn will take more time, linear less, O(1) will

Q So, which one is better?

Ans O(logn) is better because it is more efficient. So, that's why it matters.

Now let we talk a constant time complexity. So here does not matter what the size is time will always remain constant and for smaller number we don't care about smaller no.

Note:-

1) In time complexity always look at bigger numbers.
2) Always think about when your data will grow large in size in that case what will happen?

Ans:-

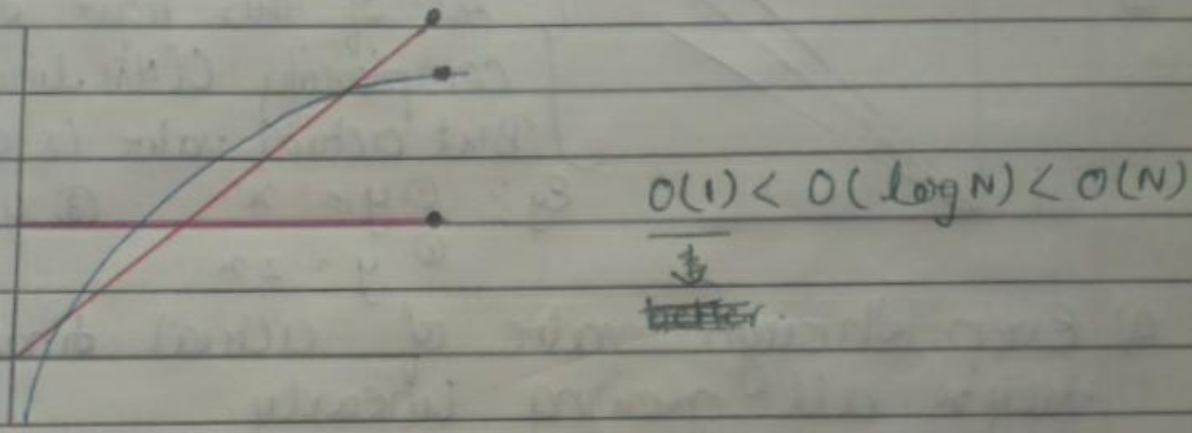


fig:- Time complexity.

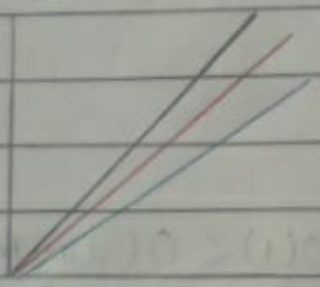
1) Now in the fig we can see that the (Red) linear is taking the most time, than log(n) then constant.

2) Thef Therefore, constant is always better than O(log(n)) & O(N)

* As you can see when the size was fixed for the same amount of data O(N) was taking the most time. then log(N) & last O(1).

Q. Which one is better?
Ans: Binary search O(log n).

* Q. What do we consider when thinking about the complexity?

Ans:
1) Always look for worst case complexity
2) Always look at complexity for large / ∞ data
3) 
All of this have the same complexity O(N) ie linear.
But actual value is different
Eg:- ① $y = x$ ③ $y = 4x$.
② $y = 2x$

a) Even though value of actual time is different they're all growing linearly.

b) "We don't actually care about what the time taken is" because that will vary from machine to machine".
We only care about the relationship of how the time will grow, when the input grows.

Q. Do we really need to worry about these constants then?

Ans: No, we only care about how its growing.

c) This is why, we ignore all constants.

4) Always ignore less dominating terms.

Okay :-

let's say you've complexity of

$O(N^3 + \log(N))$

So, from point 2. { Always look at complexity for large / ∞ data }.

∴ if we take 1million ~~time~~ amt of data.

∴ $N = 1\,mil.$

∴ $= ((1\,mil)^3 + \log(1\,mil))$

$= (1\,mil)^3\,sec + \underbrace{6\,sec}$

It is very small

So does this 6 sec as compared to $(1mil)^3$ sec has any significant.

Hence, ignore it.

Eg:- $O(3N^3 + 4N^2 + 5N + 6)$

- (Ignore the constants)

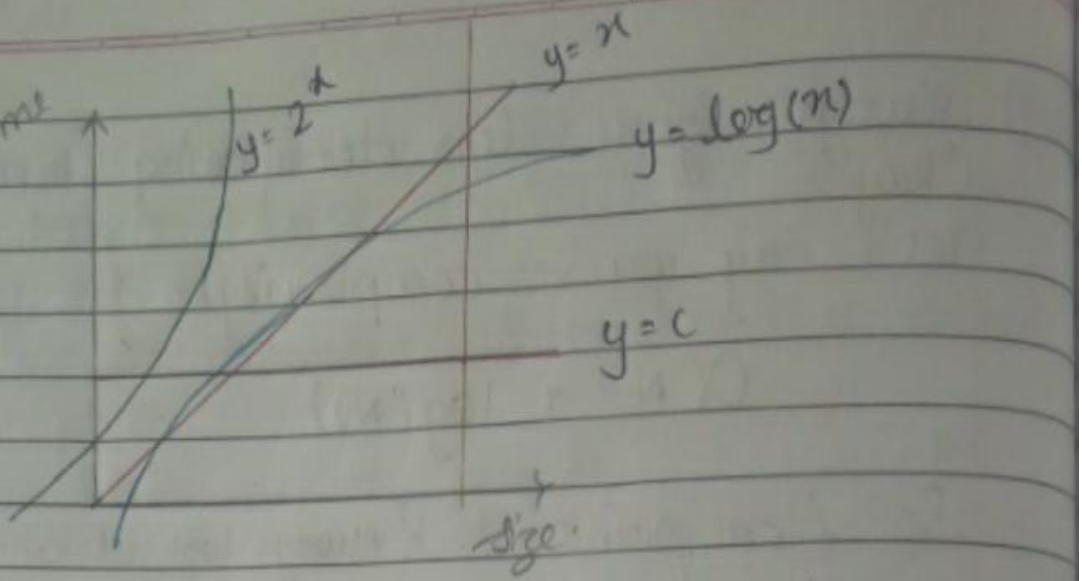
$= N^3 + N^2 + N$

- (Ignore the less dominating term)

$= N^3$

$\therefore O(N^3)$.

So, $O(3N^3 + 4N^2 + 5N + 6) = O(N^3)$.



1) $y = c$ is always be constant so, it will always be less than whatever value I you provide. So, $y = c$ will always be best optimized.

2) $\log(x)$ so, if we take some large amount of data, then
for same amount of day $y = c$ will take less amount of time.
After that $\log(x)$.
then the 'x' which will be little bit more than then as you can see 2^x which is very very poor complexity. (which is exponential complexity).

3) For such a small amount of data, time limit has exceeded alot: (2^{20}), which is not even visible on the graph.
Eg: fibonacci like for such a small amount of data time has exceeded alot, that is already not visible on the graph ∴ this is bad.

$$O(1) < O(\log(N)) < O(N) < O(2^N)$$

There other complexity also like $O(n \log n)$, $O(N^2 \log N)$ etc

Bitwise operators + Number Systems

- When we write int a = 10 ; (in any language) internally whatever the things that's you're doing. all the servers that're running, etc. So internally it's all a bunch of 0s1.
- Computers understands only binary language (0 & 1)
- We don't code in 0 & 1 because it will become very complex. So, that's why we have programming languages.
- So, that's how numbers are stored internally this is known as no system.
- There are various no. systems.
  - base 10, base 8, base 16, base 2

Q. What "base" represents?

Ans: Base basically means How many numbers do we have in the particular base concept.

* We have been studying since we're kids: decimal no system (base 10).

- Base 2 : It means binary no. system. This is what the computer understands.

Now we can imagine 0 as false And 1 as True

wise.

Bit operators :-

1) And :-

(In words) :- Here let's say if you've 2 no, all of them should be true.

Truth-table :-

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1 - (all are true)

Conditions :-

AND conditions means that all the numbers or input should be true, then only the entire expression will be true.

If one of the input is flase then the output will be false only.

If we're by saying of output of A & B & C then all the inputs should be true or equal to one (1)

In computer science we can say that 1 represents True and 0 represents False.

electronic :-

If 1 & 1 all the gate will be open so there will be flow of current, if 1&0 or any thing other it won't. AND gate is a basic logic gate.

Note:
(i) When you AND 1 with any number, the digits will never remains the same.

(ii) Eg:
```
  1 1 0 0 1   - same
& 1 1 1 1 1
-----------
  1 1 0 0 1   - same
```

3) OR:
in words :- It is opposite of AND gate. If any one of the input is true then the entire expression is true.

Truth-table :-

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Condition:-
Even if both the inputs are true output will be always positive

3) XOR (^) :- (If and only if)
In words :- It is exclusive OR. Here if you've two no. only one of them should be true and anything else will false as a output.

Truth-tables

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Condition:- If more than 2 no then, the odd no should be true.

Observation :-

1) $a \wedge 1 = \bar{a}$

Note :- $\bar{a}$:- it is basically a complementry bar that means of opposite of that no.

Eg :- ① $0 \wedge 1 = 1$

② $1 \wedge 1 = 0$

So, anything we X-OR with '1', you will get the answer opposite of that.

2) $a \wedge 0 = a$

3) $\bar{a} \wedge a = 0$

Eg :- $32 \wedge 32 = 0$

If XOR the same input the output will always be zero.

1) Complement ($\sim$) :-

Suppose :- $a = 10110$

$\bar{a} = 01001$

A	B	C	D	o/p	
0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	10 →	A
1	0	1	1	11 →	B
1	1	0	0	12 →	C
1	1	0	1	13 →	D
1	1	1	0	14 →	E
1	1	1	1	15 →	F

** Nubnber system :-

1) Decimal :- 0 to 9 & it's of base 10.

Base 10 means there are 10 nos. which we can use to represent any ~~nub~~ no. in decimal form.

Eg:- $(357)_{10}$, $(10)_{10}$

2) Binary :- 0 & 1 & it's of base 2.

Eg:- ① $(10)_{10} \rightarrow (1010)_2$

② $(7)_{10} \rightarrow (0111)_2$

3) Octal → 0 to 7 base 8

Now comparing Decimal & octal. In octal we cannot 8 & 9 because we can only use the nos of from 0 to 7.

Decimal :- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
10, 11, 12 ...

Octal :- 0, 1, 2, 3, 4, 5, 6, 7, 10,
11, 12, 13, 14, 15, 16, 17, 20 ...

So, 8 in decimal is 10 in octal & 9 in decimal is 11 in octal.

4) Hexadecimal :- 0-9 & A-F base 16

Eg :- $(10)_{10} = (A)_{16}$
$(12)_{10} = (C)_{16}$

* Conversions :-

Note:

1) Decimal ~~tens~~ to base b

Q Convert $(17)_{10}$ to base 2

pt : Q keep dividing by base. take remainders, write in opposite.

2	17	
2	8	1
2	4	0
2	2	0
	1	0

$(1001)_2$

$\therefore (17)_{10} : (1001)_2$

So, when you write int a = 17 computer will store it like (1001).
Each ram has their individual cells.
So in ram it will be stored something like

a → [1 | 0 | 0 | 1] & a will be pointing towards it the cell.

Q $(17)_{10} : (?)_8$

8	17
	2 1

$\therefore (21)_8$

$\therefore (17)_{10} : (21)_8$

② Base b to decimal

① $(10001)_2 \rightarrow (?)_{10}$

Steps:

① Multiply & add the power of base with digits.

1	0	0	0	1
2^4	2^3	2^2	2^1	2^0
$2^4 \times 1$	$2^3 \times 0$	$2^2 \times 0$	$2^1 \times 0$	$2^0 \times 1$

$= 16 + 0 + 0 + 0 + 1$

$= 17.$

$\therefore (10001)_2 = (17)_{10}.$

② $(21)_8 = (?)_{10}$

2	1
8^1	8^0
$8^1 \times 2$	$+ 8^0 \times 1$

$= 16 + 1$

$= 17$

$\therefore (21)_8 = (17)_{10}$

If you want to double the no.
Just << by 1.

continuing with operators:-

⑤ Left shift operator (<<) :- It shifts all the bits towards the left side. (one by one).

Eg:- $(10)_{10} = (1010)_2$.

a 10 << 1 - It basically means that shift i/p by 1 ;
i/p

Step I: Convert it internally it into binary no.

∴ 1010 << 1

Step II Okay. Shift it towards the left. (one by one.

∴ 1 0 1 0 << 1 = 1 0 1 0 0

Step III After shift the to the left is we will require one extra no. So whenever we need an extran no in left shift operator we add 0. as above method.

∴ 1 0 1 0 0

$= 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

$= 16 \times 1 + 0 + 4 \times 1 + 0 + 0$

$= 16 + 4$

$= 20$

Hence, any no. "a << 1 = 2a" means it's going to double that no.

General point :- If you left shift a no. "a" "b" times it's going to multiply $a \times 2^b$.

= $a << b = 2b$ I $a \times 2^b$

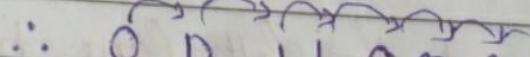
⑥ Right shift operator :- " >> "

It is opposite of left shift operator.

It moves the given i/p towards the right. Consider

① 00110011 >> 1

This question basically says that move the given i/p by 1, and by moving them one by one towards the right. we've to discard the 1.



∴ 0 0 1 1 0 0 1 ⊗ >> 1

= 00 1 1 0 0

Note:

- Similarly like in decimal no. system when the no like $(000123)_{10} = (123)_{10}$.
- Here the leading zeros are ignored
- This is same for all of the no. system. Even though because the zero from left hand side will be ignore, so the values will remained unchanged but we can't do that from right side as it will change the whole value.

∴ = 1100

Note:

In binary 1+1 = 10 & 1 is carry so ans = 0.

Eg :- $\begin{array}{r} 10 \\ +\ 11 \\ \hline 100 \end{array}$

Note: In the right shift operator we're dividing the no by 2.

$$a >> b = \frac{a}{2^b} \quad \text{- (general point).}$$

//

* Working :-

a. Given a no n find if it is odd or even.

Note:

① When you AND a 1 with any no. the digits remain the same.

② We know that any no, whatever calculation we do internally it will be calculated as a binary no. even if you do substraction, addition, etc.

Eg

$12 + 7 \rightarrow$

$$\begin{array}{r} 1100 \\ +0111 \\ \hline 10000 \end{array} \qquad \begin{array}{r} 1100 \\ +0111 \\ \hline 10011 \end{array}$$

$(19)_{10} \rightarrow (10011)_2$

$$\begin{array}{ccccc} 1 & 0 & 0 & 1 & 1 \\ 2^4 & + 2^3 & + 2^2 & + 2^1 & + 2^0 \end{array}$$

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

$$= 16 + 2 + 1$$

$$= 19 \text{ //}$$

Note :-

Every no. Eg :- 1001(1) leaving this, every other is a power of 2.

Hence this is the no. whether it determ because we know that leave the last one ~~ever~~ entire no. will be even (always) Why? Because all of those these are power of 2.

the last no will always be a^0. and anything positive no. raise to 0 is 1.

Hence, if this particular no is 1 it means that the answer to everything is +1.

if 2^0 place == 1 → e no. is odd.
otherwise no. is ~~evel~~ even.

find out what last digit is. if it is 1 then the no. is odd & if it is 0 then the no is even.

Q And the no & get the last digit.

```
  100101
And'ed 000001
--------
  000001
```

(1)

Hence, odd no.

Sum up:-

n & 1 == 1 → odd else even

Code

```
public Clas EvenOdd {
    public static void main (String [] args){
        int n = 67;
        System.out.println (isodd(n));
    }
    private static boolean isodd (int n){
        return (n & 1) == 1;
    }
}
```

Note:-

- Eg :- 0 1 1 0 0 0 [1] → it is known as LSB (Least Significant Bit).

1. ~~Your~~: you're given an array of numbers and in that array every number appears twice only one number appears once. you're to find that no.

arr = [2, 3, 4, 1, 2, 1, 3, 6, 4]
→ ans

So How'll you can find His?

Note:- Bit wise operators like in normal maths operator they also follows the associative properties

① eg :- (5 * 3) * (5 * 4) = 5 * 5 * 3 * 4 or 5 * 4 * 3 * 5, etc

So the order basically does not matter

② 2 ^ 3 ^ 3 ^ 4 = 2 ^ 4 ^ 3 ^ 3, etc.

We wanna do it in O(1) & in one single pass

Note:
As we know any no ^ with the same no.
a ^ a = 0 & 0 ^ a = a

So, if ^ of the entire array the ans. I know the all the duplicates will lead to zero

Ans: XOR all the numbers

time complexity: O(n)
space complexity: O(1)

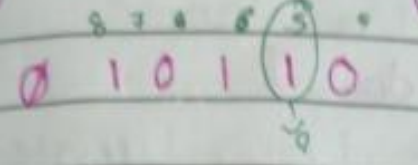
2Q. arr = [-2, 3, 2, 4, -5, 5, -4]
ans: 8

Note: ORDER does not matters

#Code:

Ans
```
public class Unique {
    public static void main(String[] args){
        int[] arr = {2, 3, -3, -2, 6, 5, -5};
        System.out.println(unique(arr));
    }

    private static int unique(int[] arr){
        int ans = 0;
        for(int i : arr){
            ans ^= i;
        }
        return ans;
    }
}
```

Q. find i^{th} bit of the number?



Ans: So we know that we were able to find How do find LSB or right most bit we just And it & 1
So And the particular digit with 1, it will give 1 no matter what no. rest And will be 0.

∴ Ans:-

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \\ \&\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \quad \text{- Mask} \\ \hline 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

Q. How do get 00010000?

Note

Mask is a sp. seperate no. or entity you've that allows us to get our answer related to it.

n = mask with n-1 zeros. Q. How do we do that?
So we need n-1 zeros in the right hand side
Q What do we need to do in order to move the 1 towards the left hand side get zeros over there
Left shift (1 << (n-1))
∴ 1 << 4 = 10000

Ans: n & (1 << (n-1))

Q. Set the i^{th} bit

Set means: turn it to 1

so if i^{th} bit 0 make it 1
if i^{th} bit 1 remain 1 only.

1 0 1 0 1 1 0

let's say you want to set the 4th bit.

Note:

0 ^ 1 : 0 OR 1 = 1

1 0 1 0 1 1 0
OR 0 0 0 1 0 0 0 - {Mask}
1 0 1 1 1 1 0

Q. Reset i^{th} bit

1 0 1 0 1 1 0

Reset means: if it's 1 make it 0
if its 0 remain 0.

Note: If we And any no with 1 we'll get the no itself but if we And any no. with 0 we'll get 0 only
0 & 1 = 0

1 0 1 0 1 1 0
& 1 0 1 0 1 1 1 - {Mask}
1 0 1 0 1 1 0

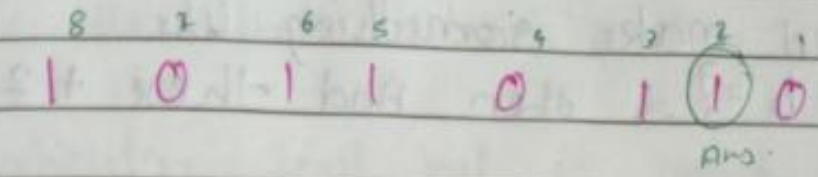
Q So how did get that Mask?

Ans By complement

$$Mask = \overline{(1 << (n-1))}$$

Q Find the position of the right most set bit?

Eg:



Looking from the right hand side which is the set bit?
The first 1 that occurs from the right hand side.

∴ Ans = 2

1 0 1 1 0 1 1 0

↓

If we're writing this no. in the form of if we do this one right most bit.

n = a 1 b

Or

1 0 1 1 0 1 1 0 0

a b Ans = 4

n = a 1 b

a = 1 0 1 1 0 1

b = 0 0

$1011011\ (1)\ 00$

a b

As we know 100 is our answer so we're not touch touching them.
And we have to create 101101 all of them as 0
we need make something like
$\bar{a}\ 1\ b = ?$ & then And these + 2.

in terms of formula it's equal to $-n$

So, let's say

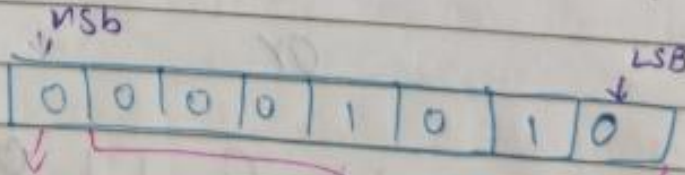
$-n = \bar{a}\ 1\ b$

Ans : $n \& (-n)$ $\because a \& \bar{a} = 0$ //

* Negative of a number

1 byte = 8 bits
So we can store 8 nos, 8 time 0 or (0, 1) in one byte.

Eg: 10 :



-10 = ?

Note:-

① MSB : Most Significant Bit [If no. is positive or neg
1 = neg & 0 = positive

② LSB : Least Significant Bit. (If no is even or odd)

Note:

- MSB is the reserved bit because if it is stored with 1 that means it is a negative no. or 0 then that means it is a positive no.

- Size of an it integer = 4 bytes. = 32 bits
So, Here the first bit is gonna be either 1 or 0.
That particular first bit is going to represent whether the no. is positive or not.
And rest of the 31 bits thats going to represents the value of the number.

- ~~Two~~ 2's Compliment

Steps

1) Take the compliment of the given input

2) Add 1 to it.

Eg: $(10)_{10} = (00001010)_2$

∴ ① 11110101

② 11110101 + 1 = 11110110

∴ Ans = $(-10)_2 = (11110110)$

This is how you calculate negative because in binary we can not apply - symbol.

Q Why does 2's complement gives negative of a number?

Ans:

- As we know when the size is 1 byte (8 bits) and you're bay having an additional bit, it that's going to get ignored.
- Eg: 1 0 1 0 1 1 0 1 1 1
  Here as we know that 1 byte is of 8 bit only so it will discard the starting 10 because of the space.
- As we know that we substract 0 from a number ans always gonna be in negative.
  Eg2: 1 0 0 0 0 0 0 0 0 0 → As this should give neg 10
  - 0 0 0 0 1 0 1 0

As since we're storing this neg no in the size of 1 byte if add additional on in the begining of extra 0 (1) that won't really matter because that's gonna get ignored either way.
because the size will not ~~swallow~~ allow ~~that~~ them.

- If we are only allowed to add 8 bits, 9 or more than that will get discarded as the limit is of 8 bits.
- We can see that 1 0 0 0 0 0 0 0 0 is a power of 2, if there is one single 1 is available in the binary representation no rest everything is 0 that means it is of power of 2.
- Eg: 8

∴ 8 = 1000

= 7+1 = 0111 + 1

∴ 0111

+ 0001

1000

or 16 = 15+1 = 1111 + 1

1111

+ 0001

10000

• so we can write this in entire thing as

eg: 100000000
- 00001010

as: 100000000 = 11111111 + 1

so the question reduce to

11111111 + 1 - 00001010

(rearrange) 11111111 - 00001010 + 1

complement why?

```
  11111111
- 00001010   } complement
  11110101
```

(1-0=1, 1-1=0)

No 2's complement's step 2: +1

* Range of numbers

1 byte : 8 bits : $2^8 = 256$

so total 256 no's we can make.

| 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 | 0,1 |

2 x 2 x 2 x 2 x 2 x 2 x 2 x 2

= 256.

Total numbers of unique no that we can store in a datatype of size 1 byte is 256.

So, actual no will be 0 to 7 bits

total bits n-1 = 7 bits.

Total no we can be make from 7 bits is

$2^7 = 128$

∴ 128 no in negative & 128 in positive
But even 0 counts & as we know neg of 0 is not
∴ -128 to 127 (range).

→ Range formula :- for n bits.

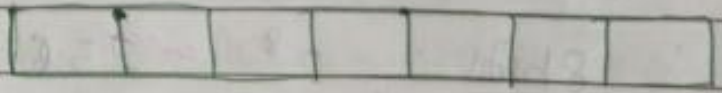
-2^{n-1} to $2^{n-1} - 1$

① Every no. is appearing 3 times & 1 no. is appearing only once : find that no.

arr = [2, 3, 2, 2, 7, 7, 8, 7, 8, 8]

idea :- we know every no is appearing 3 times than the values of its bits will also be appearing 3 times

if we take an empty array :-



2 :- 10

Basically this array says that all the bits that're set means that are 1. just keep that adding those in this set array.

so if we're adding 2 over here and then we are adding 2 again in this means that the bits available over there is it just going to added as 1.

2 :- 1 0
2 :- 1 0
2 :- 1 0
3 :- 1 1
7 :- 1 1 1
7 :- 1 1 1
8 :- 1 0 0 0
7 :- 1 1 1
8 :- 1 0 0 0
8 :- 1 0 0 0

So if we just count a total no. of bits then it'll become

1					3	3	7	4	

Imagine there was no. 3. So ↑ this will be something like | 3 | 3 | 6 | 3 |.
So obviously if 3 was not in there array it means the every single no is appearing 3 times means their set bits are also going to be appearing 3 times. So the set bits are going to be multiple of 3.
Hence the entire no is going to be divisible by 3, every digit of this no. in the array is going to be multiple of 3. one of these no.s
But here we're say is let's say add that is some extra, So some of the digits may not be multiple of 3. will be having 7 & 4 because of 1 & 1

So if we just take the modulo "%. 3" (the no which is repeating)

$\therefore$ %. 3 = 0 0 11 = 3 // ans.

an extra no. that is our answer "Everything that was appearing 3 times their set-bits will also appear 3 times Hence, the sob set-bits will be % of 3 "i.e remainder of 3 will be 0. But if there is one extra no. i.e. 1 actually addition to the 3 multiples that we have so that won't be a multiple of 3.":

Note:

Not just of for 3, we can do this for any Eg:- for 3, 5, 7 ..... do it like-wise.

Answer:-

Q. find the n^{th} magic no.

Eg:- 1) 1 = 0 0 1 = 5 (magic no.)
$5^3 + 5^2 + 5^1$

2) 2 = 0 1 0 = 25
$5^3 + 5^2 + 5^1$

3) 3 = 0 1 1 = 30
$5^3 + 5^2 + 5^1$

4) 4 = 1 0 0 = 125
$5^3 + 5^2 + 5^1$

let's say n = 6

① convert the n into binary. ∴ = 110
② We're gonna start 5⁰ and keep on multiply these with these no. & then add them.
③ Now, we have to basically check what is the last no. & we can do that with & of 1. (And of 1)

∴ n & 1 = This will give last digit in binary representation

Note:
We don't really have to convert "n" in binary. We can directly do n & 1 it will automatically converts in to binary internally.

Okay now we've 0 so leave & move to next by right shift 1
check if that 1 then proceed.

by doing n >> 1 these will go in loop

So we'll first take
n & 1 * which will give us 0 * 5⁰
n >> 1 + ...
&
∴ 0 * 5⁰ + 1 * 5² + 1 * 5³ + ~~0 * 5⁴~~

Code :-

```
pubic class MagicNumber {
    public static void main(String[] args){
        int n = 6;

        int ans = 0;
        int base = 5;

        while (n > 0) {
            int last = n & 1;
            n = n >> 1;
            ans += last * base
            base = base * 5;
        }
        System.out.println( ans);
    }
}
```

Q find no. of digits in base b.

Eg :- $(6)_{10} = 1$

$(6)_{10} = (110)_2 = 3$

formula :

No of digits in base b of no. n =
int $(\log_b n) + 1$

~~log a b~~

$$\log_b a = \frac{\log_x a}{\log_x b}$$ //

exp: $\log_b a = x$ ... (a)

$\Rightarrow a = b^x$

similarly

$\log_2 6 = x$

$6 = 2^x$ so this base thing ↑ represents the no of digits in the

Eg: $\log_2 10 = 3.32$

$10 = 2^{3.32}$

this basically means that how many times the 2 has been multiplied to form 10.

so if we take an int value of (3.32) & and 1 to it then it will gives us the no. of digits.

Q How many no. of digits are there in the binary representation of "10"?

Ans: $\log_2 10 + 1$

Code :-

```
public class NoOfDigits {
    public static void main (String[] args){

        int n = 34567;
        int b = 10;

        int ans = (int)(Math.log(n) / Math.log(b));
// if we want to convert anything to base b, just divide it by the same log of that with b.

        System.out.println(ans);

    }
}
```

meaning :-

$$\log_b a = \frac{\log_x a}{\log_x b}$$

Time complexity :- O(n)

Pascal's Triangle

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
```

find the sum of n^{th} row?

Ans:

sum of each row = sum of all the no.

$\downarrow$

$^nC_0 + \cdots$

i.e

sum of each row = $^nC_0 + {}^nC_1 + {}^nC_2 + {}^nC_3 \cdots + = 2^n$

for n^{th} row, sum = 2^{n-1}

$\therefore 1 << (n-1) = 1 \times 2^{n-1}$

Q find out the given no. if it's power of 2 or not

"Here only 1 bit will be there that will have 1 rest everything will be 0."

Eg:- ① 100000

It is a power of 2

② 100010

It is not a power of 2

Because here we have 2 1's. but only 1 '1' should be there. Rest everything should be 0.

So that's how we figure out the given no is of power of 2 or not

as we know that

$$100000 = 11111 + 1 \quad (n-1)$$

And it ⓪

$$\begin{array}{r} 100000 \\ \& \; 011111 \\ \hline 0 \end{array}$$

Eg:- ②

$$\begin{array}{r} 10010 \\ \& \; 01111 \\ \hline 00010 \end{array}$$

= not a power of 2

Ans: If $n \& (n-1) = 0$ then it is power of 2.

Code :-

```
public class PowOfTwo {
    public static void main(String[] args){
        int n = 16;
        boolean ans = (n & (n-1)) == 0;
        System.out.println(ans);
    }
}
```

Q find a^b ?

Eg :- $3^6 = 3 \times 3 \times 3 \times 3 \times 3 \times 3$ // O(b) - 1 way

$3^6 = 3^{2+4} = 3^2 \times 3^4$ - 2 way

$3^6 = 3^{110}$

$\therefore n = 110$

ans = 1

$n \& 1 \rightarrow 0$ if 0 then ignore

PTO

"If 0 then ignore because we know $3^6 = 3^{2+4}$ so we're only taking 2 & 4 the ones that are equal to the set bits".

Now it's gonna be obviously true as we know $2^0 \times 0 = 0$ so it will not do anything. Hence, we can ignore it.

& we can say that

$= ans = ans \times base$.

$\therefore ans = 9$ & $n = 1$ now $1 >> 1 = 1$

& obviously base · base $\times$ base $= 9 \times 9 = 81$ because it is doubling base everytime

because if we have

3^{110^6} = $3^4 \times 3^2 \times 3^0$

(base is doubling everytime) $= 3^4 \times 1 \times 3^2 \times 1 \times 3^0 \times 0$

Now instead of 6 times we are running it the no. of digits in 6 // $(O \log(b))$

ans = 81

$\therefore$ ans: $81 \times 9 = 729$

Note: "Checking the last value whether it is 1 or 0. if it's 1 then multiply the ans with base."

- Base = Base $\times$ Base - do this always.

even if we are skiping 3*0 that doesn't mean 3 not will get skip.

we are only not multiplying when the value is 0 of that particular index.

Code :-

```
public class Power{
    public static void main(String[] args){
        int base = 3;
        int power = 6;

        int ans = 1;

        while (power > 0){
            if ((power & 1) == 1){
                ans = ans * base;
            }

            base = base * base;
            power = power >> 1;
        }
        System.out.println(ans);
    }
}
```

$= O(\log(\text{power}))$

Q Given a no. find the no. of setbits in it

Eg: n = 9

∴ n = 1001 Ans :- 2.

n & (-n) = 0001 → right most setbit

if we subtract with this n

n - [n & (n-1)] . 1000 ——— (1) count

So keep doing this thing till n is greater than 0. "Because we are finding the right-most bit and we're deleting it one by one"

Eg: n = 1001
& 1000
1000 ——— (1) count → 8

8 & 7 = 1000
& 0111
0000 ——— (2) count.

No. of set bits = no of iterations.

O(logn)

Code :-

```
public class SetBits {
    public static void main(String[] args){
        int n = 626;
        System.out.println(Integer.toBinaryString(n));
        System.out.println(setbit(n));
    }

    private static int setbit(int n) {
        int count = 0;

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

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

Q find XOR of no. from 0 to a (codeforces)

Ans let's say a = 0

	a	XOR from 0 to a
(1)	0	0
(2)	1	0^1 = 1
(3)	2	0^1^2 = 3
(4)	3	0^3^3 = 0
(5)	4	0^0^4 = 4
(6)	5	4^5 = 1
(7)	6	1^6 = 7
(8)	7	7^7 = 0
(9)	8	0^8 = 8
(10)	9	8^9 = 1

PTO →

"Here we can see a pattern 0 = 0, 4 = 4, 8 = 8, etc. & 1, 5, 9 = 1"

So what is happening?

→ An every fourth (4th) no is 0. So basically if you see the above pattern you'd say After every 4 a. pattern is repeating.

① If a % 4 = 0 - 1st no in the patter.
Ans = 0 $\xrightarrow{\text{XOR}}$ a = a(0)

② If a % 4 = 1
Ans = 1

③ If a % 4 = 2
Ans = a+1 Eg:- for 2 = 2+1 = 3 & 6 = 6+1 = 7

④ If a % 34 = 03
Ans = 0

Conclusion:

This is going to keep on repeating (all 4).

So if you want ans of XOR for 0 to 9 ↑ is the Ans.

8. XOR of all no. betn a & b

Ans

a = 3 & b = 9 - suppose

then you've to calculate

3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9 = ?

If we ~~take~~ talk about from 0 to 9

0 ^ 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 ^ 9 - (1)

As we know that we can calculate from 0 to 9 very easily using previous cases.

But we've to find 3 till 9

So "0 ^ 1 ^ 2 ^" are the extras &

we don't want this. we only want it from 3 till 9

How do we ~~do~~ remove these extras?

These extras are already in the entire XOR. So if we XOR "0 ^ 1 ^ 2 ^" again from the eqn (1) then the 0 ^ 1 ^ 2 ^ will get removed

we basically XOR these entire thing then

0 → (a-1) Ans = f(b) ^ f(a-1)

∴ f(a) → XOR of 0 → a

So we are adding or it already added in f(b) part & we're XOR it again so this will be duplicate &

we know that x ^ x = 0

f(b) : It represents the XOR of 0 till b

Note.

If you do XOR for every single no. then it will be very bad complexity. But if you try it will TLE "Time limit exceed" error.

Code :-

```
public class RangeXor {
    public static void main(String[] args){
        int a = 3;
        int b = 9;

        int ans = xor(b) ^ xor(a-1);

        System.out.println(ans);
    }
    static int xor(int a) {

        if (a % 4 == 0) {
            return a;
        }

        if (a % 4 == 1) {
            return 1;
        }

        if (a % 4 == 2) {
            return a+1;
        }

        if (a % 4 == 3) {
            return 0;
        }
    }
```

or

```
return 0;
```

Google.

8. Flipping an image.
Given a binary matrix A, flip the image horizontally, then invert it & return the resulting image.

Ans
① Here every row is reversed.
Eg :- 1,1,0 -> 0,1,1
(reverse an array)

② Invert it 0 → 1 or 1 → 0
(It's taking a complement)

If we XOR any no. with 1. we'll get the inverse of that no.
Eg :- If no is 1 then 1^1 = 0
but if no. is 0 then 0^1 = 1
} flipped.
So 1 becomes 0
& 0 becomes 1

So after reversing every row we don't to do anything with it. So while reversing the element we can also apply XOR

Eg :-
1 1 0 → reverse → 0 1 1
1 0 1 → 1 0 1
0 0 0 → 0 0 0

↓ invert & (flipping the no)

1 0 0
0 1 0
1 1 1 // Ans.

Just reverse every single array because we know that in 2D array every single row is individually itself an array.

Code :

```
Class FlipImage{
    public int[][] flipAndInvertImage(int[][]
                                      image){

        for (int [] row : image) {

            // reversed .
            for(int i=0; i<(image[0].length +1)/2
                                      i++){

            // to Swap
                int temp = row[i] ^ 1;
                row[i] = row[image[0].length-i-
                                      ^1];

                row[image[0].length-i-1] = temp;
            }
        }
        return image;

    }
}
```