1) Programming language ✓
2) Leetcode.
   ↳ Upsolving (learn from others solution)
3) Project
4) Make resume

In some question `int` can't handle the value of the test cases so take `long` instedd of `int`.

- To iterate from left and also not going outside the no. of bit we check → while (bitmask < N) {

  bitmask = bitmask << 1

  }

Bit Manipulation

- Decimal to Binary

ex - 4

		remainder
2 | 4		0
2 | 2		0
	1	

$\Rightarrow (4)_{10} = (100)_2$

- Binary to Decimal

ex- 100

$1 \times 2^2 \; 0 \times 2^1 \; 0 \times 2^0$

$= 4 + 0 + 0$

$= 4$

0 = 000
1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111
8 = 1000

Bit-wise Operators

Binary
- AND &
- OR |
- XOR ^
- One's Complement ~
- Shif left <<
- Shif Right >>

AND &

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

ex - 101
110
100
$\Rightarrow (4)_{10}$

OR |

5 | 6

A = 0101
B = 0110
0111

$(7)_2$

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

XOR ^

⇒ 0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

5 ^ 6 A = 101 B = 110

```
 101
 110
-----
 011
```

⇒ $(3)_{10}$

Binary One's Complement ~

ex-

~0 = 1

~1 = 0

~5 A = 0101

~(101) = 010

⇒ $2_{(10)}$

But computer gives us -6

Because in computer there 8 bits

ex - 0 0 0 0 0 1 0 1

MSB — Most significant bit

LSB — least significant bit

This method is followed when we ~~eat~~ convert the negative no.

if MSB = 0 → the no. is positive

MSB = 1 → the no. is negative

So when we do ~ 0101 is actualy does

~5 → 0 0 0 0 0 1 0 1

2's complement → 1 1 1 1 1 0 1 0 → This is for the value of MSB so it is 1 ∴ no. is negative

1's complement → 0 0 0 0 1 0 1 0

Add 1 → 0 0 0 0 1 0 1 0 + 1 = 0 0 0 0 1 0 1 1 → This type of addition is done in bit wise

So it is -6

2's Complement → ① 1's Complement not (a) ~n; and ② Adding 1

ex- ~0 = -1

how → 0 → 0 0 0 0 0 0 0 0

~0 → 1 1 1 1 1 1 1 1 → For MSB

↓ MSB = 1

∴ negative

One's complement 0 0 0 0 0 0 0 0

+ 1

0 0 0 0 0 0 0 1

⇒ 1 but negative

∴ ⇒ -1 //

Binary Left Shift <<

this no. is shifted → a << b (2)

Shortcut Formula

$$a << b = a * 2^b$$

→ ex- 5<<2

⇒ 0 0 0 1 0 1

ex- 5×2^2

⇒ 20

after shift

0 1 0 1 0 0

⇒ $(1010 0)_2 \rightarrow 1 \times 2^4 + 1 \times 2^2$

⇒ 16 + 4

= 20

Binary Right shift >>

→ shift the bits right side

ex- 6 ⇒ 0 0 0 1 1 0

6>>2 0 0 0 0 1 1

Formula

$$a >> b = a/2^b$$

⇒ $(0 0 0 0 1)_{10} = 1$

Question - Check if the number is Odd or Even

1) → For this we will and (&) the no. with 1

00000001 ← This number told us that is the number is odd or even that is by LSB

* Least significant beat

2) But for this we need all other 0

∴ We will and the no. with 1

→ ex- 01011011
& 00000001
00000001 ← ∴ this no. is odd

index is counted using 0 so make bitmask make i-1

→ ex - 100 (4)10
& 001 ← And this is called as bitmask
000 ← even.

The operation we performed is & and bitmask we used is 1

Get ith bit

↳ Just to check the ith bit is 0 or 1

↳ We will take & with that bit

↳ bit of which no.

So for that we will input from user and take (1)10

→ 00000001

Taking and operator with the ith position of this no.

then n & (1<<i)

only this code is work

```
static int getithbit(int n, int i){
    int bitmask = 1<<i;
```

Set ith bit → Make a selected bit 1.

→ `n | (1<<i)` to make change make that bit 1

→ ex- 10 → 1010

we done `10 | (1 << 2)`

```
 1010
 0100  OR
 ----
 1110
```

∴ $1110 = (14)_{10}$

Clear ith bit → making that bit zero

`n & ~(1<<i)` ex- 10 → 1010

```
 1010
 0100  &
 ----
 1000
```

→ $(8)_{10}$

Update ith bit

↳ position will be given (i)
↳ and the user will tell he want 1 or 0 on that place

→

~~Update ith bit~~ → Just add a if ~~else~~ else → if (i==0) or if (i==1)

↳ ~~Write~~ Of Another Approach

int newBit → bit you want on that place 0 or 1

i) clear the bit first
ii) bitmask = `newbit << i`

```
clear_ith_bit(n, i);
int bitmask = newBit << i;
return n | Bitmask;
```

Clear last i bits

$\sim 0 = -1 = 11111111$

ex $n = 1111$, $i = 2$

⇒ 1100

⇒ for this we use $(-1<<i)$ or $(\sim 0)<<i$

00 ← i times

⇒ $(15)_{10} = 1111$

Shifting left 2 times

⇒ 11111111

→ 11111100

$n \& (-1<<2)$

⇒ 1111 &
1100

= 1100 = $(12)_{10}$

Clear Range of bits

⇒ 100111010011 , $i = 2$, $j = 7$

⇒ clear bits from 7 to 2

that is

11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 1 1 1 0 1 0 0 1 1

⇒ 1001 0000 0011 → a | b

from 7 to 2

Logic →

a ⇒ $(\sim 0)<<j+1$

⇒ 11111 00000000

b ⇒ $01 = (1)_{10} = 2^1 - 1$

$011 = (3)_{10} = 2^2 - 1$

$0111 = (7)_{10} = 2^3 - 1$

$01111 = (15)_{10} = 2^4 - 1$

∴ for 011

→

∴ for $011 = 2^2 - 1$

⇒ $(1<<i) - 1$

$2^b \rightarrow 1<<b$

∴ a | b = clear bits bit j and i

bitmask ⇒ $(\sim 0)<<j+1 \mid (1<<i) - 1$

n & bitmask
→ required answers

ex- $(10)_{10} \rightarrow$ 000 1010 $\quad i = 2$, $j = 4$

ans $\Rightarrow$ 0000010 $= (2)_{10}$

Check if a number is power of 2 or not

$4 \rightarrow 2^2 \rightarrow$ 4 → 100, 3 → 011, & → 0

$8 \rightarrow 2^3 \rightarrow$ 8 → 1000, 7 → 0111, & → 0

$\therefore$ **n & (n-1) == 0** ← If this is true then the no. is power of 2.

Fast Exponentation → to take the exponetiation fast → to take out power fast

ex - $3^5 \rightarrow$ for this we need to multiply 3 x 3 x 3 x 3 x 3

but we will use bit here

$\Rightarrow 3^{101} \rightarrow$ ① a = 3 ; ans = 1

$a^4\ a^2\ a^1$

10 1

3 ↓ (a')

at 0 multiply with 1

ans = 3

$a \rightarrow a^2 \rightarrow 9$

② a = 9, ans = 3

ans = 3 x 1

$a \rightarrow a^2 = 81$

③ a = 81, ans = 3

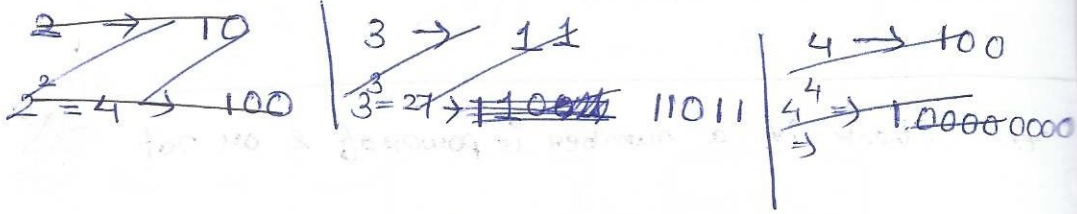
ans = 3 x 81 = 243

$a \rightarrow a^2 = 81^2 = 243$

H.W

Find the value of x^x if x is given? (Bit manipulation)

⇒



5 → 101

∧ → (XOR)

not of or

$4 = 100$

$100 \wedge 100$ → 100 ∧ 100 = 000

$\therefore$ $X \wedge X$ will always be 0

Swaping two number's without using 3rd variable.

Ex- $x = 3 \rightarrow 11$

$y = 4 \rightarrow 100$

i) $x = x \wedge y$

ii) $y = x \wedge y$

$x = x \wedge y$

$x = 100 \wedge 011 = 111 = 7$ (dec)

$y = 111 \wedge 100 = 011 \rightarrow 3$ (dec)

$x = 111 \wedge 011 = 100 \rightarrow 4$ (dec)

$\therefore$ $x = 4$, $y = 3$

after swaping.

$-x = \sim x + 1$

- Negetive of a number invert its bit and add 1 to it

∴ To add 1 using bit ⇒ $-\sim x = x + 1$

just replaced x by ~x

To Convert Uppercase ⟶ Lowercase Charecter

```
char a = 'A';
Syso((char)(a | ' '));
```

type caste from bit to charecter.

```
ch = 'A';
ch1 = 'b';
```

⇒ For Uppercase to lowercase → `(char)(ch | ' ')`

For Lower to uppercase → `(char)(ch ^ ' ')`

OOP's

Object oriented programming.

✡ Questions in interview

⇒

Classes → Blueprint of an object. → Group of this entities / Collection of objects of similar properties.

Objects → entities in real world.

ex- Pen
- Properties
  - red
  - long
  - function

Example - pen

Class
- ① → attributes (Properties) Color (String)
- ② → functions (behaviour) changeColor ()

- Conventional → Name of class start with Capital Letter
  → Name of function start with Small Letter

↳ public class is at top above
  ↳ So that no error in program

Ex:-

```
Public class OOPS {
    Public static void main(String args []){
        Pen p1 = new Pen(); ← created a pen object
        p1.setColor ("Blue"); ← assesing the color in pen object
        Syso (p1.color);
    } }
class Pen { ← another class
    String color;
    int tip;
    void setColor (String newColor){ ← Function inside the class
        color = newColor;
    }
```

☆ Classes cannot be protected or private because it will be of no use ther
× ×

Access Modifiers

	within class	within Package	outside class by subclass only	outside Pakage
1) Private	✓	×	×	×
2) Default	✓	✓	×	×
3) Protected	✓	✓	✓	×
4) Public	✓	✓	✓	✓

Grettors & Setters

- Get → to return the value.
- Set → to modify the value.

→ If the properties of the object are Private we need to used functions to used the value and change the values.

- this ↳ 'this' keyword is used to reffer to the current object.
  - ↑ Keyword
  - ↳ Usefull when objects or parameters are of same names.

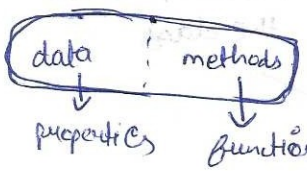
Learn defination.

OOP's
- Encapulation
- Inheritance
- Abstraction
- Polymorphism.

☆

1. Encapulation → Defined as the wrapping up of data and methods under a single unit. It also implements data hiding.

↳ just like a capsule



↳ by using access modifiers.
PDPP
Private - default - protected - public

Constructor → If not made then created automatically

↳ it is a special method which is invoked automatically at the time of object creation.

1) have same name as class or structure
2) don't have a return type (Not even void)
3) Are only called once → At object creation.
4) Memory allocation happens when constructor is called.

ex.

```
Pen p1 = new Pen();
```

Types of Constructor

① Non-parameterized → No parameter is used.

ex- Student() { }

② Parameterized → with parameters.

~~5~~ ✡ We can have as many types of constructor as many type of parameter and their sequence we give. ② ✡ And default constructor will only work if we give no constructor.

③ Copy Constructor:

↳ by default in C++
↳ need to made in java

↳ ✡ To copy the properties from one object to other.

① Shallow Copy

~~Deep Copy~~

ex-

```
main() {
    Student S1 = new Student();
    S1.name = "ABC";
    S1.roll = 456;
    Student S2 = new Student(S1);
}
Class Student {
    string name;
    int roll;
    Student(Student S1) {
        this.name = S1.name;
        this.roll = S1.roll;
    }
}
```

✡

In shallow the reffrence get copied. therefore if we make changes in first or orignal it will also reflect in copy.

Deep Copy

* Construction overloading
  ↳ When we many functions and any of that is used as per the requirement.

Destructors

↳ In C++ their need to make this functions.

↳ In java it is automatically done using garbage collector.

---

2. Inheritance → Inheritance is when properties and methods of base class are passed on to a derived class.

Class A — Base/Parent

↓

Class B — Derived/child

Class A → Class B

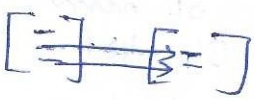
Example : →

```
class A extends B {

}
```

Types

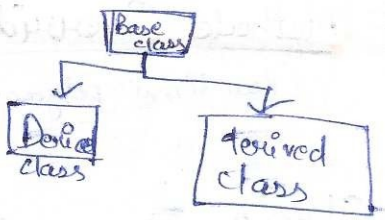
- Single level Inheritance : example → A → B



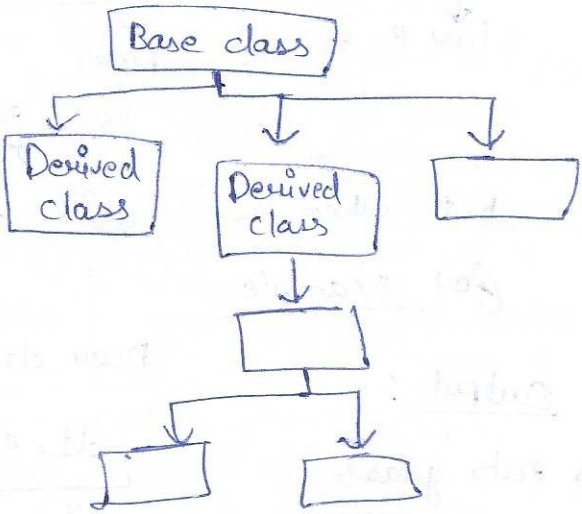
- Multi level Inheritance : example

A → B → C

- Hierarchial Inheritance :-



- Hybrid Inheritance :-



✱ Multiple inheritance → available in c++ but not in java.

---

5. #Polymorphism → When try to do the same thing in multiple forms.

many forms

Type ↓

(static)

1. Compile Time Polymorphism (Method Overloading)
2. Run Time Polymorphism (Dynamic)

type → Method Overriding

Method Overloading → Multiple function with same name but different parameter

ex- Calculator {

But all have same name.

- sum(int a, int b)
- sum(float a, float b) ← Here the data type is different
- sum(int a, int b, int c) ← Here the count is different

Methode Overriding → Parent and child classes both contains the same function with different definition.

(Run time polymorphism)

Class A ≡

↓

Class B ≡

Animal

void eat() → "eat anything"

↓

Deer

void eat() → "eats grass";

But when we will call the function

for example

Deer d1 = new Deer();

d1.eat();

this will call the function made inside the deer function.

Output :

⇒ eats grass

Prints parent class output.

Packages in Java :

classes
subpackages
interfaces

↳ Group of similar types of classes, interfaces and sub packages.

Types : 1) Inbuilt

2) User defined.

Inbuilt

ex - Scanner sc = new Scanner();

this is inside utility package → import java.util.*;

Userdefined

package name ;

(43) Abstraction → Hiding all the unnecessary details and showing only the important parts to the user.

सार्वजनिक (idea)

- Difference bet' Encapsulation and Abstraction

* Only data can be hidden that we don't want to show

* Here the we can do both things Hide the data that we don't want to see and also show the data that we want to show.

* Constructor are called hierarchical wise
  A → B → C
  Here first it will call A then B then C

Concept (1) Abstract Classes
(2) Interfaces

↳ Just add abstract word in front of class.

Properties: i) Cannot create an instance/object of abstract class.

ii) Can have abstract/non abstract methods (functions)

iii) Can have constructor

↳ this helps to pre-initialise any value or Properties in all the classes.

↳ While making the abstract function we cannot ~~tell~~ assign the work there. And if we extends the class in which the abstract function is made then it is compulsory for the other classes to use the ~~abstract~~ function that was made using abstract keyword.

Interfaces

ex- [Car1, Car2] - Object

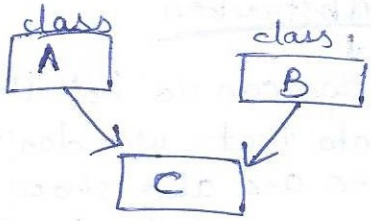
↳ It is a blueprint of class.

[Maruti 800] Class (blueprint of object)

Car [wheels, speed, engin]

Interfaces (blueprint of class)

1. Multiple Inheritance (5th) type of inheritance.
   - by default done in c++.
   - but we need to do it in java.



ex-

class Bear implements Herbivore, Carnivore {

}

2. Total Abstraction
   - 100% abstraction we get
   - Not like abstract classes in which it is needed to make each function as abstract function to get 100% abstraction.

Interface	class
1) Keyword - interface	1) keyword - class
2) extension - implements	2) extension - extends

- Properties
  1. All methods are public, abstract and without implementation.
  2. Used to achive total abstraction.
  3. Variables in Interface are (1) final, (2) public, (3) static
     - final → value will be same

Note - If it is like this

```
class vehicle();
class car extends vechicle();
```

then we can also write like this

```
vehicle v = new car();
```

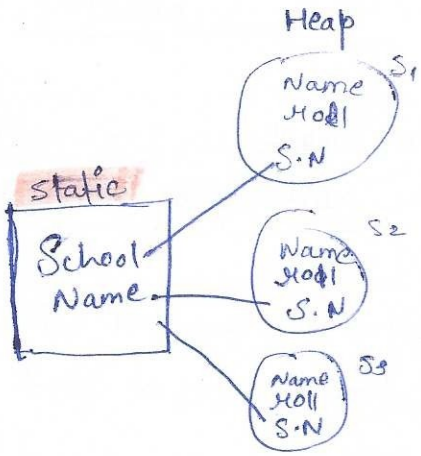
Static Keyword → Static keyword in java used to share the same variable or method of a given class

(Constant)

↳ we can make static
- → Properties
- → Functions
- → Blocks
- → Nested Classes

(Properties, Functions: important; Blocks, Nested Classes: rarely asked)

basically anything we can make static



↳ If any object declare school name means initialise it then it will be initialised for all objects

↳ If any object modify school name then it will be modified for all objects

⁂ Our main function is also a static.

↳ Using static keyword multiple things of same name are not created. (No multiple creation)

↳ Saves memory

Super. Keyword → Super keyword is used to refer immediate parent class object

1. To acces parents properties
2. to access parents functions
3. to access parents constructor.

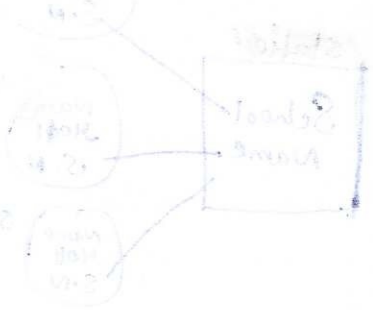
In function overriding Derive class funtion will be output.

We can access static variables directly with classes also.

ex- class Book {
static int count;
}

We can write like this
→ syso(Book.count); ✓
without making object.

While method/function overriding we can use a less restrictive or same restrictive type of (access modifier).



Recursion → It is a method of solving a ~~smaller~~ computational problem where the solution depends on solutions to smaller instances of the same problem.

- Base Case → where recursion ends.
- Recursive function → The functions that call themself again and again.

1) Top to base → calculate go down repeat
2) base to top → add the solution and go up repeat

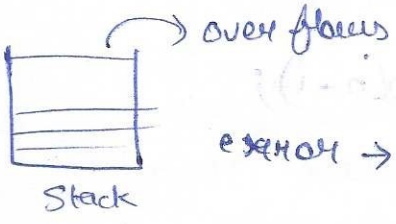
→ a) Define base case
b) Work we want to do
c) (inner call)

Print number from decreasing order

```
fun(int n) {
syso Print(n);
    fun(n-1);
if(n=1){ syso(n)
        return;
    }
```

Stack Overflow

1) To many parameters ↑
2) Too many calls.



Print numbers from n to 1 (Increasing Order)

⇒

```
function (int n)
function (n-1);
System.out.print (n);
```

if (n == 1) { sysso (n) return; }

Print factorial of number n.

⇒

```
int Fact ( int n){
    if (n==0){
        return 1; }
```

here 0 because factorial of 1 = 0

```
    int fact = n * Fact(n-1);
    return fact;
```

Time Complexity → $O(n)$ because we done n calls

Space complexity → $O(n)$ because in worst case we store n stack cases.

---

print Sum of n natural numbers

```
Sum (int n) {
```

⇒

```
if(n==1){
    return 1;
}
int sum = n + Sum(n-1);
return sum;
    }
}
```

Print N^{th} fibonacci number

↳ Each next number is sum of last two numbers.

↳ two base cases.

```
fib (int n){
  if (n=0 → 0
      n=1 ⇒ 1 } Base cases
```

$fib_{N-1} = fib(n-1);$

$fib_{N-2} = fib(n-2);$

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

return fib N

$fib\ 5 = fib\ 4 + fib\ 3$

$fib_3 + fib_2$

$fib_2 + fib_1$ ↘ $fib_1 + fib_0$

$fib_1 + fib_0$

* Space complexity : $O(n)$ → no. of calls
* Time Complexity : $O(2^n)$ → because that many leaf nodes where called.

Check if array is sorted or not.

ex - 1 2 3 4 5

check kaha ? $\boxed{i > i-1}$

=) isSorted (int arr[], int i)

if (i == arr.length) {
  return true;
}

if (arr[i] > arr[i+1]) {
  return false;
}

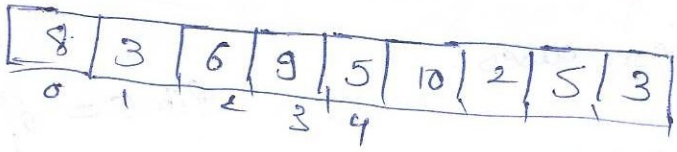
return isSorted (arr, i+1); }

i is the value from which we are starting to check is it sorted or not

First Occurrence ->

→ Here we need to return the position of key

ex-



key = 5
Output = 4.
if no key then -1.

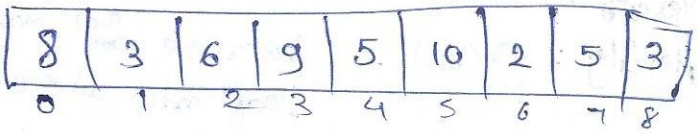
→ ∴ base case = -1
if key found return i;

→

```
firstOccur(int arr[], int i, key)
if (arr[i] == key) {
    return i;
}
return firstOccur(arr, key, i+1);
}
```

Last Occurrence →

→



→ ① Look forward
② then compare with yourself

↳ If there is no key in forward then the current i will be last occurrence is it is same as key

→
```
lastOccurence (arr, key, i+1) {
    if (i == arr.length) {
        return -1;
    }
    int isFound = lastOccurence(arr, key, i+1);
    if (isFound == -1 && arr[i] == key) {
        return i
    }
    return isFound;
```

Print x to the power n

Example - print x^n

$2^{10} = 2 \times 2^{10-1}$

$\downarrow$

$2^{(10-1)} \times 2^{(10-2)} \ldots$

base case → if (i == 1) { return n }

```
power (x, n) {
  if (n == 1) { return x; }
  int value = x * power(x, n-1);
  return value;
```

Time Complexity = O(n)

```
if (i == arr.length) {
  return -1 }
key, i+1);
```

Print x to the power n (Optimized)

int Time Complexity → O(log n)

① Case one → when n = even

ex - $x^n = 2^{10}$

⇒ $(2^5) * (2^5) = 2^{5+5} = 2^{10}$.

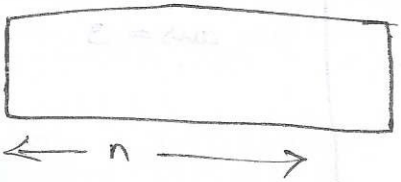
② Case two → when n = odd.

ex - $x^n = 2^5$

⇒ $x \times (2^2) \times (2^2) = 2^{1+2+2} = 2^5$

```
power(x, n) {
  if (n == 1) { return x }
  int ans = power(x, n/2)
  int ansSq = ans * ans;
// for odd
  if (n % 2 != 0) {
    ansSq = x * ansSq;
  }
  return ansSq.
```

Tiling Problem



breadth = 2

tiles = 2 x 1

↳ Either horizontaly or verticaly. or using both

Q. Find total no. of ways to place tiles?

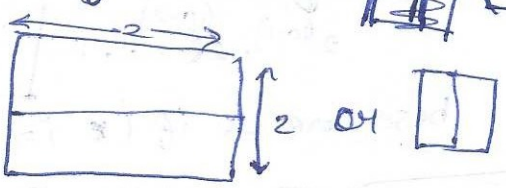
⇒ we should know ① base case
② kaam
③ call inner f(x)

n = no. of tiles.

(1) For n = 0, 2×0, ways = 1

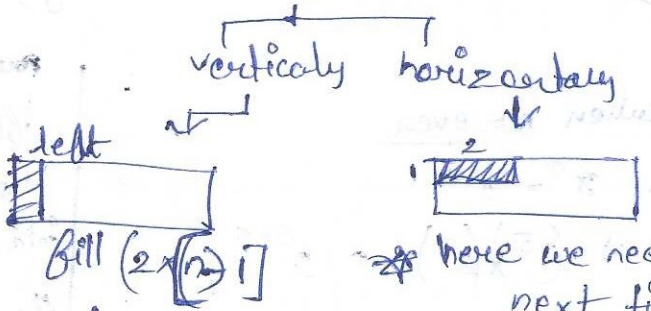
(2) n = 1, 2×1 = ways = 1

(3) n = 2, 2×2 = ways = 2.



(1) # base case : if (n==0 && n==1){
return 1;

(2) Kam
But for more than 1 choice



(3) Total ways = f(n-1) + f(n-2)

floor length = n, breadth is 1

```
int tiling(int n){
  if(n==0 || n==1){
      return 1;
  // vertical case
  int fnm1 = tiling(n-1);
  // Horizontal case
  int fnm2 = tiling(n-2);
  int NO.ways = fnm1 + fnm2;
      return NO.ways; }
```

For length = 3

ans = 3

Remove Duplicates in a String ; amazon

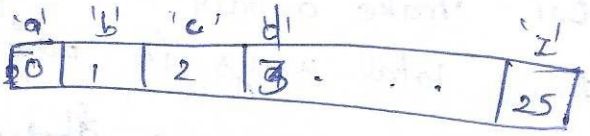
"appnacollege"

there are only lower case characters.

(1) create new strind String Builder sb

(2) index variable i

(3) map (boolean) []



index = currentchar - 'a'
(map)

⇒ example 'z' - 'a' = 25
'b' - 'a' = 1

removeDuplicates(String s)
StringBuild
bo

∴ if the charecter is their
map [i] = true;
sb.appeand(i);

and if (map[i] = true) {
~~skip off~~ continue;
}

like this only unique charecters will be their in the new string.

① base if (i == String.length) {
return sb.toString() }

② kam → is the char is present or not
✓ → new str x
x → add new str

③ ~~ret~~index → index + 1;

function(string str, int i, StringBuilder newStr, boolean map[]){

⇒

Goldman Sachs

Friends Pairing Problem :

We have n friends. A person can remain single or make a pair.
Find total ways in which they can be

⇒ total ways → standing in single + in paire.

For n = 1  (single) 1 way

For n = 2  either a, b single or pair (a, b)] 2 ways

For n = 3 

(a, b), c
a, (b, c)
(a, c), b
a, b, c
] 4 ways

choice

↙ single

↘ Pair

because it can make pair with any member ⇒

f(n-2) = (n-1) * f(n-2)

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

total ways = f(n-1) + (n-1) * f(n-2);

```
⇒ Friend ( int n) {        n = no. of people
  // base case
  if (n = 2 || n = 1) {
      return n; }

  // choice
  // single
  int fnm1 = Friend(n-1);

  // pair
// int fnm2 = Friend(n-2);
  // int pairWays = (n-1) * fnm2;

  return fnm1 + pairWays;
  }
```

Binary String Problem

o Paytm

Print all binary strings of size N without consecutive one.

that is 11

⇒ Base case

if (n=0) → "" empty.

if (n=1) → "1", "0".

if (n=2) → "00"
"01"
"10"
~~"11"~~

if (n=3)

Lp = last place

→

0	0	0	Lp=0	"000"
0	0	1	Lp=1	"001"
0	1	0	Lp=1	"010"
1	0	1		"101"
1	0	0		"100"

5 total ways.

```
Print BinString(int n, int lastPlace, String str){
  // base case
    if (n==0){
        System.out.print(str);
        return;
    // Kaam
    printBinString(n-1, 0, str + "0");
    if (lastPlace == 0){
      printBinString(n-1, 1, str + "1");
    }
```

Divide and Conquer algorithm

Merge Sort // Depth first
→ time complexity – $n(\log n)$ like inbuilt sort

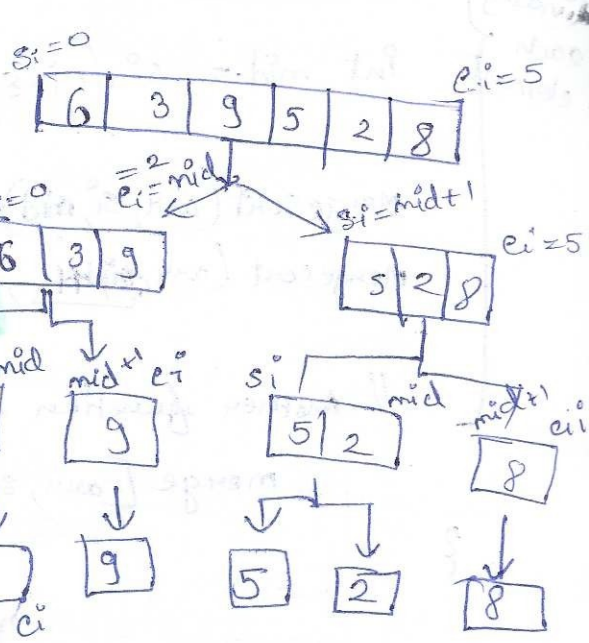
Approach

① to divide
↳ mid

to find out mid

mid → Starting index + (Ending index – starting index)/2

→ $S_i + (E_i - S_i)/2$



② mergeSort (left)
mergeSort (Right)

Now as the all elements are in single position
∴ They all are sorted.

⇒ Three iterators.

3 6 9
↑ ↑ ↑

2 5 8
↑ ↑ ↑

2	3	5	6	8	9

↳ using temrary array compare

① 3, 6 same from

then ③, ⑥ ⑨.

5	2	8

◎ Base case
$S_i > e_i$
$S_i = e_i$ (single)

◎ kam
divide
[mergeSort (left)
(right
merge.

si = starting index
ei = ending index

function (Mergesort(int arr[], int si, int ei)

dividing each element

//base case
if (si = ei){
return}

// kaam

int mid = si + (ei - si)/2 ; // or same as you can as (si+ei)/2

Mergesort(arr, si, mid); // passed ei as mid

Mergesort (arr, mid+1, ei); //passed si as mid + 1

// Another function for merging them

merge [arr, si, mid, ei]

}

merging

Public static void merge (int arr[], int si, int mid, int ei){

//creating temporary for saving array

int temp [] = new int [ei - si + 1]

int i = si;

int j = mid + 1;

int k = 0; //iterator for temp

while (i <= mid && j <= ei){

if (arr[i] < arr[j] {

temp[k] = arr[j]; //add smaller

j++; }

else { temp[k] = arr[i];

i++; } k++; }

remaining elements

while (i <= mid) {

temp[k++] = arr[i++];

}

while (j <= ei) {

temp[k++] = arr[j++];

}.

considered good & in sorting

time complexity = O(n log n) ✓

Space complexity = O(n)

∴ should not be used where space is left.

Quick Sort

Time complexity → Average cases → $O(n \log n)$

Rarely worst case → $O(n^2)$

Space complexity → $O(1)$

less than meerge sort.

=)

Approach

↳ Pivot and partition

(1) choose a pivot (the point around which we will do sorting)

but naturally we take pivot = last element

(2) Partition (parts) around pivot

a) will take pivot element as mid and then sort the small element in front of pivot

b) and big elements behinde the pivot (bigger than piviot)

(3) Quick Sort (left part)
(right part

At base case return;

Partion

here we take i = -1
the if we found a element smaller the piviot element we ① i++
② then swap the element at i and the larger element position
③ if not we do nothing and move forward
④ at last i++ and bring piviot at i++ and that element to the piviot

Quick sort's Worst time complexity

↳ When pivot is always smallest or the largest element

For example
When we have a sorted array.

Sorted and Rotated Array Search

* medium level
* important
* interview

(at pivot) (no duplicates)

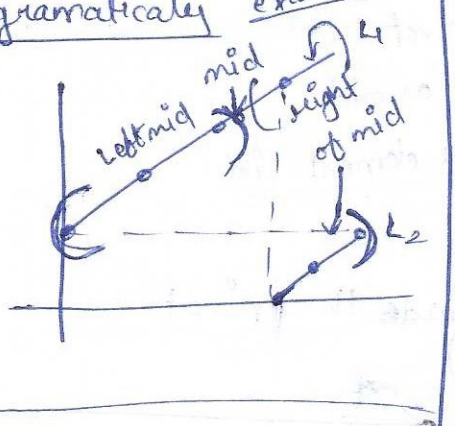
Q. We have sorted and rotated array with distinct numbers
Find the index of given element, key = 0

4	5	6	7	0	1	2

here we can se pivot was 2

⇒ to solve this we will you Divide and conquere
→ Modified binary search.

• Digramatically example



Step ①. we will find mid
$mid = si + \frac{(ei - si)}{2}$

Case 1 mid on L1

case a: we only need to search
↳ arr[si] <= arr[mid] ← range
↳ L1 left
si =< key <= mid

case b: ↓ else
right of mid.

Case 2: mid on L2
• range arr[mid] <= arr[ei]
case c: L2 right (mid =< key <= ei)
↓ else
case d: mid left

Time and Space Complexity

↳ These are two parameters to judge or compare same programs.

Time Complexity

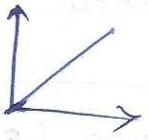
↳ Amount of time taken up by an algorithm/code as a function of input size.

Not the actual time taken.

$$time \propto Inputsize$$

- linear Search.

$$\text{time complexity} = O(n)$$



- In sorted array time complexity is constant

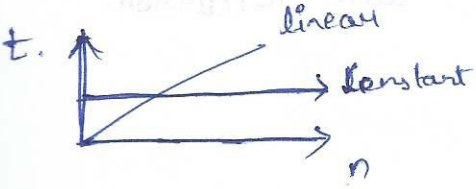
$$\therefore \text{time complexity} = O(1)$$

Constant

because same for every program

No loop

No function.



Big O notation → upper bound O()

↳ this the worst case time complexity

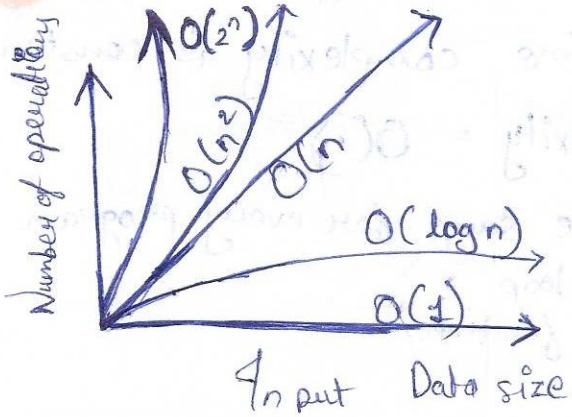
↳ program may ~~will~~ run faster than that but not slower than that.

Big Ω() → Best or lowest time complexity
Big omega
↳ lower bound.
↳ Program cannot be faster than this time.

Big Θ() → Average time complexity
Big Theta

Small 'o' of time complexity are the time complexities that are loosely packed.
↳ Any Big 'O' of time complexities are tightly packed.

Some Common Complexities



∴ if it said that input size is so short then their is good change that the code is solved with recursion.

↳ O(2^n) is the worst time complexity and we never use this in our code written in interviews.
↳ If comes when we use recursion with (memo) optimization.
↳ This time complexity is only applicable for small input size.

Space Complexity → Memory Space → Heap (1) / Stack (2)

input space + auxiliary space ✓ most of the time we are talking about this

input space: This is used in every program

For ex- If we have two

Merge sort	Quick sort
↑ Space comple	↓ Space . Complex
↓ Time . Complex	↑ Time . Complex

But we always give the priority to time complexity most of the time.

Theoretical Analysis

(1) Simple for loop

Time does not depend on the code inside the loop

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

⇒ Time complexity = $O(n \times k)$

but ignore the constant

$\therefore O(n)$

(2) Nested loop 1

```
for(i=0; i<n){
    for(j=i+1; j<n){ }
}
```

↳ here some times the inner loop work but some times not

↳ by a pattern we get (n-1) + (n-2) . . . . 1 + 0 and this is an A.P ∴

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

$\Rightarrow O\left(\frac{n^2}{2} - \frac{n}{2}\right)$

ignoring constant and taking largest no.

$= O(n^2)$

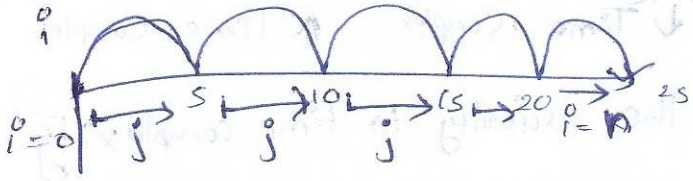
nested loop 2

T.C = $O(n^2)$

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

nested loop 3

```
for( i=0; i<n ; i= i+k) {
    for( int j= i+1 ; j<= k ; j++){
```



as j take the longer path ∴ time complexity = $O(n)$

Sorting

1) Bubble sort.

a) Worst case (array will be in reverse sorted)

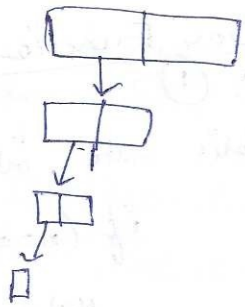
T.C = $O(n^2)$

b) Best case → here also T.C is $O(n^2)$ because the code needs to run the inner loop. T.C = $O(n^2)$

↳ to make it fast you need to write a optimised sort.

And optimised sort will give us less time complexity

Binary Search



↳ As it gets divide is two and then again divide int two

⇒ $T.C = \log(n)$

Recursive Algorithms

° Factorial code → n * k ⇒ $T.C = O(n)$

S.C = Max depth * each memory level occupied

$S.C = O(n)$

° Sum of n

↳ $T.C = O(n)$

$S.C = O(n)$

° Fibonacci

↳ here divide and conquer is used

⇒ $T.C = 2^n$ $S.C = O(n)$

° Merge Sort

↳ ① For merge function $T.C = O(n)$

② ~~divide~~

$T.C = n \times \log n$ $S.C = O(n)$

because of temp array.

) Power Function (Recursion)

Normal. (1)

```
Public static int power(int a, int n) {
    if (n==0){
        return 1;
    }
    return a * Power(a, n-1);
}
```

T.C = O(n)
S.C = O(n)

O(n)

optimized.

(2)

```
public static int power2(int a, int n) {
    if (n==0){
        return 1;
    }
    int halfpowerSq = power2(a, n/2) * power(a, n/2);

    if (n%2 != 0) { //a is odd
        return a * halfpowerSq;
    } return halfpowerSq;
```

T.C = O(n)

more optimized

(3)

```
public static int power3(int a, int n) {
    if (n==0){
        return 1;
    }
    int halfpower = power3(a, n/2);
    int halfpowerSq = halfpower * halfpower;
    if (n%2 != 0) { //a is odd
        return a * halfpowerSq;
    }
    return halfpowerSq;
}
```

T.C = O(log n)
S.C = O(log n)

How to approach Question ?

① Brute force (logical)

↓

optimize

B.F

- linear search
- Binary search
- if sorted direct.

→ in 1 sec 10^8 operations can be performed.

- watch input size for analysing time complexity.

~~Backtracking~~ - ~~Recursion~~

Backtracking

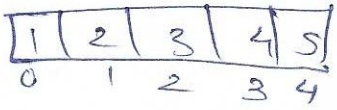
↳ Backtracking happens only after function call.

Types

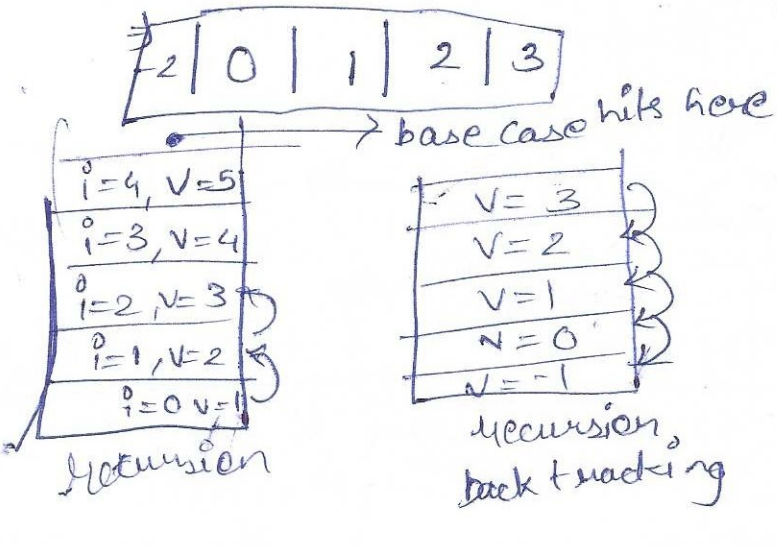
1. Decision (solution exist or not)
2. Optimization (best solution)
3. Enumeration (listing all posible solution)

Backtracking of an Array

↳ First we will make an array -



if (i = n-1) { back track and decrease each value by -2



code

```
public changeArr(int arr[], int
// base case
if(i == arr.length){
    printArr(arr);
    return;
}

// recursion
arr[i] = val;
changeArr(arr, i+1, val+1);
arr[i] = arr[i] - 2;
```

Time Complexity

→ O(n)

Find Subsets

→ Q. Find and print all subsets of a given string.

"abc"

a, b, c, ab, bc, ac, abc, " "
1 2 3 4 5 6 7 ↓ null set

8 subsets

n = no. of elements

Total subsets = 2^n

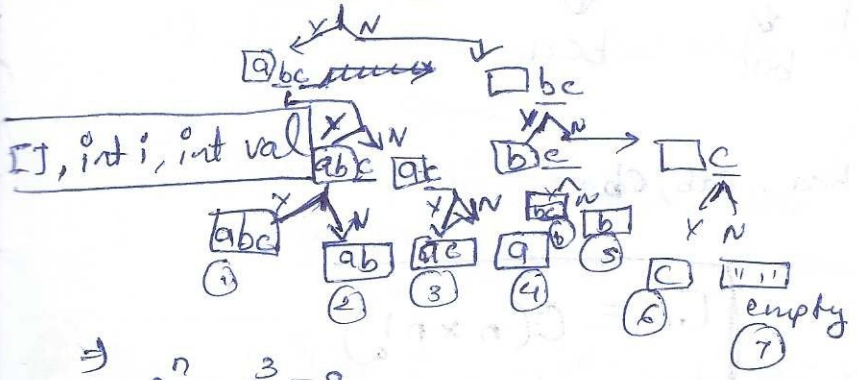
Sometimes i++ doesnot work inted of this use i+1

⇒ "abc" is a string

① we will start with a

each character has its choice to come in subset or not

⇒ a bc



⇒ $2^n = 2^3 = 8$

⇒ abc, ab, ac, a, bc, b, c, " " empty.

- Time Complexity ⇒ $O(n \times 2^n)$
- Space Complexity ⇒ $O(n)$

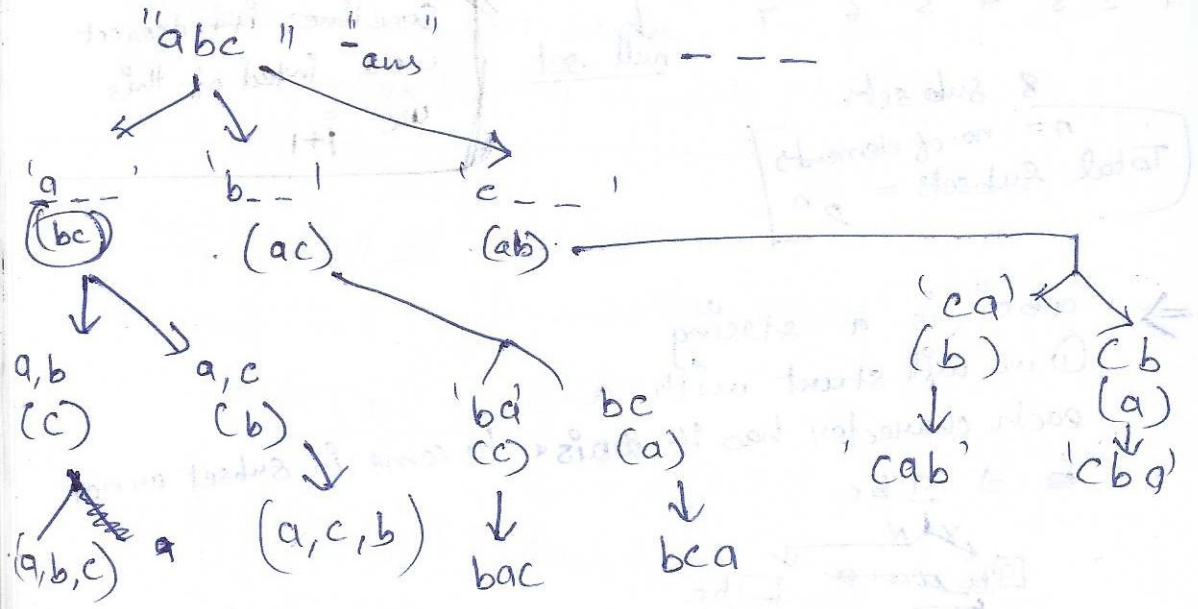
↳ If we solved this using String Builder then we also need to add a step to delete the element from the string when we come back.

Find Permutations

Q. Find and print all the permutations of a string?

→

> No. of permutation = Size of string !



abc, acb, bac, bca, cab, cba

→ $3!_{0} = 6$

$$T.C = O(n \times n!_{0})$$

→

N-Queens

* Place N queens on N×N chess board such that "no 2 queens can attack each other"

In this questions three thing is usualy asked

1) all solutions
2) Solution exists or not
3) Count all solutions.

- Conditions need to check. [box safe]
  1. Is an queen above
  2. Is a queen in above left digonal
  3. Is a queen in above right digonal

Time Complexity : the first Queen has n, other queen (n-1) like this $n(n-1)(n-2)(n-3)$

$\Rightarrow n!$

$\therefore T.C = O(n!)$

---

b) **N-Queens - Count ways.**

⇒ [in base case just make the count ++;]

↳ Make a static variable because in recursion in each level a new parameter is formed

---

c) Check is there is a sol^n and print one solution.

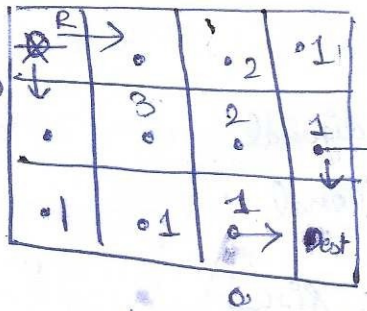
⇒ Just make the pattern function as bollean and if true then print pattern

---

Grid Ways

Q. Find the number of ways to reach from (0,0) to (N-1, M-1) in a N×M grid.

↳ Allowed moves → i) right
ii) down



→ one way that is down

$f(x,y) = f(x+1,y) + f(x,y+1)$

down | right

If we are target source = $(n-1), m-1$

→ one way

⇒ Total ways = cell → right $(x, y+1)$ w_1 | down $(x+1, y)$ way 2

total ways = way from 1 + ways from 2

∴ We get a recursive function as we move one step right or down we get a new grid and then for that new grid we need to find the ways

Time Complexity : $O(2^{n+m})$

Math trick for linear time for Grid ways

$$\text{Total ways} = \frac{(n-1+m-1)!}{(n-1)!\,(m-1)!}$$

Time complexity = $O(n+m)$

$= O(n+m)$

Sudoku

⇒ We are just solving the sudoku

→ For that we need to check the following condition

i) Is that number vertical same
ii) Is that number duplicate present horizontaly
iii) Is the number in-side the small grid.

↳ Duplicat present in?

If not then print that number.
If yes then try another number

From 1-9.

↳ in C++ this topic is Vectors.

Array List → It is a part of java collection frame work.

↳ pre defined data structure in java (inbuilt)

↳ Linear storing data

Advantages than array

(1) ↳ dynamic size (can be changed any time)

(2) → Only non primitive data types can be stored

int ✗ → Integer ✓ class

Syntax import.util.ArrayList; or util.*;

public class Classroom {

~~ArrayList<type>name~~ public static void main(String args[]) {

```
ArrayList<type> name = new ArrayList<>();
```

in type classes are used

example → <Integer>
<String>
<Boolean>

ex-

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

Operations on an Array list.

(1) Add a element.

Time - O(1)

```
object.add(element);
```

```
list.add(1);
list.add(2);
syso(list);
```

output
[1, 2]

② Get Element — Time: O(1)

```
list.get(index)
```

→ list.get(0);

ex- int a = list.get(0);
syso(a)

Output: 1

③ Remove Element — Time: O(n)

→ to delete an element on a index.

```
list.remove(index);
```

ex list = [1, 2, 3, 4, 5]
index - 0 1 2 3 4

```
list.remove(2);
syso(list);
```

Output
→ [1, 2, 4, 5]

④ Set Element at Index — Time: O(1)

```
list.set(index, element you want to store)
```

output:
[1, 2, 10, 4, 5]
remaining element moved forward.

```
list.set(2, 10);
syso(list)
```

⑤ Contains Element — Time: O(n)

↳ To check the element is present in the array not
If yes then it will return true is not then it will return false

```
list.contains(element);
```

output:
True

ex- syso(list.contains(2));

Time = O(n)

⑥ Add a element on specific index

```
list.add(index, element)
```

ex- list.add(1, 9);
syso(list);
output =
[1, 9, 2, 3, 4, 5]

Size of Array list

Syntax: `list.size()`

ex -

```
Sysо (list.size());
```

Output : 5

Print reverse of array list

```
for(i=list.size()-1; i>=0; i--){
    Sysо( list.get(i) );
}
```

Find Maximum

```
int max = Integer.MIN_VALUE;
Math.max (max, get(i));
```

Swap 2 Numbers

```
Swap( ArrayList<Integer> List, int idx1, int idx2){
    int temp = list.get(idx1);
    list.set(idx1, list.get(idx2));
    list.set(idx2, temp); }
```

Sorting an Array List

inbuilt method = `Collections.sort(name);`

accending order sort.

Sort in decending = `Collections.sort(list, Collections.reverseOrder());`

(name → list; comparator for defining the type of sort → Collections.reverseOrder())

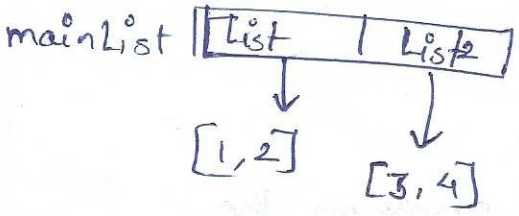
Multidimensional Array List

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

(1)
```
ArrayList<Integer> list = new ArrayList<>();
list.add(1); list.add(2);
mainList.add(list); // adding first list in main list
```

(2)
```
ArrayList<Integer> list2 = new ArrayList<>();
list2.add(3);
list2.add(4);

mainList.add(list2); // adding second list in main list.
```



Printing them

```
for(int i; i< mainList.size(); i++){
    ArrayList<Integer> curr = mainList(i);
    for(j=0; j< curr.size(); j++){
        Syso(curr.get(j));
    }
    Syso();
}
```

Container with Most Water — Time = O(n)

Companies- Flipkart, Rupzo

width = 7
height = 7
Total water = 7 × 7 = 49

Two Pointer Approach

- brute force - Time - $O(n^2)$
  - make a pair of every two lines and see which two lines can store max water
  - water ⇒ container height × width | container height = height of small side; width = $i^{th}_{last} - i^{th}_{First}$
- Optimized Solution - Time : O(n)

2pointer Approach.

↳ Water capicity of the contained depends on the
1. width
2. Height.

approach
- i = index of Left Pointer
- j = index of right pointer

while (i < j)

curr water = ht * width

↳ if we move the bar that have long height then capicity of the container will only decrease.

↳ so we move the bar that have smaller height.

ht = Min (L.P ht, Right Pointer height)

width = j - i

if (L.P height < R.P height) { L.P++ } else { RP++ }

How trapping rain water and container is different?

trapping rain water

i) here there are many boundries.

ii) Water we want that total stored in all containers.

container

i) Here it has only one container.

ii) Water we want that container has maximum capicity.

iii) here we have asked the capicity of max container.

Pair Sum → we have given Sorted Array List.

↳ find if any pair in the arraylist have targeted sum?

Optimized approach. ↳

[1, 2, 3, 4, 5, 6]

↑ L.P ↑ R.P

Two Pointer Approach

Case 1 = if (L.P [number on] + number on (R.P) == target) { return true; }

Case 2 = if (Num on (L.P) + Num on (R.P) < target) { L.P ++ }

Case 3 = if (num on (L.P) + num on (R.P) > target) { R.P-- }

→ Because we have sorted list.

Time Complexity: O(n)

Pair Sum - 2 → we have sorted and Rotated list

T.C = O(n)

Q. Find pair sum? target = 16

→ [11, 15, 6, 8, 9, 10] (breaking point)

↳ We will solve this with two pointer approach

We will take L.P = breaking point + 1

R.P = breaking point

→ then same approach.

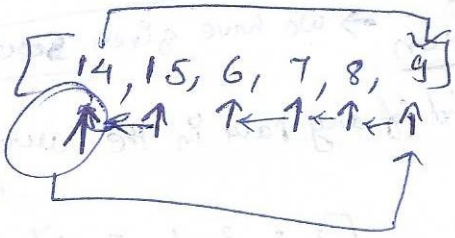
Modulo Arithmatic Properties

↳ this have many arithmatic rules or formulas which make our algorithm or logic so simple.

ex-① if we have a rotated sorted array.

And we want to run pointers

and we want



that when the pointer hits the last iterator it should go to the first iterator or jump to the last iterator.

→ property →

For moving forward.

left = (left + 1) % n

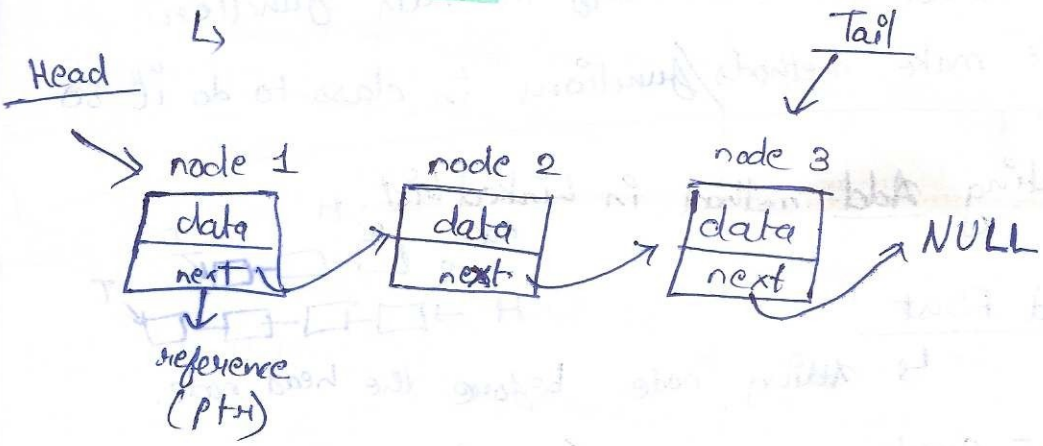
right = (right + n - 1) % n

For moving backwards

* used this

⊛ Pair Sum in Sorted rotated Array Question

Linked List



As each node stores same type of data means we know the blueprint of node.
So we will make a class and each node will be an object

Making node class.

```
Class Node {
    int data
    Node next;

    public Node(int data){ //constructor
        this.data = data;
        this.next = null;
    }
}
```

Head Node → First node

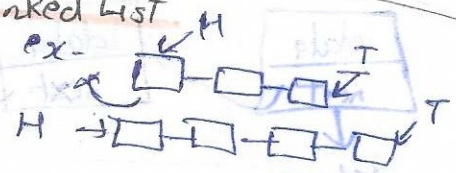
Tail Node → last valid node (Not the null one)

- Head Node = Tail Node if there is only one Node in list.

- To perform all the operations in a linked list we do not do it ourself in main function.

→ We make methods/functions in class to do it so

Creating Add method in Linked List



① Add First

↳ Adding node before the head node.

Step 1 - Create new node (Creating a node named Newnode)
step 2 - new node's next = head (Making it As head)
Step 3 - head = new Node. (transferring its data from head to Newnode)

Code -

Inside the Node class

T.C = O(1)

```
public void addFirst(int data){
    // step1 = create new node
    Node newNode = new Node(data)
    (class) (object)
        if(head == null){
            head = tail = newNode;
            return;
        }
    }
```

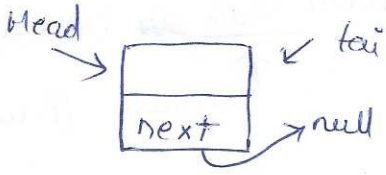
← because we have a single node in list / empty list

```
    // step 2 - newNode next = head
        newNode.next = head;
    // step 3 - head = NewNode
        head = newNode;
}
```

Creating Add Last method.

Step ① create a node (newNode)
Step ② tail.next = newNode
③ tail = newNode

If there was a null list then



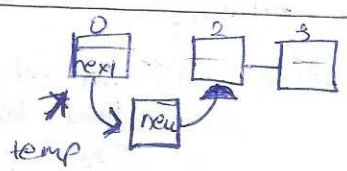
Code

```
public void addLast(int data){
    Node newNode = new Node(data); // step 1
    if (head = null){
        head = tail = newNode;
        return;
    }
    tail.next = newNode; // step 2
    tail = newNode; // step 3
```

Printing Linked List

```
if (head = null){ sout(Empty); return; }
while (temp != null){
    print(temp.data);
    temp = ~~next~~ temp.next;
}
```

Add a node in Middle




```
Node temp = head
i = 0
while (i < idx-1){
    temp → next;
    i++
}
```

Step 1; newNode.next = temp.next
Step 2; temp.next = newNode

Exception

```
if (index == 0){ addFirst(data);
    return; }
```

For finding the idx-1 position

Size of linked List

```
class LinkedList {
Public static int size;
```

↑ because it will change everywhere.

→ add size ++ in addFirst, addLast, add middle

Remove First Node

↳ We just chang the head and make the head to the second node. And when the garbage collector will come as the code run it will remove the first node thinking it as garbage.

```
public int removeFirst () {
    int val = head.data;
    size --;
    head = head.next;
    return val; }
```

Exception

① when our LL is empty

```
if (size == 0){
syso("Linkedlist empty");
}
```

② if (size == 1){

```
int val = head.data;
[redacted];
head = tail = null;
return val;
size = 0 }
```

Remove Last Node

```
Public int removeLast() {
  if (size == 0){ syso("LL is empty");
      return;
  if (size == 1){ int val = head.data;
                  head = tail = null;
                  size = 0; return val;}

Node prev = head;
for(int i=0; i< size-2; i++){
   prev = prev.next; }

   Prev.next = null;
   tail = prev;
   size --;
   return val; }
```

Iterative Search in a Linked List

↳ search a key and return its index; if not found return -1;

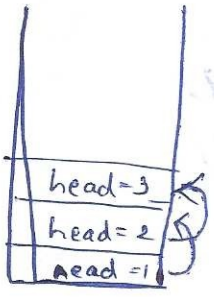
```
int i=0;
Node temp = head
while (temp != null) {
    if (temp.data = key) { return i; } else
    { temp = temp.next};
      i++;
return -1;
```

time = O(n)

Recursive Search

↳ Find the index of key? use recursion?

Base case



we will ask in each time is our key == head if not the put value again in the function.

And if key = head return return * i+1;

```
public int recSearch (int key) {
    return helper (head, key); }

Public int helper (Node head, int key) {
    if (head == null) { syso (ll is empty)
                        return -1 }
    if (head.data == key) { // base case
        return 0;
    }
        int idx = helper(head.next, key);
        if (idx == -1) { return -1; }
    return idx + 1;
}
```

3 variables 4 steps

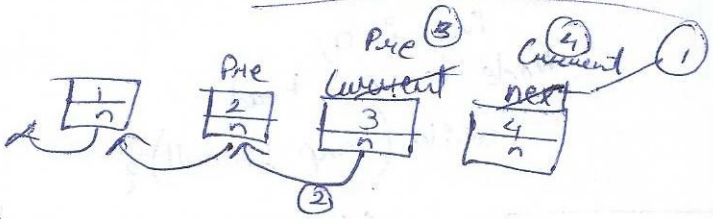
Reverse a Linked List

Q- 1 → 2 → 3 → 4 → null

Ans → null ← 1 ← 2 ← 3 ← 4

(Iterative approach)

Time = O(n)



Approach

1. nex = curr.next
2. curr.next = pre
3. current = pre
4. next = current

while (current != null)

Code:

```
public void reverse() {
    Node prev = null; // because there is no node before head.
    Node curr = tail = head; // As our new tail will be our head
    Node next;
    while (curr != null) {
        next = current.next;
        current.next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}
```

Q. # Find & Remove N^{th} node from End

amazon adobe Rubrikcom

(iterative Approach)

Ex- n = 3

1 → 2 → 3 → 4 → 5

3 2 1

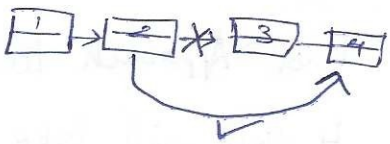
So we cannot go backward but there is a simplest approach that

n^{th} node from backward is $(size - n + 1)^{th}$ node

ex n = 3

= 5 - 3 + 1 = 3rd node.

And to remove a node just change the pointer of the previous node by pointing it towards the next node of n^{th} node.



Code :

```
public removeFrom nth() {
// to calculate size
node temp = head;
int sz = 0;
while (temp != null) {
    temp.next;
    sz++;
}
// if we have n=1 node.
if (n = sz) {
    head = head.next;
    return; }
// sz - n
int i = 1;
int iToFind = sz-n;
Node prev = head;
while (i < iToFind) {
    prev = prev.next;
    i++;
}
Prev.next = prev.next.next;
return;
}
```

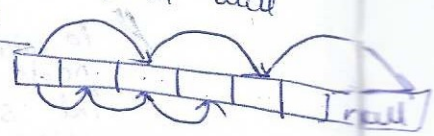
if the element is the first element that we want to remove

Corner Case

Check if L.L is a Palidrome

Slow-Fast Approach to find mid

↳ slow will take one step at a time and fast will take two step at a time



when the fast reaches at the end the slow will be at middle.

(1) for even case when
fast will at null
slow will be at mid

(2) for odd case when
fast will be at fast.next=null
Slow = mid.

Palidrome

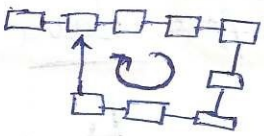
(1) find middle
(2) 2nd half reverse
(3) check if 1st half == 2nd half

• Corner Case / Exception

↳ if size of of L.L is odd then mid will be slow.next

```
if(fast!=null){ //odd case
    mid=slow.next;}
```

Detect a Loop/Cycle in a L.L



So we use Floyd's Cycle Finding Algorithm

↳ Slow Fast Approach

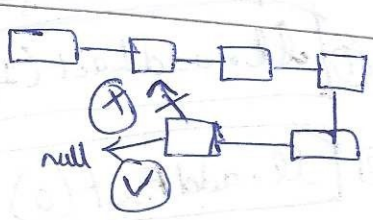
⇒ We will run slow and fast and if the

if (slow = fast) // If it is loop then it will happen once
{ then the loop exists};

slow = head
fast = head
while (fast != null && fast.next != null)
{ slow = slow.next;
fast = fast.next;

if (slow.data = fast.data)
{ return t; }
}
return false;
}

Remove the Cycle code



① find the last node
② point it towards null

Approach ① detect cycle Node prev = null
② after detecting make again
slow = head
then slow+1;
fast+1;

while (slow == fast)
prev = fast;
slow ++;
fast ++; }

③ the make the end point towards null
④ and point prev towards the fast.next or slow.

And # the node where the slow and fast meet again is the point of end.

Java Collection Frame Work

↳ Array List,
Linked List
Stack
Queue
Hashset
HashMap

→ We learn to creat a data structure from scratch because interviewer can ask us to make data structure from scratch.

→ In C.P we will use inbuilt data structures.

Linked List in Java frame work

↳ Linked list is same as array list As it is a class we have to make objects in it with using <Integer>, <Float>, <Boolean> classes as my data types.

Creating Linked List

```
LinkedList<Integer> ll = new LinkedList<>();
```
(data type: Integer, name: ll)

① `ll.addLast(1)` ← Adding elements in last.

② `ll.addFirst(0)` ← Adding elements in starting

③ `ll.removeFirst();` ← removes element from first

④ `ll.removeLast();` ← removes element from last.