

Ques) Factorial of a number.

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Static long factorial (int n)

```
for (int i = 1; i <= n; i++)  
{  
    fact = fact * i;  
}  
return (fact);
```

Ques check if string is palindrome

int ispalindrome (String s1)

String s2 = "";

```
for (int i = s1.length() - 1; i >= 0; i--)
```

```
{
```

```
s2 = s2 + charAt(i);  
}
```

```
if (s1.equals(s2))
```

```
{  
    return 1;  
}
```

```
else
```

```
{  
    return 0;  
}
```

```
}
```

Ques) get length of Integer

int num;

int len = String.valueOf(num).length();

Ques) Armstrong no = no whose sum = cube of its digit
Eg 153, 407, 371

$$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

Eg $12 = 1^2 + 2^2 \Rightarrow$ it should follow this condⁿ
 $1234 = 1^4 + 2^4 + 3^4 + 4^4$

Static String armstrongno (int ~~n~~ n)

{

int length = String.valueOf(n).length();

int n_val = n;

int arm_no = 0;

int r = 0;

while (n > 0)

{

r = n % 10;

arm_no = arm_no + (r * r * r);

n = n / 10;

}

if (arm_no == n_val)

{
return "true";

else

{
return "false";

}

}

Ques

Fibonacci series till n

0, 1, 1, 2, 3, 5, 8, 13, ... soon
find nth fibonacci series no.

Static int nthfibo(int n)

{

int sum = 0

int n1 = 0

int n2 = 1

for (int i = 2; i <= n; i++)

{

sum = (n1 + n2) % 1000000007;

n1 = n2;

n2 = sum;

}

} return sum;

if (n == 0)
< return 0;

if (n == 1)
< return 1;

Ques Reverse a String

class reverse

{
public static reverseString (String str)

{
String str2 = "";

for (int i = str.length() - 1; i >= 0; i--)

{

str2 = str2 + str.charAt(i);

}

return str2;

}

M2

```
2  
    str2 = " ";  
    for (int i = 0; i < len str.length; i++)  
        str2 = str.charAt(i) + str2;  
    }  
    return str2;  
}
```

M3

String Builder & String Buffer.

```
Public static reverseword (String str)  
{  
    StringBuffer sb = new StringBuffer (str);  
    sb.reverse();  
    return sb.toString();  
}
```

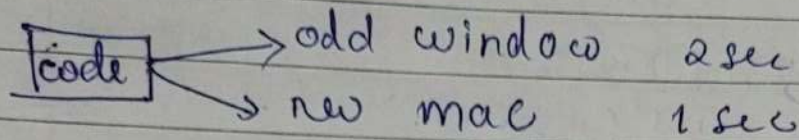
Public static reverseword (String str)

```
{  
    StringBuilder sb = new StringBuilder (str);  
    sb.reverse();  
    return sb.toString();  
}
```

Ques print a no is prime or not time ($O(\sqrt{n})$)

```
{  
    if (n == 1) { return 0; }  
    for (int i = 2; i < Math.sqrt(n); i++)  
        if (n % i == 0)  
            return 0;  
    }  
    return 1  
}
```


Q) What is time complexity & space complexity
→ This is wrong definition → Time taken to run the code

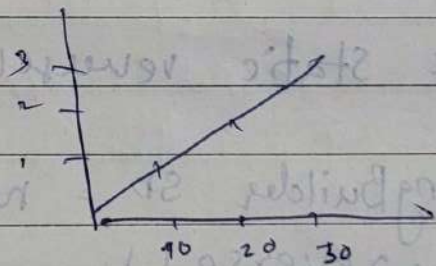
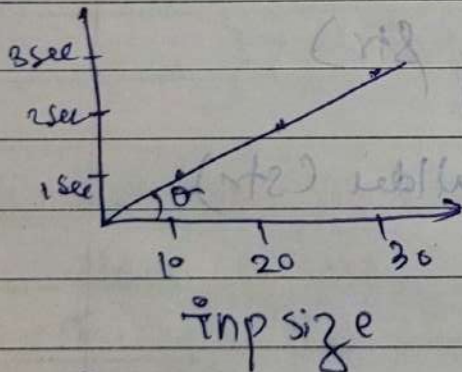


cause in different system our code might take different time to run

Ans rate of which the time increases with respect to input size.

Eg old window

new mac



is rate at which the time increases

3 Rules

- ① Time complexity, calculate in worst case
- ② avoid constants
- ③ avoid lower values


```

for (int i=1; i<N; i++)
    print(" ");

```

TC = $O(N \times 3) = O(3N)$

Eg

```

if (marks < 20) print("D grade")
else if (marks < 40) print("C grade")
else if (marks < 60) print("B grade")
else print("A grade")

```

→ best case
→ worst case

① if marks = 10 → best case
 $O(2)$ as (1 → check cond, 2 → print)

② if marks = 70 → worst case
 $O(4)$ as (1 → check, 2 → check, 3 → check, 4 → print)

③ Avg case = $\frac{\text{best} + \text{worst}}{2}$

Ques why we always check worst condition.

Eg I want to build a comp building, will make it for 1 person or 100 person
obv 100, we want to scale up our sys
so our codes extreme worst case we see

Ques why avoid constants.

Eg our time comp code is $= O(4N^3 + 3N^{10} + 2)$
and input size $N = 10^3$

So
Here constant would have no significance so in front of such huge no so we ignore it
 $= O(4 \times 10^9 + 3 \times 10^{30} + 2)$

Ex `int n=1` → this is constant so we ignore or not include in time complexity calculation
`for (int i=0; i<N; i++)`
`sop (i)`

3 Ques) avoid lower value
 here $[3 \times 10^{20} + 4 \times 10^3 + 8]$
 ↓
 this is lower value this is small in front of 3×10^{20} so ignore it

Big O	Theta (Θ)	Omega (Ω)
↓	↓	↓
worst case	Avg case	lowest Bound

Ques find time comp of below

`for (int i=0; i<N; i++)` → N

`for (int j=0; j<N; j++)` → N

so → ignoring this as constant

time comp $O(N^2)$


```

ans
for (int i=0; i < N; i++)
{
    for (int j=0; j < i; j++)
    {
        // s.o.p ( ) - ignore
    }
}

```

#0, 1, 2, 3, 4 ... N

$$\frac{n(n+1)}{2} = \left(\frac{n^2}{2} + \frac{n}{2} \right) = O\left(\frac{n^2}{2}\right)$$

\downarrow
 ignore or lower value $= O(n^2)$

Space complexity \rightarrow Memory space that code takes
 it also varies from machine to machine

Space complexity = Auxiliary space \downarrow Space that takes to solve the problem / inp	+ input space \downarrow Space that takes to store the problem input
---	--

Eg $c = a + b \Rightarrow sp = O(3)$

\downarrow
 auxiliary space
 (extra space to store sum)

\downarrow
 input space.

• $O(N) = O(N)$

$b = a + b$

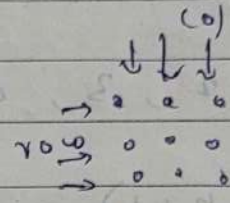
as user data is critical in industry we should not change

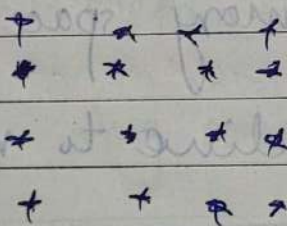
never do this as inp data should not be touch
 tell interviewer u don't want to change data for entire session

$1 \text{ sec} = 10^8 \text{ operations} \rightarrow \text{most server takes this much}$
 $2 \text{ sec} = 2 \times 10^8$
 $5 \text{ sec} = 5 \times 10^8 \text{ sec}$

Patterns are not asked in Interviews.

- ① all patterns have 2 loops
- ② outer loop \rightarrow row
inner loop \rightarrow col
- ③ for outer loop count row
- ④ for inner loop focus on col & connect them to row
- ⑤ print things inside inner loop
- ⑥ observe symmetry.



(Ques)


 $i = \text{row}$
 $j = \text{col}$

for every line / row we print 4 stars.

```

for (int i = 0 ; i < 4 ; i++)
{
    for (int j = 0 ; j < 4 ; j++)
    {
        s.o.p ("*");
    }
    s.o.p ("\n");
}
    
```


Que) Difference between below two

System.out.print("*"); → prints in same line

System.out.println("*"); → prints in new line

System.out.print(); → goes to next line

Ques print

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

```
public class void pattern(int n)
```

```
{  
    for (int i=0; i<n; i++)
```

```
    {  
        for (int j=0; j<n; j++)
```

```
        {  
            System.out.print("*");  
        }
```

```
        System.out.println();  
    }
```

Ques print the pattern

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

```
{  
    for (int i=0; i<n; i++)
```

```
    {  
        for (int j=0; j<=i; j++)
```

```
        {  
            s.o.p ("*");  
        }
```

```
        s.o.println();  
    }
```

```
}
```


Ques print pattern

```
for (int i=0; i<=n; i++)
{
    for (int j=0; j<=n; j++)
    {
        system.out.print(j + " ");
    }
    system.out.println();
}
```

Ques

```
for (int i=1; i<=n; i++)
{
    for (int j=1; j<=i; j++)
    {
        system.out.print(i + " ");
    }
    system.out.println();
}
```

Ques print

```
for (int i=0; i<=n; i++)
{
    for (int j='A'; j<='A'; j++)
    {
        System.out.print(j);
    }
}
```

A
AB
ABC
ABCD

Ques print

```
0 1 2 3 4 5 6 7 8
1 | | | | | | | |
2 | | | | | | | |
3 | | | | | | | |
4 | | | | | | | |
```

space & space

4	1	4
3	3	3
2	5	2
1	7	2
0	8	0

3 for loops

- ① print space
- ② print ~~space~~
- ③ print space

i-rows $\Rightarrow 0$ to 4

j-col = 0 to 8

SWITCH STATEMENTS

In the switch statements ~~car~~, you can jump to various cases based on your expression

Syntax

switch (expression)

{

case one:

// do something;

break;

case two:

// do something;

break;

default:

// do something;

}

• cases have to be same type as expressions, must be constants or literals

• duplicate case values are not allowed

• break is used to terminate the statement

• If break is not used, it will continue to next case.

• default will execute when none of the above does

• if default is not at the end, put break after it

• if break is not provided, it will execute till end

Eg

```
import java.util.Scanner;
```

```
public class Main
```

```
{  
    public static void main (String args[])
```

```
{  
    Scanner sc = new Scanner(System.in)
```

```
    String fruit = sc.next();
```

```
    switch (fruit)
```

```
    {  
        case "Mango":
```

```
            System.out.println("yellow fruit");  
            break;
```

```
        case "apple":
```

```
            System.out.println("red fruit");  
            break;
```

```
        case "grapes":
```

```
            System.out.println("green fruit");  
            break;
```

```
        default:
```

```
            System.out.println("enter valid fruit");
```

In clean format

```
switch (fruit)
```

```
{  
    case "Mango" -> System.out.println("yellow");
```

```
    case "apple" -> System.out.println("red");
```

```
    case "grapes" -> System.out.println("green");
```

```
    default -> System.out.println("enter correct fruit");  
}
```

For we don't need break statement
and more clean

if / else it → switch case.

or switch (day)

```
{
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
    s.o.p ("weekday");
    break;
  case 6:
  case 7:
    s.o.p ("weekends");
    break;
}
```

or

→ switch (day)

```
{
  case 1, 2, 3, 4, 5 → s.o.p ("weekday");
  case 6, 7 → s.o.p ("weekend");
  default → s.o.p ("enter valid day");
}
```


Methods & Functions

dry - donot repeat

Method = function.

Method = function within a class

Eg print sum 2 nos

```
public class Main
```

```
{
```

```
    public static void main (String args[])
```

```
{
```

```
    Scanner sc = new Scanner (System.in)
```

```
    s.o.p ("num1 enter");
```

```
    int n1 = sc.nextInt();
```

```
    s.o.p ("num2 enter");
```

```
    int n2 = sc.nextInt();
```

```
    int sum = n1 + n2;
```

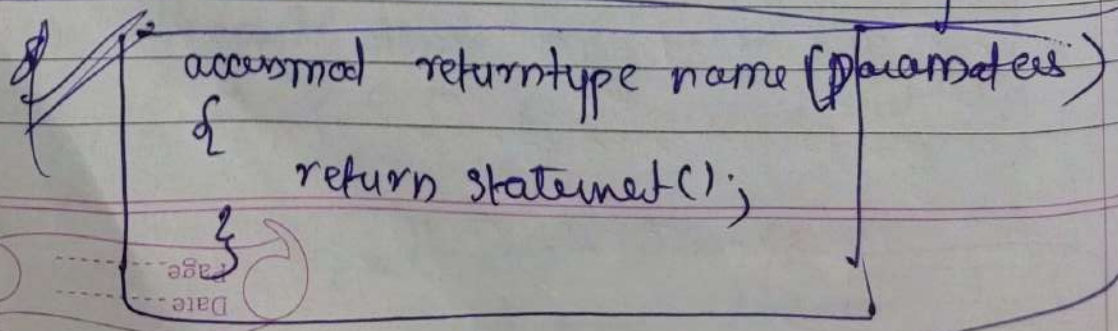
```
    s.o.p ("sum is: " + sum);
```

```
}
```

format of function.

access_modifier returntype name () {}

oop




```
import java.util.Scanner;
```

```
public class Main Sum
```

```
{  
    public static void main (String args[])
```

```
{  
    sum (); // fun call  
}
```

```
    public void sum ()
```

```
{  
    Scanner sc = new new Scanner (System.in)
```

```
    S.o.p ("Enter n1")
```

```
    int n1 = sc.nextInt ();
```

```
    int n2 = sc.nextInt ();
```

```
    int sum = n1 + n2;
```

```
    S.o.p ("S.o.p ("sum is " + sum));
```

function returns.

Void return type

when our function just prints something and does not return any value then return type of fun is void.

```
public class main
```


eg 2

```
public class greetingfun
```

```
{
    public static void main (String args[])
    {
        greetings();
    }
}
```

```
static void greetings()
{
    s.o.p ("Hi greeting");
}
```

eg 3

```
public class sumavg2not
{
    public static void main (String args[])
    {
        int ans = sum();
        s.o.p (ans);
    }
}
```

```
static int sum()
{
    int n1 = sc.nextInt();
    int n2 = sc.nextInt();
    int sum = n1 + n2;
    return sum;
}
```

calls this

→ returns this value

int ans = sum();

→ Sum fun is called.

takes 2 inp returns sum whose value is 30

returns sum = 30 which is caught by sum() fun

* If I write return then means function is finished.

Imp In a function return is always at end. If any thing is written after return it will not execute → only inside a function

```
} return sum;  
s.o.p ("this will never execute");  
}
```

§ Main is always first method that runs *** }

eg public class StringExample

```
public static void main (String args[])  
{
```

```
    String msg = greetings();  
    s.o.p (msg);  
}
```

```
static String greetings()  
{
```

```
    String g = "how r u?";  
    return g;  
}
```

Now suppose in a function I don't want to take input from user every time i.e

```
int n = sc.nextInt();
```

so we use parameters inside a function
passing parameters inside a function

* Passing parameters inside a function

```
public class sumofnos
```

```
{  
    public static void main (String args[])
```

```
{  
        int ans = sum2 (10, 20);  
        s.o.p (ans);  
    }
```

```
    static int sum2 (int a, int b)
```

```
{  
        int sum = a + b;  
        return sum;  
    }
```

* public class mygreet

```
{  
    public static void main (String args[])
```

```
{  
        Scanner sc = new Scanner (System.in)
```

```
{  
        String naam = sc.next(); // Ekta
```

```
        String personal = mygreet (naam);  
        s.o.p (personal);  
    }
```

```
    static String mygreet (String name)
```

```
{  
        String msg = "Hello " + name;  
        return msg;  
    }
```

```
}
```


In java there is no pass by reference ~~there is only~~ pass by value explain?

p.s.v.m (s.a) in previous example

stack naam \Rightarrow ekta heap
name

name & naam both are pointing towards same object
name is a copy of reference variable

Que

```
p.s.v.m (s.a)
{
    String name = "kunal"
    change(name);
    s.o.p(name);
}

static void (naam)
{
    naam = ekta
    // here creating new object
}
```

name \rightarrow ~~ekta~~ \rightarrow kunal
naam \rightarrow kunal

naam - ekta

The reference variable was first pointing kunal, then to ~~ekta~~ so final

Imp \Rightarrow we can access naam only within that function not outside the function

name \rightarrow kunal
naam \rightarrow ekta

Que Imp Primitives eg Int, float, char, double, byte
Pass BY ~~ref~~ value

Objects reference - pass ~~by~~ value of reference variable