D.S.A · C++

Transistor :- 0 or 1
off on

Binary no - $\{0, 1\}_{base=2}$

0 + 0 = 1

1 + 0 = 1

0 + 1 = 1

1 + 1 = 1 0 (Carry, Base)

Conversion :-

$(27)_{10} \rightarrow (\ ?\ )_2$

2	27	1
2	13	1
2	6	0
2	3	1
	1	

Quotient, Remender

$\Rightarrow (11011)_2$

2	43	1
2	21	1
2	10	0
2	5	1
2	2	0
	1	

$\Rightarrow (101011)_{10} \rightarrow (43)_{10}$

---

(4) Decimal → other ⇒ Divide with base.

(5) other → Decimal ⇒ Multiply with (base)^n n=0,1,2

other

octal ⟷ binary

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

Octal ⟷ hex

⇒ Change in binary then convert

hexal ⟷ binary

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Moore law : Every 2 year transistor double

Machine language → binary

Assembler Compiler Interpreter

Data = Store / Fetch.

↳ Space ↓

↳ Time ↓

Flowchart & Pseudocode

Diagram representation Instructions

① Understand
② Given value
③ Approach
④ Code
⑤ Error / debug
⑥ Other Solution

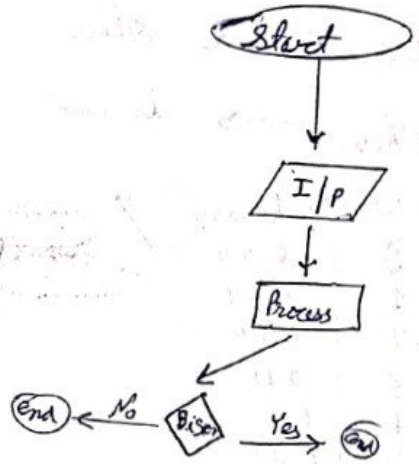
① Terminal (Start, End).

② Parallelogram (I/P, O/P)

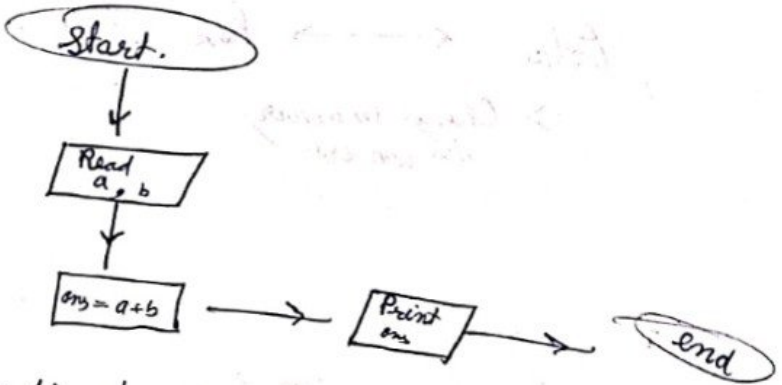
③ Process (Calculation & other)

④ Decision

⑤ Arrow (Flow)



→ Sum of 2 no



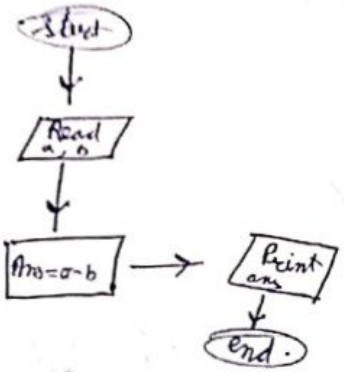
Pseudocode → basic instruction of english about code.

① Read a, b

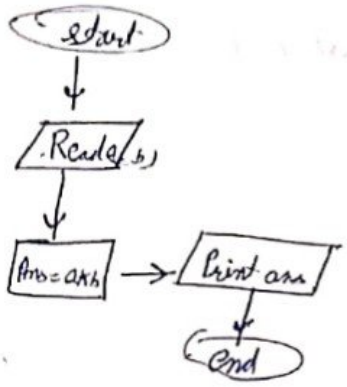
② ans = a+b

③ Print ans.

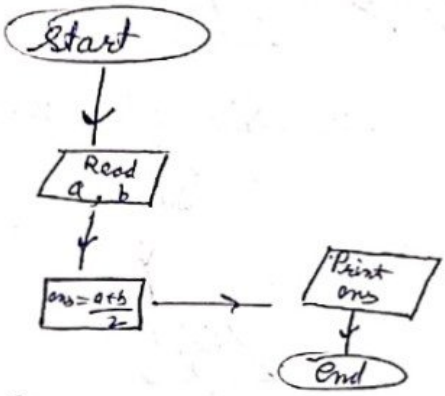
② Sub 2 no. →



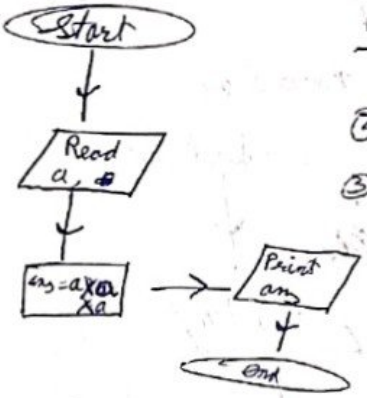
③ Product 2 no.



④ Average of 2 No.

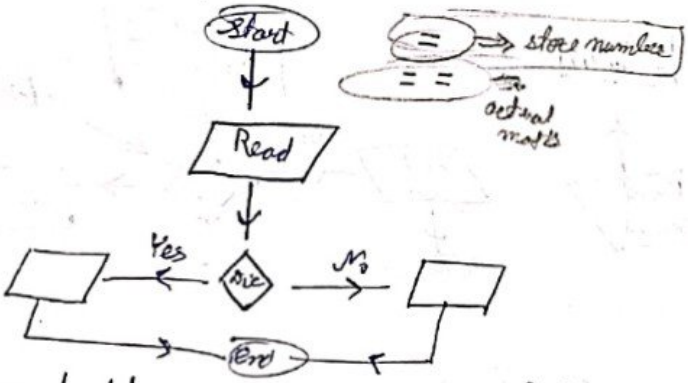


⑤ Cube of a number.

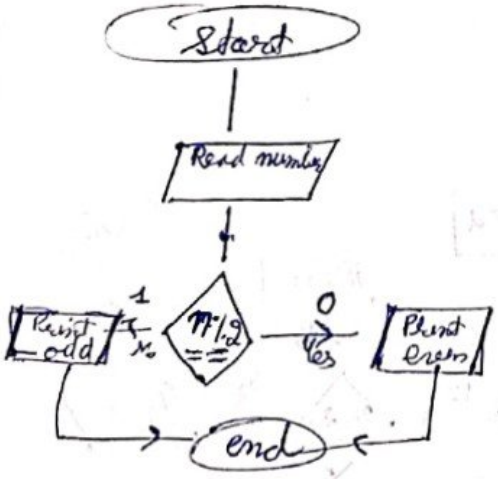


Pseudo code →
① Read n
② ans = n x n x n
③ Print ans

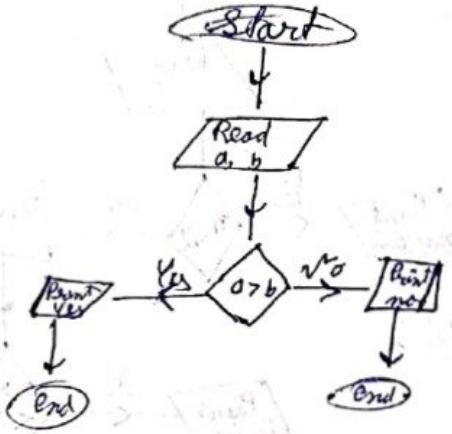
→ ~~Design~~ Decision making : ◇



⑥ Even / odd



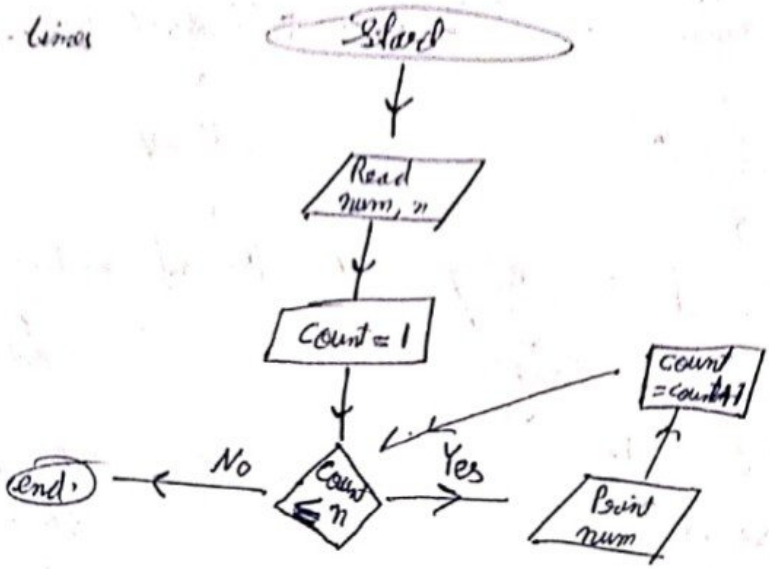
② Check a > b.



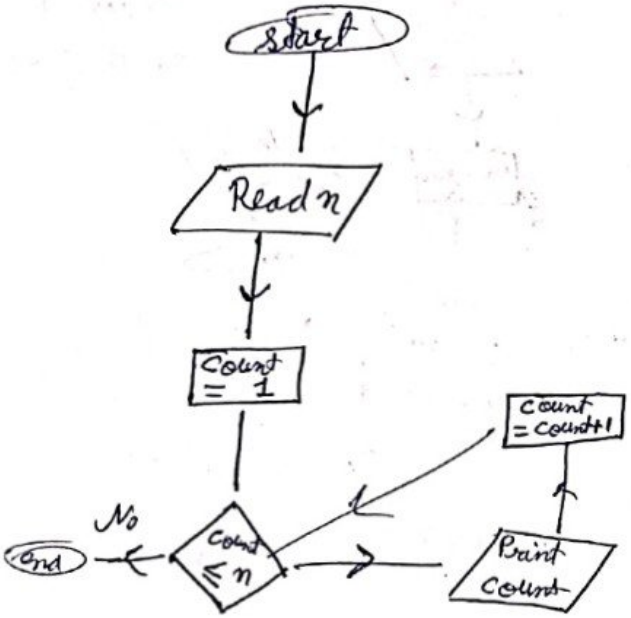
③ Check which is greater among 2 no.

⑧ Loop :-

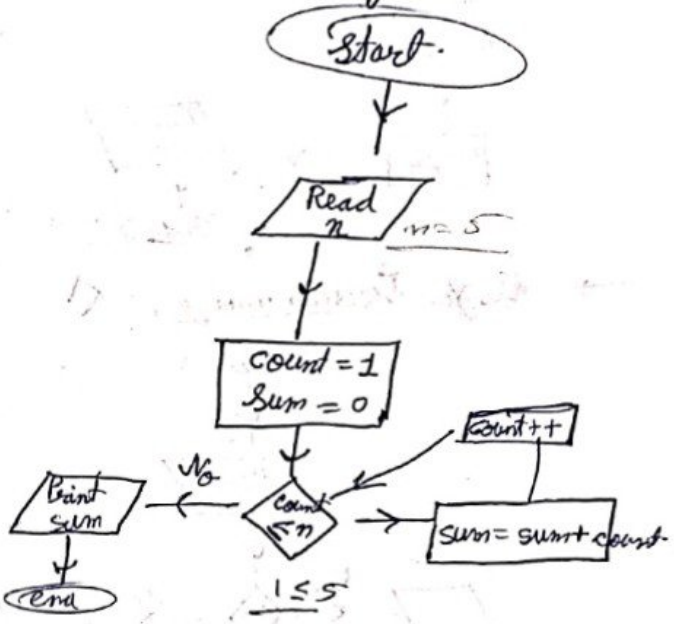
. num, Print n times



① Print n - natural no.

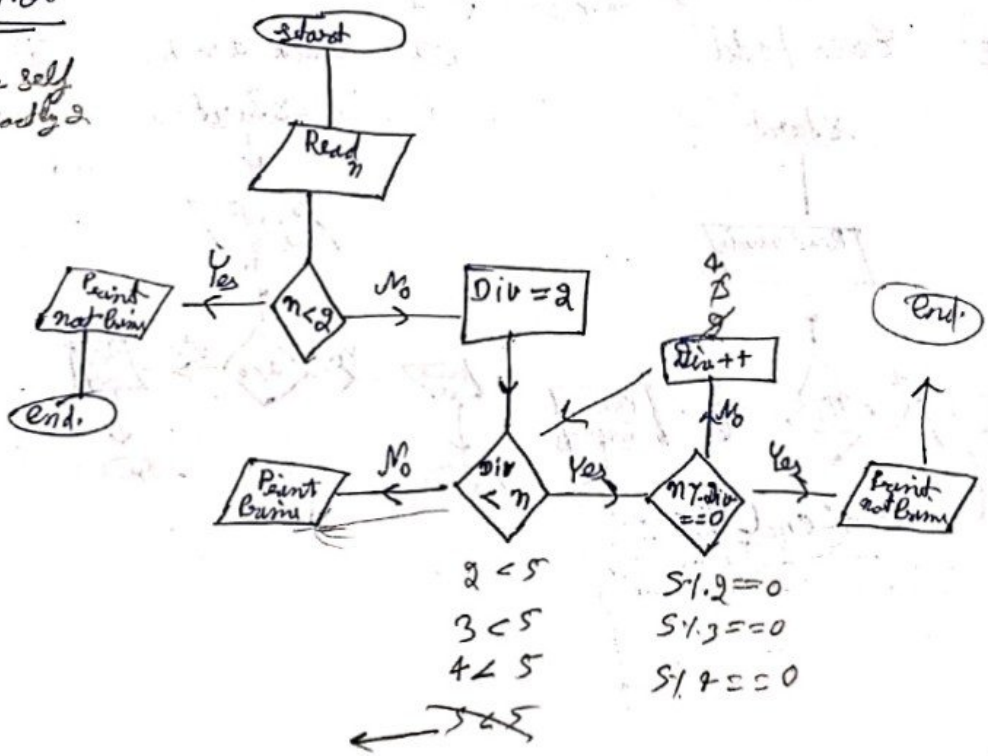


② Sum of n natural



③ Prime or Not

Prime => 1 or self
=> exactly 2



C++ - Introduction

Flowchart → [C++] → compiler → [Exe]

Transistor: → 0 off / 1 on ⇒ store 1 bit at a time

bit = 0 / 1.
8 bit = 1 byte.
1024 byte = 1 KB.
1024 KB = 1 MB.
1024 MB = 1 GB.
1024 GB = 1 TB.

ASCII: American Standard Code for Information Interchange.

A = 65 — a = 97
B = 66 — b = 98

• C++

① Syntax :

• .cpp

① <iostream>

```
#include<iostream>
int main ()
{
  std::count <<"Hello CR";
  std::cin
  std::vector
};
```

②

```
#include <iostream>
Using namespace std;
int main ()
{
   count <<"Hello CR";
   cin
   vector
};
```

→

count << 2+3; → 5.

count << "2+3"; → 2+3.

IDE = Integrated Development Environment
↓
V.S. code.

→ Next line ⇒ count << "Hello CR" << endl; ⇒ \n in C

→ also could use \n ⇒ count << "Hello CR\n";

* Variables & Datatype :

① INT : 1, 2, - - -
② float : 1.2, 2.7, 7.3
③ Double : 1.293, 2.678
④ Char : a, b, c
⑤ String : How are you
⑥ Boolean : 0 or 1

① int Name = 10 ;
↓ data Type
↓ Variable
assignment Operator

10
Name → 4 Bytes

4 Byte = 32 bit

Name
1 0 1 0 - - - 32 bit

10 = 1010 ⇒ 0 0 0 0 0 0 .. 1 0 1 0 ⇒ 32 bit

② Char ch = 'a' / 'b' / '0' / '1' / '2' ... ;

Char → data.

ch → Variable
{ Alphabet, (1–9), (_) }
→ (Not start with Number)

'a' → ascii = 97 → 1 Byte. → 8 bit
ch

Char ch = ab → only store single character

0 1 1 0 0 0 0 1 ⇒ 97
ch

③ float : 1.2, 2.6 ....

float f = 1.28 ; → 4 bytes.
data var.

1.28 = ----------
f

④ double d = 4.63326278
→ 8 bytes (64 bit)

→ int a = 29876348--58 ⇒ 40 bit number.
↳ but int store 32 bit (4 bytes).
→ Use long int a =..... ⇒ store 64 bit (8 bytes).

⑤ ~~Bol~~ Boolean : b = 0/1 ⇒ 1 byte.
data var

bool b = 0; (b = 0, b = 1, b = true, b = false)

0 → 1 byte (8 bit)
b

0 0 0 0 0 0 0 0 = 0

bool b = 0 ; / bool b = false

cout << b ; → 0

* Negative No

```
int a = -5;
```

as int = 4 byte (32 bit)
↳ Total 2^{32}

→ -2 ⇒ 2 = 0 1 0

1 0 1 ⇒ 1's complement

+1

1 1 0 → 2's complement ⟶ represent -2 in binary.

Let 3 bit

— — — ⇒ $2^3 = 8$

0	0	0	→ 0
0	0	1	→ 1
0	1	0	→ 2
0	1	1	→ 3
1	0	0	→ -4
1	0	1	→ -3
1	1	0	→ -2
1	1	1	→ -1

⇒ — — —

(-ve) ↙ ↘ (+ve)

2^2 space. ($-1 \to -2^2$)

2^2 ($0 \to 2^2-1$)

⇒ unsigned int — —

↳ Store only +ve numbers

32 bit

-ve: 2^{32} $(-1, -2, \ldots, -2^{31})$

+ve: 2^{32} $(0, 1, 2 \ldots, 2^{31}-1)$

→ Size of :-

```
int a = 123;
int size = sizeof(a);
cout << "Size of a is : " << size << endl;
```

↓

4

→ Type casting

① `int a = 'a';`
`cout << a << endl`

↓

97

② `char a = 98;`
`cout << a << endl;`

↓

b

→ operator:

+ - * / %

→ Relational oprtr

> < == != >= <=

= ↳ assignment operator

&& → and

|| → or

! → not

If Else Conditional Statement

→ input : cin

```
#include <iostream>
using namespace std;
int main () {
    int a, b;
    cin >> a;
    cin >> b;
    cout << a+b;
}
```

<< ⇒ Insertion operator.

>> ⇒ Extraction operator.

= ⇒ assignment operator

== ⇒ equal (comparision operator)

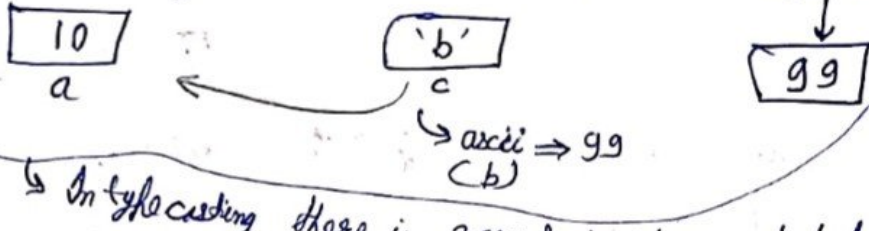
→ Typecasting

ascii ⇒ 0 - 255

int a = 10;

char c = 'b';

a = c



→ In typecasting there is very high chance of data loss if first datatype is bigger then assigned datatype then remain data will lost.

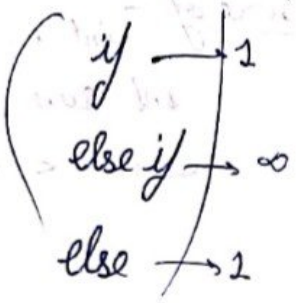
If - else.

◇ → (Division making)

Syntax

```
int Package = 15
if (Package > 10)
{
    cout << "accepted";
}
else
{
    cout << "rejected";
}
```

True then Run



① Comparison 2 number.

② Check Even or odd.

③ Vowel & Consonant. (if - elseif - else).

④ Number (1-7) print week (Mon - Sun).

loops

→ Count = 1
Count ≤ 5
Count "C R"
Count = Count + 1
variable ↓ assignment opr.

Syntax:

```
for (initilize ; loop break ; increment / Decrement)
{
    // code.
}
```

```
for(int i=1 ; i ≤ 10 ; i = i+1)
{
    cout << "Hello ";
}
```

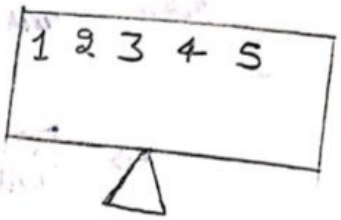
① ② ③ ④

i = 1
i ≤ 10
Print Hello
i = i + 1.

→ Print numbers (1-5)

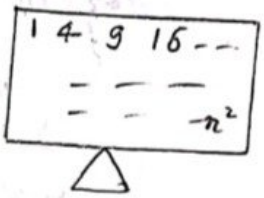
```
for (count=1 ; count≤5 ; count=count+1)
{
    cout << count;
}
```

→



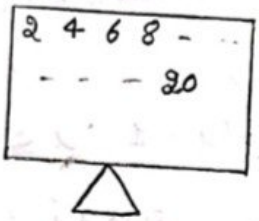
→ Print Square (1 - n)

```
for (int i=1; i ≤ n ; i++)
{
    cout << i*i;
}
```



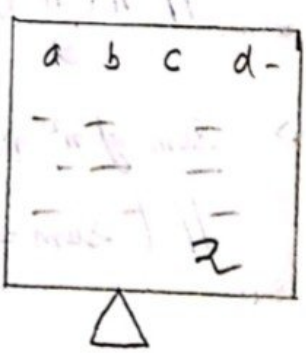
→ Print all even no (1-20)

```
for (int i=1; i ≤ 10; i++)
{
    cout << 2*i;
}
```



→ Print alphabate {a - z}

```
char name = 'a';   (Typecasting)
for (name = 'a'; name ≤ 'z'; name++)
{
    cout << name;
}
```



→ Table

```
int n;
for(int i=1; i<=10; i++)
{
cout<<n<<"X"<<i<<"="<<n*i<<endl;
}
```

⇒

```
7X1= 7
7X2 = 14
 .
 .
 .
7X10=70
```

→ Power.

n = 5 Pow = 4

5^4 → n*n*n*n

```
num = 5;
for (i=1; i<Pow; i++)
{
    num = num * n.
}
```

⇒ Debugging num=5; 4

i=1 ; i < Pow ; i++

num = 5 X 5 ①

25 × 5 - ②

125 × 5 ③

625 ⇒ Print

→

```
int num, Pow, n;
Cin >> num >> Pow;
  n = num
for(int i=1; i<Pow; i++)
  {
     num = num * n;
  }
  cout << num;
```

→ Sum of n natural no

```
Sum = 0
for (int i=1; i<=n; i++)
{
  Sum = sum + i;
}
cout << sum;
```

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

→ Sum of n^2 no.

// [Sum = sum + i * i]

→ Factorial :

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

→ Prime number :

```
if (num < 2)
{
    cout << "Not Prime";
    return 0;
}
else {
    for(int i = 2; i < num; i++)
    {
        if (num % i == 0)
        {
            cout << "Not Prime";
            return 0;
        }
    }
    cout << "Prime";
}
```

→ Fibonacci Series :

fibo : 0 1 1 2 3 5 8 13 21 34.

F0 F1 F2 F3

0 + 1 → 1

Pre Previous + Previous → current

Current = Prev + Pre Prev

Pre Previous = Previous

Prev = current.

```
int fib1, fib2, fibn, n;
    cin >> n;
if (n == 0) {
    cout << 0;
    return 0;
}
else if (n == 1) {
    cout << 1;
    return 0;
}
fib1 = 0;
fib2 = 1;
for
```

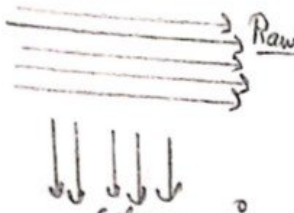
fibn = fib1 + fib2;
fib2 = fib1;
fib1 = fibn;

```
for(int i = 2; i <= n; i++)
{
    fibn = fib1 + fib2;
    fib1 = fib2;
    fib2 = fibn;
}
cout << fibn;
}
```

Pattern Printing

→
```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```
for(i=1; i<=5; i++)
{
    cout << "* " << " ";
}
cout << endl;
```




```
for(j=1; j<=5; j++)
{
    for(i=1; i<=5; i++)
    {
        cout << "* " << " ";
    }
cout << endl;
}
```

Nested loop

→ j=1, i=1 i=2 i=3 i=4 i=5

→ j=2, i=1 i=2 i=3 i=4 i=5

→
```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

```
for(i=1; i<=5; i++)
{
    for(j=1; j<=5; j++)
    {
        cout << i << " ";
    }
    cout << endl;
}
```

→
```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```
⇒ Use column to print (cout << j ;)

→
```
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
```

```
for(i=1; i<=5; i++)
{
    for(j=5; j>=1; j--)
    {
        cout << j << " ";
    }
cout << endl;
}
```

→
```
5 5 5 5 5
4 4 4 4 4
3 3 3 3 3
2 2 2 2 2
1 1 1 1 1
```

```
for(i=5; i>=1; i--)
{
    for(j=1; j<=5; j++)
    {
        cout << i << " ";
    }
    cout << endl;
}
```
↔
```
for(i=5; i>=1; i--)
{
    for(j=5; j>=1; j--)
    {
        cout << i << " ";
    }
    cout << endl;
}
```

→
```
1 4 9 16 25
1 4 9 16 25
1 4 9 16 25
1 4 9 16 25
1 4 9 16 25
```

```
for(raw=1; raw<=5; raw++)
{
    for(col=1; col<=5; col++)
    {
        cout << col*col << " ";
    }
    cout << endl;
}
```

→
a b c d e
a b c d e
a b c d e
a b c d e
a b c d e

```
for(int i=1 ; i<=5 ; i++)
{
  for(char j='a' ; j<='e' ; j++)
  {
    cout << j << " ";
  }
  cout << endl
}
```

diff ways
1 → a(5)
2 → b(5)
etc

→
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

1 - 1
2 - 6
3 - 11 ⇒ (i-1)*5
4 - 16
5 - 21

```
int i, j, count = 1
for (i=1 ; i<=5 ; i++)
{
  for(j=1 ; j<=5 ; j++)
  {
    cout << count << " ";
    count++;
  }
  cout << endl;
}
```

```
for(int row=1 ; row<=5 ; row++)
{
  for(int col=1 ; col<=5 ; col++)
  {
    cout << (row-1)*5+col
         << " ";
  }
  cout << endl
}
```

→
*
* *
* * *
* * * *
* * * * *

① row = 1
② row <= 5
③ Print '*' row times
④ row = row + 1

```
for (i=1 ; i<=5 ; i++)
{
  for(j=1 ; j<=i ; j++)
  {
    cout << "* ";
  }
  cout << endl
}
```

→
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1

① row = 1
② row <= 5
③ Print row to 1
④ row = row + 1

```
for (i=1 ; i<=5 ; i++) {
  for (j=i ; j>=1 ; j--) {
    cout << j << " ";
  }
  cout << endl;
}
```

→
a
b b
c c c
d d d d
e e e e e

① row = 1
② row <= 5
③ Print name, row times
④ row = row + 1
name = 'a' + (row - 1);

```
for(row=1 ; row<=5 ; row++) {
  char name = 'a' + (row-1);
  for(col=1 ; col<=row ; col++) {
    cout << name << " ";
  }
  cout << endl
}
```

→
* * * * *
* * * *
* * *
* *
*

row	Print
1 →	5
2	4
3	3
4	2
5	1

⇒ 5 - (row - 1).

→

$n = 5$

row	star	space
1	1	4 (5-1)
2	2	3 (5-2)
3	3	2 (5-3)
4	4	1 (5-4)
5	5	0 (5-5)

① row = 1
② row ≤ 5
③ Print space, 5-row times.
④ Print * row times
⑤ row = row + 1.

row = 1 →
col → 1 2 3 4 | 5-row = 4

```
for (row = 1; row ≤ 5; row++)
{
    for (col = 1; col ≤ (5 - row); col++)
    {
        cout << " ";
    }
    for (col = 1; col ≤ row; col++)
    {
        cout << "*";
    }
}
```

→ (23_cpp)

```
        A
      A B
    A B C
  A B C D
A C B D E
```

① row = 1
② row ≤ 5
③ Print space (5-row) times.
④ Print 'A' → 'A' + (row-1).
⑤ row = row + 1

```
for(row = 1; row ≤ 5; row++) {
  for(col = 1; col ≤ 5-row; col++)
    { cout << " "; }
  for(char name = 'A'; name ≤ 'A'-row+1
        ; name = name+1)
    { cout << name << " "; }
  cout << endl;
}
```

⇒ also
char name = 'A' + (col - 1);

→

```
      1
    2 1
  3 2 1
4 3 2 1
5 4 3 2 1
```

① row = 1
② row ≤ 5
③ Print space. (5-row) times
④ Print row do 1
⑤ row = row + 1

```
for(row = 1; row ≤ 5; row++)
{ for(int col = 1; col ≤ 5-row; col++)
  { cout << " "; }
  for(int col = row; col >= 1; col--)
  { cout << col << " "; }
  cout << endl;
}
```

→ Pyramid.

Pyramid

$n = 5$

row	space	*
1	4	1
2	3	3
3	2	5
4	1	7
5	0	9

$* (2 * row - 1)$

① row = 1.
② row ≤ 5
③ Print space (5 - row).
④ Print star (2*row - 1).
⑤ row++.

```
for(row=1; row<=5; row++){
    for(col=1; col<=5-row; col++)
    {
      cout << "_ _ ";
    }
    for(col=1; col<=(2*row-1); col++)
    {
      cout << "*_";
    }
    cout << endl;
}
```

Palandrom → Naman

```
          1
        1 2 1
      1 2 3 2 1
    1 2 3 4 3 2 1
  1 2 3 4 5 4 3 2 1
```

$n = 5$

space → no
5-row
1 → row (increasing)
(row-1) → 1 (decreasing).

① row = 1
② row ≤ 5
③ print space (5 - row)
④ print {1 → row} (inc)
⑤ print {(row-1) → 1} (dec).
⑥ row++.

```
for(row=1; row<=5; row++)
{
  for(col=1; col<=(5-row); col++)
    { cout << "_ _ ";
    }
  for(col=1; col<=row; col++)
    {
      cout << col << "_";
    }
  for(col=(row-1); col>=1; col--)
    { cout << col << "_";
    }
  cout << endl;
}
```

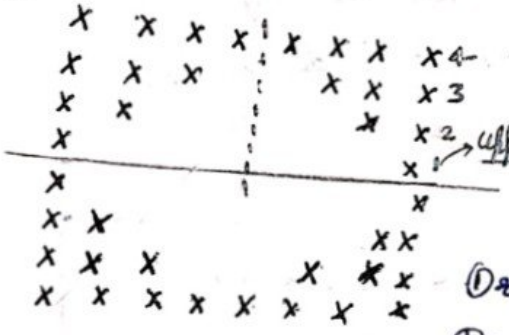
→

row	stars	spaces (row-1)
1	9	0
2	7	1 ⇒ (5 - row)
3	5	2
4	3	3
5	1	4
↓	↓	↓
run half 5 times	(2n-1)	(row-1)

```
for(row = 5 ; row >= 1 ; row--)
{
   for(col = 1 ; col <= (5-row) ; col++)
   {
      cout << " - ";
   }
   for(col = (2*row-1) ; col >= 1 ; col--)
   {  also for(col=1 ; col <= (2*row-1) ; col++)
      cout << " * ";
   }
cout << endl;
}
```

① row = 5
② row >= 1
③ print space (1 → 5-row) dec
④ print stars (2*row-1 → 1) dec
⑤ row++.

→



n = 4

row	stars	spaces
4	4, 4	0
3	3, 3	2
2	2, 2	4
1	1, 1	6
	row=star	8 - 2row ⇒ 2n - 2row

Upper half

① row = 4
② row >= 1
③ Print *, row times
④ Print space (8-2row) ⇒ 2n - 2row
⑤ Print * row times
⑥ row = row - 1 (row--)

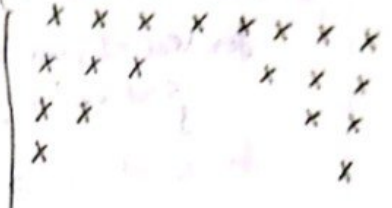
Lower half

① row = 1
② row <= 4
③ Print '*', row times
④ Print space (2n - 2row)
⑤ Print '*', row times
⑥ row++.

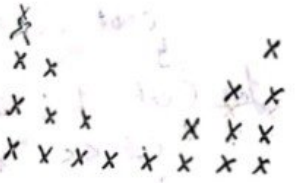
```
for(row = 4 ; row >= 1 ; row--) {
    for(col = 1 ; col <= row ; col++)
    {  cout << "*_";
    }
    for(col = 1 ; col <= 8-2*row ; col++)
    {  cout << "__";
    }
    for(col = 1 ; col <= row ; col++)
    {  cout << "*_" ;
    }
    cout << endl;
}
```

```
for (row = 4 ; row >= 1 ; row - -)
 {
   for (col = 1 ; col <= row ; col++)
    { cout << "* _ " ;
    }
   for (col = 1 ; col <= (2n - 2row) ; col++)
    { cout << " _ _ " ;
    }
   for (col = 1 ; col <= row ; col ++)
    { cout << "* _ " ;
    }
  cout << endl ;
}

for (row = 1 ; row <= 4 ; row ++)
 {
   for (col = 1 ; col <= row ; col++)
    { cout << "* _ " ;
    }
   for (col = 1 ; col <= (2n - 2row) ; col++)
    { cout << " _ _ " ;
    }
   for (col = 1 ; col <= row ; col ++)
    { cout " * _ " ;
    }
  cout << endl ;
}
```

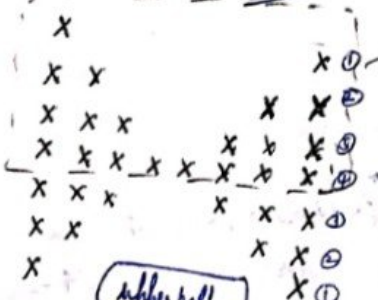
→ Print upper half pattern



→ Print lower half



→ Butterfly Pattern n = 4



row	*	space.
1	1, 1	6 ⇒ 8 - 2*row
2	2, 2	4
3	3, 3	2
4	4, 4	0
	row = star	2n - 2row

upper half

① row = 1

② row <= 4.

③ Print *, row times

④ Print space ; 2n - 2row.

⑤ Print * , row time

⑥ row++;

lower half

① row = 3

② row >= 1

③ Print * , row times

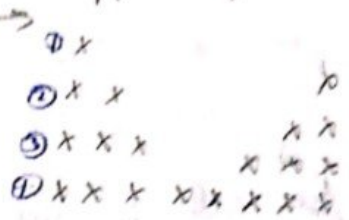
④ Print space (2n - 2row) ⇒ (8 - 2row)

⑤ Print * , row time

⑥ ~~Print space~~ row++.

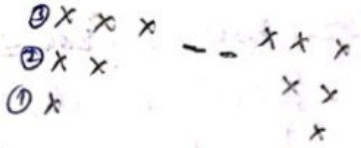
```
for (row=1 ; row<=n ; row+1) {
    for (col=1; col<=row; col+1) {
        cout << "*_ ";
    }
    for (col=1; col<=(2n-2row); col++) {
        cout << " __ ";
    }
    for (col=1 ; col<=row ; col++) {
        cout << "*_ ";
    }
} cout << endl;
```

upper half

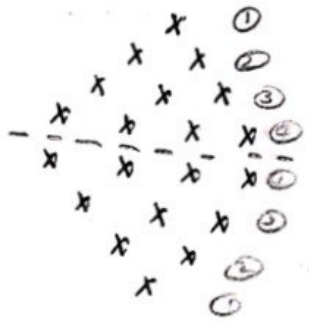


```
for (row=(n-1) ; row>=1 ; row++) {
    for (col=1 ; col<=row ; col++) {
        cout << "*_ ";
    }
    for (col=1 ; col<=(2n-2row); col++) {
        cout << " __ ";
    }
    for (col=1 ; col<=row ; col++) {
        col << "*_ ";
    }
    cout << endl;
}
```

lower half.



→ Diamond



row	space	star
1	3	1
2	2	2
3	1	3
4	0	4
	sp = (4-row)	star = row

n-row

_	_	_	X	o			
_	_	X	o	X	o		
_	X	o	X	_	X	o	
X	o	X	o	X	o	X	o

→ " _ " ⇒ 1 bar space

→ "*o" → 1 star, 1 space

```
for(row=0 ; row<=4 ; row++) {
    for(col=1; col<=(4-row); r++)
    {  cout << " _ ";
    }   → 1 bar space only
    for(col=1 ; col<=row; col++)
    {
      cout << "*_ *1 ;
    }
  cout << endl;
}
```

→ Operators in C++ (int < float < double) → Precedence

- Arithmetic Operator

Binary
+
-
*
/
% (Rem).

operator
2 + 3
operand

Precidence.
{*, /, %} > {+, -}

Associativity : left → Right

Unary
++ , --

① Post increment (a++)
② Pre increment (++a)
③ Post decrement
④ Pre decrement

a = 10

⇒ b = a++ | b = ++a
b = 10 | a = 11
a = 11 | b = 11

first assign values then operate. | first increase then assign

→ Comparision operator : {==, >, <, >=, <=, !=}
(1, 0, ↳ bit wise opte).

{>, <, >=, <=} > {==, !=}

Associativity : left to Right

→ 5 > 4 > 3
left to Right
1 > 3 ⇒ 0

⇒ 3 > 4 > 5 != 1
0 > 5 != 1
0 != 1 ⇒ 1

→ logical Operator : { && , || , ! }
And | or | Not.

&&

0	0	0
0	1	0
1	0	0
1	1	1

(a>b && a>c)

OR

0	0	0
0	1	1
1	0	1
1	1	1

[ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u']
(Vowel)

! → 0, 1

!10 = 0, !5 = 0

!0 = 1

→ Bitwise operator : { &, |, ^ (exor), ~ (complement), <<, >> }

2 & 3 = ②

2 = 1 0
3 = 1 1
2 → 1 0 = ②

2 | 3 = ③

1 0
1 1
1 1 → ③

Ex-or

0	0	0
0	1	1
1	0	1
1	1	0

2 ^ 3

1 0
0 1 → 1

{<<, >>} > {&, |, ^}

Preadens

<< (Left shift)

eg: 6 << 1 ⇒ 12

6 → 0 0 . . . 1 1 0

0 1 2 3 . . . 31

0 . . . 1 1 0 0 ← 0 padding

1100 = 12

Num << 2 ⇒ Num × 2^2

6 << 2

1 1 0

1 1 0 0 → 24

Num × 2^x → formula

>> (Right shift)

eg: 6 >> 1 ⇒ Num / 2^x

6 = 0 . . . 0 0 0 1 1 0

0 . . . 0 0 0 1 1 0

③

~ (complement)

~5 ⇒

5 = 0 . . . 0 0 0 1 0 1

(~5 = 1 . . . 1 1 1 0 1 0)

→ 2's complement ⇒ 0 . . . 0 0 0 1 1 0 → -6

Since ⇒ ~5 = -6, ~8 = -9, ~13 = -14, ~(-10) = 9

⇒ Assignment operator ⇒ {=; *=, /=, +=, -=, %=}

a = 3 / a *= 3

= a = a*3

a += 15

⇓

a = a+15

While loop :-

```
int i = 1
While (condition)
{
    // code
    increment / Decrement
}
```

→ Initialize ①
→ break ②
→ Updation ③

Do While loop :-

```
Do {
    // code
    increment /
    Decrement
} While (condition)
```

→ Initialize.
→ Updation.
→ break.

→ **Switch :-**

```
int i
Switch (i)
{
  case 1;
   break;
  case 2;
   break;
  default;
}
```

→ break :- → out from loop

→ Continue :- → skip the condⁿ

* Decimal to Binary ⇒

Dec.	Bin
0	0 0
1	0 1
2	1 0
3	1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

2	13	1
2	6	0
2	3	1
2	1	1
	0	1

⇒ 13 = 1101

⇒ a = 2 , b = 5 , c = 8 , d = 9

25 ⇒ 2×10 + 5 = a×10 + b

258 ⇒ 2×10² + 5×10 + 8 ⇒ a×10² + b×10 + c

2589 ⇒ 2×10³ + 5×10² + 8×10 + 9

a×10³ + b×10² + c×10 + d

⇒ 3 6 2 4 5

ans = 0×10 + 3 = 3

ans×10 + num (3)

3×10 + 6 = 36

ans×10 + num

36×10 + 2 ⇒ 362

362×10 + 4 ⇒ 3624

3624×10 + 5 ⇒ 36245

→ initialize = 0

→ ans = ans×10 + num

⇒ num

6 4 9 2 → 2 9 4 6

num ← 6×1 + 0

4×10 + 6 ⇒ 46 ⇒ ans

9×10² + 46 ⇒ 946.

2×10³ + 946 ⇒ 2946.

2×10³ + 9×10² + 4×10 + 6×10⁰

→ initialize ⇒ ans = 0

ans = num × 10^i + ans ⇒ Reverse order

⇒ num

2	13	1 ← rem
2	6	0
2	3	1
2	1	1
	0	1

⇒ 01101

→ Steps ⇒ ans = 0 (initialize)

ans = rem × 10^i + 1

Debugging:-

num = 13 6 3 1

rem = 13 % 2 = 1 ; 0 ; 1 ;

ans = 0 → 1

mul = 1

ans = rem × mul + ans

1 × 1 + 0 = 1

0 × 10 + 1 = 1

1 × 100 + 1 ⇒ 101

1 × 1000 + 101 ⇒ 1101

→ 1101

code:-

```
int num = 13
int rem, ans = 0, mul = 1;
while (num > 0)
{
    rem = num % 2;        → also use "rem = num & 1" use bitwise last digit
    num = num / 2;
//ans = rem × 10^i + ans;  (for each increment i++) {10^i can't use in computer}
    ans = rem × mul + ans;
    mul = mul × 10        → each time mul × 10 ⇒ 1×10, 10×10;
}
```

```
int num = ;
int rem, ans = 0, mul = 1;
while (num > 0)
{
    rem = num & 1;      // & → bit wise op
    num = num >> 1;     // >> left shift
    ans += rem * mul;
    mul *= 10;
}
```

Binary to Decimal.

$1\ 0\ 1 = 5$

$1\times2^2 + 0\times2^1 + 1\times2^0 = 5$

$1\ 1\ 0\ 1$

$2^3\ 2^2\ 2^1\ 2^0 \Rightarrow 13$

$1\ 4\ 3$

$1\times10^2\ \ 4\times10^1\ \ 3\times10^0$

$= 143$

$\Rightarrow$ if $143 \% 10 \rightarrow 3 \Rightarrow 143/10 = 14$

if $14 \% 10 \Rightarrow 4$

if $1 \% 10 =$

$\rightarrow 1\ 1\ 0\ 1$

Mod 10 (%)

Divide 10 (/)

1		110
0	% 10	11
1	% 10	1
1	% 10	1
0 → stop.	% 10	

Quotient

num

10	1101	1
10	110	0
4	11	1
	1	

$\Rightarrow$ 1101

$\Rightarrow 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$ans = rem \times 2^i + ans$

Debugging

rem = 1 0 1

mul = 1 2 4 8

num = 1 1 0 / 110 / 11 / 1 / 0

ans = rem × mul + ans.

$0 \times 1 + 0 = 1$

$0 \times 2 + 1 = 1$

$1 \times 4 + 1 = 5$

$1 \times 8 + 5 = 13$

Code:

```
int num = ;
int rem, ans = 0, mul = 1;
while (num > 0)
{
    // rem
    rem = num % 10;
    // ans
    ans = rem * mul + ans;
    // Quotient
    num /= 10;
    // mul
    mul *= 2;
}
cout << ans;
```

Leetcode Problem

→ **Add digits**

68 → 6+8 = 14 ⇒

10	68	8
0	8	
	0	6+8 = 14

⇒ ans = ans + rem

- Single digit

249 ⇒ 17 ⇒ 8

```
int num;
int rem, ans = 0;
while (num != 0)
{
    rem = num % 10;
    num /= 10
    ans = ans + rem
}
cout << ans;
```

A

→ for single digit ⇒

```
while(num > 9)
{
    A
    num = ans;
}
cout<<num;
```

→ **Leap year**

① 400 se divide → L.Y.

② 4 se divide And 100 se nhi → L.Y.

```
int N;
if (N % 400 == 0)
    cout << "LY";
else if (N % 4 == 0 && N % 100 != 0)
    cout << "LY";
else
    cout << "Not L.Y.";
```

→ **Reverse Integer:**

234 → 432

4 ⇒ 4 + 0 × 2

3 ⇒ 43 × 10

2 ⇒ 43 + 2 × 100

432

→ ans = ans × 10 + rem

```
int x;
int ans = 0, rem;
while (x)
{
    rem = x % 10;
    x /= 10;
    ans = ans * 10 + rem;
}
```

INT_MAX.
INT_MIN

→ for overflow

ans × 10 + rem > INT_Max

ans > (INT_Max − rem) / 10

if(ans > INT_Max / 10)

↳ return 0;

as rem (0 → 9) ⇒ rem / 10 = 0 (Integer)

Also

if(ans < INT_Min / 10)

return 0;

→ Power of 2

1 - 1_{2^0}
2 - 10_{2^1}
4 - 100_{2^2}
8 - 1000_{2^3}
11 - 10000_{2^4}
32 - 100000_{2^5} ⇒ 26

2	52	
2	26	0
2	13	0
2	6	1
2	3	0
2	1	1
	0	1

⇒ 110000
2^5

{ last me 1 mile to wo 2 Ki Power me exist karta hai }

```
x
While(x != 1)
{ if (x % 2 == 1)
    return 0;
  x /= 2;
}
//cout << 1;
return 1;
}
```

→ Sqrt x:

123

1 = 1
2 = 4
3 = 9
4 = 16
5 = 25
6 = 36
7 = 49
8 = 64
9 = 81
10 = 100 → return
11 = 121 → Greater then 123

→ Pallindrom: (mom, naman).

121 → 121
-121 ⇒ 121 - ✗

```
int num = x;
int rem; ans = 0;
while( num)
{
  rem = num % 10;
  num /= 10
  ans = ans * 10 + rem;
} if (ans == x)
  cout << "Pal";
```

12	121	
10	12	1
10	1	2
	0	1

1 2 1

ans = ans × 10 + rem
0 × 10 + 1 = 1
1 × 10 + 2 = 12
12 × 10 + 1 ⇒ 121

→ Complement of a no:

5 ⇒ 1 0 1
5 comp = 0 1 0 = 2

2	27	
2	13	1 → 2^0 ⇒ 0
2	6	1 → 2^1 ⇒ 0
2	3	0 → 2^2 ⇒ 2^2
2	1	1 → 2^3 ⇒ 0
	0	1 → 2^4 = 0

4

⇒ ans = ans + rem × 2^i

```
While (n)
{
  rem = n % 2;
  rem = rem ^ 1;
  n = n / 2;
  ans = ans + rem * mul;
  mul = mul * 2
}
```

^ (exor)

0 ^ 1 → 1
1 ^ 1 → 0

↳ Convert 0 to 1 or 1 to 0

Function

→ Reusability
→ Readability

Syntax :-

```
return_type fx^n name (Parameter 1, Parameter 2 ....)
{
    // code
    return value;
}
```

return type
→ int,
char,
double,
void,

→ Prime no.

```
bool Prime(int n)
{
  if(n < 2)
    return 0;
  for(i=2; i<n; i++)
  {
    if(n % i == 0)
  }     return 0;
  return 1; → bool se 1 if true.
}
```

→ Factorial

```
int fact (int n)          ---- fx^n declare.
{
    int ans = 1;
    for(i=1; i<=n; i++)
    {
      ans = ans * i;      ( fx^n define
    }
    return ans;
}
```

→ Main fx^n

```
int main()
{
   int a, b;
cout << Prime(a);   → argument.
<< fact(b);          fx^n call
<< Prime(a-b);
```

→ if :-

```
int Fact(int n=3)
// Default Parameter
```

↳ it pass 3 in fact if we don't pass any argument in main function.

→ Pass by value :-

```
void Incr(int n)
{
  n++;
}
int main()
{ int a=10;
  Incr(a);
  cout << a;
}
```

a → 4
n → 4 5 (n++)

→ does not increase value of 'a' it increase value of 'n'.
→ n में a की copy जाती है

→ Pass by Refrence :-

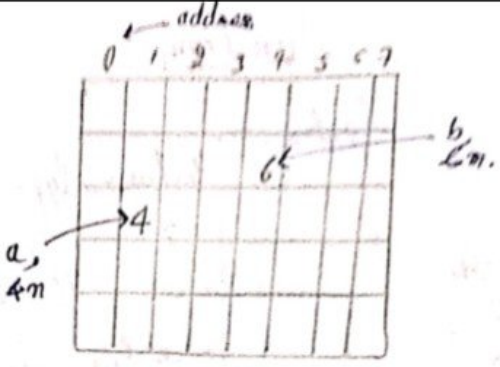
```
Void Incr(int &n)
{
    n++;
}
int main()
{
  int a = 10;
  Incr(a);
  cout << a;
}
```

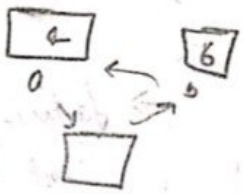
a 4 ← n

→ 'n' one 'a' जैसा होता है no कि 'a' की copy.
→ n store in form of address of 'a'.

→ Swapping two numbers:

```
void Swap(int &n, int &m)
{
    int c;
    c = a;
    a = b;
} b = c;
int main()
{
    int a, b;
    cin >> a >> b;
    Swap(a, b);
cout << a << b;
}
```





→ Swap fn is also a inbuild fn in C++:

swap ⟵ ~~Swap~~

→ Convert 'a' to 'A'. {a, b, c ---- z} → {A, B, C ---- Z}.

```
char convert(char name)
{
  char ans = name - 'a' + 'A';
  return ans;
}

cout << convert(name);
```

Debugging:

name - 'a' + 'A'

↓ 97 ↓ 65

'h' - 'a' + 'A'

↓ 107 - 97 + 'A' →

7 + 'A' ⇒ 'H'

→ Armstrong No.:

2 3 ⇒ 2 digit

→ $2^2 + 3^2 \Rightarrow 13$ ✗

1 5 3 (3 digit) / $1^3 + 5^3 + 3^3 \Rightarrow 153$ ⟵ → Armstrong no.

```
int CountDigit(int n)
{
  int count = 0;
  while(n)
  {
    count++;
    n /= 10;
  }
  return count;
}
```

↳ Count digit in the number

```
bool Armstrong(int num, int digit)
{
  int n = num, ans = 0, rem;
  while(n)
  {
    rem = n % 10;
    n /= 10;
    ans = ans + Pow(rem, digit);
  }
  if (ans == num)
    return 1;
  else
    return 0;
}
```

Pow(rem, digit) → Inbuild

(rem)^digit → 3^3

```
int main()
{
  int num;
  cin >> num;
  int digit = CountDigit(num);
cout << Armstrong(num,
                  digit);
}
```

→ Find Trailing zero in a factorial :-

6! → 1 × 2 × 3 × 4 × 5 × 6

1 2 3 2^2 5^1 2×3

⇒ $2^4 \times 3^2 \times 5$

→ $10 = 5 \times 2$ → how many "5 × 2" will help find the total 0's

∴ 6! ⇒ 720 → 1 zero.

8! ⇒ 40320 → 1 zero.

10! → 3628800 ⇒ 2 zero.

1 × 2 × 3 × 4 × 5 × 6 × 7 × 8

$2^4 \times 3^1 \times 5^1 \times 7$

⇒ $2^x 3^y 5^z$

always $x > z$ → To find z

1 × 2 × 3 × 4 × 5 × ... × 10 × ... × 15 × ... 20 ... 25 ... 30 ... 35 ... 50.

5¹ 5 × 2 5 × 3 × 2 5×2^2 5 × 5 (2 fives) 5 × 2 × 3 35 50 5 × 5 × 2

→ logic ⇒ !n ⇒ ans = n/5 for n < 5

if n > 5

!n ⇒

- ans = n/5 = 5
- ans = 5/5 = 1

ans = 5 + 1 = 6

eg: ⇒ 100!

⇒ 100/5 ⇒ 20;

20/5 = 4 ⇒ 20 + 4 ⇒ 24 zeros

eg ⇒ 148!

148/5 = 29

29/5 = 5

5/5 = 1

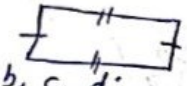
⇒ 29 + 5 + 1

⇒ 35 zeros

Code :-

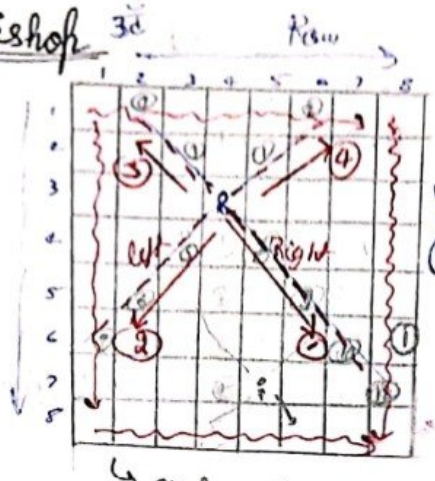
```
int num = ;
int ans = 0;
while (num >= 5)
{
    ans = ans + num / 5;
    num /= 5;
}
cout << ans;
}
```

→ Rectangle



```
int a, b, c, d;
if ((a == b && c == d) || (a == c && b == d) || (a == d && b == c))
    return 1;
else
    return 0;
```

→ Bishop



(Digonal चलता है)

↳ ye hamesha corner me yata last 'row' me. hoga ya last 'column' me.

Row | col → Total out put = 11
Present → A B → to go
3 4

when it go RIGHT side

① → Total : row = col
8 8

Present | 3 4
(5, 4) → Min = 4

① → min(8-A, 8-B). ↳ goes Right down.

when it go left side

② → Row col
8 1

min(8-A, B-1). ↳ goes left down.

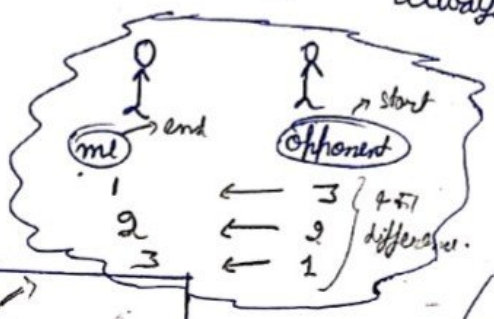
③ → min(A-1, B-1) ↳ goes Upper left.

④ → min(A-1, 8-B) ↳ goes Upper Right.

```
{
int count = 0;
Count += min(8-A, 8-B);
count += min(8-A, B-1);
count += min(A-1; B-1);
count += min(A-1, 8-B);
return count;
}
```

→ Nim Game:

1 → 20 → (जो भी last me 20 bolega wo jeetega)
always go with 4's factor.



N = x ↳ first make it in form of 4's factor

N = 23 → 20 (1, 2, 3).

N = 8 → me haranya agar me gone start karunga.

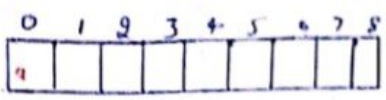
N = 21 → 20 (1) → 4 ke factor me bano or 4 ke factor me Fasao

N = 17 → 16 (1)

N = 15 → 14 (1, 2, 3).

```
int N
if (N%4 == 0)
    no. 4 ka factor
    return 0; me haranga.
else
    return 1; me jeetunga
```

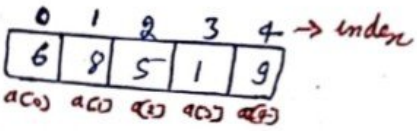
Arrays



→ Array store same type of data type at contigous location. (1000, 100)
address of arr[] — size of array

int a[1000];
data type. name — size of array

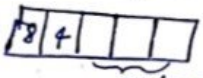
→ ① int a[5] = {6, 8, 5, 1, 9}



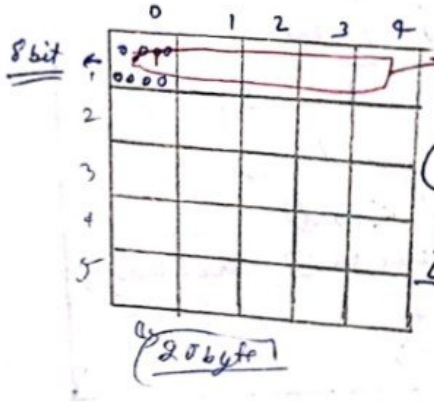
② int a[] = {6, 7, 9, 12};

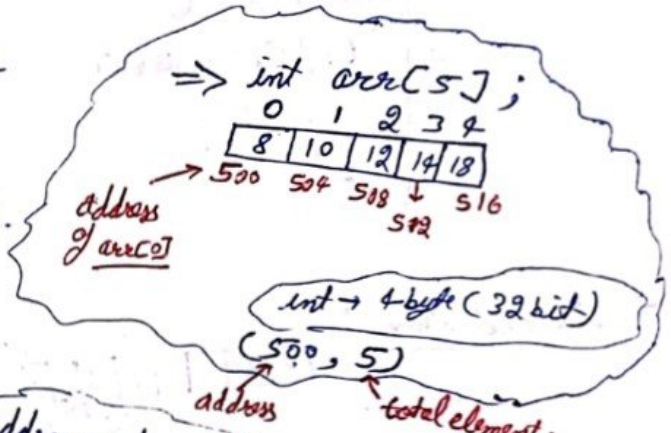
③ int arr[10]; => loop (cin >> arr[i]) // user input.

④ int arr[5] = {8, 4}

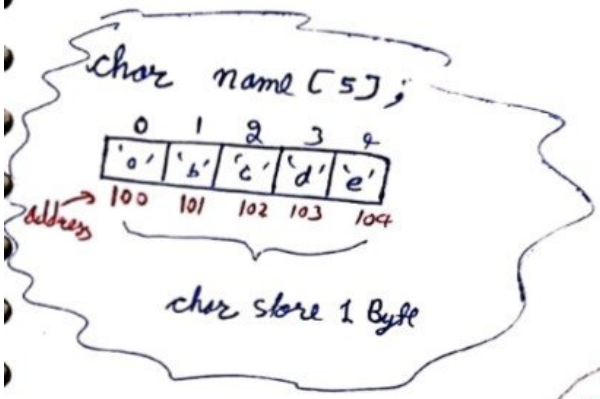


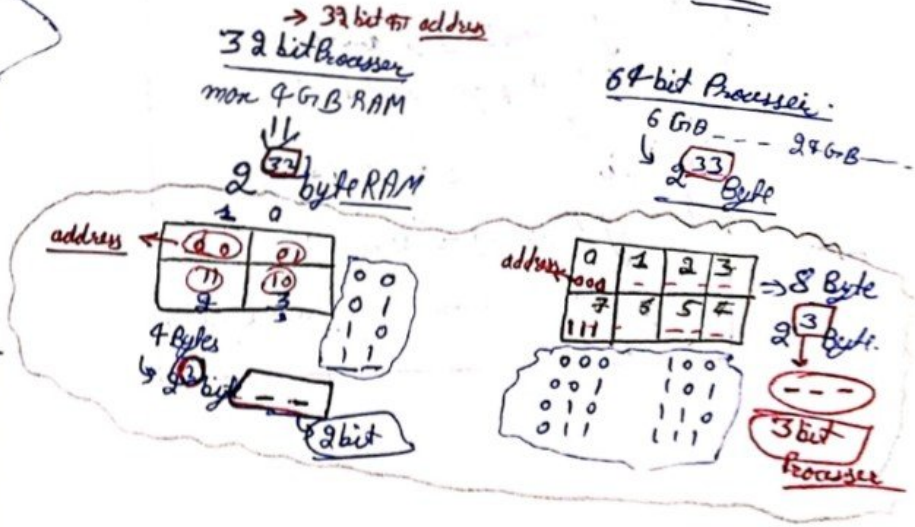
⑤ int arr[5] = {0}; // initialise Zero: {0 0 0 0 0} => only true for '0'.





→ address of index :=>
arr + index × size of data type.
for 2: index 500 + 2 × 4 => 508

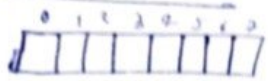




=> find min in arr:

```
int arr[5] = {...};
int ans = arr[0];
for (int i=0; i<5; i++)
{
    if(arr[i] < ans){
}       ans = arr[i];
cout << ans;
```

→ Search Element :

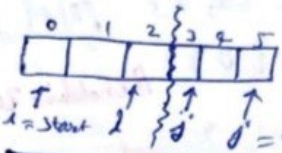


```
int arr[6] = { ... };
int x = ?;
int index = 0;
for (i = 0; i <= 5; i++)
{
    if (arr[i] == x)
    {
        index = i;
    } break;
    else {
        } return -1;
}
cout << index;
```

→ Reverse Array :

```
int arr[6], temp[6]
int i = 5, j = 0
while (i >= 0)
{
    temp[j] = arr[i];
    j++;
    } i--;
```

&




```
int arr[6] = {      };
int start = 0, end = 5;
while (start < end)
{
    swap (arr[start], arr[end]);
    start++;
} end--;
```

→ Second Max :


```
int arr[6] = {...};
int ans = 0 ~~arr[0]~~;
for (i=0; i <= 5; i++)
{
    if (arr[i] > ans ~~arr[0]~~)
    {
        ans = arr[i];
    }
}
for (i=0; i < ~~6~~ ; i++)
{
    if (ans != arr[i])
        ans = max (ans, arr[i]);
}
```

→ Missing No :-

as N = 6

then size of arr = N-1

$1 \le arr[i] \le N$ → | 1 | 2 | - | - | N |

1	3	4	5	6	is expected	1	2	3	4	5	6

sum = 19 sum = 21

⇒ sum arr[B] − sum arr[A]

21 − 19 ⇒ 2 ⇒ element missing

→ Fibonacci Series :-

0 1 1 2 3 5 8 13 - -

1st 2nd 3rd 4 5 6

0	1	1	2	3	5	8	-
0	1	2	3	...			

→ arr i = arr[i-1] + arr[i-2];

```
int n;
int arr[1000];
arr[0] = 0;
arr[1] = 1;
for (int i=2; i<n; i++)
  { arr[i] = arr[i-1]+arr[i-2];
  }
cout << arr[n-1];
```

→ Rotate array by 1

2	7	11	4	5	⟶	5	2	7	11	4

```
int arr[n] = { - - - - - - - }
int temp = arr[n-1].
for(i = n-2; i >=0; i--)
  { arr[i+1] = arr[i];
  }
  arr[0] = temp;
  Print array
```

→ Array as function :-

Not a array ⇒ it is address

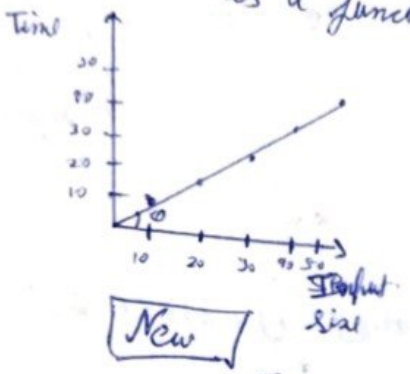
Pointer

```
void fun (int arr[], int n)
{ for (i=0; i<5; i++)
    cout << arr[i];
}
int main()
{ int arr[5] = {1, 2, 3, 5, 7}
    fun (arr, 5);
}
```

Time & Space Complexity

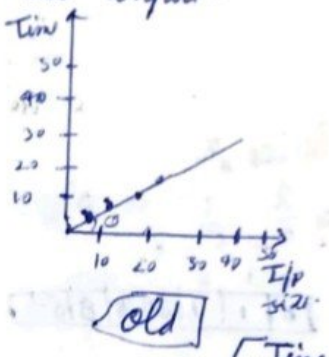
Time Complexity != Time taken

↳ It is the total time taken by an algorithm to run as a function of length of the input.



New

(Time = n)



Old

(Time = $\frac{n}{2}$) ≈ n

Big (O) {worst case}

$O(n)$

Ω {Best case}

$\Omega(n)$

Θ {Avg case}

→

```
for(i=1 ; i<=n ; i++)   // 1 → ①, i=1 → ②, i<=n → ③, i++ → ④
{
    cout << "chamka";
}
```

program 1st time ⇒ 4 steps.
2nd time ⇒ 3 steps

$\underset{1st}{4} + \underset{2nd}{3} + \underset{3rd}{3} + \dots + \underset{n^{th}}{3} \Rightarrow 3n+1$

→ Ignore constant ⇒ $n = O(n)$

→ If multiple dependency

$2N^3 + 3N^2 + N$ ⇒ take max ⇒ $N^3 = O(n)$

→

```
cin >> n
for(i=0 ; i<10 ; i++)
{
    cout << "chamka";
}
```

for n = 10 } output 10 times
for n = 100 } 10 times

doesn't depend on 'n'.

$O_{10} = O_{(1)} = \Omega_{(1)} = \Theta_{(1)}$

↓ constant.

order 1

→ [cout << n(n+1)/2;] ⇒ Order ⇒ $O(1)$

→

```
for(i=1 ; i<=n ; i++)      → n times
    for(j=1 ; j<=n ; j++)  → n times
    {
        cout << "chamka";
    }
```

Order = $O(n^2)$.

⇒ n^2

Debugging

$i=1$ } run n times
$i=2$ } $j=n$ times
$i=3$ } $j=n$ times
⇒ $n+n+n+\dots$ ⇒ $n \cdot n = n^2$

→
```
for(i=1 ; i ≤ n ; i++)
    for(j=1 ; j ≤ i ; j++)
        cout << "chamka";
```

i=1 → j→1 (1 time)
i=2 → j→1 to 2 (2 times)
i=3 → j→1 to 3 (3 times)
i=n → j=1 to n (n times)

→ 1+2+3+1 ---- n = $\frac{n(n+1)}{2} = \frac{n^2+n}{2}$

→
```
for(i=1 ; i^2 ≤ n ; i++)
    for(j=1 ; j ≤ i^2 ; j++)
        cout << "chamka";
```

i=1 → j→1 time
i=2 → j→4 times
i=3 → j=9 times
--- i=n → j=n² times

1+4+9+ --- n^2 ⇒ $\frac{n(n+1)(2n+1)}{6}$ ⇒ $O(n^3)$

→ Time comp → $O(n^2)$

→
```
for(i=1 ; i ≤ n ; i++)        → n times
    for(i=1 ; i ≤ m ; i++)    → m times
        cout << "chamka";
```

⇒ $O(nm)$.

→
```
for(i=1 ; i ≤ n ; i++)
    for(j=1 ; j ≤ i^2 ; j++)
        for(k=1 ; k = n/2 ; k++)
            cout << "chamka";
```

Debugging
i=1 → j=1 to 1 → k=1 to n/2 → $\frac{n}{2}$ times Print
i=2 → j=1 to 4 → k=1 to n/2 → $4\frac{n}{2}$ times Print
i=3 → j=1 to 9 → k=1 to n/2 → $9\frac{n}{2}$ Print
i=n → j=1 to n^2 → k=1 to n/2 ⇒ $(n^2 \times \frac{n}{2})$ times Print

→ $\frac{n}{2} + 4\frac{n}{2} + 9\frac{n}{2} + \cdots + n^2\left(\frac{n}{2}\right) \Rightarrow \frac{n}{2}(1+2^2+3^2+\cdots+n^2)$

$\frac{n}{2}\left[\frac{n(n+1)(2n+1)}{6}\right]$ → $O(n^3)$.

→
```
for(i=1 ; i ≤ n ; i=i*2)
    cout << "chamka";
```

i=1 → 1
i=2 → 1
i=2² → 1
i=2^k(n) → 1

Print (k+1) times ⇒ $\log_2 n + 1$

$n = 2^k$
$\log n = k \log 2$
$k = \frac{\log n}{\log 2} \Rightarrow \log_2 n$

{1 — 2 — 4 — 8 — 16 — 32 — --- n} ⇒ $\log_2 n$
{1 — 3 — 9 — 27 — --- n} ⇒ $\log_3 n$

→
```
for(i=n/2 ; i ≤ n ; i++)        → n/2 times
    for(j=1 ; j ≤ n/2 ; j++)    → n/2 times
        for(k=1 ; k ≤ n/2 ; k++) → n/2 times
            cout << "chamka";
```

⇒ $\frac{n}{2} \times \frac{n}{2} \times \frac{n}{2} \rightarrow \frac{n^3}{8}$

→
```
for(i=n/2 ; i ≤ n ; i++)        → n times
    for(j=1 ; j ≤ n ; j=2*j)    → log2 n times
        for(k=1 ; k ≤ n ; k=2*k) → log2 n times
            cout << " n ";
```

Time comp → $O(n^3)$.

⇒ $n.(\log_2 n)^2$

if n + log n
ignore log n

→
```
for(i=1; i<=n; i++)
  for(j=1; j<=n; j=j+i)
  {
    cout<<"*";
  }
```

$i=1$, $j=1 \to n$ (n times Bad) | $i=2$, $j=\frac{n}{2}$ jumps ($\frac{n}{2}$ times) | $i=3$, $j=\left(\frac{n}{3}\right.$ jumps ($\frac{n}{3}$ times Bad) | ... $i=n$, $j=\frac{n}{n}=1$ (1 time Good)

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

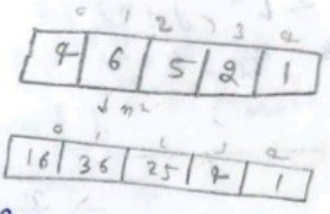
$$\Rightarrow n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$$

(HP)

$$n \log_e n \Rightarrow O(n \log_e n).$$

* **Space Complexity :-** it is the amount of space taken by an algorithm as a fn of length of input.

- auxillary space.
- Total space complex.



only → Aux $\Rightarrow n \to O(n)$.

both ⇒ Total $\Rightarrow n+n \Rightarrow 2n \to O(n)$.

→
```
for(i=1; i<n; i++)
{
  cout<<i
}
```

spaces → $i \to 1$

$n \Rightarrow 1$ Auxilary = 1 $\Rightarrow O(1)$.

Total = 1+1 $\Rightarrow O(1)$.

→
```
int a, b, c;
cin>>a>>b>>c;
int d = (a+b)/2;
int e = (a+c)/2;
```

$\to 1+1 \Rightarrow$ Aux

Auxilary = 1+1 = 1 $O(1)$

Total = 1+1+1+1+1 = 5 $O(1)$.

⇒ $O(n!)$ → worst

$O(2^n)$

$O(n^3)$

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\sqrt{n})$

$O(\log n)$

$O(1)$ → best

→ $n! = 2^n$

1	1	2
2	2	4
3	6	8
4	24	16
5	120	25
6	720	36
	↓	↓
	big	small
	=	big
	worst	

Sorting

3	8	1	2	5

ascending → | 1 | 2 | 3 | 5 | 8 |

descending → | 8 | 5 | 3 | 2 | 1 |

0	1	2	3	4
9	7	3	1	6

Round 1: 1 7 3 9 6 (Round 1)

R2 → 1 3 7 9 6

R3 → 1 3 6 9 7

R4 → 1 3 6 7 9

→ n^{th} element

→ no of Round = $(n-1)$.

→ Selection Sort

→ Smallest no. (index) in array:

```
int index = 0;
for(i=1; i<=4; i++)
{
    if(arr[i] < arr[0])
    {
        index = i;   → store index
    }
}
```

```
for(i=0; i<(n-1); i++)
{
    int index = i;
    for(j=i+1; j<n; j++)
    {
        if(arr[j] < arr[index])
        {
            index = j;
        }
    }
    swap(arr[index], arr[i])
}
```

Time complexity.

i=0	i=1	...	i=n-2
j=1 to n	j=2 to n		j=(n-1) to n
(n-1) times	(n-2) times		1 time.

$(n-1) + (n-2) + (n-3) + \dots + 1$

$\rightarrow \frac{n(n-1)}{2} \Rightarrow O(n^2)$.

Space complexity ←

Auxillary : $O(1)$

Total space : $1+n \Rightarrow O(n)$

→ Descending sort :- → `if(arr[j] > arr[index]) index = j;`

→ Bubble Sort

7	4	8	5	3

4 7 5 3 | 8
4 3 | 5 7 8
3 | 4 5 7 8

7	4	8	5	3

Round 1: 4 7 8 5 3
4 7 8 5 3
4 7 5 8 3

4	7	5	3	8

Round 2: 4 5 7 3 8
4 5 3 | 7 8
Round 3: 4 5 3 7 8
4 3 | 5 7 8
Round 4:

3	4	5	7	8

n
Round = n-1

Ascending → Descending

5	4	3	2	1

Round 1: 4 5 3 2 1
4 3 5 2 1
4 3 2 5 1
4 3 2 1 | 5

n = 5
Round = n-1
(0 - 3) times loop
R1

Round 2: 4 3 2 1 5
3 4 2 1 5
3 2 4 1 5
3 2 1 | 4 5

(0 - 2)
R2

Round 3: 3 2 1 4 5
2 3 1 4 5
2 1 | 3 4 5

(0 - 1)
R3

Round 4 → 2 1 3 4 5
1 2 3 4 5

(0 - 0)
R4

→ 0 - 3
0 - 2
0 - 1
0 - 0

i

Code →

```
for (i = n-2; i >= 0; i--)
{
    for (j = 0; j <= i; j++)
    {
        if (arr[j] > arr[j+1])
            swap(arr[j], arr[j+1])
    }
}
```

→ Time complexity

i = (n-2)	i = n-3	...	i = 0
j = 0 to (n-2)	j = 0 to n-3	...	j = 0 to 0
(n-1) times	(n-2) times	...	1 time

(n-1) + (n-2) + ... + 1 → n(n-1)/2

O(n²)

→ Space comp → O(1).

→ Best case already sorted →
1 2 3 4 5
→ no need to shift

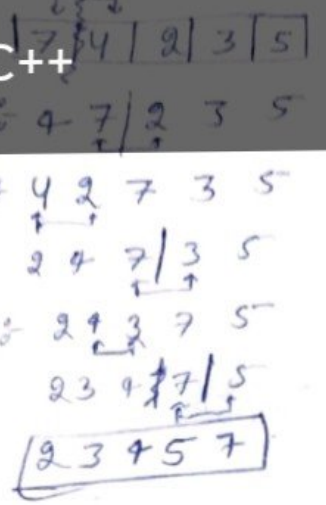
O(n), Ω(n)

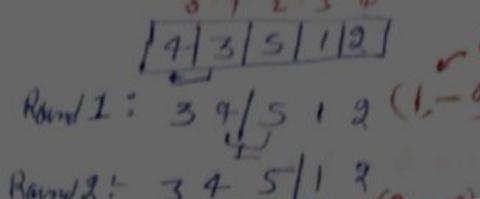
→ worst case → O(n²).
→ best case → Ω(n).
→ avg case = O(n²).

```
for (i = n-2; i >= 0; i--)
{
    bool swapped = 0;
    for (j = 0; j <= i; j++)
    {
        if (arr[j] > arr[j+1])
            swap(arr[j], arr[j+1])
            swapped = 1;
    }
    if (swapped == 0)
        break;
}
```

→ Jab swap = 0 (No need for further check)

Insertion Sort





Round 3:- 3 4 1 5 2 (3-0)

3 1 4 5 2

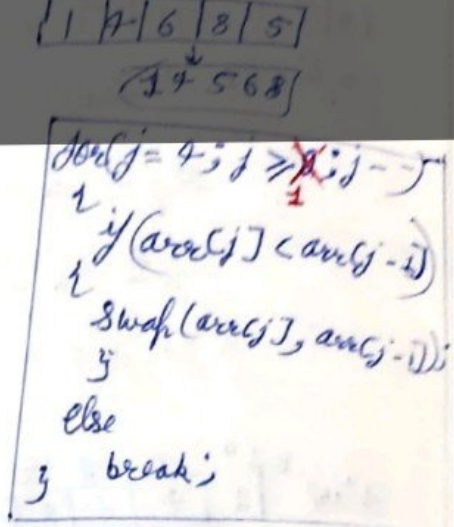
1 3 4 5 2

Round 4:- 1 3 4 2 5 (4-0)

1 3 2 4 5

1 2 3 4 5

No of Round → n - 1




```
for (i=1; i<n; i++)
{
    for(j=i; j>0; j--)
    {
        if(arr[j] < arr[j-1])
            swap(arr[j], arr[j-1]);
        else
            break;
    }
}
```

→ Space complexity :- O(1).
our

→ Time complexity :-

i=1	i=2	...	i=n-1
j=1→1	j=2 to 1	...	j=(n-1) to 1
1 time	2 time	...	(n-1) time

1 + 2 + 3 + ... (n-1) → $\frac{n(n+1)}{2}$

⇒ $O(n^2)$

→ Best case. already sorted [1 2 3 4 5]

⇒ 1 + 1 + 1 + 1 - - - 1 → (n-1) comp → $\Omega(n) = \Omega(n)$.

→ Avg ⇒ $\frac{\text{sum of all cases}}{\text{total cases}}$ our → $\Theta(n^2)$

→ Decreasing Order ⇒ [if(arr[j] > arr[j-1])
swap(arr[j], arr[j-1]);]

Binary Search.

0	1	2	3	4	5	6	7	8
3	7	11	12	17	18	23	27	29

→ find 27 ⇒ cond ⇒ "array should be sorted."

→ went to middle part.

$\frac{0+8}{2} = 4 \Rightarrow a[4] = 12 \neq 27$.

→ $\frac{5+8}{2} \Rightarrow 6 \Rightarrow a[6] = 23 \neq 27$.

→ $\frac{6+8}{2} \Rightarrow 7 \Rightarrow a[7] = 27$.

3 steps

"for linear search need 8 steps"

start

0	1	2	3	4
2	4	6	8	10

end

start = 0 , end = 4

① Mid = $\frac{start + end}{2}$

② if (arr[mid] == Key)
cout << "Got it"
break;

else if (arr[mid] < Key)
Right side. → [start = mid + 1;].

else
left side. → [end = mid - 1;].

0	1	2	3	4
4	6	10	12	17

find 11.

→ start = 0, 3 end = 4

mid = $\frac{0+4}{2}$ = 2 = 10 < 11 Right

mid = $\frac{3+4}{2}$ → 3 ⇒ 12 > 11 left

⇒ loop doesn't end cuz 11 does not exist ⇒

→ while (start <= end)
{
// code
}

Code :-

```
start = 0;
end = n - 1;
While( start <= end)
{
  mid = (start + end) / 2
  if (arr[mid] == Key)
  {
     cout << "Got it";
  }  break;
  else if (arr[mid] < Key)
     start = mid + 1;
  else // (arr[mid] > Key).
     end = mid - 1;
}
```

Time complexity

0	1	2	3	4	5

→ size of array

① N = 1 step. (0+1) ...

② N/2 = 1 step (2+1) ...

③ N/4 = 1 step. (2+1) ...

⋮

⑤ $N/2^K$ = 1 step. (K+1) ...

$\frac{N}{2^K} = 1$ → 1 array left at last (worst case)

$N = 2^K$

$\log N = K \log 2$

$K = \log_2 N$

⇒ $O(\log_2 N + \text{constant})$

⇒ $O(\log_2 N)$

⇒ Best case ⇒ $O(1) = \Omega(1)$.

⇒ Avg case ⇒ $O(\log_2 N)$.

↳ Overflow integer ⇒ mid = $\frac{start + end}{2}$ $(2^{30}+1)$ $(2^{31}-1)$ ⟶ write as (mid = start + $\frac{end - start}{2}$) (End index)

→ First and last position of element:

- Sorted array.

0	1	2	3	4	5
5	7	7	8	8	10

Target = 8

if target does not exist (-1, -1)

↳ Time complexity = O(n)

first occur = 3.
last occur = 4.

```
start = 0, end = n-1;
first = -1, last = -1;
// first occurence
while (start <= end)
{
    mid = start + (end - start)/2;
    if (arr[mid] == target)
    {
        first = mid;
        end = mid - 1;
    }
    else if (arr[mid] < target)
        start = mid + 1;
    else
        end = mid - 1;
}
```

T.C. = log(n)

```
start = 0, end = n-1;
// last occurence.
while (start <= end)
{
    mid = start + (end - start)/2;
    if (arr[mid] == target)
    {
        last = mid;
        start = mid + 1;
    }
    else if (arr[mid] < target)
        start = mid + 1;
    else
        end = mid - 1;
}
```

T.C. = log(n)

→ Time complexity → O(log n)

→

0	1	2	3	4	5	6	7
1	4	6	8	10	14	16	18

⇒ Strictly increasing order.

⇒ if target = 4 ⇒ return → 1 (index)

if target = 5 ⇒ return = 2

Code:

```
index = n, mid;
while (start <= end)
{
    mid = start + (end - start)/2;
    if (arr[mid] == target)
    {
        index = mid;
        break;
    }
    else if (arr[mid] < target)
        start = mid + 1;
    else
    {
        index = mid;
        end = mid - 1;
    }
}
```

1	4	6	8	10	14	16	18
0	1	2	3	4	5	6	7

find 5

mid ≠ 5 → index = 3 (goes left)

↳ (at least 3 ≥ 2)

1	4	6
0	1	2

2+0/2 → 1 (4 ≠ 5) → goes Right

↳ index update nhi hoga

2
6

→ 6 ≠ 5 → index = 2

⇒ Why index = n → (if find 20 in array)

Jab "only else if" wali cond" chalo Jab Tab index pr kuch bhi store nhi hoga. Tab 'index = n.' repeat element not exist in array, even greater than last index.

→ Sqrt(x)

```
start = 1 ; end = x , ans
while (start <= end)
{
   mid = start + (end - start)/2;
   if(mid * mid == x)
   {
      ans = mid;
      break;
   }
   else if (mid * mid < x)
   {
      ans = mid; // Potential ans
      start = mid + 1;
   }
   else // (mid * mid > x)
      end = mid - 1;
}
```

overflow (mid == x/mid)

(mid <= x/mid)

→ T.C. ⇒ log n

O(log n)

Sqrt(80)

ans = 5 ⇒ Potential ans

6 7 8 9 → ans = 7

(6+9)/2 → 7 ⇒ 49 < 80

7 8 9 → ans = 8 ✓

(7+9)/2 ⇒ 8 ⇒ 64 < 80

start, end → 8 9

(8+9)/2 ⇒ 9 ⇒ 81 > 80

⇒ ans = 8

→

2	3	4	7	11	12

K^th Positive missing integer.

1 2 3 4 5 6 7 8 9 10 ||

1st 2nd 3rd 4th 5th 6th 5th missing

0	1	2	3	4	5
2	3	4	7	11	12

arr[i] - i - 1

1/1 2/1 3/1 4/3 5/5 6/6

7+1 ⇒ 4th

7+2 ⇒ 5 5th

7+3 ⇒ 6th

mid

0	1	2	3	4	5
1	1	1	3	6	6

K = 5

ans = 4

3	4	5
3	6	6

< 5

3

3 < 5

```
start = 0, end = n-1 , ans = ans + K
while (start <= end)
{
   mid = start + (end - start)/2;
   if(arr[mid] - mid - 1 >= K)
   {
      ans = mid;
      end = mid - 1;
   }
   else
      start = mid + 1;
}
return ans + K;
```

n

ans store index of element.

obtain index

ans + K ← Given

Proof :-

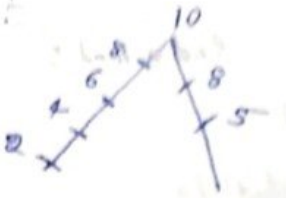
K^th missing No.

arr[index] - (arr[index] - index - 1 - K + 1)

⇒ index + K

→ Peak Index in Mountain Array :

2	4	6	8	10	8	5



Web 1

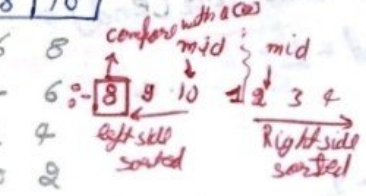
```
if (arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1])
        return mid;
else if ( arr[mid] > arr[mid-1])
        start = mid + 1;
else    // (arr[mid] >= arr[mid+1]
        end = mid - 1;
```

→ mid → (end + (start - end)/2)

→ Rotated Array :- → find min element.

8	4	6	8	10

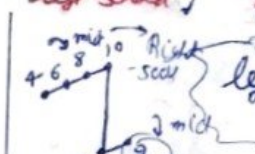
1 shift → 10 2 4 6 8
2 shift → 8 10 2 4 6
6 8 10 2 4
4 6 8 10 2



4	6	8	10	1	2

(Ek side sorted होगा)

start a[0]
Inc
Inc
Left sorted
Right sorted.
{ arr[0] < arr[mid] left sorted



left side sorted hai
Right side. (सीधा element dhundne ke liye)
जाना
left side.
Right side sorted hai
left side (जाना है)
Potential ans
ans = 2

if array rotate nhi hua ho
तब ye arr[0] return
karega

arr[0] → already sorted array

```
ans = a[0], start = 0, end = n-1;
while (start <= end)
{  mid = start + (end - start)/2;
   if (arr[mid] >= arr[0])        // left side sorted
      start = mid + 1;
   else // (arr[mid] < arr[0])    // Right side sorted.
      ans = arr[mid];
      end = mid - 1;
}
```

→ Search in Rotated array :-

4	5	6	1	2	3

0 1 2 3 4 5
mid

Target = 4

mid = a[2] = 0 < 4 → Right side sorted

0 <= 4 <= 5 → arr[end]

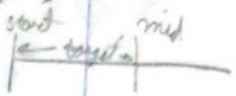
arr[mid] ↓ mid

4	5

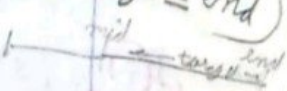
4 == 4

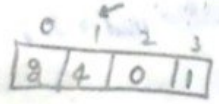
```
ans = -1 , start = 0 ; end = n-1;
while ( start <= end)
{
   mid = start + (end-start) /2 ;
   if( arr[mid] == target)
         return mid;
   else if (arr[mid] >= arr[0])  // left side sorted.
   {
         if( arr[start] <= target && arr[mid] >= target)
                end = mid -1;
         else
   }           start = mid +1;
   else  {  // (arr[mid] <= arr[0])   // Right side sorted.
         if (arr[mid] <= target && arr[end] >= target)
                start = mid +1;
         else
   }           end = mid -1
}
```

start ≤ target ≤ mid



mid ≤ target ≤ end





ans = -1

target = 1

start = ~~0~~ ~~2~~ 3

end = 3

$0 \le 1 \le 1$

$2 \le \not 1 \le 9$

→ 1

→ Top Questions

→ Book Allocation :

1. Every student get atleast 1 Book.
2. Book will be allocated in contiguous way.
3. Out of all permutation student with most number of books get min pages.

12	34	67	90

M = 2 → 2 student.

I st	II nd	
12 → 12	34+67+90 → 119	→ 119
12+34 → 46	67+90 → 157	→ 157
12+34+67 → 113	90 → 90	→ 113 min

(difference min है)

| 12 | 34 | 67 | 90 | → end = sum → M=1

start = Max of arr.

I: (12+34+67) II nd: (90)

90 . 91 . 92 - - 108 [113] . - - - 150 . - - - 203

90 — 108 — 117 — 146 — 203

start = ~~90~~ ~~109~~ 113

end = ~~203~~ ~~145~~ ~~116~~ 113

ans = ~~146~~ 117

12, 34, 67, 90

I: 12, 34, 67 | II: 90

109 — 112 — 116 ⇒ 113 — 114 — 116 ⇒ 113 113

12	34	67	90

Page = 0 count = 1

108 → Page = ~~34~~ ~~67~~ 90

Count = ~~1~~ ~~2~~ 3

```
start = 90 , end = 203 , mid, ans
while ( start <= end)
{
  mid = start + (end - start) /2;
  Page = 0 , count = 1;
  for( i = 0 ; i < n; i++)
  {
     Page += arr[i];
     if( Page > Mid)
     {
        count++; → // 2nd student
        Page = arr[i];
     }
  }
  if (count <= M)
  {
     ans = mid;            // leftword.
     end = mid - 1;
  }
  else  // (count > M)
     start = mid + 1;      // Rightword.
}
cout << ans;
```

→ T.C. ⇒ O(N. log N) / S.C. → O(1).

[count <= M] (M = 4)

[19 9 30 22 7]

I	II	III	IV
19+9	30	22+7	×

→ count <= M

ans = 30

end = mid - 1;

[अगर 3 में distribute होगा तो 4 में भी distribute हो जायेगा]

30 — 32 — 33 — 58 — 87

start = 30

end = ~~87~~ ~~57~~ ~~32~~ 35

ans = ~~58~~ ~~43~~ ~~36~~ 32

30 — 31

I	II	III	IV
19, 9	30	22, 7	
19, 9	30	22, 7	
18, 9	30	22, 7	
18, 9	30	22	7

(30) →

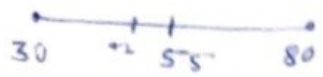
→ Painter Partition :

5	10	30	20	15

K = 3 (painters)

I	II	III	
5+10	30+20	15	→ 50 min
5+10	15	20+15	→ 35 min

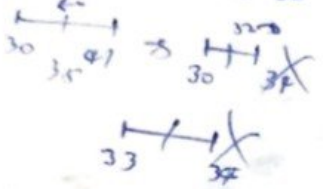
(1 length = 1 unit of time)



I	II	III	
5	30	20	
10		15	
			35

Start : 30
End : 80.

ans = 55 ~~92~~ ~~35~~ ~~80~~



⇒ Same code as Book allocation.

```
int arr[1000], n, k;

int start = 0, end = 0, mid, ans;
for (int i = 0; i < n; i++)
{
    if (start < arr[i])
    {
        start = arr[i];
    }
    end += arr[i];
}
while (start <= end)
{
    mid = start + (end - start) / 2;
    int Pages = 0, count = 1;
    for (i = 0; i < n; i++)
    {
        Pages += arr[i];
        if (Pages > mid)
        {
            count++;
            Pages = arr[i];
        }
    }
    if (count <= M)
    {
        ans = mid;
        end = mid - 1;
    }
    else
        start = mid + 1;
}
cout << ans;
```

start = max of array

end = sum of array

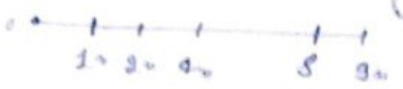
→ Aggressive cow :-

1	2	4	8	9

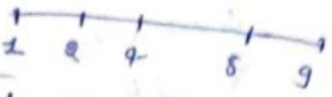
- Assign stall to K cows
- Minimum distance b/w any two of them is maximum possible.

K = 3

Min (Max)



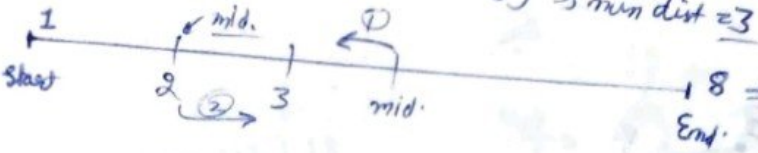
1 → 2 → 4 → 8 → min 1
1 → 2 → 4 → 9 → 1
1 → 2 → 8 → 9 → 1
1 → 4 → 8 → 9 → 3



[1 2 4] → min dist = 1
[1 2 8] → min dist → 2
[1 4 9] → min dist = 3

```
for (i=1; i<=INT_MAX; i++)
{
    ...
    break;
}
```

K = 3



start = ~~1~~ 3
End = ~~8~~ 3
ans = ~~2~~ 3

[INT_MAX − INT_MIN]
9 − 1

mid = 4 → 1 2 4 8
2 → 1 3 4 8
3 → 1 3 4 8

```
stalls[] , n , K
start = 1 , end , mid , ans ;
        Sort (stalls);
end = stall[n-1] - stall[0];
While (start <= end)
{
    mid = start + (end - start)/2;
    int count = 1 , Pos = stalls[0];
    for (i = 1 ; i < n ; i++)
    {
        if (Pos + mid <= stalls[i])
        {
            count++;
            Pos = stalls[i];
        }
    }
    if (count < K)
            end = mid - 1 ;
    else        // (count >= K).
    {
        ans = mid ;
        start = mid + 1;
    }
}
```

min 2 distance Pe cow milti

first cow always get a[0] Pos.

→ Time complexity ⇒ O(N log N).

→ KOKO Eating Banana:

3	6	11	7

Piles

h = 8

min speed to eat banana → min.

if speed = 4 banana/hr.

banana	hour
3	1 hr
6	2 hr.
11	3 hr.
7	2 hr.
	8 hr

5 banana/hr

3 = 1 hr
6 ⇒ 2 hr
11 = 3 hr
7 = 2 hr
8 hr.

```
for(i=1; i<INT_MAX; i++)
{
    ____
    break;
}
```
brute force method

(sum of all index) 27/8 ← 3 ——— 11 (INT_MAX)

start = 3 (left+1) 4 7 end

start = 3
end = ~~11~~ 6 ~~X~~
mid = ~~7~~ 4
ans = ~~7~~ 4

~~7 banana/hr~~

	4 ban/hr	3 bana/hr
3 — 1	3 — 1	3 — 1
6 — 1	6 — 2	6 — 2
11 — 2	11 — 3	11 — 4
7 — 1	7 — 2	7 — 3
5 hr	8 hr	10 hr

⇒ 5 hr.

```
int Piles[ ], h;
~~sum~~ start=0, end=0, mid, ans, n;   // total index
long long sum = 0  → Prevent overflow.
for(int i=0; i<n; i++)
{
    sum += Piles[i]; // sum of all elements.
    end = max(end, Piles[i]); // max in array
}
start = sum/h;   (start = sum/hour)
if(!start) => // start = 0
    start = 1;
While(start <= end)
{
    mid = start + (end - start)/2;
    int total_time = 0;
    for(int i=0; i<n; i++)
    {
        total_time += Piles[i]/mid;
        if(Piles[i] % mid)
            total_time++;
    }
    if(total_time > h)
    {
        start = mid+1;
    }
    else
    {
        ans = mid;
        end = mid-1;
    }
}
return ans;
```

STL / Vector

Standard Template library

→ How to declare vector : vector <datatype> name;

→ How to insert value in vector : vector <int> V (size of vector, initialize)

- vector <int> V(4) ⇒ | | | | |
- vector <int> V(4, 2) ⇒ | 2 | 2 | 2 | 2 |

→ vector <int> a;

a.push_back (4); | 4 | a

a.push_back (8); | 4 | 8 |

a.push_back (5); | 4 | 8 | 5 | |

a.push_back (7); | 4 | 8 | 5 | 7 | | | |

a.push_back (17); | 4 | 8 | 5 | 7 | 17 | | |

→ vector <int> V = {2, 4, 6, 8, 10};

| 2 | 4 | 6 | 8 | 10 |

→ vector <int> V(5);
for (i=0; i<5; i++)
{
cin >> V[i];

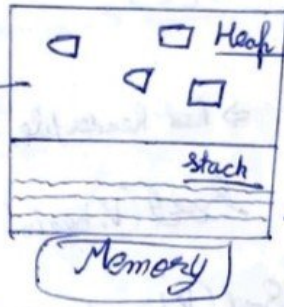
→ arr[n] ✗ (dynamic nA)

arr[1000] ✓

stack (Static memory allocation)

vector <int> V(n) → in heap

RAM



→ how to remove value from vector :

V | 2 | 4 | 6 | 8 | (0 1 2 3)

- V.pop_back(); → | 2 | 4 | 6 | | T.C. O(1)
- V.clear(); → | | | | |
- V.erase(V.begin()+2) → | 2 | 4 | 8 | | T.C. O(n)

(remove 2nd index value.)

→ Size & Capacity :

vector <int> V;

	size	capacity
V.push_back(1);	1	1
V.push_back(2);	2	2
V.push_back(10);	3	4
V.pop_back();	2	4

→ front, back, at, empty :

| 2 | 4 | 6 | 8 |

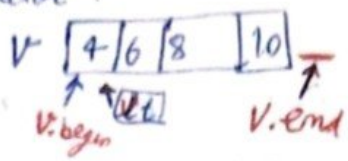
cout << V.front(); → 1

cout << V.back(); → 8

cout << V.empty(); → 0 (false)

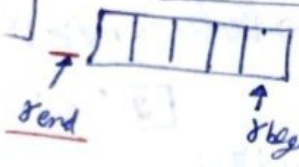
→ Iterator in a vector :

[V.begin()
V.end()]




```
for (auto it = V.begin(); it != V.end ; it++)
{
    cout << *it << " ";
}
```

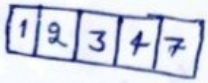
[V.rbegin();
V.rend();]

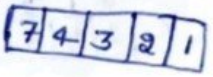


→
```
for (i = 0 ; i < V.size() ; i++)
    cout << V[i];
```

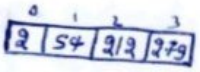
```
for (auto i : V)
cout << i ;
```

→ Sorting ⇒ Must header file : (#include <algorithm>).

Sort(V.begin(), V.end()); → Inc → 

Sort(V.begin(), V.end, greater<int>()); → Dec → 

→ Search • Sort vector before binary search.



cout << binary_search(ans.begin(), ans.end(), 54); → 0/1 true false.

- find :- cout << find(ans.begin(), ans.end(), 212) - ans.begin(); → 2

(212 ke jo iterator point kr rha hai) - (begin() wala iterator)

→ Insert :- V.insert(V.begin() + 1, 50);

↳ insert 50 at 2 V[1]

→ Count :-

int ct = count(V.begin(), V.end(), 5); ⇒ count how many times specific value occurs.

→ Max & Min :-

int maximum = max_element(V.begin(), V.end()).
min

↳ find max/min element

↳ find iterator pointing to the largest / smallest element in given range

Two Pointer

→ Segregate 0 and 1 :

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

⟶

0	0	0	1	1	1

① Sorting → $O(N^2)$

② Sort(arr.begin(), arr.end) → $O(N \cdot \log N)$.

③ count 0 = 0 + 0 + 3 → 3 0's

count 1 = 0 + 0 + 3 → 3 1's → $O(n)$

```
int count0 = 0
int count1 = 0
for (i=0; i<n; i++)
{
    if (arr[i] == 0)
        count0++;
    else
        count1++;
}
```

} n

// Print

```
for (i = 0; i < count0; i++)
    arr[i] = 0;
for (i = count0; i < n; i++)
    arr[i] = 1;
```

} n, } n

T.C. → $n + n + n = 3n$ → $O(n)$.

S.C. → $O(1)$.

→

0	1	1	0	0	1	0

start ↑ end ↑ ⇒ 2 Pointers

single Traverse

```
int start = 0, end = n-1;
while (start < end)
{
    if (arr[start] == 0)
        start++;
    else // (arr[start] == 1)
    {
        if (arr[end] == 0)
        {
            swap(arr[start], arr[end]);
            start++;
            end--;
        }
        else // (arr[end] == 1)
            end--;
    }
}
```

→ T.C. → $O(n)$.

→ Two Sum :

2	7	11	15	27

Target = 22

7 + 15 = 22.

```
vector<int> ans;
for(i=0 ; i<n-1 ; i++)          → n
{
  for(j=i+1; j<n ; j++)         → n
  {
    if(arr[i]+arr[j] == target)
    {
      ans.Push_back(arr[i]);
      ans.Push_back(arr[j]);
      break;
    }
  }
}
```

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

brutt force

by binary search

- if $2 + x = 22$

  $x = 20$ → search, in binary

```
for(i=0; i<n-1; i++)            → n
{
  x = target - arr[i];
  start = i+1 ; end = n-1;
  while( )                      → logn
  {
    binary search
  }
}
```

↳ T.C. ⇒ $O(n \log n)$.

→ By 2 Pointer :

2	7	11	15	27

start, end.

```
[sorted array]
start = 0 , end = n-1 ;
while (start < end)
{
  if(arr[start]+arr[end] == target)
  {
    return 1; // store (start, end)
  }
  else if(arr[start]+arr[end] < target)
      start++;
  else
      end--;
}
return 0;
```

T.C. $O(n)$

0	1	2	3	4	5
1	2	4	6	18	25

start, end

Target = 10

start end

1 + 25 = 26

1 + 18 ⇒ 19

1 + 6 ⇒ 7

2 + 6 ⇒ 8

4 + 6 = 10

arr[start] + arr[end] == target

< target

arr[start], arr[end] > target

arr[start], arr[end]

arr[start], arr[end]

arr[start], arr[end]

→ Pair with Given difference :-

0	1	2	3	4	5
2	3	5	10	50	80

diff = 95

start, end.

(Sorted array)

```
Start = 0, end = 1;
While (end < n)
{
  if (arr[end] - arr[start] == target)
        return 1;
  else if (arr[end] - arr[start] < target)
        end++;
  else
        start++;
}
return 0;
```

T.C. → O(n)

start
↳ increase
↳ decrease

end
↳ increase
↳ decrease

Pairs :- start increase, end decrease.

→ Prefix & suffix :-

arr →

6	4	5	-3	2	8

6	10	15	12	14	22

→ Prefix sum

22	16	12	7	10	8

→ Suffix sum

```
vector<int> prefix(n);
prefix[0] = arr[0];
for (i = 1; i < n; i++)
{
   Prefix[i] = Prefix[i-1] + arr[i];
}
```

suff[n-1] = arr[n-1]

(for(i=n-2; i>=0; i--);
suff[i] = suff[i+1] + arr[i];)

→

4	3	7	2

→ n

subarray

1 size : [4] [3] [7] [2] → n

2 size : [4 3] [3 7] [7 2] → n-1

3 size : [4 3 2] [3 7 2] → n-2

4 size : [4 3 7 2] → 1

→ Divide array in 2 subarray with equal sum :-

0	1	2	3	4	5	6	7
3	4	-2	5	8	90	-10	8

```
for(i=0; i<(n-1); i++)          → n
{
  int sum1=0, sum2=0;
  for(j=0; j<=i; j++)           → n
      sum1 += arr[j];
  for(j=i+1; j<n; j++)          → n
      sum2 += arr[j];
  if (sum1 == sum2)
      return 1;
}
return 0;
```

n * 2n → 2n²

T.C. → O(n²)

→ Pair with Given difference :-

0	1	2	3	4	5
2	3	5	10	50	80

Diff = 95

start, end.

(sorted array)

```
Start = 0, end = 1;
while (end < n)
{
  if (arr[end] - arr[start] == target)
        return 1;
  else if (arr[end] - arr[start] < target)
        end++;
  else
        start++;
}
return 0;
```

T.C. → O(n)

start
↳ increase
↳ decrease

end
↳ increase
↳ decrease

Pairs ⇒ start increase, end decrease.

→ Prefix & suffix :-

arr → | 6 | 4 | 5 | -3 | 2 | 8 |

6	10	15	12	14	22

→ Prefix sum

22	16	12	7	10	8

→ suffix sum

```
vector<int> prefix(n);
prefix[0] = arr[0];
for (i = 1; i < n; i++)
{
   Prefix[i] = Prefix[i-1] + arr[i];
}
```

suff[n-1] = arr[n-1]

(for(i=n-2 ; i>=0 ; i--);
suff[i] = suff[i+1] + arr[i];)

→

4	3	7	2

→ n

subarray

1 size : [4] [3] [7] [2] → n

2 size : [4 3] [3 7] [7 2] → n-1

3 size : [4 3 2] [3 7 2] → n-2

4 size : [4 3 7 2] → 1

→ Divide array in 2 subarray with equal sum :-

0	1	2	3	4	5	6	7
3	4	-2	5	8	20	-10	8

```
for(i=0; i<(n-1); i++)          → n
{
   int sum1=0, sum2=0;
   for(j=0; j<=i; j++)          → n
        sum1 += arr[j];
   for(j=i+1; j<n; j++)         → n
        sum2 += arr[j];
   if (sum1 == sum2)
        return 1;
}
return 0;
```

n * 2n → 2n²

T.C. → O(n²)

3	4	-2	5	8	20	-10	8

→ Total sum − Prefix sum = Suffix sum

Total sum − Prefix = ans

if (Prefix = ans) ✓

```
Total sum = 0, ans;
for(i=0; i<n; i++)          → n
    Total sum += arr[i];
int prefix = 0;
for(i=0; i<(n-1); i++)      → n
{
    prefix += arr[i];
    ans = Total sum - Prefix;
    if(ans == Prefix)
        return 1;
}
return 0;
```

T.C.
↳ O(n).

→ Largest Sum contiguous subarray:

0	1	2	3	4	5	6	7
3	4	-5	8	-12	7	6	-2

0-3
1-3
2-3
3-3
start
n-1
index

Prefix sum

{0} {0,1} {0,1,2} {0,1,2,3}
{1} {1,2} {1,2,3}.
{2} {2,3}
{3}

```
int Maxi = INT_MIN;
for(i=0; i<n; i++)          → n
{
    prefix = 0;
    for(j=i; j<n; j++)      → n
    {
        Prefix = arr[j];
        Maxi = max(Maxi, Prefix);
    }
}
return Maxi;
```

→ O(n²).

Kadane's algorithm

3	4	-5	8	-12	7	6	-2

Prefix sum: 3 7 2 11 0 7 13 11

↑ if -ve appear make it 0.

Maxi = INT_MIN ~~7 10~~ 13 = ans

```
Maxi = INT_Min;
Prefix = 0
for (i=0; i<n; i++)
{
    Prefix += arr[i];
    maxi = max(maxi, Prefix);
    if (Prefix < 0)
        Prefix = 0;
}
```

→ Max Difference b/w 2 element :

0	1	2	3	4	5	6	7
9	5	8	12	2	3	7	4

↑ (find Right side max difference)

```
ans = INT_MIN;
for(i=0; i<(n-1); i++)
{
  for(j=i+1; j<n; j++)
  {
    ans = max(ans, arr[j] - arr[i]);
  }
```

→ T.C. = $O(n^2)$

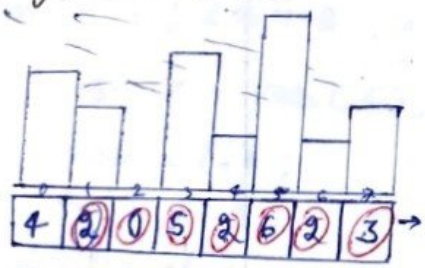
M_2 (Right से सबसे बड़ा element)

9	5	8	12	2	3	7	4
12	12	12	12	7	7	7	4

← Suffix max

⇒ $O(n)$.

→ Trapping Rain water :-



4	2	0	5	2	6	2	3	→ height

Store → 0 2 4 0 3 0 1 0 ⇒ 10

max left →	0	4	4	4	5	5	6	6	→ N

max Right →	6	6	6	6	6	3	3	0	→ N

min (max left, max Right) ⇒	0	2	4	0	3	0	1	0	→ N

(4-2) ↓ ↓ height - [min(max left, max Right)]

(T.C. = $O(N)$, S.C. ⇒ $O(N)$)

(min height) = min (left max, Right max).

```
Vector<int> height;
Leftmax[n], Rightmax[n];
leftmax[0] = 0;
for(i=1; i<n; i++)                          n ←
  leftmax[i] = max(leftmax[i-1],
                   height[i-1]);
Rightmax[n-1] = 0;
for(i=n-2; i>=0; i--)                       n ←
Rightmax[i] = max(Rightmax[i+1],
                  height[i+1]);
int water = 0;
for(i=0; i<n; i++)                          n ←
{
  min height = min(Leftmax[i],
                   Rightmax[i]);
  if(minheight - height[i] >= 0;
    water += minheight - height[i];
}
return water;
```

⇒ 3n

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

S.C. ⇒ $O(n)$.

```
height[]);
max left=0, maxRight=0, water;
max height = height[0], index=0;
for(i=1; i<n; i++)
{
  if(height[i] > max height)
  {
    max height = height[i];
  } index = i;
}
// left Part.
for(i=0; i<index; i++)
{
  if(leftmax > height[i])
    water += leftmax - height[i];
  else
    leftmax = height[i];
}
// Right Part.
for(i=n-1; i>index; i--)
{
  if(Rightmax > height[i])
    water += Rightmax - height[i];
  else
    Right max = height[i];
}
return water;
```

→ Time comp:

$O(n)$

→ space comp.

$(O(1))$

→ 3 Sum

1	4	45	6	10	8

x = 13

brute force

```
for(i=0; i<n-2; i++)
{
  for(j=i+1; j<(n-1); j++)
  {
    for(k=j+1; k<n; k++)
    {
      if(arr[i]+arr[j]+arr[k] == x)
        return x;
    }
  }
}
return 0;
```

→ T.C. = O(n³).

Sort →

1	4	6	8	10	45

Pointer1 (i), P2 (start), P3 (end)

x = 24

```
for(i=0; i<n-2; i++)
{
  ans = x - arr[i];
  start = i+1;
  end = n-1;
  while(start < end)
  {
    if(arr[start]+arr[end] == ans)
      return 1;
    else if(arr[start]+arr[end] > ans)
      end--;
    else
      start++;
  }
}
return 0;
```

→ T.C. → O(n²)

→ 4 Sum :

2D Array

int arr[4][3];

① Row major

	0	1	2
0	00	01	02
1	10	11	12
2	20	21	22
3	30	31	32

Visual

row_index, column_index

0	1	2	3	4	5	6	7	8	9	10	11
00	01	02	10	11	12	20	21	22	30	31	32

Memory

[Index = row_index × col + col_index]

[row_index = Index / col ;
col_index = Index % col;]

Index = row_index × col + col_index

(0 <= col_index < col)

row_index × col / col + col_index / col

=> row_index

index = row_index × col_index + col_index.

(row_index × col_index) % col + col_index % col

=> col_index % col -> col_index

-> ② int arr[5] →

0	1	2	3	4
500	504	508	512	516

arr = 500

arr represent/store location of 0th index. base address

arr = base address

arr[ ] = base_index

[arr[index] = base-address + index × size of element]

arr[3] = 500 + 3 × 4

= 512

-> [arr[i][j] = base-address + (i × col + j) × size of element]

-> ① int arr[4][3];

② Initialise : int arr[4][3] = {1, 2, ... 12};

③ Upgrade : arr[3][0] = 15,

user info. -> cin >> arr[1][2]

④ Print.

```
for(i=0 ; i<row, i++)
{
   for(j=0 ; j<col, j++)
      cout << arr[i][j];
}
```

→ Search Element :-

```
int main()
{
  int arr[3][3] = {1,2,3 ... 12};
  int x = 7;
  for(i=0; i<4; i++)
  {
    for(j=0; j<3; j++)
    {
      if (arr[i][j] == x)
      {
        cout << "yes";
        return 0;
      }
    }
  }
  cout << "no";
}
```

→ Pass by function.

```
void Print(int arr[][4])
{
  cout << arr[2][1];
}
int main()
{
  int arr[3][4] = {1,2, ... 12};
  Print(arr);
}
```

arr[i][j] = base address + (row_index × col + col_index) × SIZE of elem

500 + (2 × 2 + 1) × 4

500 + 36 = 536

(col is needed to Print.)

→ Add 2 Matrix :- [ans[i][j] = arr1[i][j] + arr2[i][j] .]

Time comp → O(row × col)

ans[i][j] ← space comp → O(row × col)

→ Print Row index with max Sum :-

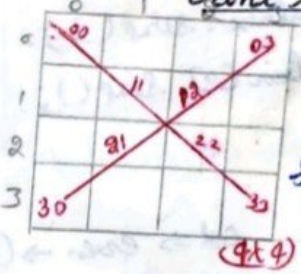
```
int sum = INT_MIN, index = -1;
for(int i=0; i<row; i++)
{
  int total = 0
  for(int j=0; j<col; j++)
    total += arr[i][j];
  if (sum < total)
  {
    sum = total;
    index = i;
  }
}
cout << index;
```

→ Reverse each Row of Matrix :-

```
for (int i = 0; i < row; i++)
{
  start = 0, end = col-1;
  while (start < end)
  {
    swap(arr[i][start], arr[i][end]);
    start++;
    end--;
  }
}
```

→ Print Sum of Diagonal Elements :-

→ only for square matrix



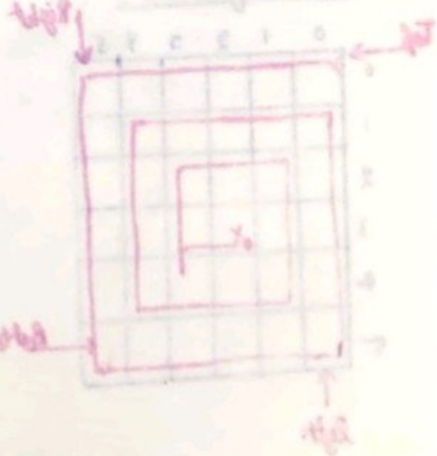
first = 00, 11, 22, 33.

second = 03, 12, 21, 30.

i j

```
int first = 0;
for(i=0; i<row; i++)
  first += arr[i][i];
int second = 0;
int i = 0, j = col-1;
while(j >= 0)
{
  second += arr[i][j];
  i++; j--;
}
```

Time Com →
O(2row)
||
O(row)



→ Column major order

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

arr[0][3]

0	1	2	3	4	5	6	7	8	9	10	11
1	4	7	10	2	5	8	11	3	6	9	12

Index =

(Column major → Banking system
Row major → Email excess)

→ Vector in 2-D :

↳ vector < vector < int > > matrix ;

→ vector < vector < int > > matrix (rows, vector < int > (col , initline)) ;

→ row = matrix . size ( ) ;.
col = matrix [0]. size ( ) ;.

→ Wave form :

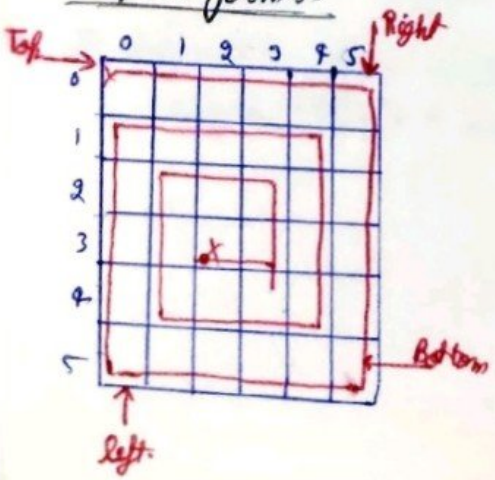
col ⇒ even → (up → down). ⇒ for(i=0 ; i < row ; i++)
col ⇒ odd → (down → up) . ⇒ for(i=row-1 ; i >= 0 ; i--)

```
for(int j = 0 ; j < col ; j++)
{
    if (j % 2 == 0)
        for(i = 0 ; i < row ; i++)
            cout << arr[i][j] << " ";
    else
        for(i = row-1 ; i >= 0 ; i--)
            cout << arr[i][j] << " ";
}
```

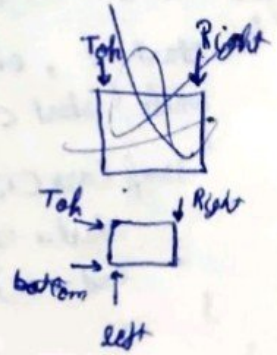
→ Time comp. ⇒ O(row * col)

→ Space com. ⇒ O(1)

→ Spiral form :



Top ⇒ Row (0).
Right ⇒ column (5).
Bottom ⇒ Row (5).
Left ⇒ column (0).

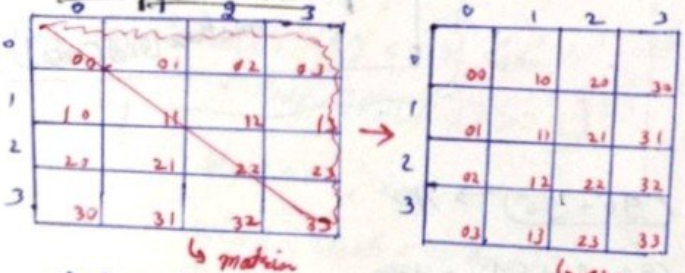


```
Toph, Right, Bottom, left;
while(toph <= bottom && left <= Right)
{
for(int j = left ; j <= Right ; j++)
    cout << matrix[toph][j];
      toph++;
for(int i = toph ; i <= bottom; i++)
    cout << matrix[i][Right];
      Right--;
if(toph <= bottom) --> //edge cases.
{
  for(int j = Right ; j >= left; j--)
    cout << matrix[bottom][j];
}     bottom--;
if(left <= Right) --> // edge cases.
{
  for(int i = bottom; i >= toph ; i--)
    cout << matrix[i][left];
}     left++;
}
```

→ O(row × col) ⇒ T.C.

→ O(1) ⇒ S.C.

→ Transpose Matrix ⇒



↳ matrix ↳ arr

→ Digonal ke ek side ~~ko~~ swap kar do.

```
for(i=0; i<n; i++)
{
  for(j=0; j<n; j++)
  {
    arr[j][i] = matrix[i][j];
  }
}
```

↳ T.C. = O(n²) ↳ S.C. = O(n)

```
for(int i = 0; i < row-1); i++)
{
  for(int j = i+1; j < col; j++)
  {
    swap(matrix[i][j], matrix[j][i]);
  }
}
```

→ T.C. ⇒ O(n²).

→ S.C. ⇒ O(1).

→ Rotate Matrix by 90° clockwise

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

90°

	0	1	2	3
0	13	9	5	1
1	14	10	6	2
2	15	11	7	3
3	16	12	8	4

(col → Reverse) ⇒ Row

[0][3] → [3][3] [0][0] → [0][3]
[3][1] → [1][3] [1][0] → [0][2]
[2][3] → [3][2] [2][0] → [0][1]
[0][2] → [2][3] [3][0] → [0][0]
i j J (X) ? i j J X

↳ as always ⇒ i + X = 3

i + x = n - 1 ⇒ X = n - 1 - i

```
for(i=0; i<(n-1); i++) // transpose.
{
   for(j=i+1; j<n; j++)
      swap(matrix[i][j], matrix[j][i]);
}

for(i=0; i<n; i++) // Reverse each Row.
{
    start = 0, end = n-1;
    while (start < end)
    {
      swap(matrix[i][start],
           matrix[i][end]);
      start++, end--;
    }
}
```

→ T.C. ⇒ $O(n^2)$

→ S.C. ⇒ $O(1)$

```
for(i=0; i<n; i++)
{
   for(j=0; j<n; j++)
   {
     ans[j][n-1-i] = matrix[i][j];
   }
}
```

T.C. ⇒ $O(n^2)$ S.C. $O(n^2)$

(for anticlockwise) Reverse each column is more preferred.

```
swap(matrix[start][i];
     matrix[end][i]);
```

→ Rotate matrix 180°

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

180

	0	1	2	3
0	16	15	14	13
1	12	11	10	9
2	8	7	6	5
3	4	3	2	1

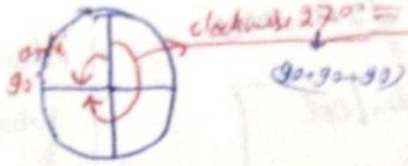
(90° + 90°) → 180

(90 + 90 + 90) → 270.

① column reverse
② Row Reverse

```
for(j=0; j<n; j++) // Reverse each column
{
   start = 0, end = n-1;
   while(start < end)
   {  swap(matrix[start][j], matrix[end][j]);
      start++, end--;
   }
}
for(i=0; i<n; i++) // Reverse each Row.
{
   start = 0, end = (n-1);
   while(start < end)
   {
    swap(matrix[i][start], matrix[i][end]);
   Start i++ ; end--;
   }
}
```

→ Rotate Matrix by K times :-



rotate
- 1 time = 90°
- 2 time = 180°
- 3 time = 270°
- 4 time = 360°
- 5 time = 90°
- 6 time = 180°

K → K = K % 4

```
Rotate 90°(matrix)
{
   // Rotate 90 code.
}
int main() {
  int K;
  cin >> K;
  K = K % 4;
  while (K)
  {
    Rotate 90(matrix)
    K--;
  }
```

→ Binary Search in 2D Array :-

	0	1	2	3	4
0	2	6	10	14	18
1	20	24	27	29	38
2	47	52	78	93	102
3	108	111	204	218	325
4	350	388	401	482	452

→ Sorted array

row = 5(n) x = 52
Col = 5(m)

Brutt force.

```
for(i=0; i<N; i++)
{
  for(j=0; j<m; j++)
  {
    if (Matrix[i][j] == x)
      return 1;
  }
}
return 0;
```

→ T.C. → O(nm)
→ S.C. → O(1)

N M
↑ ↑
row, col, x

```
for(i=0; i<N; i++)
{
  if (Matrix[i][0] <= x &&
      Matrix[i][M-1] >= x)
  {
    start = 0, end = M-1;
    while (start <= end)
    {
      mid = (start + end)/2;
      if (Matrix[i][mid] == x)
        return 1;
      else if (Matrix[i][mid] < x)
        start = mid+1;
      else
        end = mid-1;
    }
    return 0;
  }
}
```

→ T.C. → O(N+logM) → S.C. → O(1)

→ binary search

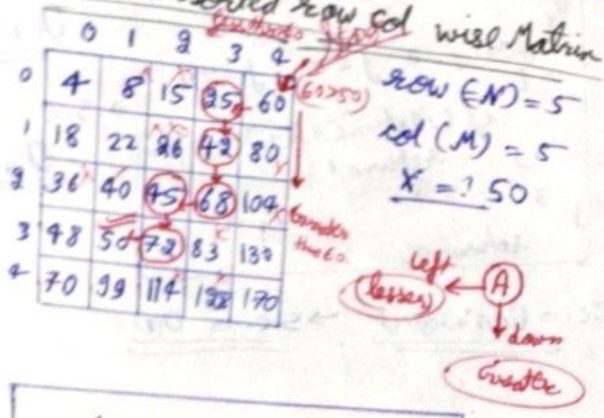
direct jump in mid of 2D - array:

row_index = Index / col
col_index = Index % col

```
start = 0 , end = row * col - 1 ;
while (start <= end)
{
  mid = (start + end) / 2;
  row_index = mid / col;
  col_index = mid % col;
  if (Matrix[row_index][col_index] == x)
      return 1;
  else if (Matrix[row_index][col_index] < x)
      start = mid + 1;
  else
      end = mid - 1;
}
return 0;
```

T.C. → $O_{(row \times col)}$
S.C. → $O_{(1)}$

→ Search in sorted row col wise Matrix:




```
if A == x
    return 1;
else if (A < x)
    i++;
else // (A > x)
    j--;
```

```
N, M
i = 0 , j = M-1;
while (i < N && j >= 0)
{
  if (Matrix[i][j] == x)
      return 1;
  else if (Matrix[i][j] < x)
      i++;
  else
      j--;
}
```

→ T.C. → $O(n+m)$
$\Omega(1)$
→ S.C. → $O_{(1)}$.

→ How to store 2 number in 1 Position :-

N = 1 to 99 → 100

Store no & it's occurance.

→ 2, 2, 2, 2, 2, 2

2 (no), 6 (occurance)

Store → (2 + 6 × 100) ⇒ 602

- no = 602 % 100 = 2
- occurance = 602 / 100 = 6

→ 12, 12, 12, 12, 12, 12 → (12 × 4 × 100) ⇒ 4812

→ Find Missing & Repeating :-

	0	1	2	3	4	5	6
arr →	4	3	2	1	2	7	6

→ 1 to N

↳ 1 number is missing & 1 is repeated.

(1) Brutt : Repeating : 2 } T.C. ($O(n^2)$)
Missing : 5 } T.C. ($O(n^2)$)

(2) sort → 1 2 2 3 4 6 7 → T.C. = O(nlogn)
missing = 5
Repeating = 2 → n ⇒ O(n log n)

(3)

	0	1	2	3	4	5	6
arr	4	3	2	1	2	7	6

→

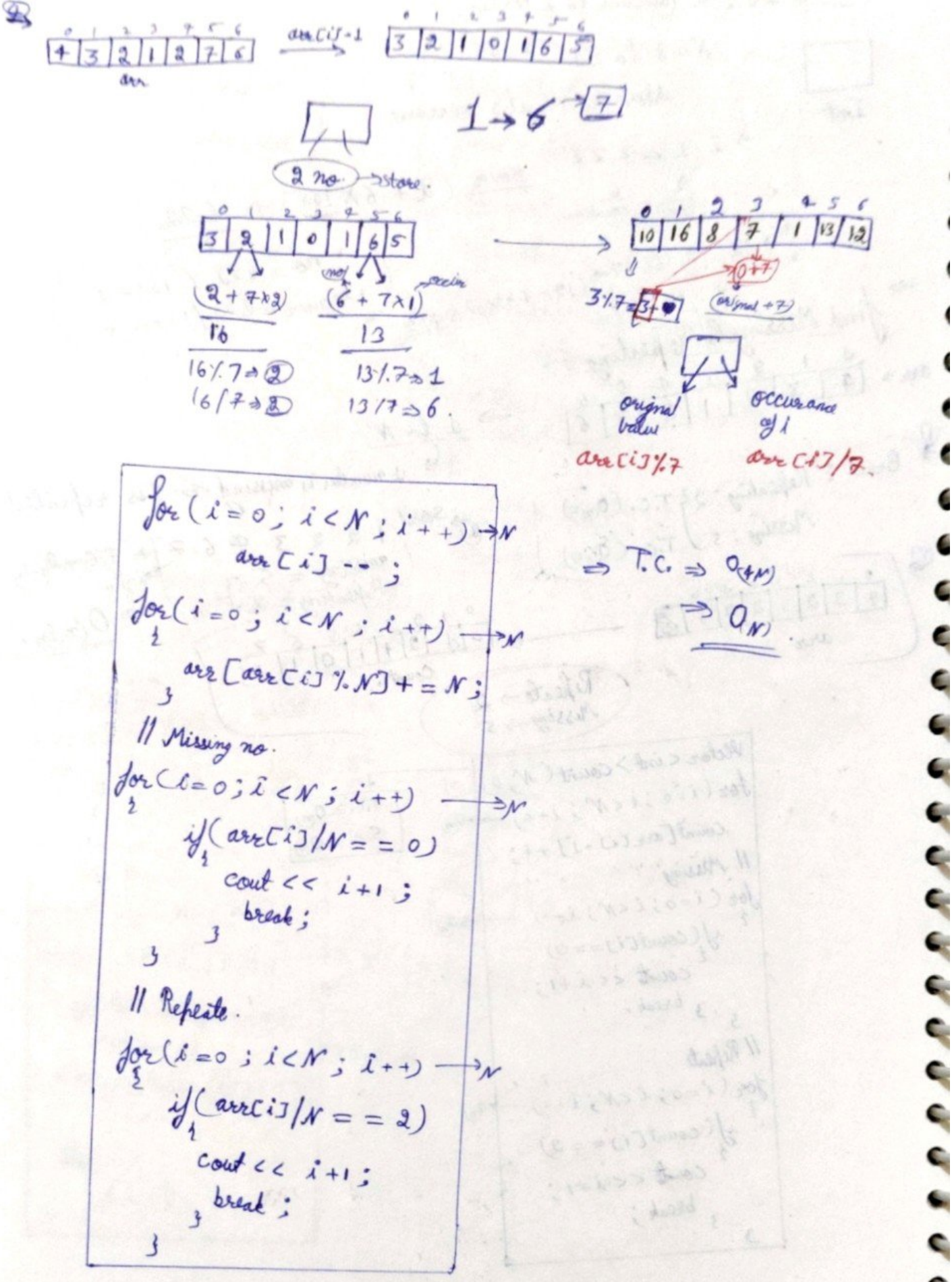
	0	1	2	3	4	5	6	7
Count.	0	1	2	1	1	0	1	1

↳ use 1 base index also

Repeate → 2
Missing → 5

```
Vector<int> count(N, 0)
for(i=0; i<N; i++)                    → O(n)
    count[arr[i]-1]++;
// Missing
for(i=0; i<N; i++)                    → O(n)
{
    if(count[i]==0)
    {
        cout << i+1;
        break;
    }
}
// Repeate
for(i=0; i<N; i++)                    → O(n)
{
    if(count[i]==2)
    {
        cout << i+1;
        break;
    }
}
```

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

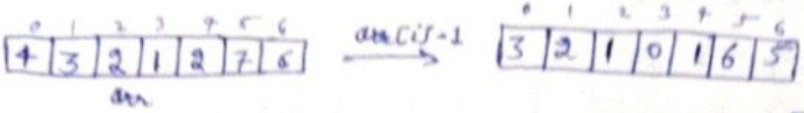


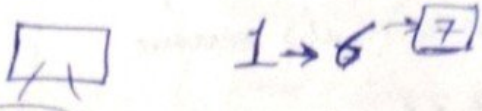

```
for (i=0; i<N; i++)          → N
    arr[i] --;
for (i=0; i<N; i++)          → N
    arr[arr[i]%N] += N;
// Missing no.
for (i=0; i<N; i++)          → N
    if (arr[i]/N == 0)
        cout << i+1;
        break;
// Repeate.
for (i=0; i<N; i++)          → N
    if (arr[i]/N == 2)
        cout << i+1;
        break;
```

⇒ T.C. ⇒ $O(4N)$

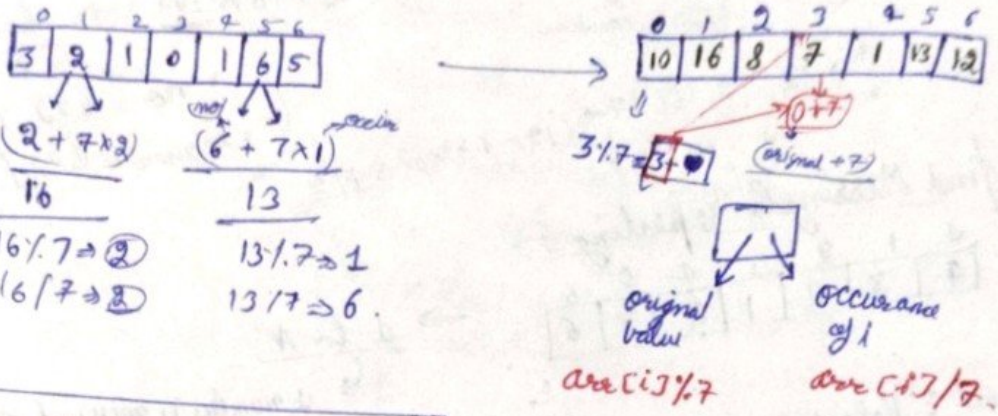
⇒ $O(N)$

Q3





2 no. → store.

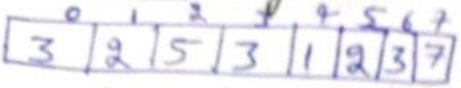



```
for(i=0; i<N; i++)          → N
    arr[i] --;
for(i=0; i<N; i++)          → N
{
    arr[arr[i] % N] += N;
}
// Missing no.
for(i=0; i<N; i++)          → N
{
    if(arr[i]/N == 0)
    {
        cout << i+1;
        break;
    }
}
// Repeate.
for(i=0; i<N; i++)          → N
{
    if(arr[i]/N == 2)
    {
        cout << i+1;
        break;
    }
}
```

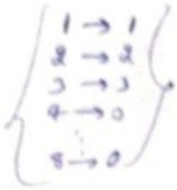
⇒ T.C. ⇒ $O_{(4N)}$

⇒ $O_{(N)}$.

→ Find the Occurence of Number :-

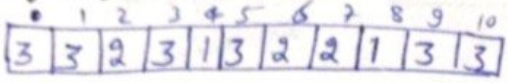


0 → 7 → 8



→ Same as Repeated code Question Last Problem...

→ Majority Element :-

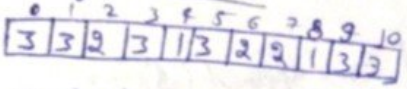


N=11

brute force → O(N²)

{number > N/2}

→ Majority in election



→ Moore voting algorithm

candidate → 3

count → 0 1 2 1 2 1 0 1 2

index 10

Verify :- number > N/2.

T.C. → O(N)

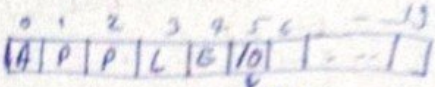
S.C. → O(1)

```
int candidate, count = 0;
for (i = 0; i < N; i++
{
   if (count == 0)
   {
      count = 1;
      candidate = arr[i];
   }
   else
   {
      if (candidate == arr[i])
         count++;
      else
         count--;
   }
}
   count = 0; // count total no. of time candidate occur.
   for (i = 0; i < N; i++)
   {
      if (arr[i] == candidate)
         count++;
   }

   if (count > N/2) // check candidate occurs (> N/2);
      return candidate;
   else
      return -1;
```

String → "Dynamic memory allocation"

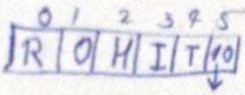
'\0' → Null character (heap)

```
char arr[20];
cin >> arr;
cout << arr;
```



APPLE

```
String S;
cin >> S;
cout << S;
```



cin >> → tab / space / Enter stop reading.

→ getline (cin, S); → write in a paragraph form.

→ S.size ⇒ size of string.

→ string $S_3 = S_1 + S_2$ ⇒ attach two strings. / S_1.append(S_2);

→ S.Push_back('P'); ⇒ add 'P' at last of string. / S = S + "P";

→ S.Pop_back(); ⇒ remove last char of string.

→ Escape character '\' ⇒ String S = "Harsh is a \"bad\" boy";

String S = "\\"; → \

→ Reverse string ⇒

```
String S = "Harsh";
            ↑     ↑
          start  end
start = 0 ,  end = S.size() - 1;
while( start < end)
{
   swap (S[start], S[end]);
   start++ , end--;
}
```

→ find size without S.size.

S = H a r s h \0

```
size = 0;
While( S[size] != '\0')
{
   size++;
}
cout << size;
```

→ Sort vowel in String : ascii value

S = "leetcode"

ee oe ⇒ eeeo

← sort "Leetcedo"

→ A → 65 a → 97

→ Select the vowel

→ Sort the vowel

→ Insert the vowel at right position.

S = "lEetcOde"

"lEOtcede"

(Upper case or lower case are different).

Ascii

```
vector<int> lower(26, 0);
vector<int> upper(26, 0);

for(int i=0; i<s.size(); i++)                    → O(n)
{
    // lower
    if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u')
        lower[s[i] - 'a']++;
        s[i] = '#';
    }
    // upper
    else if(s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O'||s[i]=='U')
        upper[s[i] - 'A']++;
        s[i] = '#';          → // vowel replace by '#'.
}

String vowel;
// upper
for(i=0; i<26; i++)                              → O(1)
{
    char c = 'A'+i;
    while(upper[i])
    {
        vowel += c;
        upper[i]--;
    }
}
// lower
for(i=0; i<26; i++)                              → O(1)
{
    char c = 'a'+i;
    while(lower[i])
    {
        vowel += c;
        lower[i]--;
    }
}
int first=0, second=0;
while(second < vowel.size())
{
    if(s[first] == '#')
    {
        s[first] = vowel[second];
        second++;
    }
    first++;
}

return s;
```

// l # # t C # d #
// E o e e → sorted vowels.
(first) (second)
O(n)

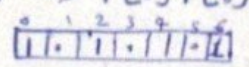
→ T.C.
⇒ O(2n)
⇒ O(n)

→ S.C.
O(n)

→ Add string :- (Big Int)

→ Defanging an I.P. address :-

address = 1.1.1.1 → 1[.]1[.]1[.]1

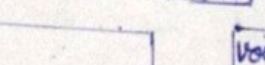


```
int index = 0
String ans;
While (index < address.size())
{
   if(address[index] == '.')
        ans = ans + "[.]";
   else
        ans = ans + address[index];
   index++;
}
return ans;
```

T.C. → O(n)
S.C. → O(n).

→ Check if string is rotated by 2 Places :-

S amazon → clockwise
→ anticlockwise

azonam → 

```
str1, str2;
String clockwise = Str1,
       anticlockwise = Str;

rotate clockwise(clockwise)
rotate clockwise(clockwise)
if (clockwise == str2)
     return 1;

rotate anticlockwise(anticlockwise)
rotate anticlockwise(anticlockwise)
if(anticlockwise == str2)
     return 1;
return 0;
```

& → reference

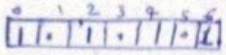
```
void clockwise(string &S)
{
   char C = S[S.size()-1];
   int index = S.size()-2;
   while(index >= 0)
   {
     S[index+1] = S[index];
      index--;
   }
   S[0] = C;
}
```

```
void anticlockwise
{       (string &S)
   char C = S[0];
   int index = 1;
   while(index < S.size())
   {
     S[index-1] = S[index];
      index++;
   }
   S[S.size()-1] = C;
}
```

→ Add String :- (Big Int)

→ Defanging an I.P. address :-

address = 1.1.1.1 → 1[.]1[.]1[.]1

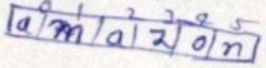


```
int index = 0
String ans;
while (index < address.size())
{
   if (address[index] == '.')
         ans = ans + "[.]";
   else
      ans = ans + address[index];
   index++;
}
return ans;
```

T.C. ⇒ O(n)
S.C. ⇒ O(n).

→ Check if string is rotated by 2 Places :-

S amazon → clockwise
→ anticlockwise

azonam → 

```
str1, str2;
String clockwise = str1,
       anticlockwise = str;

rotate clockwise(clockwise)
rotate clockwise(clockwise)
if (clockwise == str2)
      return 1;

rotate anticlockwise(anticlockwise)
rotate anticlockwise(anticlockwise)
if (anticlockwise == str2)
      return 1;
return 0;
```

& → reference

```
void clockwise(string &S)
{
   char c = S[S.size()-1];
   int index = S.size()-2;
   while(index >= 0)
   {
      S[index+1] = S[index];
      index--;
   }
   S[0] = c;
}
```

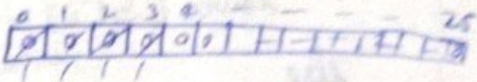
```
void anticlockwise
        (string &S)
{
   char c = S[0];
   int index = 1;
   while(index < S.size())
   {
      S[index-1] = S[index];
      index++;
   }
   S[S.size()-1] = c;
}
```

→ Check Pangram :

sentence = "the quick brown fox jumps over the lazy dog".

```
vector < bool > alpha (26, 0)
for( i = 0; i < sentence.size(); i++)
{
    int index = sentence[i] - 'a';
    alpha[index] = 1;
}
for( i = 0 ; i < 26 ; i++)
{
    if( alpha[i] == 0)
        return 0;
}
return 1;
```

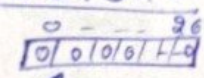
T.C. → O(n)
S.C. → O(26) = O(1).



→ Sort a String :

S = e a b c a b d e

a = 2
b = 2
c = 1
d = 1
e = 1
f = 0
g = 0
.
.
z



alpha

```
vector <int> alpha (26, 0)
for(i = 0; i < s.size(); i++)
{
    index = s[i] - 'a';
    alpha[index]++;
}
string ans;
for(i = 0 ; i < 26 ; i++)
{
    char ch = 'a' + i;
    while (alpha[i])
    {
        ans += ch;
        alpha[i]--;
    }
}
return ans;
```

O(n)

O(n)

T.C → O(2n)
→ O(n)

S.C → O(n)

→ Longest Palindrom :

S = a b c c c d d

cc → 2
cccc → 4
dccccd = 6
dccaccd = 7

S = a a a a b b a c c c c

a a b c c b c c b a a → 11

count → even → direct
odd → [n - (n-1)]

* (lower + Upper case)

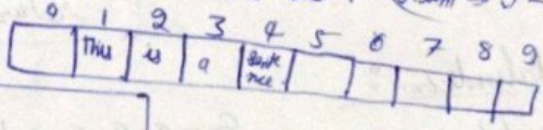
String s;

→ T.C. → O(n)
S.C. → O(26) = O(1)

```
Vector < int > lower(26, 0), Upper(26, 0);
for (i = 0 ; i < s.size() ; i++)                    → O(n)
{
    if (s[i] >= 'a')
        lower[s[i] - 'a']++;
          index
    else
        Upper[s[i] - 'A']++;
          index
}
int count = 0;
bool odd = 0;
for(i = 0; i < 26; i++)                             → O(26) = O(1)
{
    if(lower[i] % 2 == 0)
        count += lower[i];
    else
    {
        count += lower[i] - 1;
        odd = 1;
    }
    if (Upper[i] % 2 == 0)
        count += Upper[i];
    else
    {
        count += Upper[i] - 1;
        odd = 1;
    }
}
return count + odd;
```

② Sorting the sentence : s = "is2 sentence4 This1 a3". (Num → 0 - 9)

→ ans




```
string s;
vector < string > ans(10);
string temp;
int count = 0, index = 0;
while (index < s.size())
{
    if (s[index] == ' ')
    {
①       int Pos = temp[temp.size() - 1] - '0';
        temp.pop_back();
        ans[Pos] = temp;
        temp.clear();
        count++;
    }
    else temp += s[index];
    index ++
}
```

copy ① // last sentence case.

```
// Print string.
for (i = 1; i <= count; i++)
{
    temp += ans[i];
    temp += ' ';
}
temp.pop_back(); // Remove last sentence space ' '
return temp;
```

→ Add string :- long long int → 19 digit
Big Int

↳ "long integers store as string.

num1 = "2 6 5 5 3".

num2 = "6 9 8".

num1 ≥ num2

27 25 1

Function → string add (string num1, string num2)

```
int index1 = num1.size() - 1 , index2 = num2.size() - 1;
String ans;
int carry = 0, sum;
While (index2 >= 0)
{
  sum = (num1[index1] - '0') + (num2[index2] - '0') + carry;
    carry = sum / 10;
    char ch = '0' + sum % 10;
      ans += ch;
  } index2--, index1--;

// remaining's of index 1.
while (index >= 0)
{
  sum = num1[index] - '0' + carry;
  carry = sum / 10;
  char ch = '0' + sum % 10;
    ans += ch;
  } index1--;

// to carry last carry remain
  if (carry)
    ans += '1';   // last carry mein 1 hi aayega,

// (Reverse the ans. to get correct output)
  reverse (ans.begin(), ans.end())
  return ans;
}
```

⇒

```
void int main()
{
  int num1; num2;

if (num1.size() > num2.size())
    return add (num1, num2);
else
  return add (num2, num1);
}
```

→ Roman to integer :-

I → 1	III ⇒ 3
V → 5	LXX ⇒ 70
X → 10	CD ⇒ −100+500 ⇒ 400
L → 50	CCCXXV ⇒ 325
C → 100	XL III ⇒ −10+50+3 ⇒ 43
D → 500	MCCXLVIII ⇒ 1000+200+100+(−10)+50+5+3 ⇒ 1948
M → 1000	

```
int sum = 0, index = 0;
while (index < s.size() - 1)
{
   if( Num(s[index]) < num(s[index + 1]))
        sum -= num(s[index]);
   else
        sum += num(s[index]);
   index++;
}
sum += num(s[s.size() - 1]);
return sum;
```

num function

```
int num(char c)
{
   if(c == 'I')
        return 1;
   else if(c == 'V')
        return 5;
   else if(c == 'X')
        return 10;
   else if(c == 'L')
        return 50;
   else if(c == 'C')
        return 100;
   else if(c == 'D')
        return 500;
   else
        return 1000;
}
```

→ Integer to Roman ⇒ [1 → 39999]

1948 ⇒ 1000 + 900 + 40 + 8

M + CM + XL + VIII

MCMXLVIII

How to define →

1 → I	10 → X	100 → C	1000 → M
2 → II	20 → XX	200 → CC	2000 → MM
3 → III	30 → XXX	300 → CCC	3000 → MMM
4 → IV	40 → XL	400 → CD	
:	:	:	
9 → IX	90 → XC	900 → CM	

* Factorial of a Number :-

32 bit → INT-overflow, max = $2^{31}-1$

$\frac{2^{32}}{2} \Rightarrow 2^{31}, 2^{31}$ (+ve, -ve)

$2^{31}-1$ → exceed 0 case: for +ve 2^{32} (0000 → 0)

```
vector <int> ans(1, 1)
while (N > 1)
{
  int carry = 0, res, size = ans.size();
  for (i = 0; i < size; i++)
  {
    res = ans[i] * N + carry;
    carry = res / 10;
    ans[i] = res % 10;
  }
  while (carry)
  {
    ans.push_back(carry % 10);
    carry /= 10;
  }
  N--;
}
reverse(ans.begin(), ans.end())
```

→ longest substring without repeating char :- Sliding window.

① a b c d e c b e a d f. → a b c d e ⇒ 5 char
→ c b e a d f ⇒ 6 char

first, second

[| | | | a | b | c | d | e | f | | | |] (0, 1, ..., 255)

char should be distinct in window.

length ⇒ 0, 1, 2, 3, 4, 5, 6

// any character present in string
// 256 char ascii

a b c a b (first, second)

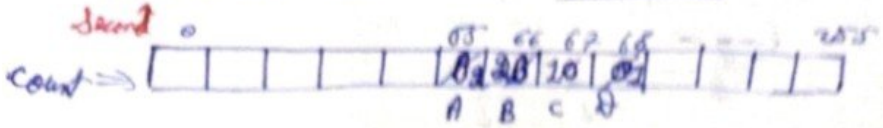
```
string s;
vector <bool> count(256, 0);
int first = 0, second = 0, len = 0;
while (second < s.size())
{
  while (count[s[second]]) // Repeating char
  {
    count[s[first]] = 0;
    first++;
  }
  count[s[second]] = 1;
  len = max(len, second - first + 1);
  second++;
}
return len;
```

count → T.C. ⇒ O(n).

-> Smallest distinct window :-

first ⇒ A A B B B C B B A C. → BAC = 3

second

count ⇒ 

→ T.C. = O

```
vector<int> count(256, 0)
int first=0, second=0, len=str.size(), diff=0;
  // calculate total unique character.
while(first < str.size())
{
    if(count[str[first]] == 0)
        diff++;

    count[str[first]]++;
    first++;
};

for(i=0; i<256; i++) // count me sabhi char ko 0 karo.
    count[i] = 0;

first = 0;  // first ko 0 pe le aao.
while (second < str.size())
{
    // diff exist. // Jab-tak diff '0' nhi ho jata second ko badhaye karo.
    while (diff && second < str.size())
    {
        if(count[str[second]]==0)
            diff--;

        count[str[second]]++;
        second++;
    }

    len = min(len, second-first);

    // diff ke value 1 na ban jaye.
    while(diff != 1)
    {
        len = min(len, second-first);
        count[str[first]]--;

        if(count[str[first]]==0)
            diff++;
        first++;
    }
}
return len;
```

→ Longest prefix suffix (LPS) KMP algorithm

A B C D E A B C D

Brute force → Q(n²) TC:

Prefix	Suffix
A	D
A B	C D
A B C	B C D
A B C D	A B C D
A B C D E	E A B C D
A B C D E A	D E A B C D
A B C D A B	C D E A B C D
A B C D E A B C	B C D E A B C D

→ Q(n²) → Q(n²)

A B C D A B D A B C → 1 - 2

suffix. → O(n²)

A B C D E A B C D

LPS: 0 0 0 0 0 1 2 3 4 → Last ans

S → A B C A B D A B D A B C D

LPS: 0 0 0 1 2 0 1 2 3 4 5 3 4 5 6 → Last ans

pre = 0 suf = 1

(here 4 represent all value before index 4 has prefix = suffix)

index 11 → mismatch

Prefix - A B C A B D

suffix - A B C A B C

S1 = longest Prefix

S2 = longest Suffix

A B C A B → suff

Pre → longest suff & longest Pre

matching → index + 1

LPS

T.C. ⇒ O(n).

S.C. ⇒ O(n).

```
vector<int> lps(s.size(), 0);
int pre = 0, suf = 1;
while (suf < s.size())
{
    if (s[pre] == s[suf])
    {
        lps[suf] = pre + 1;
        pre++, suf++;
    }
    else
    {
        if (pre == 0)
        {
            lps[suf] = 0;
            suf++;
        }
        else
        {
            pre = lps[pre - 1];
        }
    }
}
return lps[s.size() - 1];
```

→ **String Matching** :-

a b c a b d e f g. → haystack

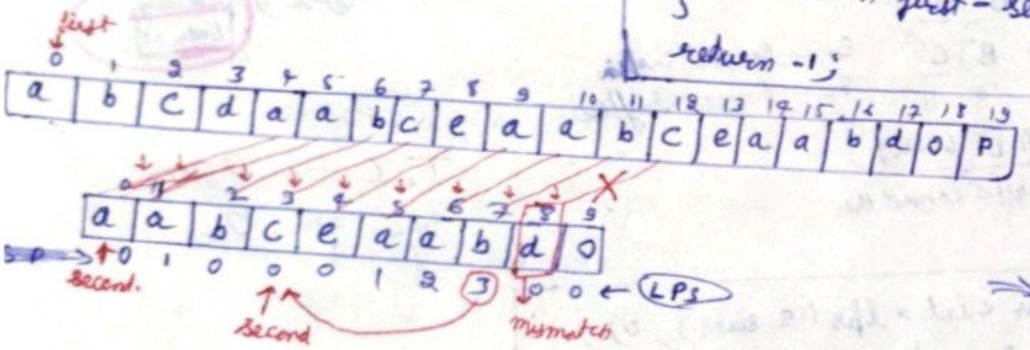
d e f. → needle

Brute force.

```
n = haystack.size(), m = needle.size();
for(i=0 ; i<=n-m ; i++)
{
    first = i , second = 0;
    while(second < m)
    {
        if(haystack[first] != needle[second])
        {
            break;
        }
        else
        {
            first++ , second++;
        }
    }
    if(second == m)
        return first - second;
}
return -1;
```

T.C. ← O(n)

S.C. ← O(1)



⇒ T.C. → O(n+m)

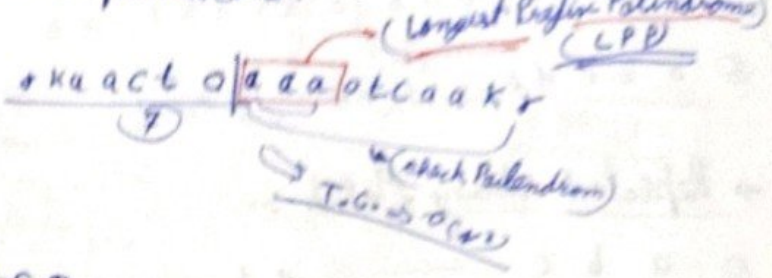
S.C. → O(m).

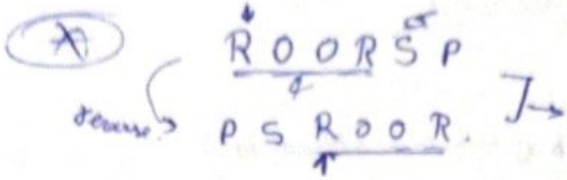
```
vector<int> lps(needle.size(), 0);
lpsfind(lps, needle);
int first=0 , second=0;
while(second < needle.size() && first < haystack.size())
{
    if(needle[second] == haystack[first])
    {
        second++;
        first++;
    }
    else
    {
        if(second == 0)
            first++;
        else
            second = lps[second-1];
    }
}
if(second == needle.size)
    return first - second;
else
    return -1;
```

→ LPS (previous code)

→ Make a string Palindrome :-

Add min char at start to make it palindrome.





P S R O O R S P $ P S R O O R (LPS)

Longest Prefix suffix.

6 - 4 = 2

LPS → R O O R S P $ P S R O O R
0 0 0 1 0 0 0 0 0 1 2 3 4

steps:
1. S
2. Rev = S (Reverse)
3. S += $
4. S += Rev.
5. LPS(S)

```
string str;
string rev = str;
reverse(rev.begin(), rev.end());
int size = str.size();
str += $; // seprator
str += rev; // add reverse string.

// LPS nikalna hai
int n = str.size(); // new size.
vector<int> lps(n, 0);
int pre = 0, suf = 1;
while (suf < str.size())
{
    if (str[pre] == str[suf])
    {
        lps[suf] = pre + 1;
        pre++, suf++;
    }
    else
    {
        if (pre == 0)
        {
            lps[suf] = 0;
            suf++;
        }
        else
        {
            pre = lps[pre - 1];
        }
    }
}

return size - lps[n-1];
```

⇒ Circular Pattern matching :-

① c d e a b r o a b | c d e a b r o a b

② a b c d e ... ↑ find string if exist

⇒ Simple LPS

⇒ Repeated string Match :-

① a b c → a bc a bc

② b c a b c → Ref = 2

Part of substring

① a b c ⇒ make atleast 15

② a b c a b c a b c a b c a b c ⇒ 15 char

Substring

① a b c a b c a b c a b c a b c

Ref = 5 6

step :-

① Repeated wale Ka size ↑ >=

② check substring

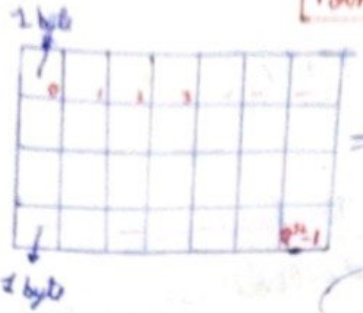
③ 1 more step only

④ if substring (return ref) / if not (return -1)

```
String a , string b ;
if(a == b)
   return 1;
int repeat = 1;
string temp = a;
while ( temp . size() < b . size() )
{
   temp + = a;
   repeat ++;
}
// KMP pattern search
if ( KMP_MATCH(temp, b) == 1)
     return repeat;

// Temp + a , KMP search
if (KMP_MATCH(temp + a, b) == 1)
     return repeat + 1 ;

return -1 ;
```

KMP_ Search. ⇒ full code of "String Matching".

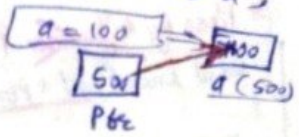
Pointers in C++

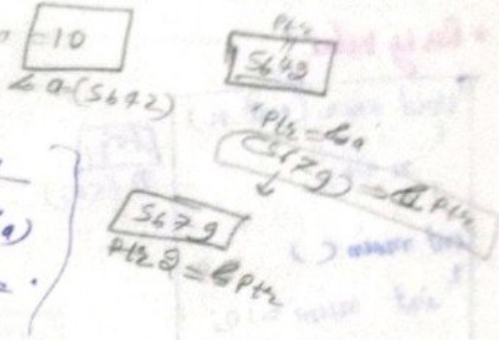


$\Rightarrow$ 4GB
$\rightarrow 2^{32}$ byte
$\rightarrow$ 32 bit address

8GB
$\rightarrow 2^{33}$ byte
33 bit $\rightarrow$ 64 bit


```
int * Ptr = &a;
```

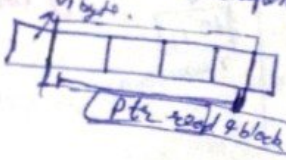




Ptr = &a
$\rightarrow$ *Ptr = *(&a)
$\Rightarrow$ a.

$\Rightarrow$ address $\rightarrow$ hexadecimal (0X fd120 ...) $\rightarrow$ store in binary form.

$\rightarrow$ Size of Ptr $\rightarrow$ depend on O.S. (Pointer size is constant)

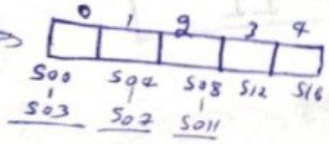


int *Ptr
4 byte.

/

char* ptr1
1 byte.

$\rightarrow$ int arr[5] $\Rightarrow$



arr - 500
$\rightarrow$ store the address of 0th index.

cout << arr + 2; $\rightarrow$ 2nd index value.

$\Rightarrow$ arr+2 $\Rightarrow$ *(arr+2)

(i th index address = Base address + i x datatype size)

arr[i] = *(arr + i)
i[arr] *(i + arr).

$\rightarrow$ // Print value.

```
for(i=0 ; i<5 ; i++)
    cout << *(arr+i) << endl;
```

// Print address

```
for(i=0 ; i<5 ; i++)
    cout << arr+i << endl;
```

$\rightarrow$

```
int * Ptr = arr;
for(i=0 ; i<5 ; i++)
{
    cout << * Ptr;
    Ptr++;
}
```

Ptr = Ptr+1
Ptr = 500 + 1 x 4
Ptr = 504 $\rightarrow$ Now Ptr changes

(increment / decrement ++ --)

$\rightarrow$ ~~arr++~~ ~~arr--~~ $\Rightarrow$ arr store address of 0th index
a++ change the address which cause data loss.

→ Pointers with char array :-

```
char arr[5] = "1234"
char * ptr = arr (address)
cout << arr;   → 1 2 3 4
cout << ptr;   → 1 2 3 4
```
(This print values till '\0')

output:

exploded
500 ✗
500 ✗

arr → 500

0	1	2	3	4
1	2	3	4	\0
500	501	502	503	504

To print address:
cout << (void*) arr;
cout << (void*) ptr;
<typecasting>

500
500

cout << static_cast<void*>(arr) << endl;

* Pass by value

```
void incr(int n)
{
  n++;
}
int main()
{
  int num = 10;
  incr(num);
  cout << num;
}
```

n (500): 10 → 11

num (800): 10

output: 10

* Pass by ~~reference~~ Pointer

```
void incr(int * ptr)
{
  *ptr++;
}
int main()
{
  int num = 10;
  incr(&num);
  cout << num;
}
```

ptr (500): 200

* ptr = value at address of ptr

num (940): 10 → 11

output: 11

→ In array.

```
void dob(int * ptr) // (int p[])
{
  for(i=0; i<5; i++)
    p[i] = 2 * p[i];
}
int main()
{
  int arr[5] = {1,2,3,4,5};
  dob(arr);
  for(i=0; i<5; i++)
    cout << arr[i]
}
```

This store address of 0th index

arr →

0	1	2	3	4
2	4	6	8	10
200	204	208	212	216

p[0] = 2 * 2 = 4

*(p+0) = 2 * [*(p+0)]
2 * 2 = 4.

p[1] = 2 * p[1]
*(p+1) = 2 * *(p+1);
2 * 4 ⇒ 8.

→ Swap: Pass by Pointer

```
void swapping(int *p1, int *p2)
{
  int temp = *p1;
  *p1 = *p2;
  *p2 = temp;
}
int main()
{
  int first = 10;
  int second = 20;
  swapping(&first, &second);
  cout << first << second;
}
```

temp (500)

p1 (300): 100

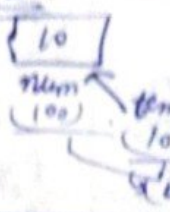
p2 (400): 200

first (100): 10

second (200): 20

→ Reference variable :- Pass by Reference

```
int num = 10;
int &temp = num
```



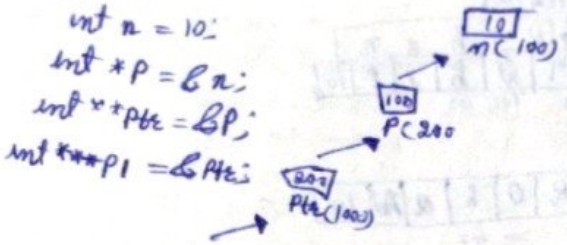

```
void Pass(Vector<int>& V1)
{
  for(i=0; i<5; i++)
      V1[i] = 10;
}
int main()
{
  Vector<int> V(5, 0);
  Pass(V);
}
```

```
void swapping(int &P1, int &P2)
{
  int temp = P1;
  P1 = P2;
  P2 = temp;
}
int main()
{
  first = 10, sec = 20;
  swapping(first, sec);
  cout << first << sec;
}
```

→ Double pointer :-

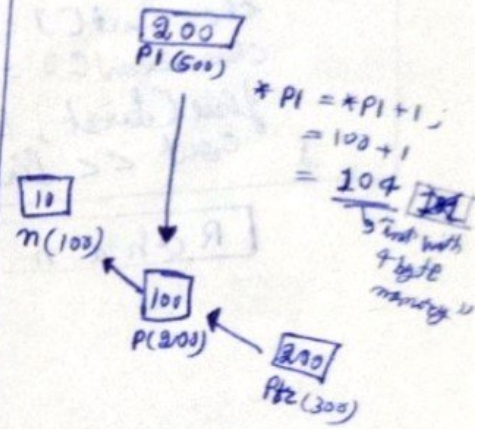
"*→ Dereference operator"

```
int n = 10;
int *p = &n;
int **ptr = &p;
int ***P1 = &ptr;
```



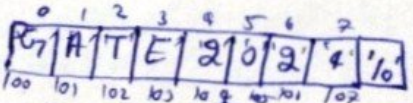
n^th Pointer = (n-1) Pointer address.

```
void fun(int **P1)
{
  *P1 = *P1 + 1;
}
int main()
{
  int n = 10;
  int *P = &n;
  int ** ptr = &P;
  fun(ptr);
  cout << P;
}
```



→

```
int main()
{
  char C[] = "GATE2024";
  char *P = C;
  cout << P + P[3] - P[1];
}
```



P + P[3] - P[1];

address + 'E' - 'A';

add ⇒ (100 + 4*1)bs 104 → 2004

output

→

```
void second(int *P1, int *P2)
{
  P1 = P2;
  *P1 = 2;
}
int main()
{
  int i=0, j=1;
  second(&i, &j);
  cout << i << j;
}
```

output: 0 2

→

```
int *ptr;
int x = 0;
ptr = &x;
int y = *ptr;
*ptr = 1
cout << x << y;
```

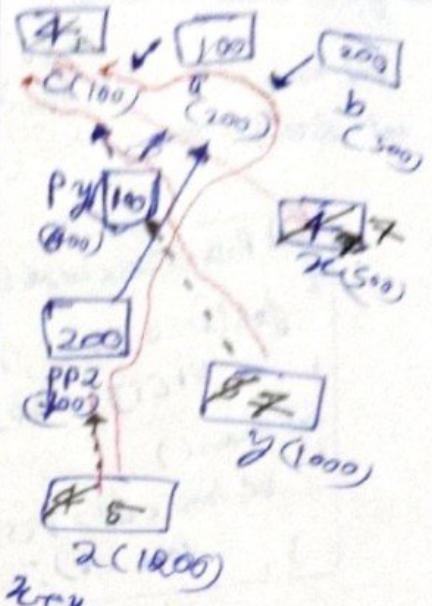
output: 1 0

```
int a = 5, b = 10;
int &name = a;
int * Ptr = &a;
(*Ptr)++;
Ptr = &b;
*Ptr = *Ptr + 5;
name += 5;
cout << a << b.
```

output: 11 15

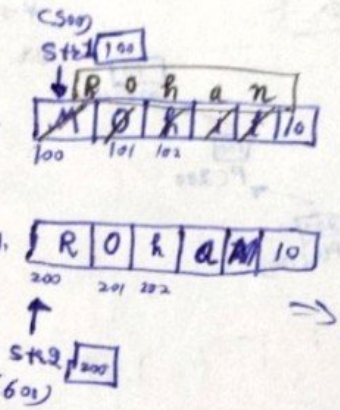
```
int four(int x, int *py, int **ppz)
{
    int y, z;
    **ppz += 1; z = **ppz;
    *py += 2; y = *py;
    x += 3;
    return x+y+z;
}
int main()
{
    int c, *b, **a;
    c = 4, b = &c, a = &b;
    cout << four(c, b, a);
}
```

output: 19



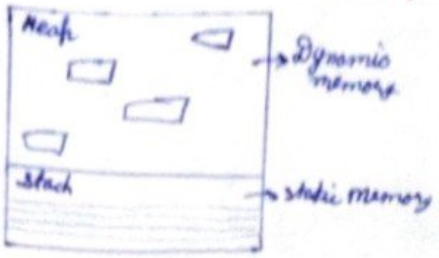

```
void fine(char *str1, char *str2);
{
    while((*str1 = *str2))
    {
        str1++; str2++;
    }
}
int main()
{
    char first[] = "Mohit";
    char second[] = "Rohan";
    fine(first, second);
    cout << first;
}
```

Rohan



⇒ String Copycode

[Memory Management]




```
int n;
cin >> n;
int arr[n];
```

```
vector<int> v(n);
```

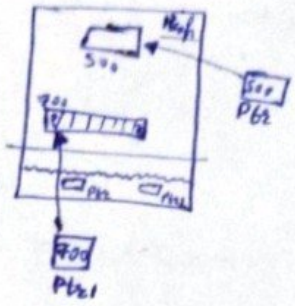
```
int main()
{
  f1()
}
```

store in stack

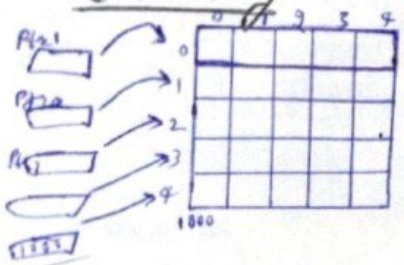
→

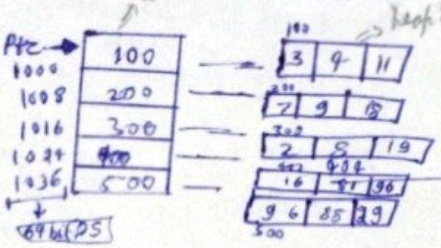
```
int main()
{
  int * Ptr = new int;
  int * Ptr1 = new int[20];
  delete Ptr;
  delete [] Ptr1;
}
```

delete



→ 2-D Array :-





Heap 2D array ⇒ use double Pointer (int * * ptr)

(*ptr+2) = ptr[2]

((Ptr+2)+1) ≡ Ptr[2][1].

*(300+1) → *(404)

→ 81.

create 2D array → (base of array)

```
int ** Ptr = new int*[n];
for(i=0; i<n; i++)
  Ptr[i] = new int[m];
```

n = rows
m = columns.

fill values →

```
for(i=0; i<n; i++)
  for(j=0; j<m; j++)
    cin >> Ptr[i][j];
```

fill values.

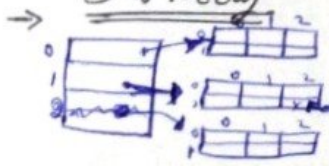
release ~~delete~~ memory. →

```
for(i=0; i<n; i++)
{
  delete [] Ptr[i];
}
delete [] Ptr;
```

Heap 2D array

stack array

3 D Array



Ptr[2][1][3]; → (*(*Ptr+2)+1)+2)

L B H.

(triple Pointer)

```
int *** Ptr = new int**[L]
for(i=0; i<L; i++)
{
  Ptr[i] = new int*[B];
  for(j=0; j<B; j++)
    Ptr[i][j] = new int[H];
}
```

(create 2D array)

Recursion

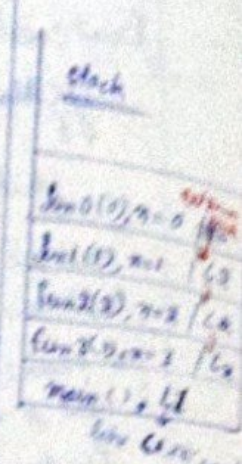
* A function which call itself again and again.
* Until a specific condition met.

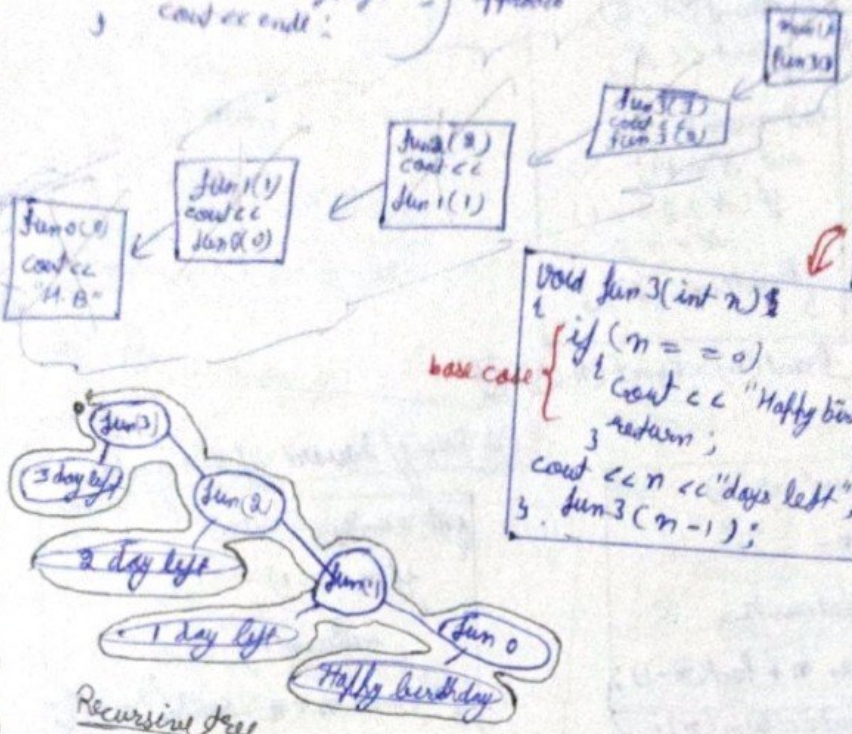
```
for (i = 5; i >= 0; i--)
{
    cout << i << "days left";
}
cout << endl;
```

} iterative approach

```
void fun3(int n)
{
    cout << n << "day left";
    fun2(n-1);
}
void fun2(int n)
{
    cout << n << "day left";
    fun1(n-1);
}
void fun1(int n)
{
    cout << n << "days left";
    fun0(n-1);
}
void fun0(int n)
{
    cout << "Happy birthday";
}
int main()
{
    fun3(3);
}
```

Stack






```
void fun3(int n)
{
    if (n == 0)                          // base case
    {
        cout << "Happy birthday";
        return;
    }
    cout << n << "days left";
    fun3(n-1);
}
```

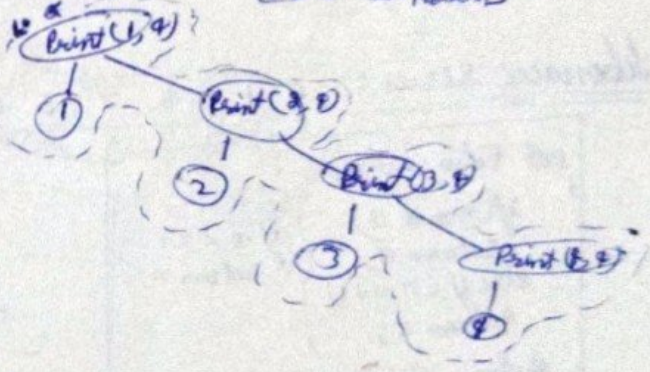
base case.

iterative way = = Recursive way

Recursive tree

→ Recursion : Print 1 to N :

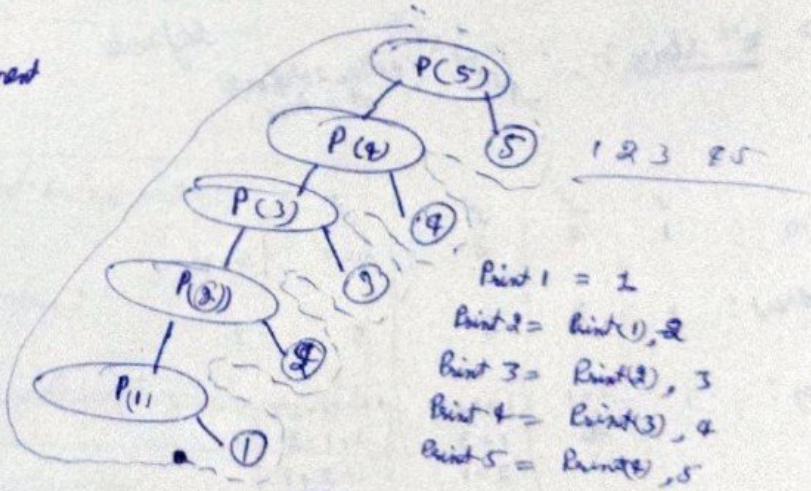
```
void Print(int num, int N)
{
    if (num == N)
    {
        cout << num;
        return;
    }
    cout << num << endl;
    Print(num+1, N);
}
int main()
{
    int N;
    Print(1, N);
}
```



Print(num, N) = num, Print(num+1, N)

1 Argument

```
void Print(int num)
{
    if (num == 1)
    {
        cout << 1;
        return;
    }
    Print(num-1);
    cout << num;
}
```



Print(num) = Print(num-1), num

→ Print even (1 - 10)

```
Print even(int num, int N)
1
   if(num > N)  // (10 > 10)
   {
      return;
   }
   cout << num;
3  Printeven(num+2, N);
int main()
2
   int N = 9;
   Printeven(2, 9);
3
```

<->

```
Printeven(int N)
1
   if(N == 2)
   {
      cout << N;
      return;
   }
   Print(N-2);
3  cout << N;
int main(){
   int N = 11;
   if(N%2 == 1)
      N--;
   Printeven(N);
3
```

Print(n) = Print(n-2), n

→ factorial

```
int fact(int n)
1
   if(n == 1)
   {
      return 1;
   }
3  return n * fact(n-1);
cout << fact(n);
```

→ Sum

```
int Sum(int n)
1
   if(n == 1)
   {
      return 1;
   }
3  return n + fact(n-1);
cout << Sum(n);
```

Sum n = n + Sum(n-1)

→ Sum of Square of N number

```
int sumSqr(int n)
1
   if(n == 1)
   {
      return 1;
   }
3  return n*n + sumSqr(n-1);
(cout << sumSqr(n);)
```

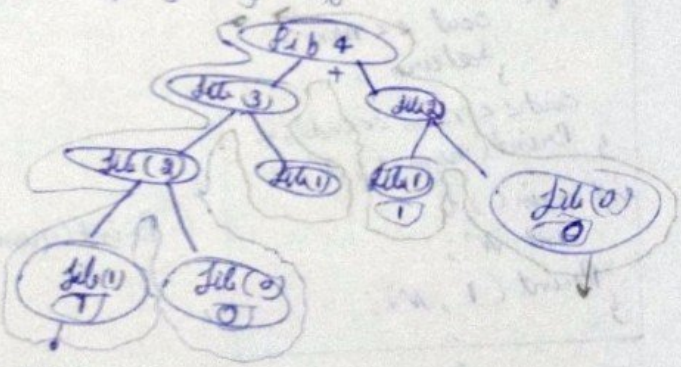
sumSq = n² + sumSq(n-1)

→ Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55

0 1 2 3 4 5 6 7 8 9 10

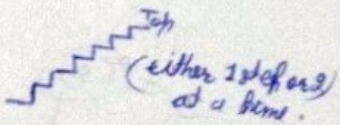
```
int Fibo(int n)
1
   if(n == 0)
      return 0;
   else if(n == 1)
      return 1;
   return Fibo(n-1) + Fibo(n-2);
3
cout << Fibo(n);
```

(if n <= 1 return n;)



leaf node

→ nth stair :-



n :	1	2	3	n=4	n=5
method :	1	2	3	5	8
way :	1	1+1 2	1+1+1 1+2 2+1	1+1+1+1 1+1+2 1+2+1 2+1 2+2	

(Total ways.)

```
int Fibo(int n)
{
   if n <= 2 return 1; else if(n == 2) return 2;
   return Fibo(n-1) + Fibo(n-2);
}
cout << Fibo(n);
```

f(n) = f(n-1) f(n-2) base case. f(1) = 1, f(2) = 2

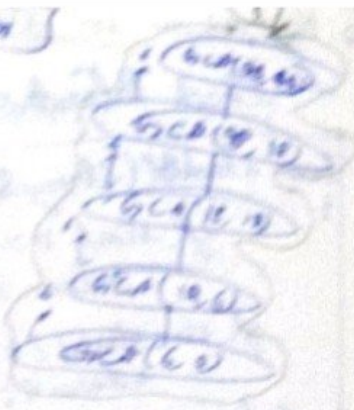
→ Greatest common Division of 2 number in { euclidean algorithm }

num 1 = 18 → 6
num 2 = 98

```
void GCD(a, b)
{
    if(b == 0)
    {
        cout << a;
        return;
    }
    GCD(b, a % b);
}
```

cout << GCD(18, 98)

(18, 98) → 98 % 18 = 8
(98, 18) → 18 % 98 = 18
(18, 18) → 18 % 18 = 0
(18, 0) → 18 % 0 = 18

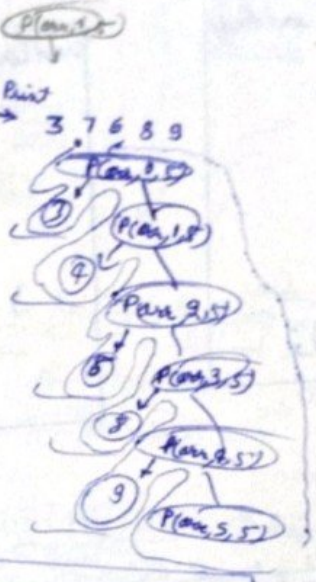


→ Recursion in Array:

→ Print Array :-

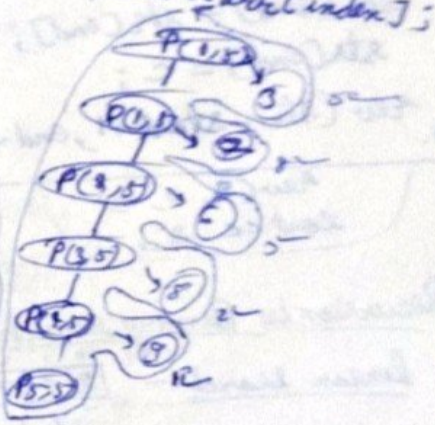
0	1	2	3	4
3	7	6	8	9

```
void Print(int arr, int index, int n)
{
    if(index == n)
    {
        return n;
    }
    cout << arr[index];
    Print(arr, index+1, n);
}
int main()
{
    int arr[] {3, 7, 6, 5, 9, -3};
    Print(arr, 0, 5);
}
```



To Print → 9 8 6 7 3

```
// Print(arr, index+1, n);
// cout << arr[index];
```



→ Sum of all index :-

```
int sum(int arr[], int index, int n)
{
    if(index == n)
        return 0;
    return arr[index] + sum(arr, index+1, n);
}
```

→ Min element :

```
int minelement(int arr[], int index, int n)
{
    if(index == n-1)
        return arr[index];
    return min(arr[index], minelement(arr, index+1, n));
}
```

→ Recursion in strings :-

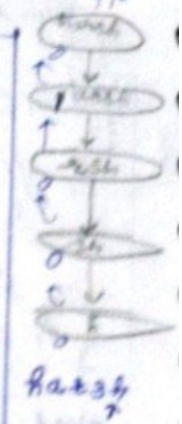
→ Check Palindrome :

```
bool checkPal(String Str, int start, int end)
{
    if(start >= end)
        return 1;
    if(str[start] != str[end])
        return 0;
    return checkPal(str, start+1, end-1);
}
```

cout << checkPal(str, 0, n-1)

→ Count vowels

```
int count(string str, index)
{
    if(index == -1)
        return 0;
    if(str[index] == 'a' ||
       ... || str[index] == 'u')
    return 1 + count(str, index-1);
    else
    return count(str, index-1);
}
```



→ Reverse a String :-

```
void rev(string &str, int start, int end)
{
    if(start >= end)
        return;
    char c = str[start];
    str[start] = str[end];
    str[end] = c;
    rev(str, start+1, end-1);
}
```

rev(s, 0, 4)
cout << s;

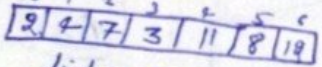
→ lower case to upper case.

```
void lowtoupp(string &S, int index)
{
    if(index == -1)
        return
    str[index] = 'A' + S[index] - 'a';
    lowtoupp(str, index-1);
}
```

string = "Rahul";
lowtoupp(str, 4)
cout << str;

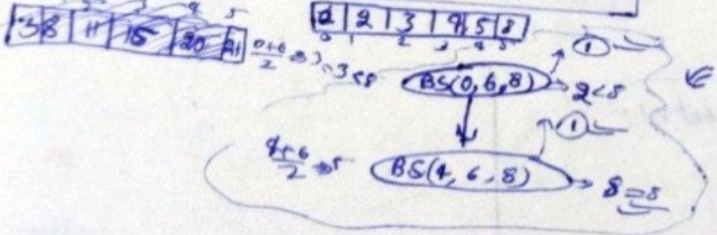
→ Recursion in Binary search :

→ linear search :-



find x = 8?

```
bool linearsearch(int arr[], int x, int index, int N)
{
    if(index == N)
        return 0;
    if(arr[index] == x)
        return 1;
    return ls(arr, x, index+1, N);
}
```



→ Binary Search

Non decreasing order
Non increasing order

```
bool BinarySearch(int arr[], int start, int end, int X)
{
    if(start > end)
        return 0;
    int mid = start + (end - start)/2;
    if(arr[mid] == X)
        return 1;
    else if(arr[mid] < X)
        return BS(arr, mid+1, end, x);
    else
        return BS(arr, start, mid-1, x);
}
```

Time Complexity in Binary Search & Recursive Function

Time taken by an algorithm to run as a function of it's input size.

```
void Print(int n)
{
    if (n == 0)
        return;
    cout << n;
    Print(n-1);
}
```

Print(5) → Print(4) → Print(3) → Print(2) → Print(1) → Print(0)

Print(n) → Print(n-1) → Print(n-2) → ... → Print(1) → Print(0)

⇒ O(n)

Print(n) = cout << n, Print(n-1)

$T_0 = 1$

$T(n) = 1 + T(n-1)$

$= 2 + T(n-2)$ ≡ $[1+1+T(n-1)]$

$= 3 + T(n-3)$

$= k + T(n-k)$

$= n + T(n-n) = 0 + T_0$

$T_n = n+1 \Rightarrow O(n)$

"Substitution Method."

$n+1 \Rightarrow O(n)$

Space Complexity

stack

Print(5), n=5
Print(4), n=4
Print(3), n=3
Print(2), n=2
Print(1), n=1
Print(0), n=0
main()

$n+1 \Rightarrow O(n)$

```
void Print(int arr[], int index)
{
    if (index == -1)
        return;
    cout << arr[index];
    Print(arr, index+1);
}
```

0 1 2 3 4 -- index = n-1

P(arr, n-1) → P(arr, n-2) → ... → P(arr, 1) → P(arr, 0) → P(arr, -1)

⇒ (n+1) time

$O(n)$ = T.C.

⇒ size of array = n+1

S.C. ⇒ $O(n)$

```
int Sum(vector<int> arr, int index, int n)
{
    if (index == n)
        return 0;
    return arr[index] + Sum(arr, index+1, n);
}
```

stack: arr, n, index+1

S(arr, 0, n) → S(arr, 1, n) → S(arr, 2, n) → ... → S(arr, n, n)

⇒ (n+1) times

⇒ T.C. ⇒ O(n)

S.C. ⇒ arr ⇒ 4n space

index ⇒ 4(n+1) space

n ⇒ 4 space

Total $(n+1)n = n^2 + n$

S.C. ⇒ $O(n^2)$

Sum(vector<int> &arr, int index, n)

(reference)

Total S.C. = O(n)

```
bool B.S(int arr[], start, end, x)
{
    binary Search
}
```

BS(arr, 0, (n-1), x) — n

BS(arr, mid+1, end, x) — n/2

BS(arr, start, (mid-1), x) — n/4

T.C. ⇒ $\log_2 n$

S.C. ⇒

⇒ $n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \frac{n}{16}, \ldots, \frac{n}{2^k}$

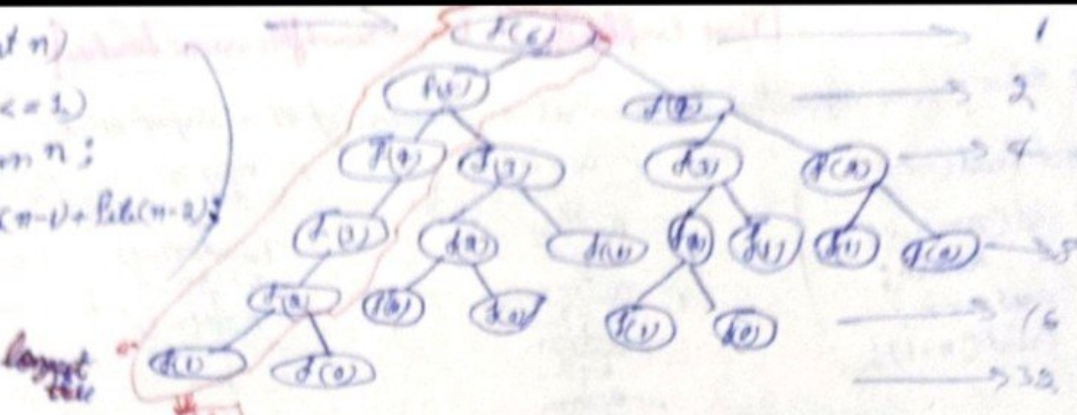
$1 + 1 + 1 + 1 + \ldots + (k+1)$ times $= \frac{n}{2^k} + 1$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$

$k \log 2 = \log n$

$k = \log_2 n$

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



$T_n = 1 + T(n-1) + T(n-2)$

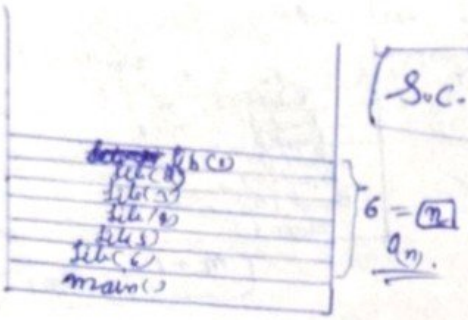
$1 + 2 + 4 + 8 + 16 + 32 \ldots$

$(2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \ldots - 2^{n-1})$.

$\Rightarrow \text{sum}(2^n - 1)$

T.C. $\Rightarrow O(2^n)$

S.C. $\Rightarrow n \Rightarrow O(n)$

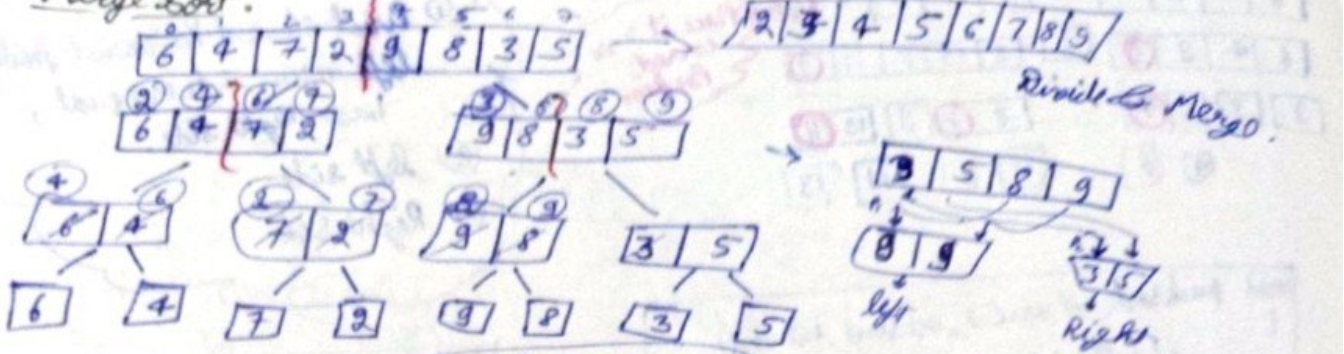


6 = T_n

$O(n)$.

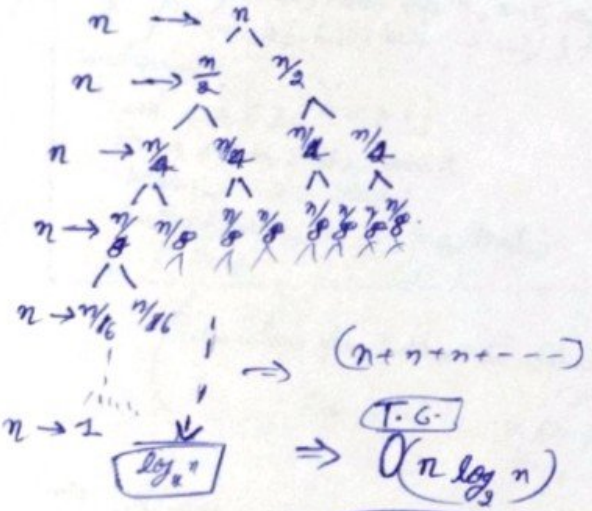
[Merge sort] Quick sort

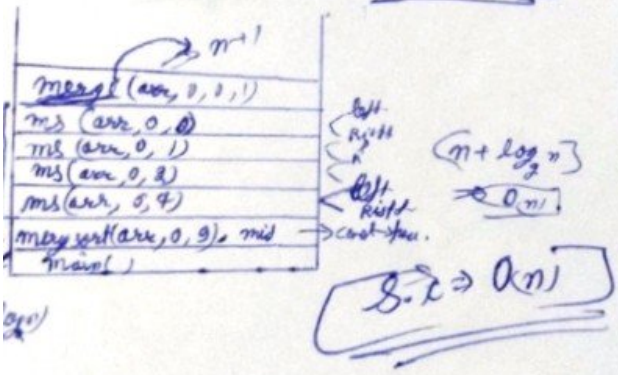
→ Merge sort :



Left (start, mid)
Right (mid, end)

```
void mergesort(int arr[], int start, int end)
{
    if (start == end)
        return;
    int mid = start + (end-start)/2;
    mergesort(arr, start, mid); // Left
    mergesort(arr, mid+1, end); // Right
    merge(arr, start, mid, end);
}
```

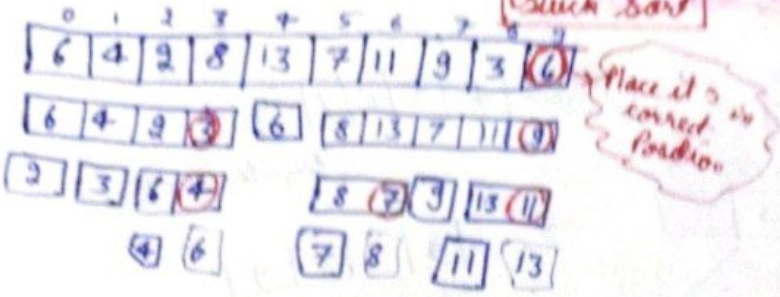




S.C ⇒ O(n)

```
void merge(int arr[], int start, int end, int mid)
{
    vector<int> temp(end-start+1);
    int left = start, right = mid+1, index = 0;
    while (left <= mid && Right <= end)
    {
        if (arr[left] <= arr[Right])
        {
            temp[index] = arr[left];
            index++, left++;
        }
        else
        {
            temp[index] = arr[Right];
            index++; Right++;
        }
    }
    // left array elements remain.
    while (left <= mid)
    {
        temp[index] = arr[left];
        index++, left++;
    }
    // Right array elements remain.
    while (Right <= end)
    {
        temp[index] = arr[Right];
        index++, Right++;
    }
    index = 0
    while (start <= end).
    {
        arr[start] = temp[index];
        start++, index++;
    }
```

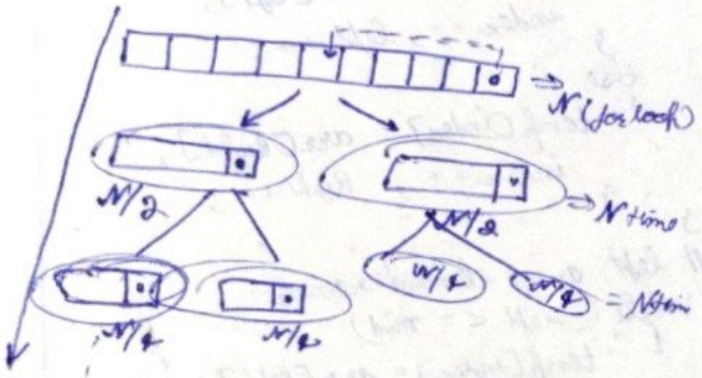
Quick Sort



① Pivot element ko correct position, left main, chote or equal, bade Right side.
② Left side
③ Right side

```
void quicksort(int arr[], int start, int end)
{
    if (start >= end)
        return;
    int Pivot = Partition(arr, start, index);
    quick sort(arr, start, Pivot - 1); //Left
    quick sort(arr, Pivot+1, end); //Right.
}
```

```
int Partition(int arr[], int start, int end)
{
    int Pos = start;
    for(i=start; i<=end; i++)
    {
        if(arr[i] <= arr[end])
        {
            swap(arr[i], arr[Pos])
            Pos++;
        }
    }
    return Pos-1;
}
```

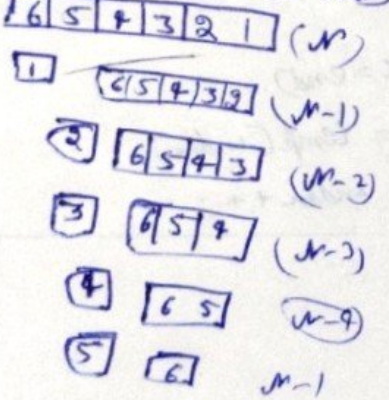


Avg ⇒ T.C. ⇒ (N log N)

log N times

= S.C. ⇒ O(log N)
Avg

⇒ Worst case. Khrab case also.

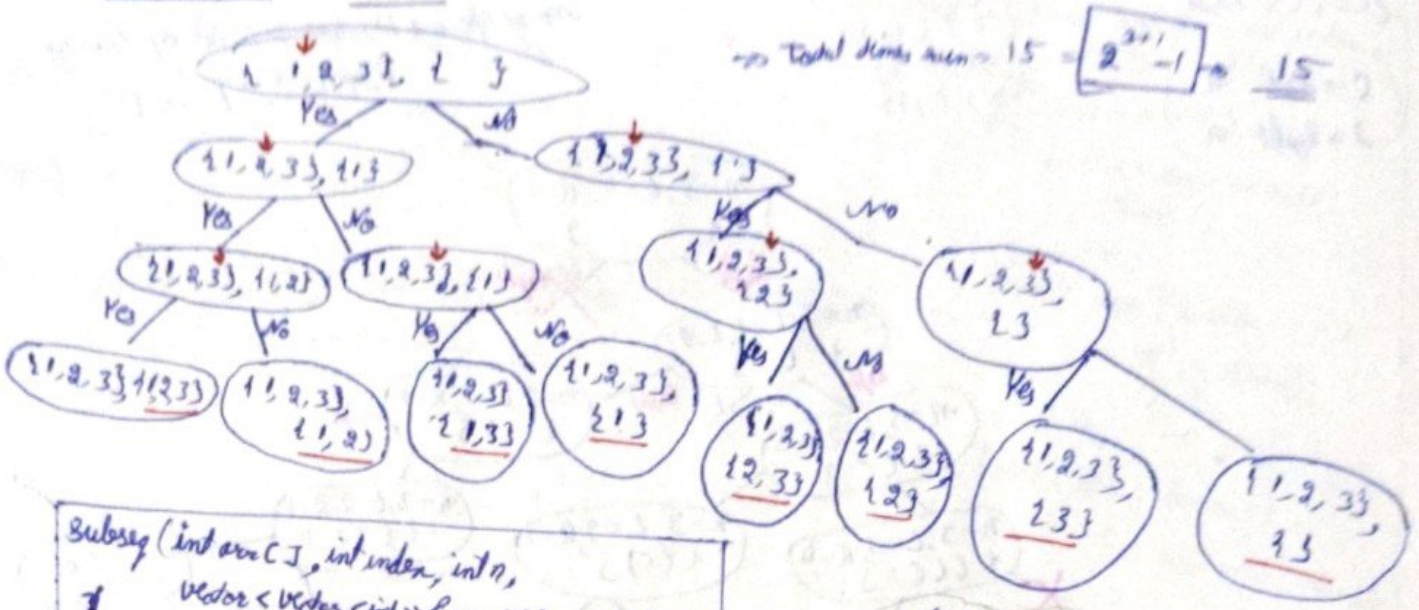


T.C. ⇒ N + (N-1) + (N-2) ---- → O(N²)

S.C. ⇒ O(N).

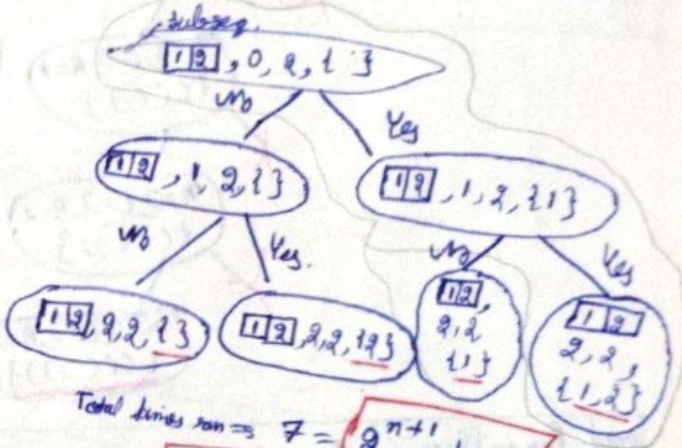
→ Subsequence : Powerset → { 1 }, { 2 }, { 3 }, { 1, 2 }, { 1, 3 }, { 2, 3 }, { 1, 2, 3 }, { } .

→ $2^3 = 8$



→ Total lines ran = 15 = $2^{3+1} - 1$ = 15

```
Subseq (int arr[], int index, int n,
        vector<vector<int>> &ans, vector<int> temp)
{
    if ( index == n)          (// vector<int> &temp)
        ans.push_back(temp);
    return;
// No → Subseq(arr, index+1, n, ans, temp);
// Yes → { temp.push_back(arr[index]);
    Subseq(arr, index+1, n, ans, temp);
}   (// temp.pop_back(arr[index]); } → (& temp)
int main()
{
    int arr[] = { 1, 2 };
    vector<vector<int>> ans;
    vector<int> temp;
    Subseq(arr, 0, n, ans, temp);
}  // Print
```



Total lines ran → $7 = 2^{n+1} - 1$

T.C. → $O(2^n)$

$O(n^2)$

→ S.C. → { 1 }, { 2 }, { 1, 2 }, { } → $2^n * n$ (n = 2)

→ $(2^n * n + n^2)$

→ S.C. → $O(2^n * n)$

auxiliary space : space except input & output

↳ temp → & temp → n (ans)

temp → n^2 (ver)

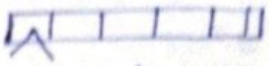
→ String

→ Generate Parentheses :-

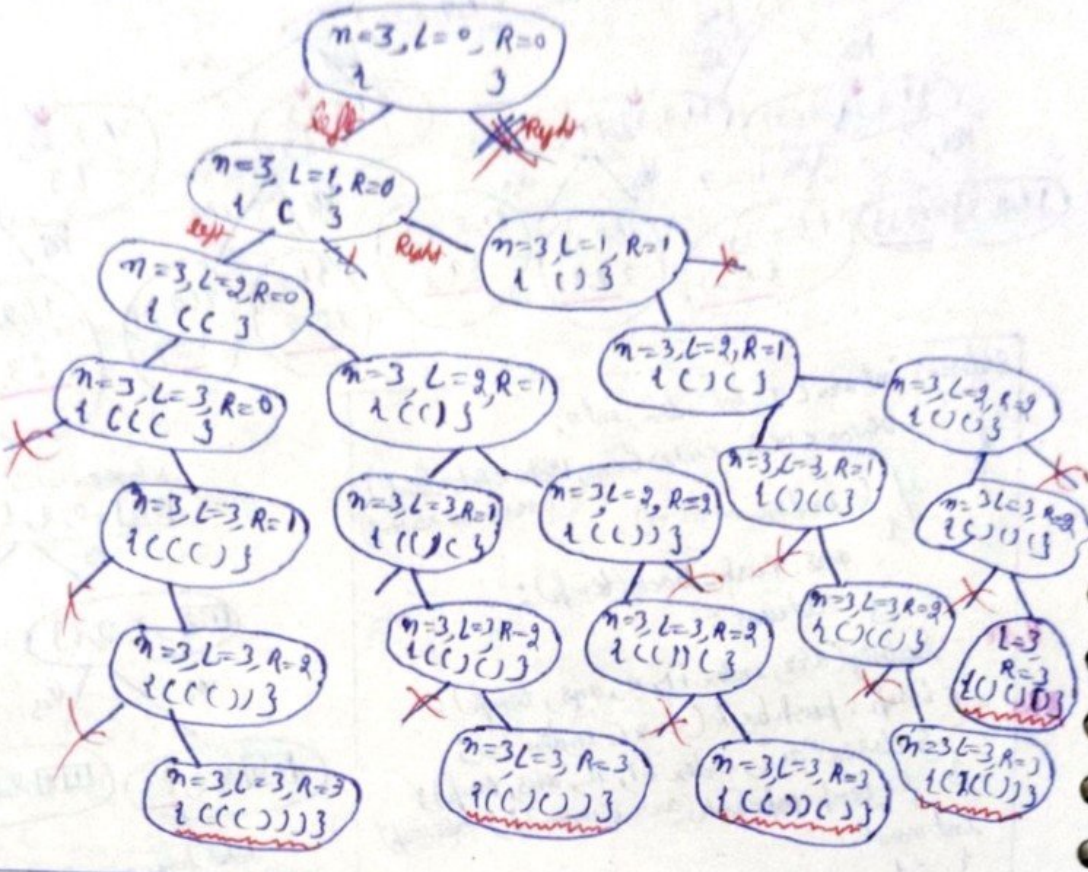
n = 2
{ (), (()) }

((→ left n
) → Right n)

n = 3
{ ((())),
(()) (),
() (()),
(()) (),
() (()) }

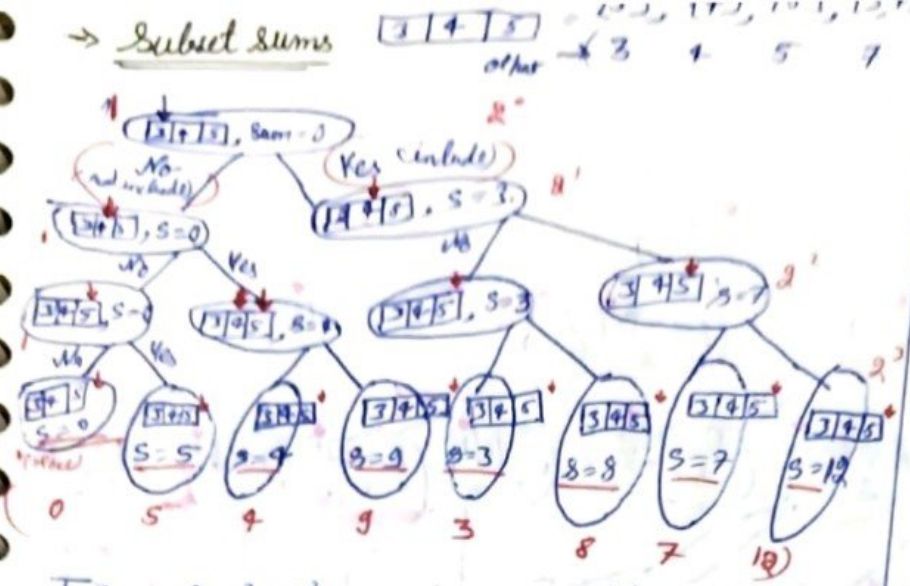


no of opening bracket >= No of closing
R > L
L >= R




```
void Parentheses (int n, int left, int Right,
                  Vector<string> &ans, string &temp)
{
  if (left + Right == 2*n)
  {
      ans.Push_back (temp);
      return;
  }
  if (left < n)  // left Parantheses
  {
      temp.push_back ('(');
      Parentheses (n, left+1, Right, ans, temp);
      temp.Pop_back ();
  }
  if (Right < left)  // Right Parantheses.
  {
      temp.Push_back (')');
      Parentheses (n, left, Right+1, ans, temp);
      temp.Pop_back ();
  }
}
```

→ Subset Sums




```
void Print(int arr[], int index,
           int n, int sum)
{
  if (index == n)
  {
    cout << sum << endl;
    return;
  }
  // No
  Print(arr, index+1,
        n, sum);
  // Yes
  Print(arr, index+1,
        n, sum+arr[index]);
}
```

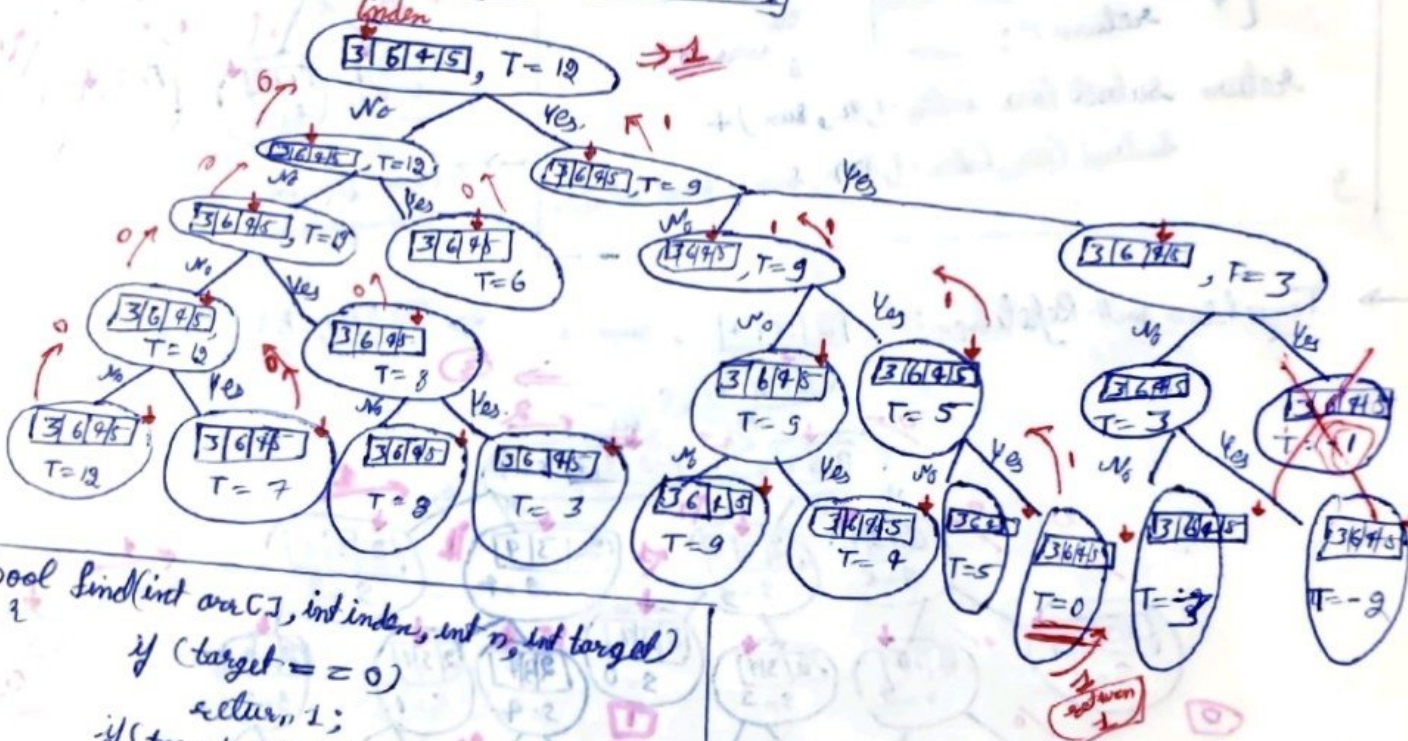
T.C. ⇒ $2^0 + 2^1 + 2^2 + 2^3 + \dots = 2^{n+1} - 1$

$= O(2^n)$

S.C. ⇒  ⇒ $O(n)$.

→ Target Sum:

3 6 4 5 Target = 12

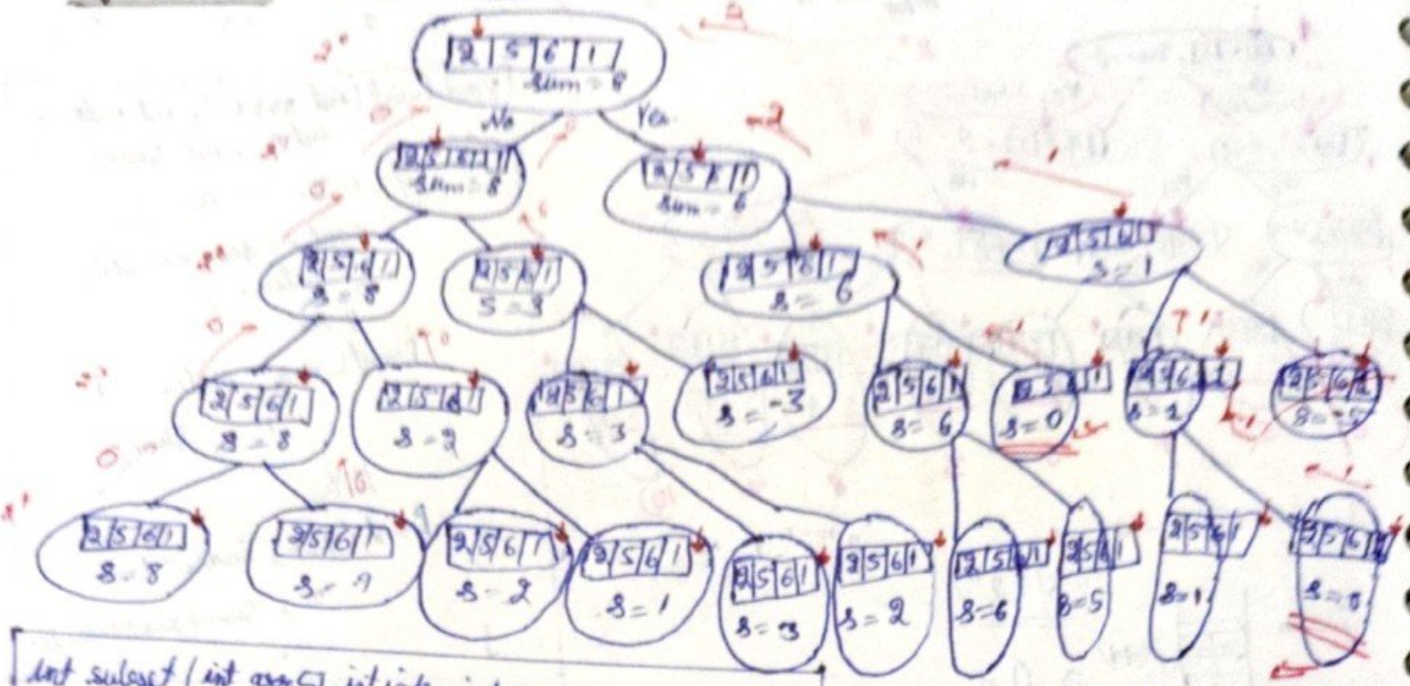



```
bool find(int arr[], int index, int n, int target)
{
  if (target == 0)
    return 1;
  if (target < 0 || index == n)
    return 0;
  return find(arr, index+1, n, target) ||
         find(arr, index+1, n, target
              - arr[index]);
}
```

T.C. ⇒ 2^n ⇒ $O(2^n)$.

S.C. ⇒ $O(n)$

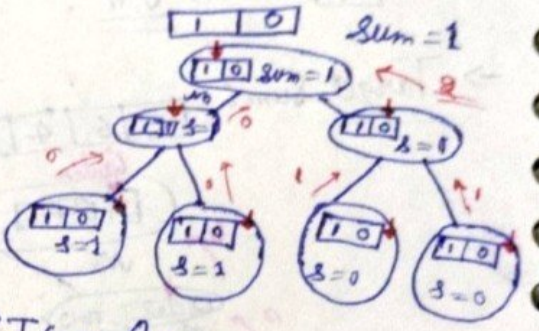
→ Perfect Sum :- [2 | 5 | 6 | 1] sum = target = 8




```
int subset (int arr[], int index, int n, int sum)
{
    if (sum == 0)
        return 1;
    if (sum < 0 || index == n)
        return 0;

    return subset (arr, index+1, n, sum) +
           subset (arr, index+1, n, sum - arr[index]);
}
```

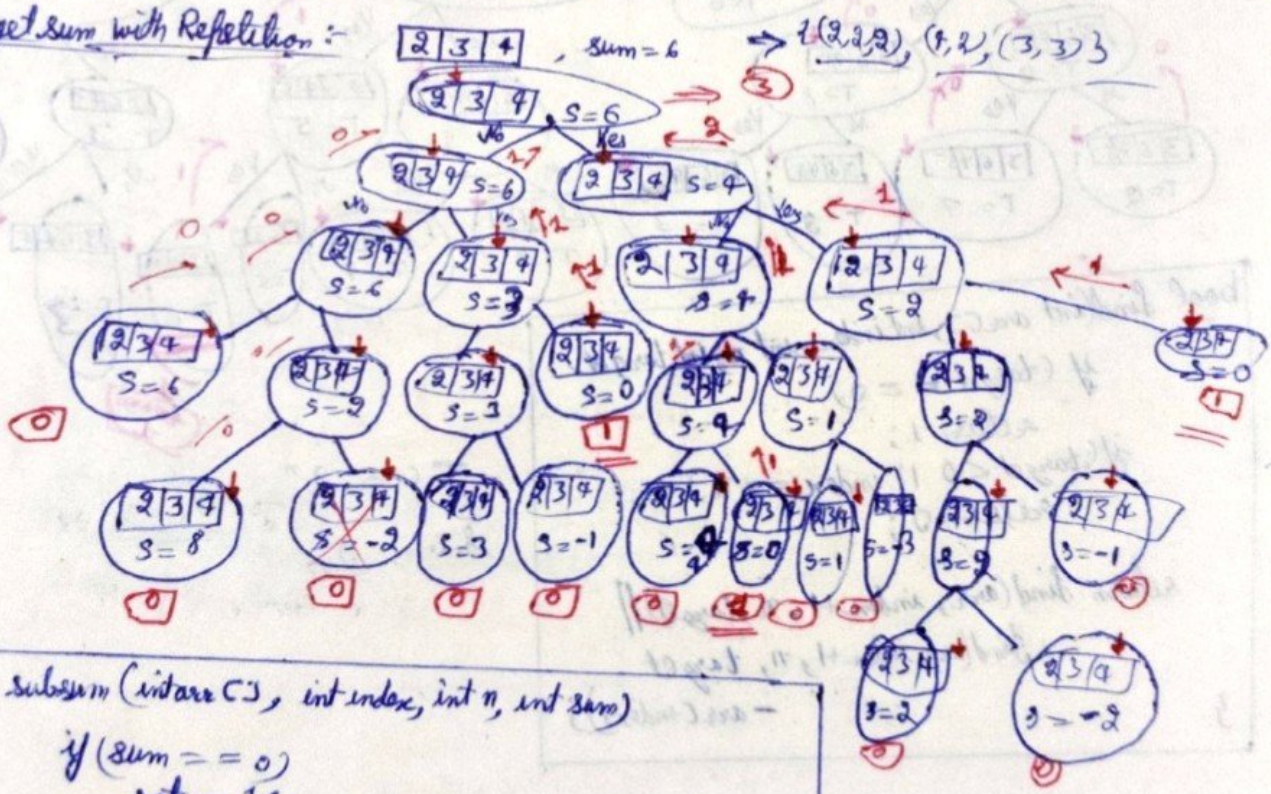
```
if (index == n)
{
    if (sum == 0)
        return 1;
    else
        return 0;
}
```

// if only arr[index] = 0



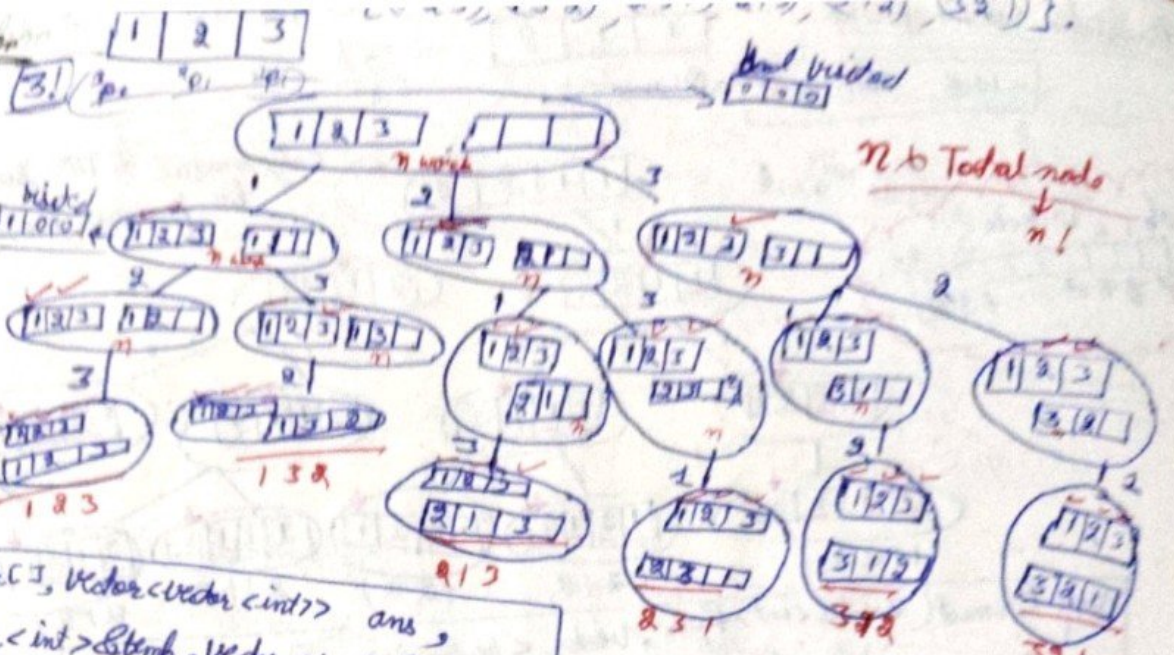
→ T.C. → $O(2^n)$

→ Target sum with Repetition :- [2 | 3 | 4] , sum = 6 → {(2,2,2), (4,2), (3,3)}




```
int subsum (int arr[], int index, int n, int sum)
{
    if (sum == 0)
        return 1;
    if (index == N || sum < 0)
        return 0;
    return subsum (arr, index+1, n, sum) +
           subsum (arr, index, n, sum - arr[index]);
}
```

→ Permutation



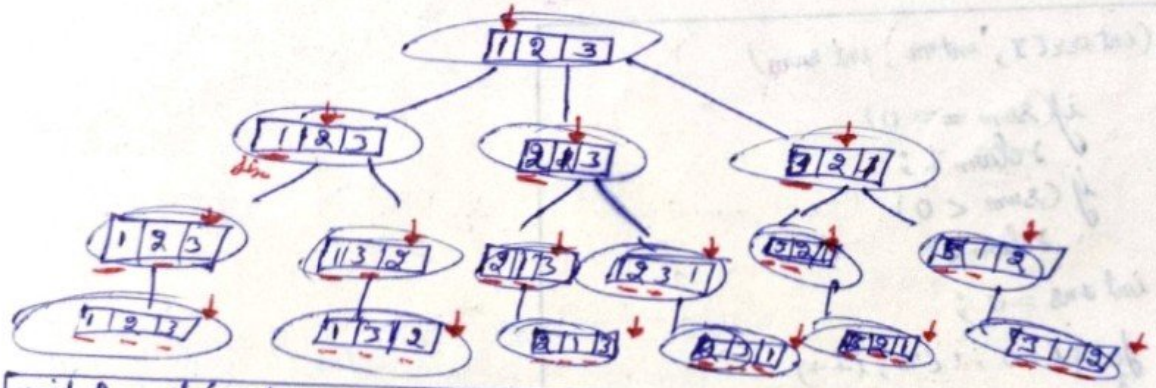

```
Permut(int arr[], vector<vector<int>> ans,
       vector<int> &temp, vector<bool> &visited)
{
    if (visited.size() == temp.size())
    {
        ans.push_back(temp);
        return;
    }
    for(int i=0; i<visited.size(); i++)
    {
        if (visited[i] == 0)
        {
            visited[i] = 1;
            temp.pushback(arr[i]);
            Permut(arr, ans, temp, visited);
            visited[i] = 0;
            temp.pop_back();
        }
    }
}
```

n work

$= n! \Rightarrow n!$ = total nodes.

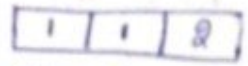
T.C. $\Rightarrow O(n \times n!)$

S.C. $\Rightarrow O(n)$ → ans (vector) / n + n + n / stack



```
void Permut (vector<int> &arr, vector<vector<int>> &ans, int index)
{
    if (index == arr.size())
    {
        ans.push_back(arr);
        return;
    }
    for(int i = index; i < arr.size(); i++)
    {
        swap(arr[i], arr[index]);
        Permut(arr, ans, index+1);
        swap(arr[i], arr[index]); // return to back position
    }
}
```

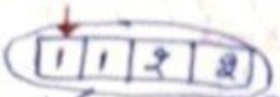
Permutation with repetition



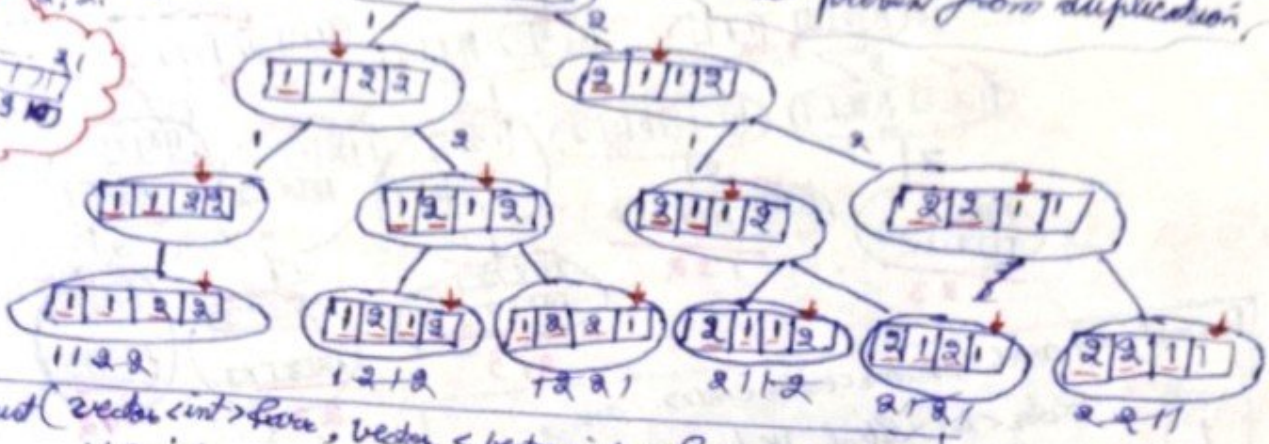
$\frac{3!}{2!} = 3$

$-10 \le num \le 10$ → 21 numbers

$\frac{4!}{2!2!} = \frac{24}{4} = 6$

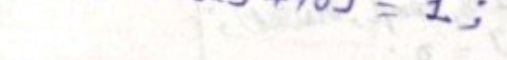


⇒ Only unique 6 will swap because back to prevent from duplication.



```
void Permut(vector<int>&arr, vector<vector<int>>&ans, int index)
{
    if(index == arr.size()-1)
    {
        ans.push_back(arr);
        return;
    }
    vector<bool> Track(21,0);
    for(i=index; i<arr.size(); i++)
    {
        if(Track[arr[i]+10] == 0)
        {
            swap(arr[index], arr[i]);
            Permut(arr, ans, index+1);
            swap(arr[index], arr[i]);
            Track[arr[i]+10] = 1;
        }
    }
}
```

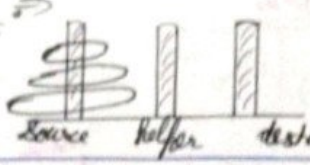
→ Way to Sum N:



{ 1+1+1+1+1+1 , 1+1+5, 1+6, 1+5+1, 5+1+1, 6+1 }

```
int way(int arr[], int m, int sum)
{
    if(sum == 0)
        return 1;
    if(sum < 0)
        return 0;
    int ans = 0;
    for(i=0; i<m; i++)
    {
        ans += way(arr, m, sum - arr[i]);
    }
    return ans;
}
```

→ Tower of Hanoi :-



① 1 baar me 1 hi disk move kar sakte ho, only top wali.
② Choti disk ke upar badi disk nahi rakh sakte.

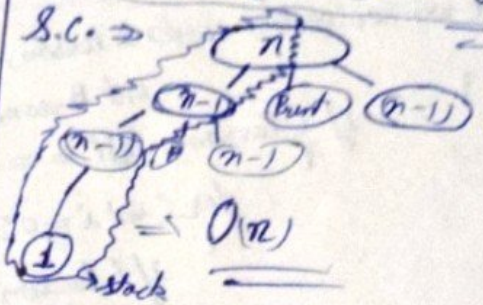
```
void Toh(int n, int sour, int help, int dest)
{
  ① if(n==1)
    {
      cout << "Move disk" << n <<
              "from" << sour << "to" << dest;
      return;
    }
  ② Toh(n-1, sour, dest, help);
  ③ cout << "Move disk" << N << "from" << sour
          << "to" << "dest";
  ④ Toh(n-1, help, sour, dest);
}
```

n = 3 ⇒ 7 steps

n = 4 ⇒ 15 steps

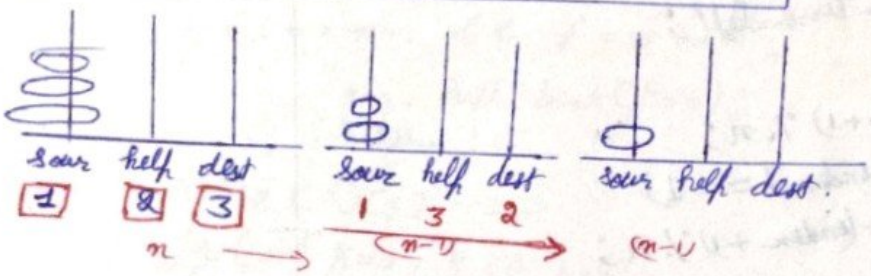
T.C. ⇒ $(2^n - 1) = O(2^n)$

S.C. ⇒

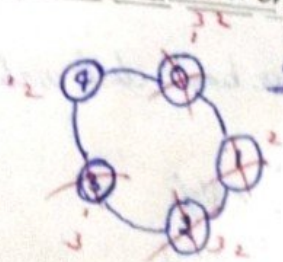


$= O(n)$

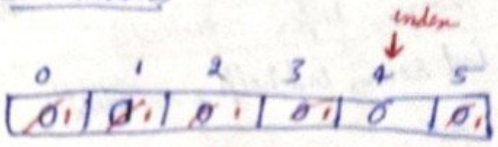
① → last one.
② → first execute till n = = 1)
③
④ → after ② till n = = 1).



→ Predict the winner :- Josephus Problem



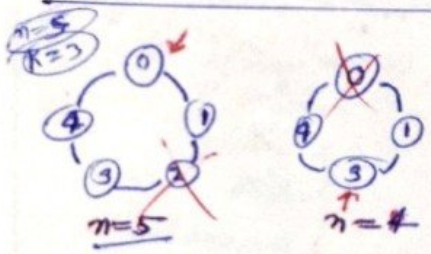
n = 5 , K = 3 (eleminate)

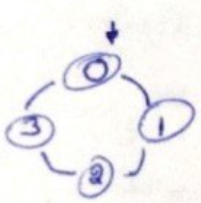


index = (index + 1) % n.

```
int winner (vector <bool> &Person, int n, int index,
            int Person_left, int K).
{
    if (Person_left == 1)
    {
        for (int i=0; i<n; i++)
        {
            if (Person[i] == 0)
                return i;
        }
    int kill = (K-1) % Person_left;
    while (kill --)
    {
        index = (index+1) % n;
        while (Person[index] == 1)
            index = (index+1) % n;
    }
    Person[index] = 1;
    while (Person[index] == 1)
    {
        index = (index+1) % n;
    }
    return Winner (Person, n, index, Person_left-1, K);
}
```

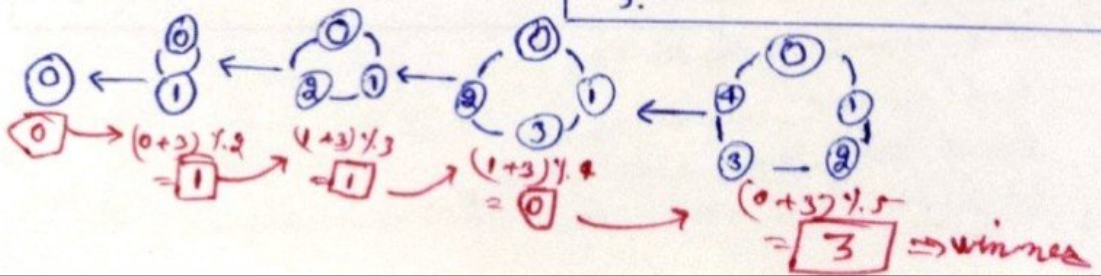
→ T.C. = O(n²).
S.C. = O(n).



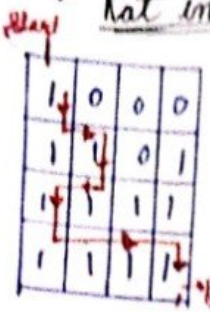



```
int winner (int n, int k)
{
    if (n == 1)
        return 0;
    return (Winner(n-1, k)+K) % n;
}
```

→ T.C. = O(n).
S.C. = O(n).



→ Rat in a Maze :



n (row, column)

up : U
Down : D
Left : L
Right : R

① Don't go outside matrix.
② Only go with 1 and with 0.
③ Never visit back to visited box again.

Path : DRDLDRRR
DRDDRDR } 2 Paths

(i, j)

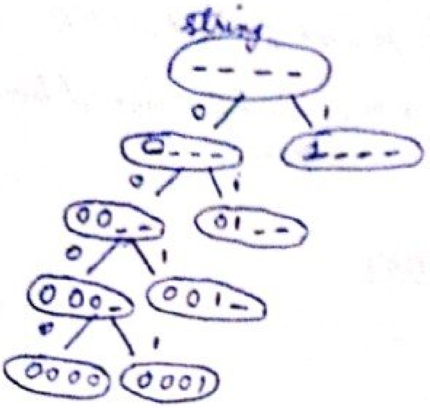
	row, col
up : i-1, j.	(-1, 0)
Down : i+1, j.	(1, 0).
left : i, j-1.	(0, -1).
Right : i, j+1.	(0, 1).

```
void Total(vector<vector<int>>& matrix, int i, int j,
           int n, string Path, vector<string>& ans, vector<vector<bool>>& visited)
{
    if (i == n-1 && j == n-1)
    {
        ans.push_back(Path);
        return;
    }
    visited[i][j] = 1;
    for (int K=0; K<4; K++)
    {
        if (valid(i+row[K], j+col[K], n) && matrix[i+row[K]][j+col[K]]
            && !visited[i+row[K]][j+col[K]])
        {
            ~~if(valid(i+row[K], j+col[K], n) && matrix[i+row[K]][j+col[K]] && visited[i~~
            Path.push_back(dir[K]);
            Total(matrix, i+row[K], j+col[K], n, Path, ans, visited);
            Path.pop_back();
        }
    }
    visited[i][j] = 0;
}
```

```
bool valid(int i, int j, int n)
{
    return i>=0 && j>=0 && i<n && j<n;
}
```

T.C. = 3^{N^2}

⇒ N - Bit binary Number : find all N bit binary numbers having ... equal to 1 than 0 for any prefix.



1 1 0 1

one ≥ zero

zero 0 1

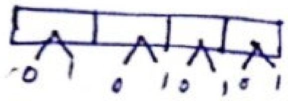
one ~~0 1 2~~ 3

1 0 0 0

zero 0 1 2

one 0 1

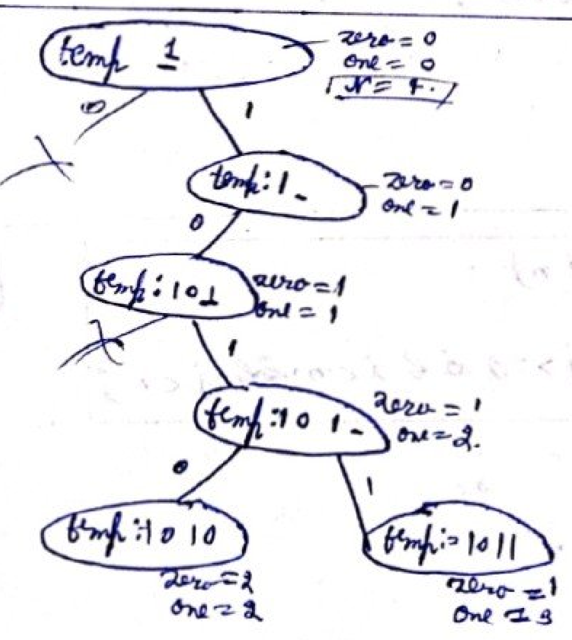
zero > one



zero : 0
one : 0

if (zero <= one)

```
void find(int N, vector <string> &ans, string& temp,
          int zero, int one)
{
    if(temp.size() == N)
    {
        ans.push_back(temp);
        return;
    }

    if(zero < one)
    {
        temp.push_back('0');
        find(N, ans, temp, zero+1, one);
        temp.pop_back();
    }
    temp.push_back('1');
    find(N, ans, temp, zero, one+1);
    temp.pop_back();
}
```



Object Oriented Programming

It is an approach or Programing pattern where the programs are structured around objects rather than function and logic.

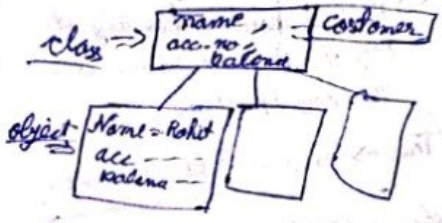
```
class student → Data Type (user define)
{
  public:
    string name;
    int age, roll_no;
    string grade;
};
int main
{
    student S1; → object
    S1.name = "Harsh";
    S1.age = 10;
    S1.roll_no = 2139;
    S1.grade = "A+";
}
```

→ Private : (Default)
Public :
Protected :

```
class student
{
  Private:
  string name;
  int age, rollno;
  // function → setter or getter.
  Public:
  void setname(string n)   ← function
  {
    name = n;
  }
  void getname()
  {
    cout << name << " ";
  }
int main()
{
  student S1;
  S1.setname("Harsh");
  S1.getname();
```

→ Class : • It is user defined data type.
• Blue print for created object.

→ Object : • It is an entity that has a state and Behaviour.
• Anything that exist in Physical World.

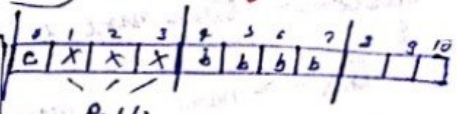


Empty class size = 1

* Padding

```
class a
{
  char c;
  int b;
};
int main()
{
  a obj
  cout << sizeof(obj);
}
```

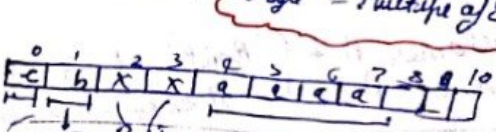
8



32 bit OS
64 bit OS
→ 4 byte

1 byte = Multiple of 1.
2 byte = Multiple of 2.
4 byte = Multiple of 4
8 byte = Multiple of 8.

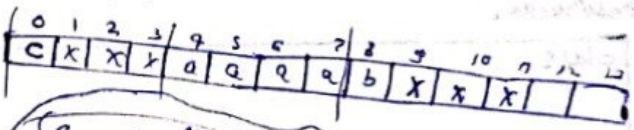
```
char c; → 1
char b; → 1
int a;  → 4
```



⇒ 8 byte.

```
char c; → 1
int a;  → 4
char b; → 1
```

→ 4 byte



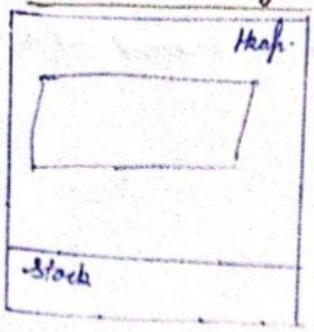
Greedy allignment → Sabse bada Sabse upper

⇒ 12 byte.

but 9 does not divide 4 completly so data divisible of 4 nearest by increasing 9× → 12 ✓

Correcty allignment : Biggest byte data type upper

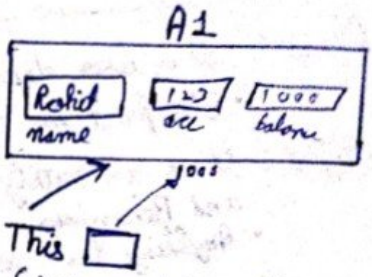
→ Static vs Dynamic Memory Allocation:




```
Student *s = new Student;
(*s).name = "Harsh";
s -> name = "Harsh";
```

* **Constructor**: It is a special fx^n that is invoked automatically at the time of object creation.
  - Name of the construction should be same as class name.
  - It doesn't have any return type.
  - It is used to initilize the value.

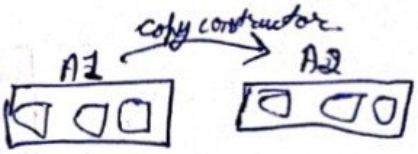
```
Class customer
{
    string name;
    int account_number;
    int balance;
    customer ()    // Default construction.
    {
      cout << "Construction is called";
    }

    // Parameterized construction.
    customer (string name, int account_number, int balance)
    {
      this -> name = name;
      this -> account_number = account_number;
      this -> balance = balance;
    }
};

int main()
{
      customer A1 ("Rohit", 123, 1000);
      customer A2(A1) // copy constructor
}
```



() this is a pointer pointing a object A1 & store it's memory location("1001")

This -> name ≡ (*this). name.

```
customer (string a, int b, int c)
{
    name = a
    account_number = b
    balance = c
}
```



display the objects

```
Void display()
{
    cout << name << acc_no << balance;
}
int main()
{
    customer A1();
    A1.display();
}
```

A1 → A2

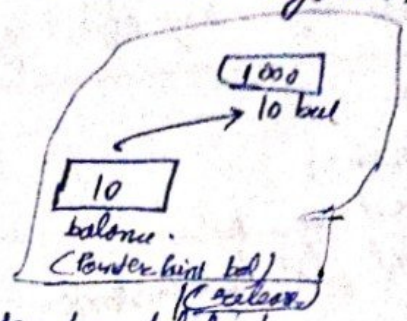
Copy Constructor

```
class - -
{

    // copy constructor.
    customer (customer &B)
    {
      name = B.name;
      account_number = B.account_number;
      balance = B.balance;
    }
};
int main()
{
      customer A2 (A1)
}
```

Copy Constructor :-

* Destructor :- It is an instant member fx" that is invoked automatically whenever an object is going to be destroyed.
  - it is a last fx" that is going to be called before an object is destroyed.

```
class customer
{
    string name;
    int * balance;
  public:
  customer (string name, int bal)
  {
    this -> name = name;
     balance = new int;
    * balance = bal;
  }
~ customer ()                    // destructor
    {
        delete balance;
    }
};
int main ()
{
  customer A1("Rohit", 1000);
}
```



destructor delete dynamic memory "balance".
(memory used in heap) → release heap memory.

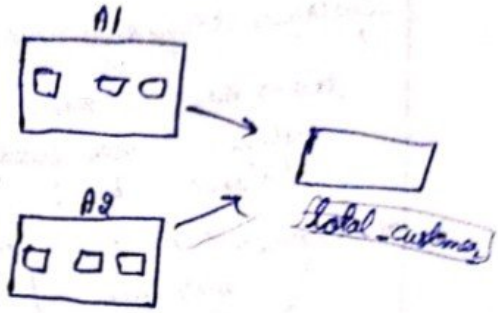
→ compiler automatic made destructor if user can't.
→ destructor created only once.

constructor call order wise objects.
but destructor call reverse order objects.

* Static data member

- They are attribute of class or class member.
- it is declared using static keyword.
- only copy (one) of that member is created for the entire class & is shared by all the class
- It is initialized before any object of this class is created.

```
class customer
{
    string name;
    int account_no, balance;
    static int total_customer;
    public:
        customer (string a, int b, int c)
        {
            name = a;
            acc_no = b;
            balance = c;
            total_customer ++;
        }
};

int customer :: total_customer = 0;
int main()
{
    customer A1("Rohit", 1, 1000);
    customer A2("Mohit", 2, 200);
}
```



// const -

* Encapsulation : Wrapping up of data & information in a single unit, while controlling access to them.

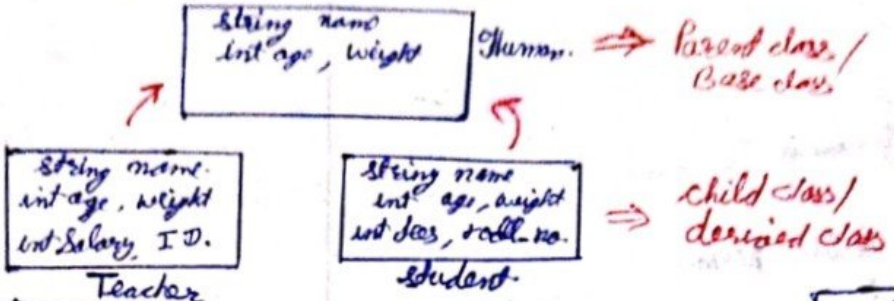
- data hiding

```
class customer
{
    string name;
    int balance, age;
    public:
    customer(string a, int b, int c)
    {
        name = a;
        balance = b;
        age = c;
    }
    void deposit(int amount)
    {
        if (amount > 0)
            balance += amount;
        else
            cout << "Invalid amount";
    }

int main() {
    customer A1("Hasan", 5000, 19);
    deposit(10000);
}
```

→ to secure data, data should be valid given by the user.

Abstraction :- Display only essential information & hiding the details.

* Inheritance :- The capability of a class to derive property & characteristic from another class.



	external class	within class	derived class
Public :	✓	✓	✓
Protected :	X	✓	✓
Private :	X	✓	X

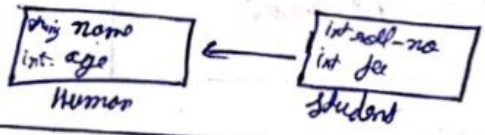
```
class
Private :
    int a ;
Protected
    int b ;
Private :
    int c ;
```

→ (class student : access_modifier Human
{
};)
(Public / Private / Protected)

Base class	child class	
Public	Public	Public
Public	Protected	Protected
Public	Private	Private.
Protected	Public	Protected
Protected	Protected	Protected
Protected	Private	Private.

Type of Inheritance :

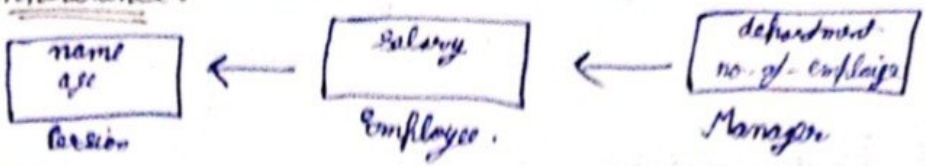
① Single Inheritance :




```
class Human
{   public :
    string name ;
    int age, weight ;
    // constructor
    Human ( )
    {
        cout << "Hello Human \n";
    }
};
class student : public Human
{
        int roll_no, fees ;
        public :
    student ( ) // constructor
    {
        cout << "Hello Student \n";
    }
};
int main ( )
{
        student A1 ;
}
```

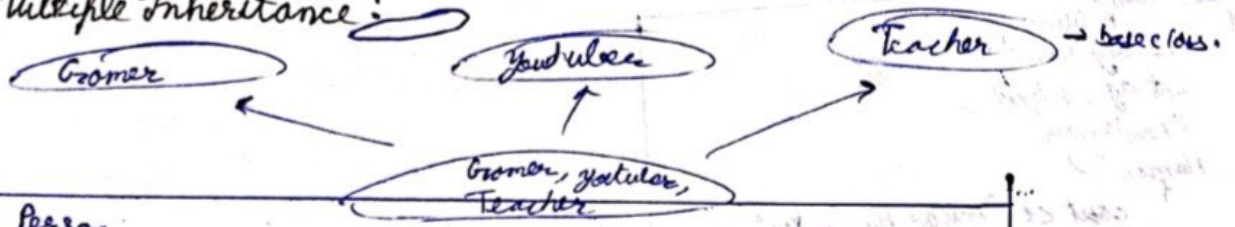
Output ⇒ (Hello Human
Hello Student)

* Multilevel Inheritance:



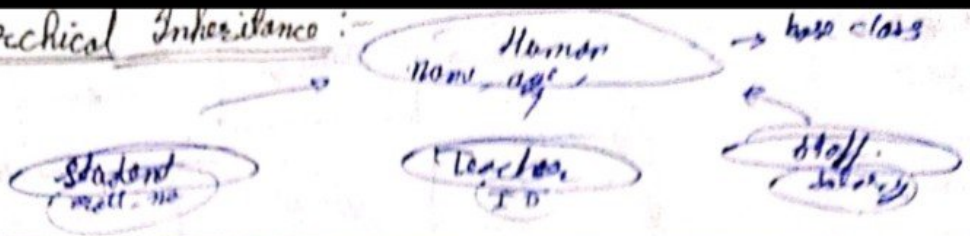

```
class Person
{
    Protected :
     string name;
    Public :
    void intro()
    {
      cout << "My name is " << name ;
    }
};
class Employee : Public Person
{
      Protected :
      int salary ;
      Public :
        void monthly_sal()
        {
          cout << "My monthly salary is " << salary ;
        }
   };
class Manager : Public Employee.
{
      Protected :
string ~~int salary~~ department;
      Public :
      manager (string name, int salary, string department)
      {
        this → name = name;
        this → salary = salary;
        this → department = department;
      }
};  display
```

③ Multiple Inheritance:



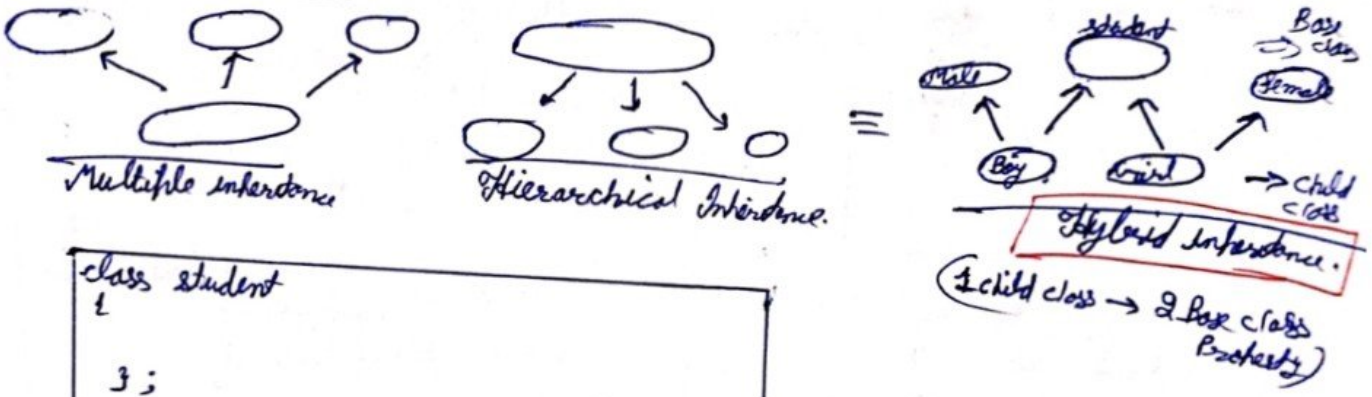

```
class Person
{  Public :
     string name;
};
class Teacher
{   Public :
     string specialization;
};
class youtuber
{   Public :
     int subscriber;
};
class code_teacher : Public Person, Public Teacher, Public youtuber.
{
     cout << name << specialization << subscriber.
}
```

Hierarchical Inheritance :



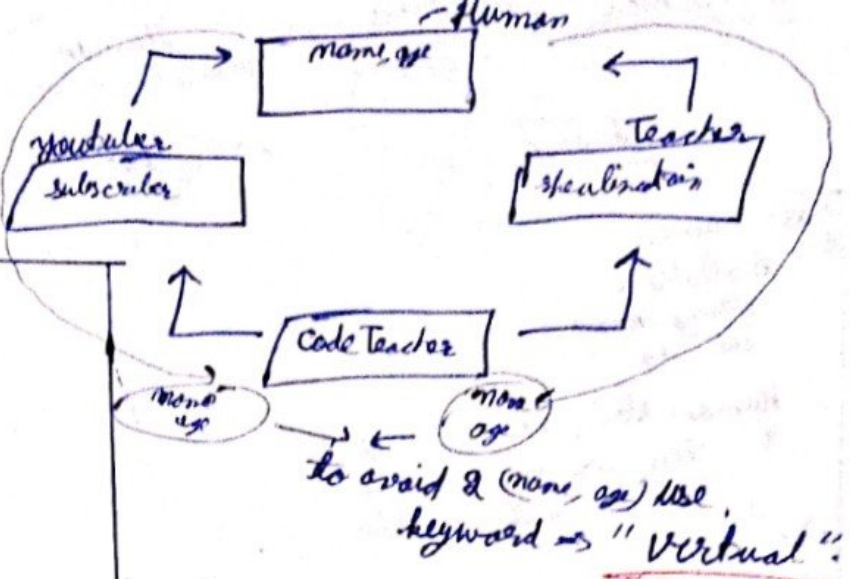

```
class Human
{
   Protected :
   string name ;
   int age ;

   Human(string name , int age)
   { this → name = name ;
     this → age = age ;
   };

class Student : Public Human
{
     Public :
     int roll_no , fees ;
     Student ( string name, int age , int roll_no, int fees): Human (name, age)
     {
          this → name = name ;
          this → age = age ;
          this → rollno = roll_no ;
       } this → fees = fees ;
```

Hybrid Inheritance :




```
class student
{

 };
class Male
{

 };
class Female
{

 };
Class Boy : Public student, Public Male
{

 };
class Girl : Public student, Public Female
{

 };
```

Multipath Inheritance



to avoid 2 (name, age) use keyword → "virtual".

```
class human
{
  };
class youtuber : public virtual human
{
};
class Engineer : public virtual human
{
  };
class codeTeacher : public Engineer, public youtuber
{
  };
```

* Polymorphism

Many form

- Compile time
  - Function overloading
  - operation overloading
- Run time

• fn" overloading :

```
class area
{
  public :
    int calculate area (int r)
    { return 3.14 * r * r ; }
    int calculate area (int l, int b)
    { return l * b ; }
};
int main ()
{
    Area A1, A2 ;
    A1. Calculate area (3) ;
    A2. Calculate area (3, 4) ;
}
```

• operation overloading

```
class my complex
{
  int real, img ;
  public :
    my complex ( ) { }
    my complex (int real, int img)
    {
      this → real = real ;
      this → img = img ;
    }
    void display ( )
    {
      cout << real << " + i" << img << endl ;
    }
    my complex operation + (my complex & C)
    {
      my complex ans ;
      ans. real = real + C. real ;
      ans. img = img + C. img ;
      return ans ;
    }
};
int main ( )
{
    my complex C1 (3, 5) ;
    my complex C2 (7, 5) ;
    my complex C3 = C1 + C2 ;
    C3 . display ( ) ;
}.
```

* Virtual

```
class animal
{
  public :
    virtual void speak ( ) :
    { cout << "hehe \n" ; }
};
class dog : public animal
{
  public :
    void speak ( )
    { cout << "bark \n" ; }
};
int main ( )
{
    animal * P ;
    P = new dog ( ) ;
    P → speak ( ) ;
}
```

*P (Pointer Pointing animal class)

P = new dog ( ); ⇒ heap memory

but in run time.

Since virtual hold "P → speak ( );" till the run time not decide output is compile time.

* Pure virtual function :- // Abstract class.

⇒ (virtual void speak ( ) = 0 ;)

⇒ No object form in pure virtual function class.

* Exception Handling :- An exception is an unexpected problem that arises during the execution of a program i.e. our program terminates suddenly with some errors / issues.

Try :- It represent a block of code that may through an exception placed inside the try block.

Catch :- It represent a block of code that is executed when a particular exception is thrown from the try block.

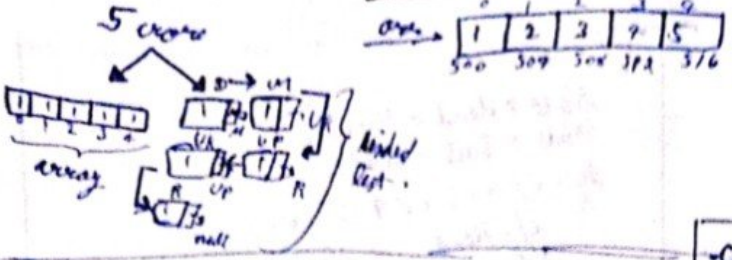
Throw :- An exception in C++ can be thrown using the throw keyword.

```
class customer
{
  string name;
  int balance, acc-no;
  Public
  customer(string name, --- -)
  {
  }
  - - -
  void deposit(int amount)
  {
    if(amount > 0)
    {
      balance += amount;
    } cout << amount;
    else
    {
    } throw "amount should greater then zero";
  } or // throw run-time error("amount should greater then zero");
};
int main()
{
  customer A1("Rohit", 5000, 998);
  try {
    A1.deposit(500);
    A1.deposit(-200);
  }
  catch (const char *e) or // catch (const runtime_error& e)
  {
    cout << "exception occured : " << e << endl;
  }
                                  or // << e.what() << endl;
```

-> The try keyword represents a block of code that may throw an exception placed inside the try block. It's followed by one or more catch blocks. If an exception occurs, try block throw that exception.

```
int main()
{
  int a = 4, b = 0;
  try {
    if (b == 0)
      throw "Divide by 0 is not Possible";
    int c = a/b;
    cout << c << endl;
  }
  catch (const char *e)
  {
    cout << "Exception occured : " << e << endl;
  }
}
```

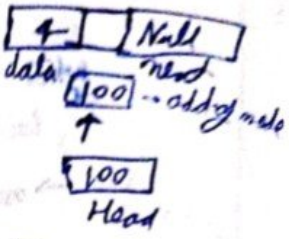
* Linked list



- Insertion.
- Deletion.
- Search.
- Update.

```
class Node
{
  Public:
    int data;
    Node * next;
  Node ( int value)
  {
      data = value;
      next = NULL;
  }; }
int main()
{
    Node * Head
    Head = new Node(4);
}
```

```
class Node
{  Public:
     int data;
   Node *next;
};
int main()
{
   Node * Head;
   Head = new Node();
   Head → data = 4;
   Head → next = Null;
};
```

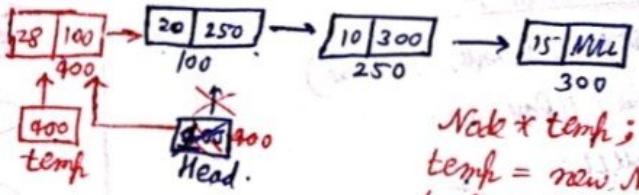


(Dynamic way to create a node)

① Insertion

- start
- end
- Middle.

① start



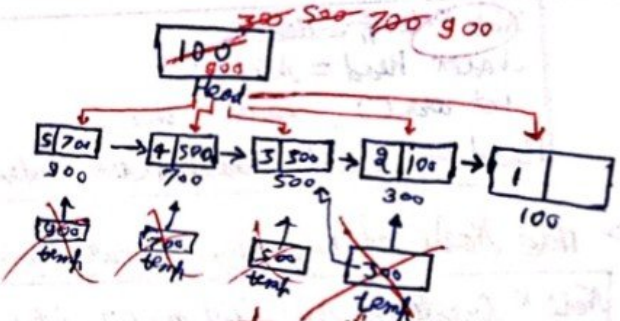

```
Node * temp;
temp = new Node(28);
temp → next = Head;
Head = temp;
```

if linked list does not exist

```
if(Head == Null)
{
  Head = new Node(
}
else
```

```
class Node
{
  Public:
  int data;
  Node * next;
  Node ( int value)
  {
      data = value;
      next = NULL;
  };
int main()
{
    Node * Head;
    Head = NULL;
    int arr[] = {1,2,3,4,5};
    for (int i=0; i<5 ; i++)
    {
        if( Head == NULL)
        {
            Head = new Node(arr[i]);
        }
        else {
            Node * temp;
            temp = new Node (arr[i]);
            temp → next = Head;
            Head = temp;
        }
    }
    // Print values of node.
    Node * temp = Head;
    while (temp != Null)
    {
        cout << temp → data << " ";
    }; temp = temp → next;
}
```

arr 1 2 3 4 5



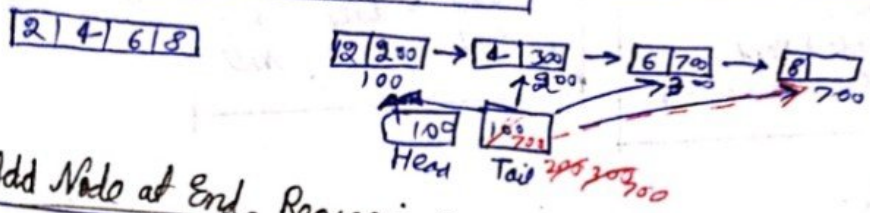
temp will delete automatically after a loop completes because it create static "Node * temp"

but temp = new Node(arr[i]); does not delete because it create dynamically

Q) End :

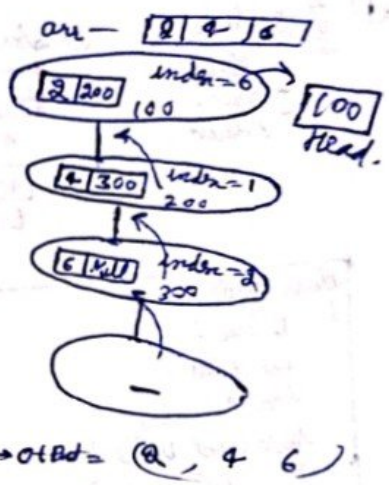
```
int main()
{
  Node * Head = Null;
  for (int i=0; i<4; i++)
  {
    if (Head == NULL))
    {
      Head = new Node(arr[i]);
    }
    else
    {
      Node * tail = Head;
      while (tail -> next != NULL)
        tail = tail -> next;
      tail -> next = new Node(arr[i]);
    }
  }
}
```

```
int main()
{
  Node * Head = NULL;
  Node * tail = NULL;
  for (i=0; i<4; i++)
    if (Head == Null)
    {
      Head = new Node(arr[i])
      tail = Head;
    }
    else
    {
      tail -> next = new Node(arr[i]);
      tail = tail -> next;
    }
}
```



→ Add Node at End, Recursion :-

```
Node * CreateLinkedlist(int arr[], index, int size)
{
    if (index == size) //Base case)
    {
      return NULL;
    };
    Node * temp;
    temp = new Node(arr[index]);
    temp -> Next = CreatedLinkedlist(arr, index+1, size);
    return temp;
}
```

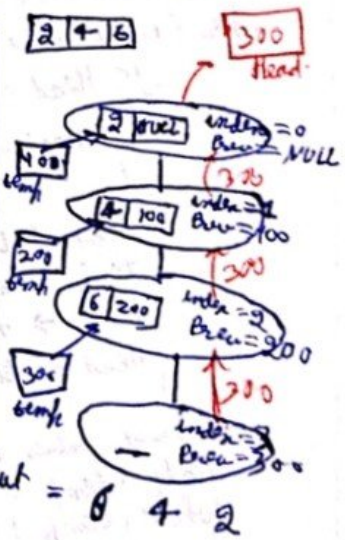


→ Output = (2, 4 6)

```
~~Node * temp = Head~~;
Node* Head = NULL;
int arr[], index, size;
Head = createLinkedlist(arr, index, size);
```

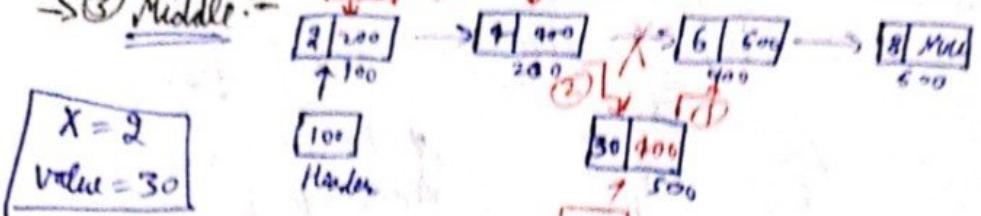
→ Add Node at Beginning, Recursion :-

```
Node * CreateLinkedlist (int arr[], int index, int size, Node* Brev)
{
    if (index == size)
        return prev;
    Node * temp;
    temp = new Node(arr[index]);
    temp -> next = Brev;
    return create linkedlist(arr, index+1, size, temp);
}
```



Output = 6 4 2

→ ③ Middle :-



↳ move (x-1) times

temp

① temp2 → next = temp → next;
② temp → next = temp2;

```
Node * createLinkedList (int arr[], int index, int size)
{
    // Same code as "Add node at End_Recursion".
}
int main()
{
    Node * Head = NULL;
    int arr[] = {1, 2, 3, 4, 5};
    Head = createLinkedList(arr, 0, 5);
    //
    int x = 3;
    int value = 30;
    Node * temp = Head;
    x--;
    while(x--)
    {
        temp = temp → next;
    }
```

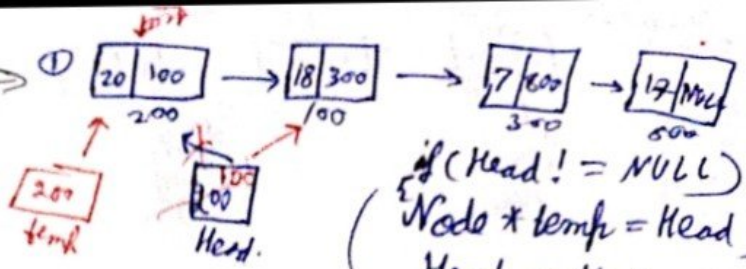
```
Node * temp2 = new Node(value);
temp2 → next = temp → next;
temp → next = temp2;
// Print nodes
temp = Head;
while(temp)
{
    cout << temp → data << " ";
    temp = temp → next;
};
}
```

- Array - static memory → couldn't increase size (use vectors)

  Linked list - Dynamically allotion → could increase size

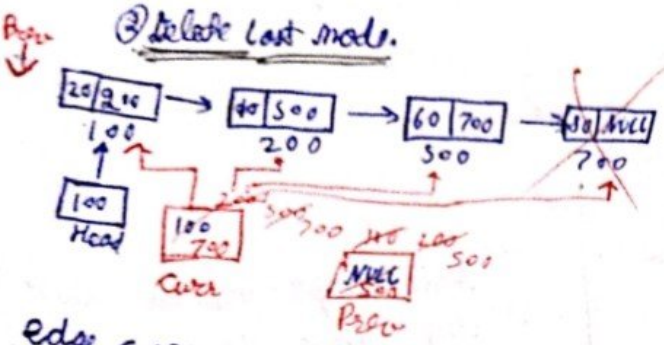
→ Deletion in Singly linked list :-

① Delete first node.
② Delete Last node.
③ Delete Particular node.




```
if (Head != NULL)
{
Node * temp = Head;
Head = Head → next;
delete temp;
}
```

② Delete Last node.



Edge cases :-

① Linked list doesn't exist
(Head != NULL)

② Linked list with only 1 Node.
(Head → next == NULL)

③ Linked list with greater than 1 Node.
(else ⇒ remain code)

```
if(Head != NULL)
{
  if (Head → next == NULL)
  {
    Node * temp = Head;
    Head = NULL;
    delete temp;
  }
  else
  {
    Node * curr = Head;
    Node * Prev = NULL;
    while (curr → next != NULL)
    {
      Prev = curr;
      curr = curr → next;
    }
    delete curr;
    Prev → next = NULL;
  }
}
```

③ Delete a particular Node :-

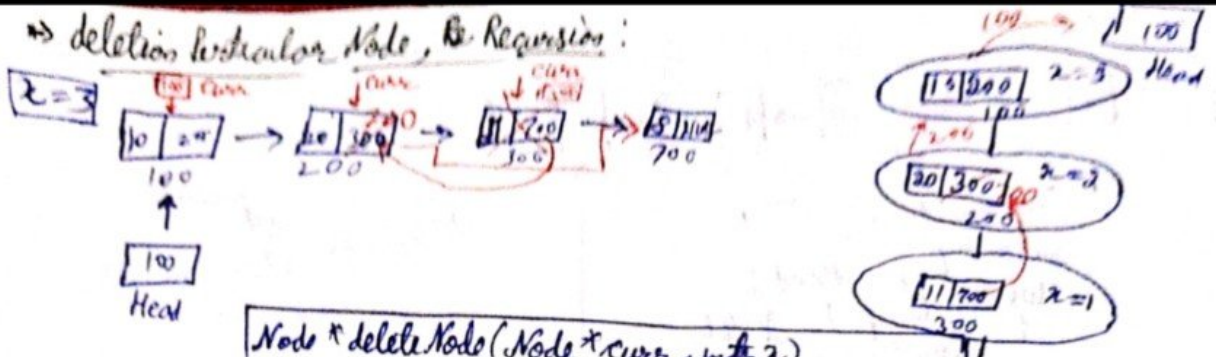
```
if (x == 1)
{
  Node * temp = Head;
  Head = Head → next;
  delete temp;
}
else
{
  Node * curr = Head;
  Node * Prev = NULL;
  x--;
  while (x--)
  {
    Prev = curr;
    curr = curr → next;
  }
  Prev → next = curr → next;
  delete curr;
}
```

Edge cases :-

① At first node.
(x == 1)

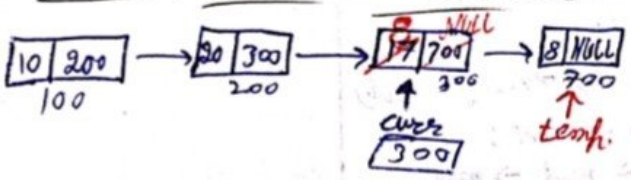
② remaining nodes.
(else ⇒ remain code)

→ deletion Node, By Recursion :



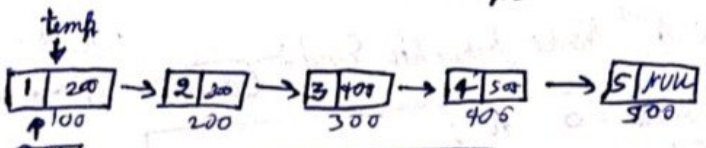

```
Node * deleteNode(Node * curr, int x)
{
    if (x == 1)
    {
        Node * temp = curr → next;
        delete curr;
        return temp;
    }
    curr → next = deleteNode(curr → next, x-1);
    return curr;
}
```

→ delete ~~test~~ Node without Head Pointer :- delete data of Node pointing by curr & curr can't point last Node.




```
Node * temp = curr → next;
curr → data = temp → data;
curr → next = temp → next;
delete temp;
```

→ Reverse linked list



Reverse value :-

```
vector <int> ans;
Node * temp = Head;
while (temp != NULL)
{
    ans.push_back(temp → data);
    temp = temp → next;
}
int i = ans.size() - 1;
temp = Head;
while (temp)
{
    temp → data = ans[i];
    i--;
    temp = temp → next;
}
```

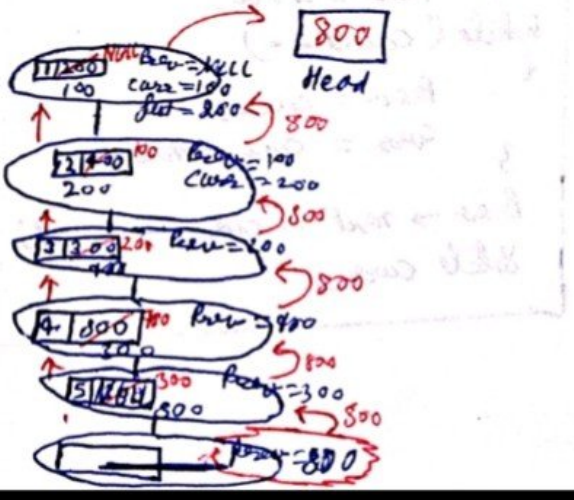
1 2 3 4 5

Reverse whole Node :-

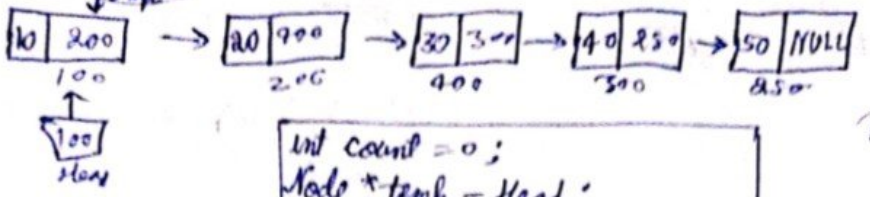
```
Node * curr = Head;
Node * Prev = NULL;
Node * fut = NULL;
while (curr)
{
    fut = curr → next;
    curr → next = Prev;
    prev = curr;
    curr = fut;
}
Head = Prev;
```

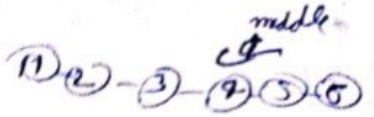
Reverse by recursion →

```
Node * Reverse(Node * curr, Node * Prev)
{
    if (curr == NULL)
        return Prev;
    Node * fut = curr → next;
    curr → next = Prev;
    return Reverse(fut, curr);
}
int main()
{
    Node * Head;
    Head = Reverse(Head, NULL);
}
```



→ Middle of linked list :-





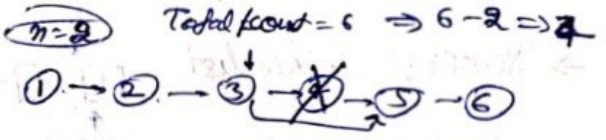

```
int count = 0;
Node * temp = Head;
While (temp != NULL)
{
    count++;
    temp = temp -> next;
}
count /= 2;
temp = Head;
While (count--)
{
    temp = temp -> next;
}
```

- Slow Pointer & fast Pointer :-

```
Node * slow = Head;
Node * fast = Head;
while (fast != NULL && fast -> next != NULL)
{
    slow = slow -> next;
    fast = fast -> next -> next;
}
return slow;
```

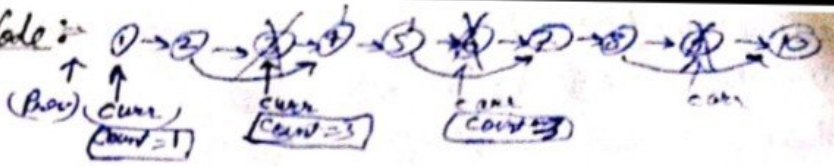
→ Rotate list :-

→ Remove N^{th} Node from the End :- [N :-]



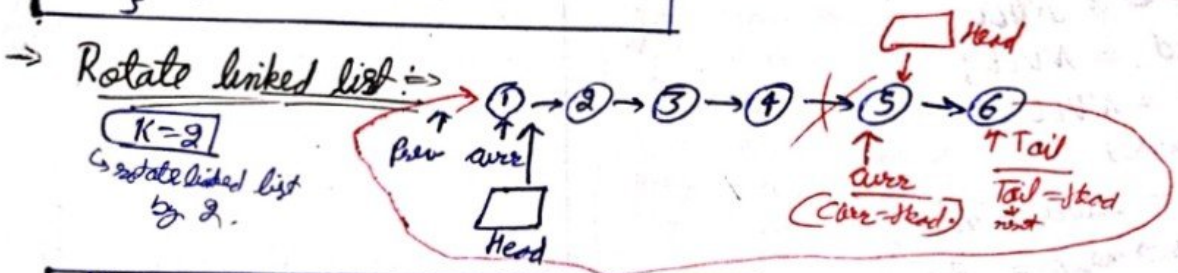

```
int count = 0, n;
Node * temp = Head;
while (temp) // count total no. of Nodes
{
    count++;
    temp = temp -> next;
}
count -= n;
if (count == 0) // (edge case) ⇒ delete first node.
{
    temp = Head
    Head = Head -> next
    delete temp;
}
Node * curr = Head;
Node * Prev = NULL;
while (count--)
{
    Prev = curr;
    curr = curr -> next;
}
Prev -> next = curr -> next;
delete curr;
```

→ Remove Every K^th Node :-



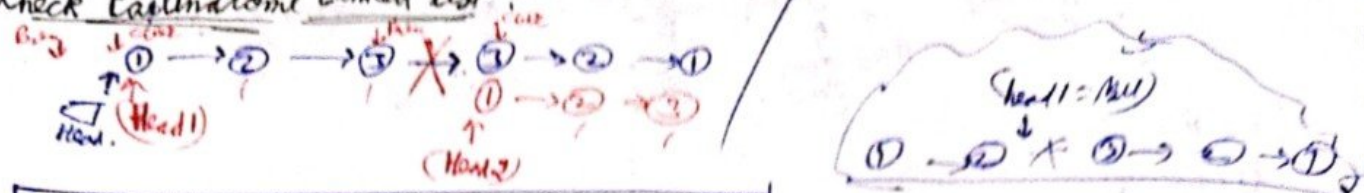

```
int K;
if (k == 1)
    return NULL;
Node *curr = Head, *prev = NULL;
int count = 1;
while (curr)
{
    if (K == count)
    {
        Prev -> next = curr -> next;
        delete curr;
        curr = Prev -> next;
        count = 1;
    }
    else {
        Prev = curr;
        Curr = curr -> next;
        count++;
    }
}
```

→ Rotate linked list :-



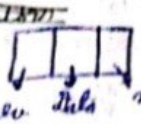

```
if (Head == NULL || Head -> next == NULL) => // edge cases
        return head;
int count = 0;
Node *temp = head;
while (temp)
{
    count++;
    temp = temp -> next;
}
K = K % count;
if (K == 0)          // edge cases
    return head;
count -= K;
Node *curr = head, *prev = NULL;
while (curr--)
{
    Prev = curr;
    curr = curr -> next;
}
Prev -> next = NULL;
Node *tail = curr;
while (tail -> next != NULL)
{
    tail = tail -> next;
}
tail -> next = head;
head = curr;
return head;
```

→ Check Palindrome Linked list :




```
Node * temp = Head;
int count = 0;
while (temp)
{
    count ++;
    temp = temp -> next;
};
count /= 2;
Node * curr = Head;
Node * prev = NULL;
while (count--)
{
    prev = curr;
    curr = curr -> next;
}
prev -> next = NULL;
Node * fut = NULL;
    prev = NULL;
while (curr)
{
    fut = curr -> next;
    curr -> next = prev;
    prev = curr;
    curr = fut;
}
Node * Head1 = Head;
Node * Head2 = prev;
while (Head1)
{
    if (Head1 -> data/val != Head2 -> data/val)
        cout << "Not a Palendrom";
    Head1 = Head1 -> next;
    Head2 = Head2 -> next;
}
    cout << " Linked list is Pallendrom";
```

* Doubly linked list :-

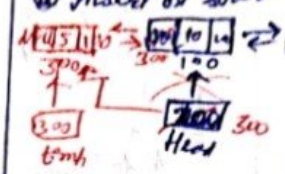



```
class Node
{
    Public :
    int data ;
    Node * next ;
    Node * Prev ;
    Node ( int value )
    {
        data = value ;
        next = NULL ;
        Prev = NULL ;
    } ;
};
```

① Insertion → Insert at Start.
→ Insert at end.
→ Insert at Particular Place.

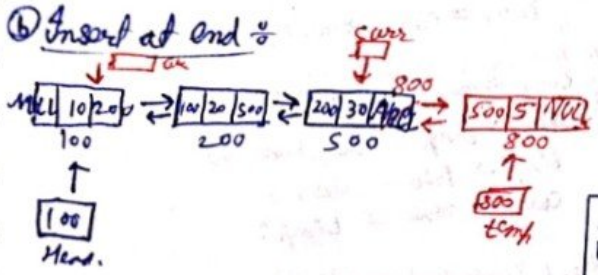
② Deletion

ⓐ Insert at start →




```
if (Head == NVLL)  // if linked list is empty
    Head = new Node (5) ;
else {
    Node * temp = new Node (5) ;
    temp -> next = Head ;
    Head -> Prev = temp ;
    Head = temp ;
}
```

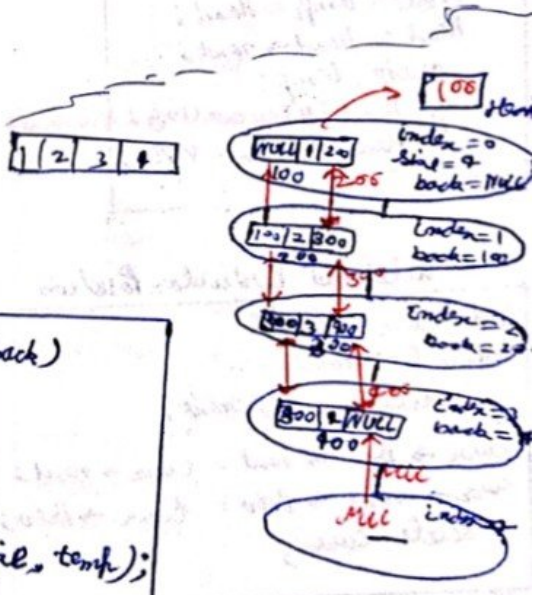
ⓑ Insert at end :-




```
Node * curr = Head ;
while ( curr -> next != null )
    curr = curr -> next ;
Node * temp = new Node (5) ;
curr -> next = temp ;
temp -> Prev = curr ;
```

→ Array to Doubly linked list :- (Insert at end)

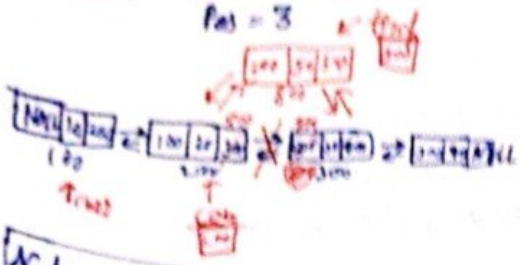
```
Node * Head = NULL ;
Node * Tail = Head ;
for (int i=0 ; i <5 ; i++)
{
    if (Head == NULL)
    {
        Head = new Node (arr[i]) ;
        Tail = Head ;
    }
    else
    {
        Node * temp = new Node (arr[i]) ;
        tail -> next = temp ;
        temp -> prev = Tail ;
        Tail = temp ;
    }
}
```



→ Using Recursion :-

```
Node * CreateDLL (int arr[], int index, int size, Node * back)
{
    if (index == size)
        return NULL ;
    Node * temp = new Node (arr[index]) ;
    temp -> prev = back ;
    temp -> next = CreateDLL (arr, index+1, size, temp) ;
    return temp ;
}
int main ()
{
    Head = CreateDLL (arr, 0, 5, NULL) ;
}
```

Insertion at Particular Node :




```
Node* curr = Head;
while(--Pos)
{
    curr = curr->next;
}
Node* temp = new Node(5);
temp->next = curr->next;
temp->Prev = curr;
curr->next = temp;
temp->next->Prev = temp;
```

edge cases :-

① Insert at start ①

② Insert at End (last Node)

```
int Pos;
if(Pos == 0) // Insert at start ①
{
    if(Head == NULL) // Linked list does not exist
    {
        Head = new Node(6);
    }
    else {
        Node* temp = new Node(6);
        temp->next = Head;
        Head->Prev = temp;
    } Head = temp;
}
else {
    Node* curr = Head;
    // Go to Particular Pos.
    while(--Pos)
    {
        curr = curr->next;
    }
    // Particular Pos is Last Node ②
    if(curr->next == NULL)
    {
        Node* temp = new Node(6);
        temp->Prev = curr;
        curr->next = temp;
    }
    else { // Insert at middle.
        Node* temp = new Node(6);
        temp->next = curr->next;
        temp->Prev = curr;
        curr->next = temp;
        temp->next->Prev = temp;
    }
}
```

★ Deletion :

- at start ①
- at Middle ②
- at End ③

→ delete at start.

```
if(Head != NULL)
{
    Node* temp = Head;
    Head = Head->next;
    delete temp;
    if(Head) // Edge case (only 1 Node exist)
        Head->Prev = NULL;
};
```

→ Delete at End.

```
if(Head != NULL)
{
    if(Head->next == NULL) // only single node exist.
    {
        delete Head;
        Head = NULL;
    }
    else {
        Node* curr = Head;
        while(curr->next)
        {
            curr = curr->next;
        }
        curr->Prev->next = NULL;
    } delete curr;
}
```

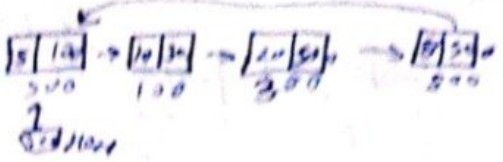
→ Delete at Particular Position

```
Node* curr = Head;
while(--Pos)
{
    curr = curr->next;
}
curr->Prev->next = curr->next;
curr->next->Prev = curr->Prev;
delete curr;
```

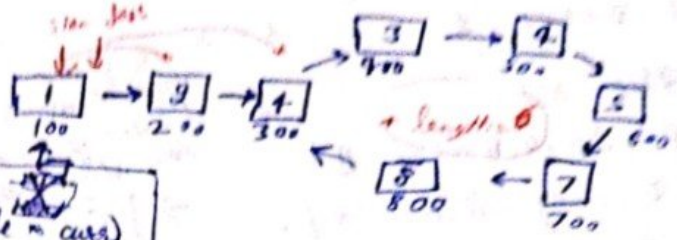
edge cases : delete at first node.
delete at Last node.
⇒ Same code as above → ~~Particular node~~ Particular node.

- Doubly linked list in real life
  → Undo / Redo

→ Circular linked list :



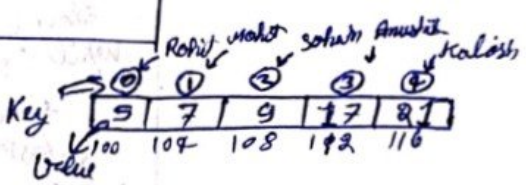
→ Detect Loop in Linked List :



```
bool check (vector <Node *> visited, Node * curr)
{
    for (i = 0; i < visited.size(); i++)        → O(n)
    {
        if (visited[i] == curr)
            return 1;
    }
    return 0;
}

Node * Head = ____;
Node * curr = Head;
vector <Node*> visited;
while (curr != NULL)                            → O(n)
{
    if (check (visited, curr))
        return 1;
    visited.push_back (curr);
    curr = curr -> next;
}
return 0;
```

T.C. → $O(n^2)$

- Unordered maping :



Key	Value
Rohit/0	5
Mohit/1	7
sohan/2	9
Anshit/3	17
Kailash/4	21

- Detect loop.

```
Node * curr = Head;
unordered_map <Node*, bool> visited;
while (curr != NULL)
{
    if (visited[curr] == 1)
        return 1;
    visited[curr] = 1;
    curr = curr -> next;
}
```

→ $O(n)$ = T.C.

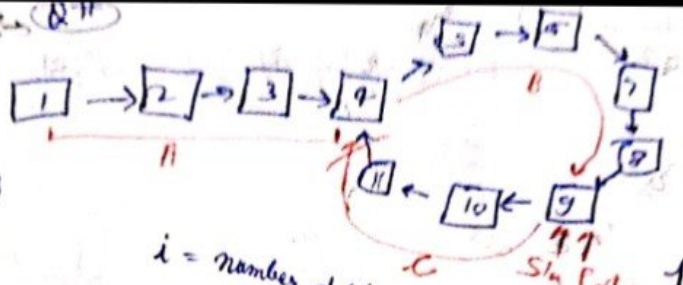
- Fast and Slow Pointer

```
Node * slow = Head;
Node * fast = Head;
while (fast != NULL && fast -> next != NULL)
{
    slow = slow -> next;
    fast = fast -> next -> next;
    if (slow == fast)
        return 1; // cout << "loop exist";
}
```

→ length of loop :

```
Node *slow = Head, *fast = Head;
while (fast != NULL && fast -> next != NULL)
{
    slow = slow -> next;
    fast = fast -> next -> next;
    if (slow == fast)
        break;
}
if (fast == NULL || fast -> next == NULL)
    return 0; // "loop does not exist"
int count = 1;
slow = fast -> next;
while (slow != fast)
{
    count++;
    slow = slow -> next;
}
cout << count;
}
```

→ Remove loop in Linked list :- (Q 141)



Mathematical Proof :-

Slow dist :- $A + (B+C)i + B$

Fast dist :- $A + (B+C)j + B$

Fast dist = 2 Slow dist.

$A + (B+C)*j + B = 2A + 2(B+C)*i + 2B.$

$A + B = (B+C)*j - 2(B+C)*i.$

$A + B = (B+C)*(j - 2i)$

$A + B = T * (B+C)$

$T \neq 1,$

$T = 2, 3, 9,$

($T = 2$) $A + B = 2(B+C)$

$A = (B+C) + B$

($T = 3$) $A = 2(B+C) + B$

i = number of times slow iterate the loop.

j = number of times fast iterate the loop.

→ $j > 2i$

fast slow

1. - loop detect.
2. Slow = Head.
3. Starting of loop.

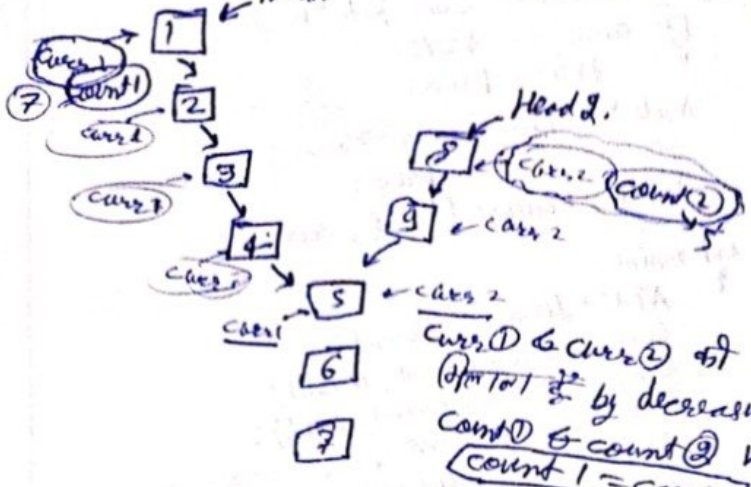
```
Node *slow = Head, *fast = Head;
while(fast != NULL && fast -> next != NULL)
{
    slow = slow -> next;
    fast = fast -> next -> next;
    if(slow == fast)
        break;
}
if(fast == NULL || fast -> next == NULL)
    return;
slow = head;
while(slow != fast)
{
    slow = slow -> next;
    fast = fast -> next;
}
while(slow -> next != fast)
{
    slow = slow -> next;
}
slow -> next = NULL;
```

(M2)

1. loop detect
2. find number of nodes in loop.
3. slow = Head, fast = slow + no of nodes
4. iterate fast to meet Head
5. break the loop.

Insertion Point in Y Shaped Linked list

#8



curr① & curr② ko by decreasing count① & count② when count 1 = count 2 curr① & curr② ki same position

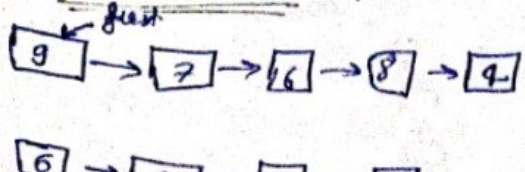
```
Node * curr1 = Head1, * curr2 = Head2;
int count1 = 0, count2 = 0;
while (curr1)
{
    count1++;
    curr1 = curr1 → next;
}
while (curr2)
{
    count2++;
    curr2 = curr2 → next;
}
curr1 = Head1, curr2 = Head2;
while (count1 > count2)
{
    curr1 = curr1 → next;
    count1--;
}
while (count2 > count1)
{
    curr2 = curr2 → next;
    count2--;
}
while (curr1 != curr2)
{
    curr1 = curr1 → next;
    curr2 = curr2 → next;
}
if (curr1 == Null)
    return -1;
return curr → data;
```

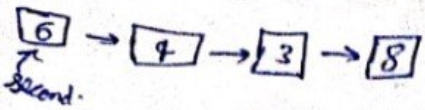
Alternate sol^n

① *Tail, Head 1 ka.

② Loop ka starting Head ① Pointer.

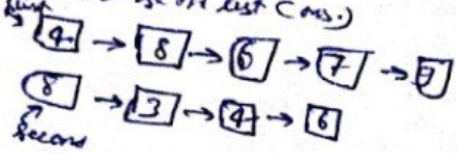
→ Add 2 Numbers :-



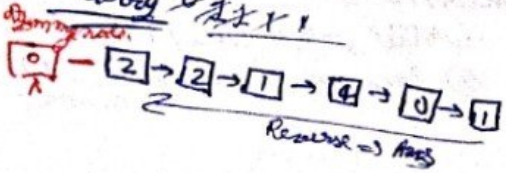


9 7 6 8 4
6 4 3 8
1 0 8 1 2 2

① Reverse both linked list.
② Addition
③ Reverse the list (ans.)



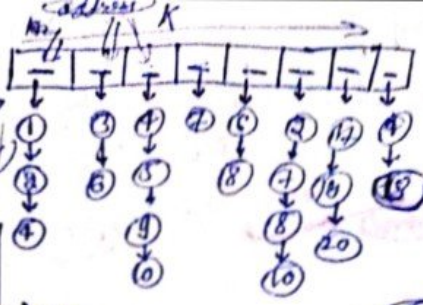
Carry 0 आया !




```
//Reverse the Node
Node *Reverse(Node *curr, Node *prev)
{
    if(curr == NULL)
        return prev;
    Node *front = curr->next;
    curr->next = prev;
    return Reverse(front, curr);
}
int main()
{
    Node *first, *second;
    first = Reverse(first, NULL);
    second = Reverse(second, NULL);
    Node *curr1 = first, *curr2 = second;
    Node *Head = new Node(0); //Dummy node
    Node *Tail = Head;
    int carry = 0, sum;
    while(curr1 && curr2)
    {
        int sum = curr1->data + curr2->data + carry;
        Tail->next = new Node(sum % 10);
        Tail = Tail->next;
        curr1 = curr1->next;
        curr2 = curr2->next;
        carry = sum / 10;
    }
    while(curr1)
    {
        int sum = curr1->data + carry;
        Tail->next = new Node(sum % 10);
        Tail = Tail->next;
        curr1 = curr1->next;
        carry = sum / 10;
    }
    while(curr2)
    {
        int sum = curr2->data + carry;
        Tail->next = new Node(sum % 10);
        Tail = Tail->next;
        curr2 = curr2->next;
        carry = sum / 10;
    }
    while(carry)
    {
        Tail->next = new Node(carry % 10);
        carry /= 10;
    }
    Head = Reverse(Head->next, NULL);
    return Head;
}
```

→ Merge K Sorted linked list :-

(Node* arr[K]) ⇒ arr



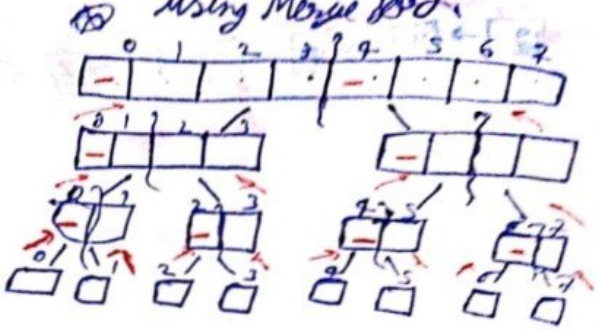
```
class Node
{ Public:
  int data;
  Node *next;
};
```

M1

```
// Merge fn same as previous question
Node* Head = arr[0];
for (i=1; i<k; i++)
{
   Head = merge (Head, arr[i]);
}
return Head;
```

→ T.C ⇒ $T(NK^2)$

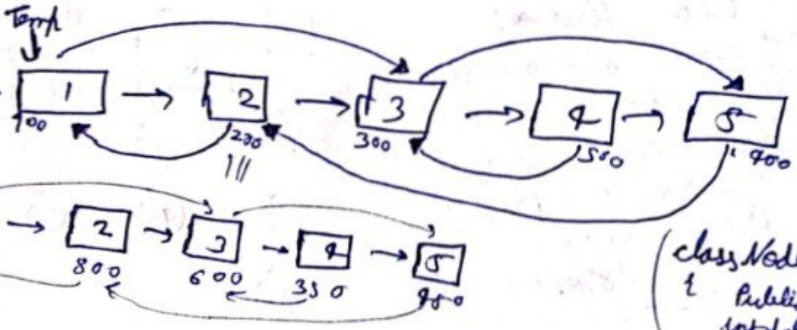
M2 Using Merge sort :-



```
void Mergesort (Node *arr[], int start, int end)
{
   if (start >= end)
      return;
   int mid = start + (end - start)/2;
   mergesort (arr, start, mid); // left
   mergesort (arr, mid+1, end); // Right
   arr[start] = merge (arr[start], arr[mid+1]);
}
```

⇒ T.C. → $O(KN\log K)$

→ Clone a linked list :-



steps :-
1. create all nodes without random pointers.
2. Assign random pointers from start.

```
class Node
{ Public:
  int data;
  Node *next, *arb;
};
```

→ find fxn

```
Node* find (Node* curr1, Node* curr2, Node* x)
{
   if (x == NULL)
      return NULL;
   while (curr1 != x)
   {
      curr1 = curr1->next;
      curr2 = curr2->next;
   }
   return curr2;
}
```

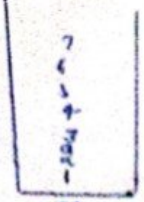
Time complexity :- $O(n^2)$

```
Node *Headcopy = new Node (0);
Node *tailcopy = Headcopy;
Node *temp = Head;
while (temp)
{
   Tailcopy->next = new Node (temp->data);
   Tailcopy = Tailcopy->next;
   temp = temp->next;
}
Tailcopy = Headcopy;
Headcopy = Headcopy->next;
delete Tailcopy;
Tailcopy = Headcopy;
temp = Head;
while (temp)
{
   Tailcopy->arb = find (Head, Headcopy, temp->arb);
   Tailcopy = tailcopy->next;
   temp = temp->next;
}
return Headcopy;
```

Create duplicate linked list

O(n) O(n) O(n)

Stack



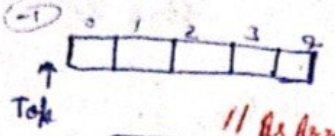
Stack

- It is a linear data structure, in which insertion & deletion only allowed at the end, called the top of stack
- When we define stack as an Abstract data type then we are only interested in knowing the stack operation from user POV.
  If it simply means, we are not intrested in knowing the details, we only intrested in what type of o/p we can perform.

Operations on Stack :

Pop :
Push :
Top/Peek :
isSize :
isEmpty :

Last in first out LIFO



First in first out
First in Last out
Last in Last out
Last in first out

// As Array

```
class Stack
{
    int *arr;
    int size;
    int top;
  Public :
bool flag ;
// Constructor
Stack (int s)
{
    size = s;
    top = -1;
    arr = new int[s];
  } flag = 1;

    // Push
void Push(int value)
{  if (top == size - 1)
    {
        cout << "Stack overflow \n";
        return;
    }
    else {
    top ++;
    arr[top] = value;
    cout << "Pushed" << value << "in Stack";
} } flag = 0
```

```
    // Pop
void Pop()
{   if(top == -1)
        cout << "Stack Underflow
    Else
    {  cout << "Popped" << arr[top]
            << From the Stack \n"
    top --;
    if(top == -1).
} } flag = 1;
    // Peek
int peek()
{   if(top == -1)
    {  cout << "Stack empty";
       return -1;
    }
    else
    {
} } return arr[top];
    // IsEmpty
bool IsEmpty()
{   return top == -1;
} // Is Size
int IsSize()
{   return top + 1;
};
```

// As Linked list.

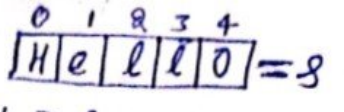
```
class Node
{
    // Same code to create node.
};
class Stack {
{   Node * Top;
    int size;
    Public :
    Stack ()
    {   Top = NULL;
        size = 0;
    }
void Push(int value)
{  Node * temp = new Node(value);
   temp -> next = Top;
   Top = temp;
     size++;
}
Void Pop()
{ if(Top == NULL)
  {  cout << "Stack Underflow";
  }
  else
  {
    Node * temp = Top;
    Top = Top -> next;
    delete temp;
} }
int Peek()
{ if(top == NULL)
  {  cout << "Stack is empty";
     return -1;
  }
  else {
     return top -> data;
  }
};
```

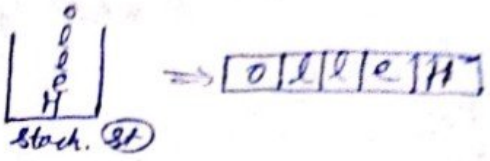
STL

→ implementation as "deque."

```
stack<int> s;
s.push(5);
s.push(6);
s.pop();
cout<<s.top;
```

- Reverse array :

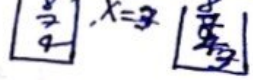





```
vector<char> s;
stack<char> st;
for(int i=0; i<s.size(); i++)
{
    st.push(s[i]);
}
int i=0;
while(!st.empty())
{
    s[i] = st.top();
    i++;
    st.pop();
}
```

→ TC ⇒ O(n)
SC ⇒ O(n)

- Insert at bottom :

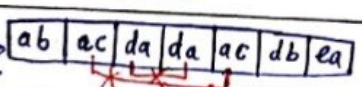



```
stack<int> st;
int x;
```

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

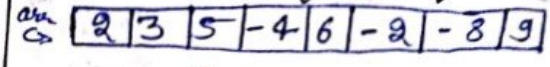
```
stack<int> temp;
while(!st.empty())
{
    temp.push(st.top());
    st.pop();
}
st.push(x);
while(!temp.empty())
{
    st.push(temp.top());
    temp.pop();
}
return st;
```

- String Manipulation :




```
stack<string> s;
for(i=0; i<v.size(); i++)
{
    if(s.empty())
        s.push(v[i]);
    else if(s.top() == v[i])
        s.pop();
    else
        s.push(v[i]);
}
return s.size();
```

- Make the array beautiful



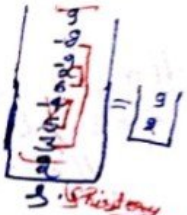
2 3 5 -4 6 -2 -8 9 (different)

2 3 -4 6 -2 -8 9

2 -8 9

stack neg

2 9




```
stack<int> s;
for(i=0; i<arr.size(); i++)
{
    if(s.empty())
        s.push(arr[i]);
    else if(arr[i] >= 0)
        if(s.top() >= 0)
            s.push(arr[i]);
        else
            s.pop();
    else // arr[i] <= 0 (-ve)
        if(s.top() < 0)
            s.push(arr[i]);
        else
            s.pop();
}
vector<int> ans(s.size());
int i = s.size() - 1;
while(!s.empty())
{
    ans[i] = s.top();
    i--;
    s.pop();
}
return ans;
```

- Parenthesis is valid or not

str = (((())())) → valid

```
bool check(string str)
{
    stack<char> s;
    for(i=0; i<str.size(); i++)
    {
        if(str[i]=='(')
            s.push(str[i]);
        else
        {
            if(s.empty())
                return 0;
            else
                s.pop();
        }
    }
    return s.empty();
}
```

→ T.C. = O(n)
→ S.C. = O(n)

```
bool check(string str)
{
    int left=0;
    for(i=0; i<str.size(); i++)
    {
        if(str[i]=='(')
            left++;
        else
        {
            if(left==0)
                return 0;
            else
                left--;
        }
    }
    return left==0;
}
```

→ T.C. = O(n)
S.C. = O(1)

- Minimum add to make Parenthesis valid

str =))((⇒ 2

```
stack<char> st;
int count=0;
for(i=0; i<s.size(); i++)
{
    if(s[i]=='(')
        st.push(s[i]);
    else
    {
        if(st.empty())
            count++;
        else
            st.pop();
    }
}
return count+st.size();
```

- Valid Parentheses :

s = ([]) {} [()]

st.

```
stack<char> st;
for(i=0; i<s.size(); i++)
{
    if(s[i]=='('||s[i]=='{'||s[i]=='[')
        st.push(s[i]);
    else
    {
        if(st.empty())
            return 0;
        else if(s[i]==')')
        {
            if(st.top()!='(')
                return 0;
            else
                st.pop();
        }
        else if(s[i]=='}')
        {
            if(st.top()!='{')
                return 0;
            else
                st.pop();
        }
        else
        {
            if(st.top()!='[')
                return 0;
            else
                st.pop();
        }
    }
}
return st.empty();
```

- Print Bracket Number :

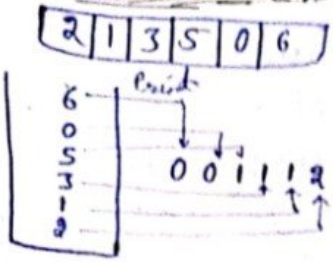
s = (aa(bdc))P(de) → 3

count = 0 1 2 3

ans = 1 2 2 1 3 3

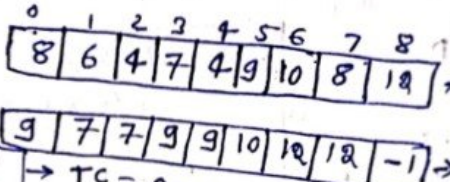
```
string s;
int count=0;
stack<int> st;
vector<int> ans;
for(i=0; i<s.size(); i++)
{
    if(s[i]=='(')
    {
        count++;
        st.push(count);
        ans.push_back(count);
    }
    else if(s[i]==')')
    {
        ans.push_back(st.top());
        st.pop();
    }
}
return ans;
```

Get Minimum at pop :




```
int arr[];
int size;
stack<int> st;
for(int i=0; i<n; i++)
{
    if(i==0)
        st.push(arr[i]);
    else
        st.push(min(arr[i], st.top()));
}
// print stack.
```

→ Next Greater Element :-



(M1) brute force :

```
vector<int> ans(n,-1);
for(i=0; i<n; i++)
{
    for(j=i+1; j<n; j++)
    {
        if(arr[j]>arr[i])
        {
            ans[i]=arr[j];
        }
    }
    break;
}
return ans;
```

→ TC = O(n²).
→ SC = O(n).

(M2) ⇒ (stack by store index)

```
stack<int> st;
vector<int> ans(n,-1);
for(i=0; i<n; i++)
{
    while(!st.empty() && arr[st.top()]<arr[i])
    {
        ans[st.top()]=arr[i];
        st.pop();
    }
    st.push(i);
}
return ans;
```

T.C. ← O(n)
S.C. ← ⇒ O(n).

(M3) ⇒ (stack by store data from the last element)

```
stack<int> st;
vector<int> ans(n,-1);
for(i=n-1; i>=0; i--)
{
    while(!st.empty() && arr[st.top()]<arr[i])
    {
        st.pop();
    }
    if(!st.empty())
        ans[i]=arr[st.top()];
    st.push(i);
}
```

→ Next Smaller Element

0	1	2	3	4	5	6	7	8
7	9	12	10	14	8	3	6	9

(3 8 10 8 8 3 -1 -1 -1)

- (Stack key & store indexes)

```
Stack<int> st;
Vector<int> ans(n, -1);
for(i=0; i<n; i++)
{
  while(!st.empty() &&
        arr[st.top()] > arr[i])
  {
      ans[st.top()] = arr[i];
      st.pop();
  }
  st.push(i);
}
return ans;
```

→ Smallest Number on left (Nearest)

0	1	2	3	4	5	6	7
4	13	11	05	9	7	8	6	→ a

(-1 4 4 4 5 5 7 5)

```
Stack<int> st;
Vector<int> ans(n, -1);
for(i = n-1; i>=0; i--)
{
  while(!st.empty() && a[st.top()] > a[i])
  {
      ans[st.top()] = a[i];
      st.pop();
  }
  st.push(i);
}
return ans;
```

→ Next Greatest Element 2 (circular array)

0	1	2	3	4	5	6	7	8	9	10	11	12	13
6	10	7	4	8	9	4	6	10	7	4	8	9	4

(6 -1 8 8 9 10 6)

↳ create a extra array.

(loop (0 → (2n-1))

⇒ next Greatest element code replace i by i%n

i%n

→ Stock Span Problem ⇒ left me uss price se less or equal price ka count store karona hai.

(similar next greatest on left side with itself also)

0	1	2	3	4	5	6
100	80	55	70	60	75	85

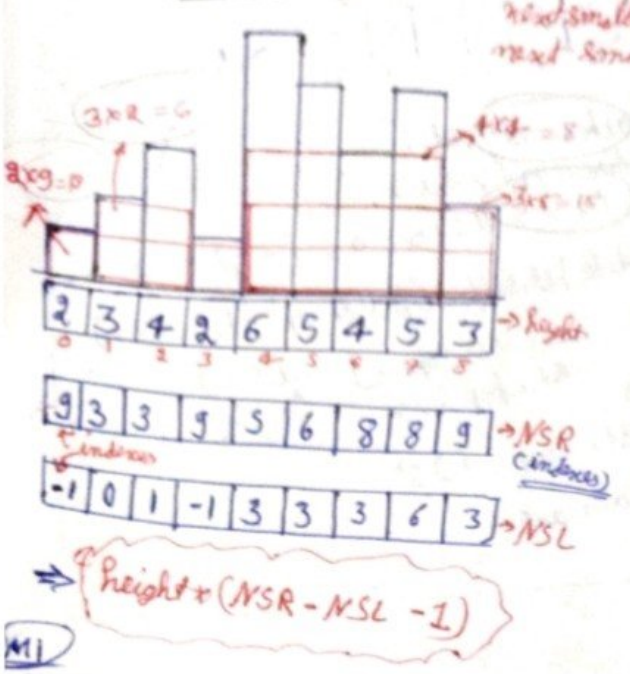
Price

(1 1 1 2 1 4 6)

4	60
5	75
6	85

```
stack<int> st;
vector<int> ans(n, 1);
for(i = n-1; i>=0; i--)
{
  while(!st.empty() && price[i] > price[st.top()])
  {
      ans[st.top()] = st.top() - i;
      st.pop();
  }
  st.push(i);
}
while(!st.empty())
{
    ans[st.top()] = st.top() + 1;
    st.pop();
}
return st;
```

- Largest Rectangle in histogram :-



⇒ height * (NSR - NSL - 1)

M1

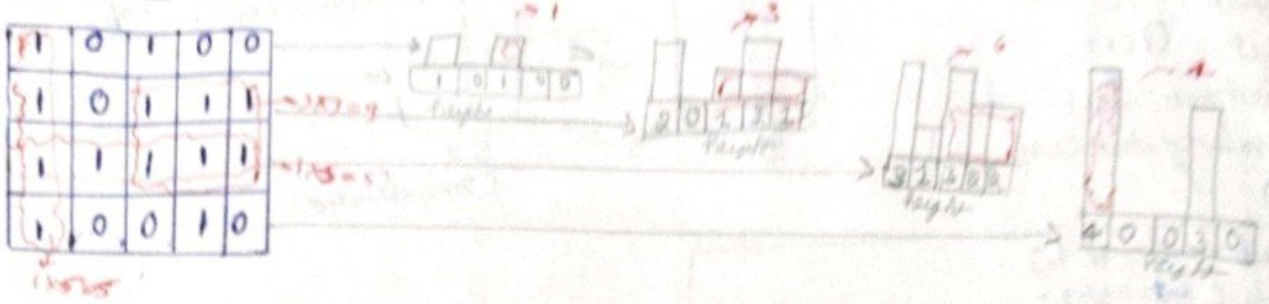
```
vector<int> Right(n);
vector<int> left(n);
stack<int> st;
for (i=0; i<n; i++)
{
  while(!st.empty() && height[st.top()]
                        > height[i])
  {
      Right[st.top] = i;
      st.pop();
  }
  st.push(i)
}
while (!st.empty())
{
   Right[st.top()] = n;
   st.pop();
}
for (i=n-1; i>=0; i--)
{ while(!st.empty() && height[st.top()]
                        > height[i])
  {
      left[st.top] = i
      st.pop();
  }
  st.push(i);
}
while(!st.empty())
{
   left[st.top()]] == -1;
   st.pop();
}
int ans = 0;
for (i=0; i<n; i++)
{
   ans = max(ans, height[i] *
              (Right[i] - left[i-1]));
}
return ans;
```

↳ T.C. ⇒ O(n).
S.C ⇒ O(n).

M2 more optimized.

```
int ans = 0, index, n = height.size();
stack<int> st;
for (i=0; i<n; i++)
{
  while(!st.empty() && height[st.top()] <
                        height[i])
  {
     index = st.top();
     st.pop();
     if(!st.empty())
     ans = max(ans, height[index] * (i - st.top() - 1));
     else
     ans = max(ans, height[index] * i);
  }
  st.push(i);
}
while(!st.empty)
{
  index = st.top();
  st.pop();
  if(!st.empty())
     ans = max(ans, height[index] * (n - st.top() - 1));
  else
     ans = max(ans, height[index] * (n));
}
return ans;
```

Maximal Rectangle only 1 containing binaries



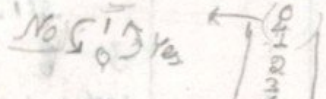
```
int row = matrix.size();
int col = matrix[0].size();
int ans = 0
vector<int> height(col, 0);
for(i=0; i<row; i++)
{
    for(j=0; j<col; j++)
    {
        if matrix[i][j] == '0'
            height[j] = 0;
        else
            height[j] += 1;
    }
    ans = max(ans, Rectangle(height));
}
return ans;
```

// Rectangle(height) ⇒ function
↳ code same as last question (Largest Rectangle)

→ The Celebrity Problem :-

celebrity ⇒ जिसे सब जानते हो ⇒ 1
वो किसी को ना जानता हो ⇒ 0

n ⇒ Whole ⇒ 5 (0, 1, 2, 3, 4)



first	second
Yes	No
Yes	Yes
No	No
No	Yes

↳ cand. celebrate

```
vector<vector<int>> M = ?;
int n;
stack<int> st;
for(i=n-1; i>=0; i--)
    st.push(i);
while(st.size()>1)
{
    int first = st.top();
    st.pop();
    int second = st.top();
    st.pop();
    if (M[first][second] && !M[second][first])
        st.push(second);
    else if (!M[first][second] && M[second][first])
        st.push(first);
}
if (!st.empty())
    return -1;
```

```
int num = st.top();
    st.pop();
int row=0, col=0;
for(i=0; i<n; i++)
{
    row += M[num][i];
    col += M[i][num];
}
return row==0 && col==(n-1) ? num : -1;
```

→ Get minimum element from stack

① Push : O(1) time
② Get : O(1)
③ Get min : O(1)

1 ≤ value of stack ≤ 10

min element (top)

M1

```
stack<int> st1;
stack<int> st2;
void push(int x)
{
    if(st1.empty())
        st1.push(x);
        st2.push(x);
    else
    {
        st1.push(x);
        st2.push(min(x, st2.top()));
    }
}
int pop()
{
    if(st.empty())
        return -1;
    else
    {
        int element = st1.top();
        st1.pop();
        st2.pop();
        return element;
    }
}
int getmin()
{
    if(st2.empty())
        return -1;
    else
        return st2.top();
}
```


if 2 pow 2 numbers of a int use %.

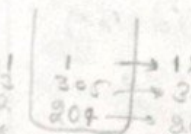
original * 101 + min_element

3 12

3 X 101 + 12 = 317

317 / 100 = 3

317 % 100 = 17



M2

```
stack<int> st1;
stack<int> st2;
void Push(int x)
{
    if(st1.empty())
        st1.push(x*101+x);
    else
        st1.push(x*101 + min(x, st1.top()
                              % 101));
}
int pop()
{
    if(st.empty())
        return -1;
    else
    {
        int element = st1.top()/101;
        st1.pop;
        return element;
    }
}
int getmin()
{
    if(st1.empty())
        return -1;
    else
        return st1.top()%101;
}
```

↳ work on limited range questions.

→ Maximum of Minimum for every Window Size :

10	20	15	50	10	70	30

K = 1 → {10}, {20}, {15}, {50}, {10}, {70}, {30} ⇒ max = 70.

K = 2 → {10, 20}, {20, 15}, {15, 50}, {50, 10}, {10, 70}, {70, 30} ⇒ max = 30

K = 3 → {10, 20, 15}, {20, 15, 50}, {15, 50, 10}, {50, 10, 70}, {10, 70, 30} ⇒ max = 15

K = 4 → {10, 20, 15, 50}, {20, 15, 50, 10}, {15, 50, 10, 70}, {50, 10, 70, 30} ⇒ max = 10

K = 5 → {10, 20, 15, 50, 10}, {20, 15, 50, 10, 70}, {15, 50, 10, 70, 30} ⇒ max = 10

K = 6 → {10, 20, 15, 50, 10, 70}, {20, 15, 50, 10, 70, 30} ⇒ max = 10

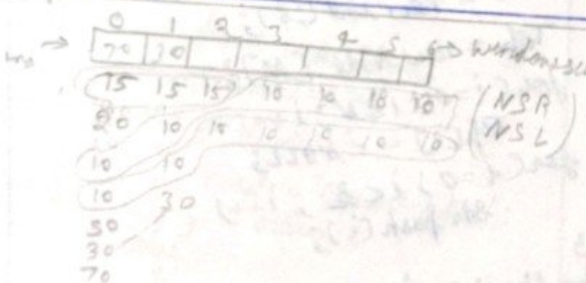
K = 7 → {10, 20, 15, 50, 10, 70, 30} ⇒ max = 10

(n-1)

(M1 = brute force)

```
vector<int> ans(n,0);
for(i=0; i<n; i++)
{
    for(j=0; j<n-i; j++)
    {
        int num = INT_MAX;
        for(k=j; k<j+i+1; k++)
        {
            num = min(num, arr[k]);
        }
        ans[i] = max(ans[i], num);
    }
}
return ans;
```

→ T.C. ⇒ O(n³)

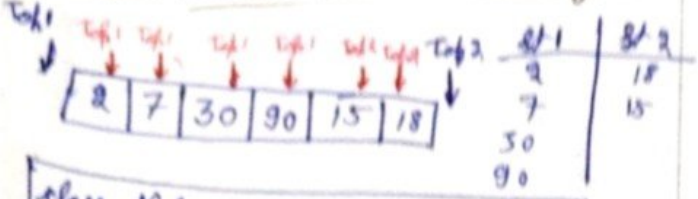


(M2)

```
vector<int> ans(n,0);
stack<int> st;
for(i=0; i<n; i++)
{
    while(!st.empty() && arr[st.top()] < arr[i])
    {
        int index = st.top();
        st.pop();
        if(st.empty())
        {
            range = i;
            ans[range-1] = max(ans[range-1], arr[index]);
        }
        else
        {
            range = i - st.top() - 1;
            ans[range-1] = max(ans[range-1], arr[index]);
        }
    }
    st.push(i);
}
while(!st.empty())
{
    int index = st.top();
    st.pop();
    if(st.empty())
    {
        range = n;
        ans[range-1] = max(ans[range-1], arr[index]);
    }
    else
    {
        range = n - st.top() - 1;
        ans[range-1] = max(ans[range-1], arr[index]);
    }
}
for(i=n-2; i>=0; i--)
    ans[i] = max(ans[i], ans[i+1]);
return ans;
```

T.C. ⇒ O(n)

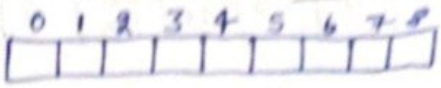
Implement two stack in an array



st 1	st 2
2	18
7	15
30	
90	

```
class NStack
{
    Public:
    int *arr;
    int top1, top2;
    int size;
    NStack (int n)
    {
        size = n;
        arr = new int[n];
        top1 = -1;
        top2 = n;
    }
    void push1(int x)
    {
        if (top1+1 == top2)
            return;
        top1++;
        arr[top1] = x;
    }
    void push2(int x)
    {
        if (top2-1 == top1)
            return;
        top2--;
        arr[top2] = x;
    }
    int pop1()
    {
        if (top1 == -1)
            return -1;
        int elem = arr[top1];
        top1--;
        return element;
    }
    int pop2()
    {
        if (top2 == size)
            return -1;
        int element = arr[top2];
        top2++;
        return element;
    }
```

N stack in an array.



Let stack = 3 space = 9

store top indexes (modify)

(Top is a ptr which is pointing to a ptr (index) store Node type values)

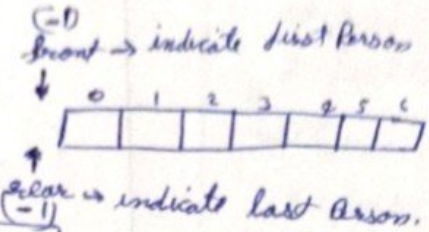
```
// Create Node (same as linked list)
class Node
{
    Public:
    int index;
    Node* next;
    Node (int x)
    {
        index = x;
        next = NULL;
    }
};
class NStack
{
    Public:
    int *arr; // store original data
    Node **Top; // Top element's index of particular stack
    stack<int> st; // Empty indexes (store)
    NStack (int N, int S) // size of array S
    {
        arr = new int[S];
        Top = new Node*[N];
        for(i=0; i<N; i++)
            Top[i] = NULL;
        for(i=0; i<S; i++)
            st.push(i);
    }
    bool push (int x, int m)
    {
        if st.empty())
            return 0;
        arr[st.top()] = x;
        Node *temp = new Node(st.top());
        temp -> next = Top[m-1];
        Top[m-1] = temp;
        st.pop();
        return 1;
    }
    int Pop()
    {
        if (Top[m-1] == NULL)
            return -1;
        st.push = Top[m-1] -> index;
        int element = arr[Top[m-1] -> index];
        Top[m-1] = Top[m-1] -> next;
        return element;
    };
```

queue →

FCFS (first in first out)

Operations in queue
1. Push
2. Pop
3. front
4. isfull
5. isEmpty

→ Implementation queue using Array:-




```
class queue
{
  int * arr;
  int front, rear;
  int size;
  Public:
  Queue (int n)
  {
    arr = new int[n];
    front = -1; rear = -1;
    size = n
  }
  bool IsEmpty()
  {
    return front == -1;
  }
  bool IsFull()
  {
    return rear == size-1;
  }
  void push (int x)
  {
    if (IsEmpty())
    {
      front = rear = 0;
      arr[0] = x;
    }
    else if (IsFull())
    {
      cout << "Queue Overflow";
      return;
    }
    else
    {
      rear = rear + 1;
      arr[rear] = x;
    }
  }
  void Pop()
  {
    if (IsEmpty())
    {
      cout << "Queue Underflow";
      return;
    }
    else
    {
      if (front == rear)
        front = rear = -1;
      else
        front = front + 1;
    }
  }
  int stack()
  {
    if (IsEmpty())
    {
      cout << "Queue is Empty";
      return -1;
    }
    else
      return arr[front];
  }
};
```

⇒ Circular Queue as array:

- front = (front + 1) % size;
- rear = (rear + 1) % size;

circular Queue.

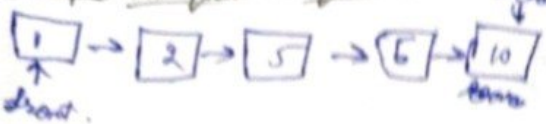
⇓

return (rear + 1) % size == front;

rear = (rear + 1) % size;

front = (front + 1) % size.

→ Implement queue using linked list




```
class Node {
    // create linked list
    code;
};

class Queue
{
    Node* front;
    Node* rear;
    Public:
    Queue ()
    {
      front = rear = NULL;
    };
    bool IsEmpty()
    {
      return front == NULL;
    }
    void Push (int x)
    {
        if( Is Empty())
        {
            front = new Node (x);
            rear = front;
        } return;
        else
        {
            rear -> next = New Node (x);
        } rear = rear -> next;
    }
    void Pop ()
    {
        if (Is Empty())
        {
            cout << "Queue Underflow";
            return;
        }
        else
        {
            Node * temp = front
            front = front -> next;
        } } delete temp;

    int start ()
    {
        if (Is Empty())
        {
            cout << "Queue is Empty";
            return -1;
        }
        else
        {
            return front -> data;
        }
    }
```

→ STL : Queue

① Push
② Pop
③ Size
④ front
⑤ Empty.
⑥ back.

```
queue <int> q;
q. Push (5);
q. Pop ();
```

→ Print all element in queue :→

q → | 2 | 4 | 6 | 3 | 5 |

Approach 1

```
Vector<int> ans;
while(!q.empty())
{
  cout << q.front();
  ans.Push_back(q.front());
}q.Pop();
for(i=0; i<ans.size(); i++)
    q.push(ans[i]);
```

Approach 2

```
int n = q.size();
while(n--)
{
  cout<<q.front();
  q.push(q.front());
}q.pop();
```

→ Reverse first K element of queue:

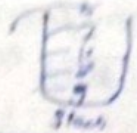
3	7	10	13	8	5	4
0	1	2	3	13	8	5	4

① Push first K element of queue in stack.
② Store the size of remain queue.(n)
③ Push all element of stack into queue.
④ Push first n element of queue at last.

```
stack<int> st;
while(k--)
{
   st.push(q.front())
   q.Pop();
}
int n = q.size();
while((st.empty()))
{
    q.push(st.top());
    st.pop();
}
while(n--)
{
    q.push(q.front());
}  q.pop();
```

→ Queue Reversal:

9	8	11	7	5




```
stack int > st;
while(!q.empty())
{
   st.push(q.front());
   q.pop();
}
while(!st.empty())
{
   q.push(st.top());
   st.pop();
}
```

→ Time needed to buy tickets :→

0	1	2	3	4
1	5	2	3	7

Ticket[i] — K=2 — i — j — ticket

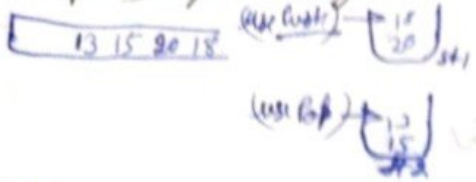
K=2
Time need to buy all ticket for person ②.
He need 2 (ticket)

1 sec for 1 ticket.
1 ticket to 1 Person at a time

① Push all index in queue.
② Time = 0 sec;
③ Give ticket & Push to back by subtract 1 (ticket--)
④ Time = increase by 1/
⑤ Ticket requirement which is required more ticket.

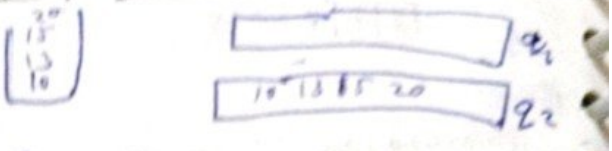
```
queue<int> q; //n = ticket.size();
for(i=0; i<n; i++)
    q.push(i);
int time =0;
while(Ticket[k]!=0)
{
    Ticket[q.front()]--;
    if(Ticket[q.front()])
        q.push(q.front());
q.Pop()
   Time++;
}
return Time;
```

→ Implement queue using stack :-




```
class Queue
{
    stack<int> st1;
    stack<int> st2;
public:
bool empty()
{
return st1.empty() && st2.empty();
void push(int x)
{
    st1.push(x);
int pop()
{
    if(empty())
        return 0;
    if(!st2.empty())
    {
        int element = st2.top();
        st2.pop();
        return element;
    }
    else
    {
        while(!st1.empty())
        {
            st2.push(st1.top());
            st1.pop();
        }
        int element = st2.top();
        st2.pop();
        return element;
    }
}
int peek()
{
    if(empty())
        return 0;
    if(!st2.empty());
        return st2.top();
    else
    {
        while(!st1.empty())
        {
            st2.push(st1.top());
            st1.pop();
        }
        return st2.top();
    }
};
```

→ Implement stack using queue :-



Push → Jo queue exist karti ho

Pop → Jis queue me order element ka sahi ho (shift to another queue & delete last element)

```
class stack
{
    queue<int> q1;
    queue<int> q2;
public:
bool empty()
{
    return q1.empty() && q2.empty();
void push(int x)
{
    if(empty())
        q1.push(x);
    else if(q1.empty())
        q2.push(x);
    else
        q1.push(x);
int pop()
{
    if(empty())
        return 0;
    else if(q1.empty())
    {
        while(q2.size()>1)
        {
            q1.push(q2.front());
            q2.pop();
        }
        int element = q2.front();
        q2.pop();
        return element;
    }
    else
    {
        while(q1.size()>1)
        {
            q2.push(q1.front());
            q1.pop();
        }
        int element = q1.front();
        q1.pop();
        return element;
    }
int top()
{
    if(empty())
        return 0;
    else if(q1.empty())
        return q2.back;
    else
        return q1.back;
```

→ Print all number in every window of size K →

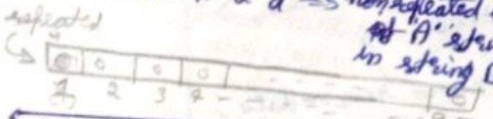
0	1	2	3	4	5
3	6	2	7	8	11

① Push that index element
② Print the queue.
③ Pop.

```
void display(queue<int> q)
{
    while (!q.empty())
    {
        cout << q.front() << " ";
        q.pop();
    }
    cout << endl;
}
queue < int > q
for (i=0; i<k-1; i++)
    q.push(arr[i]);
for(i=k-1; i<n; i++)
{
    q.push(arr[i]);
    display(q);
    q.pop();
}
```

→ first non repeating char in a stream of character.

A = "a b a b d c"
B = "a a b # d d" → first non repeated char of A' string placed in string B.




```
string B = "";
vector<int> repeated(26,0)
queue < char > q;
for (i=0; i<A.size(); i++)
{
    if (repeated[A[i]-'a']>=1)
        repeated[A[i]-'a']++;
        while(!q.empty() && repeated
                [q.front()-'a']>1)
            q.pop();
        if (q.empty())
            B += '#';
        else
            B += q.front();
    else {
        repeated[A[i]-'a']++;
        q.push(A[i]);
    }
```

→ first negative integer in every window of size k

A → | 2 | -3 | -4 | -2 | 7 | 8 | 9 | -10 |

-3 -3 -4 -2 0 -10

① Push the element in queue
② Print first negative element in queue.
③ Pop.

(M1)

```
queue<int> q
for (i=0; i<k-1; i++)
    q.push(A[i]);
vector<int> ans;
for(i=k-1; i<n; i++)
{
    q.push(A[i]);
    ans.push_back(display(q));
    q.pop();
}
return ans.
```

```
int display(queue<int> q)
{
    while (!q.empty())
    {
        if (q.front() < 0)
            return q.front();
        q.pop();
    }
    return 0;
}
```

(M2)

2	-3	-4	-2	7	8	9	-10

store indexes of -ve number in queue of array

① Push -ve num. indexes
② Pop the first element if number index of queue < k
③ Print first -ve num.

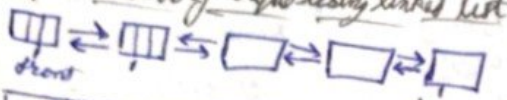
```
queue<int> q;
for(i=0; i<k-1; i++)
{
    if (A[i] < 0);
        q.push(i);
}
vector<int> ans;
for (i=k-1; i<n; i++)
{
    if (A[i] < 0)
        q.push(i);
    if (q.empty())
        ans.push_back(0);
    else
    {
        if(q.front() <= i-k)
            q.pop();
        if(q.empty())
            ans.push_back(0);
        else
            ans.push_back(A[q.front()]);
    }
}
return ans;
```

```
while(repeated[q.front()-'a']>1)
{
    q.pop()
}
B += q.front();
}
return B;
```

→ Dequeue :-

1. push - front.
2. push - back.
3. pop - front.
4. pop - back.
5. start.
6. end.

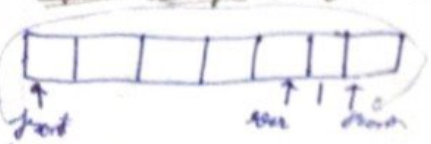
→ Implementation of Dequeue using linked list.




```
class Node
{
};// create a dubly linked list
class Dequeue
{
  Node *front, *rear;
  Public:
  Dequeue()
    { front = rear = NULL;
    }
  Push - front (int x)
  { if (front = NULL)
      { front = rear = new Node (x);
      } return ;
    else
      { Node *temp = new Node (x);
        temp -> next = front;
        front -> Prev = temp;
        front = temp;
      } return ;
  }
  void push - back (int x)
  {
    if (front = = NULL)
      { front = rear = new Node (x);
      } return ;
    else
      { Node *temp = new Node (x);
        rear -> next = temp ;
        temp -> Prev = rear;
        rear = temp;
      } return ;
  }
  void pop - front ()
  {
    if (front = = NULL)
    { cout << "queue underflow";
      } return ;
    else
      { Node *temp = front;
        front = front -> next;
        delete temp;
        if (front)
          front -> prev = NULL;
        else
          } rear = NULL;
  }
```

```
void pop - back()
{
    if (front = = NULL)
        return ;
    else
      { Node* temp = rear;
        rear = rear -> prev;
        delete temp;
        if (rear)
            rear -> next = NULL;
        else
  } }     front = NULL;

int start ()
{
    if (front = = NULL)
        return -1;
    else
        return front -> data;
}
int end ()
{
    if (front = = NULL)
        return -1;
    else.
        return rear -> data ;
  }
};
```

⇒ Implement Dequeue Using Array



↳ make circular array:

- front = (front + 1) % size ; (for pop front)
- front = (front - 1 + size) % size ;
  ↳ for pop back when front is at 0
- full ⇒ front == (rear + 1) % size .

```
class Dequeue
{
    int front, rear, size;
    int * arr ;
    public :
    Dequeue (int n)
    {
       size = n;
       arr = new int [n];
       front = rear = -1;
    }
    bool IsEmpty()
    {
       return front == -1;
    }
    bool IsFull ()
    {
       return (rear+1) % size == front;
    }
    void push_front (int x)
    {
        if ( IsEmpty() )
        {
            front = rear = 0;
            arr[0] = x;
        } return ;
        else if ( IsFull() )
        {
            return ;
        }
        else
        {
            front = (front - 1 + size) % size;
            arr[front] = x;
        } return ;
    }
    void push_back (int x)
    {
        if ( IsEmpty() )
        {
            front = rear = 0;
        } arr [0] = x;
        else if ( IsFull() )
        {
            return ;
        }
        else
        {
            rear = (rear+1) % size ;
            arr [rear] = x
        } return ;
```

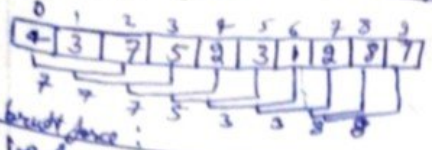
```
    void pop_front ()
    {
        if (IsEmpty ())
            return ;
        else
        {
            if (front == rear)
                front = rear = -1;
            else
                front = (front + 1) % size;
        }
    }
    void pop_back ()
    {
        if ( IsEmpty())
            return;
        else
        {
            if (front == rear)
                front = rear = -1;
            else
                rear = (rear - (+ size)) % size;
        }; }
    int start ()
    {
        if ( IsEmpty ())
            return -1;
        else
            return arr [front];
    }
};
```

⊛ STL :- `#include <deque>`

```
deque <int> d;
d. push_back (x);
d. push_front (x);
d. pop_front ();
d. pop_back ();
d. front ();
d . back ();
```

→ Sliding Window Maximum :-

K=3



brute force :

```
vector<int> ans;
for (i=0; i<n-K; i++)
{
    int total = INT_Min;
    for (j=i; j<i+K; j++)
    {
        total = max (total, nums[j]);
    }
    ans.push_back (total);
}
return ans;
```

(M2)

```
deque <int> d;
vector<int> ans;
for (i=0; i<K-1; i++)
{
    if(d.empty ())
        d.push_back (i);
    else
    {
        while(!d.empty && nums[i] > nums[d.back()])
            d.pop_back();
        d.push_back (i);
    }
}
for (i=K-1; i<n; i++)
{
    while(!d.empty && nums[i] > nums[d.back()])
        d.pop_back();
    d.push_back (i);
    if (d.front() <= i-K)
        d.pop_front();
    ans.push_back (nums[d.front()]);
}
return ans;
```

→ Minimum Number of K consecutive flip bits :-

→ all members should be '1'.

K=3 → flip = 0 K 2

K=2 → flip = 0 1 → return -1

can't convert all element in 1

brute force

T.C. = O(n K)

```
int flip = 0;
for (i=0; i<n; i++)
{
    if(nums[i] == 0)
    {
        for(j=i; j<=i+K-1; j++)
            nums[j] = (!nums[j]);
        flip++;
    }
}
return flip;
```

(if (i+K-1 >= n) return -1;)

flip = 1 2 3 3 2 1

1 → no. of flip odd = 0.
no of flip even = 1.

0 → no of flip odd = 1.
no of flip even = 0.

```
queue <int> q;
int flip = 0
for (i=0; i<n; i++)
{
    if(!q.empty && q.front()<i)
        q.pop();
    if (q.size()%2 == nums[i])
    {
        if(i+K-1 >= n)
            return -1;
        q.push(i+K-1);
        flip++;
    }
}
return flip;
```

nums[i] = 1 ← even = 1, odd = 0

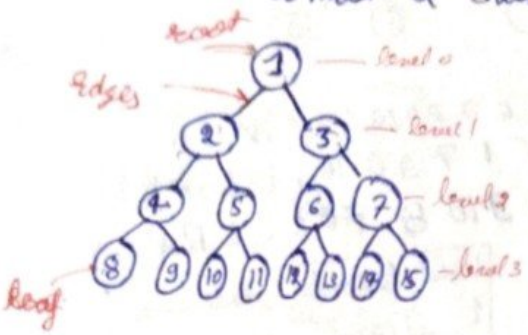
nums[i] = 0 ← even = 0, odd = 1

(q.size()%2 == nums[i])

Tree

- It is a type of Data structure that represent a hierarchial relationship between data elements called nodes.

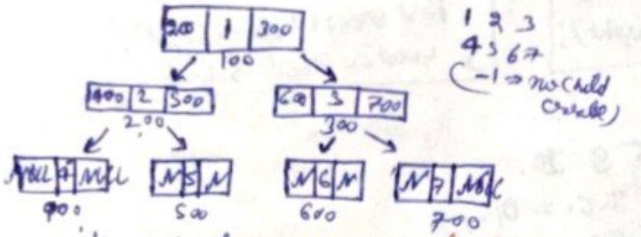
- Binary tree :- It is defined as a tree Data structure where each node has almost 2 children.



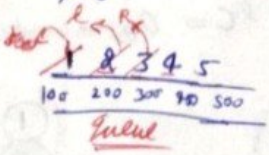
Node
Root
Parent
child
leaf
Ancestor

Descendent
level
Sibling
Height
Degree
Edges.

if n nodes → "(n-1) edges"



if x = -1, return NULL
① Create root node.
② left child.
③ Right child.

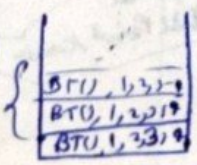


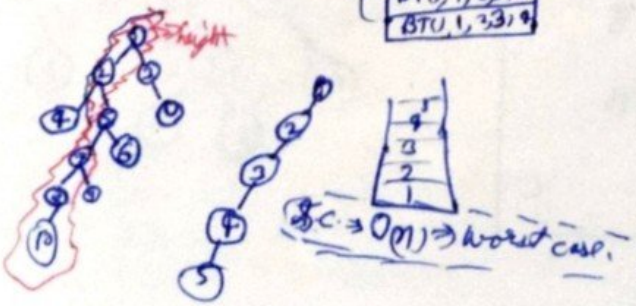

```
class Node
{
    Public :
    int data;
    Node * left, Node* Right;
    Node (int value)
    {
        data = value;
        left = Right = NULL;
    }
};
int main()
{
    int x ; first, second;
    cin >> x;
    queue <Node *> q; // store address.
    Node *root = new Node (x);
    q. push (root);
    while (! q.empty ())
    {
        Node * temp = q. front ();
        q. pop ();
        cin >> first; // left
        if (first != -1)
        {
            temp -> left = new Node (first);
            q. push (temp -> left);
        }
        cin >> second;
        if (second != -1)
        {
            temp -> Right = new Node (second)
            q. push (temp -> Right);
        }
    }
```

```
class Node
{
    // above class node same code
};
Node * Binary Tree ()
{
    int x;
    cin >> x;
    if ( x == -1)
        return NULL;
    Node *temp = new Node (x);
    temp -> left = Binary Tree (); // left side
    temp -> right = Binary Tree (); // Right side
    return temp;
}
```

T.C. ⇒ O(n)

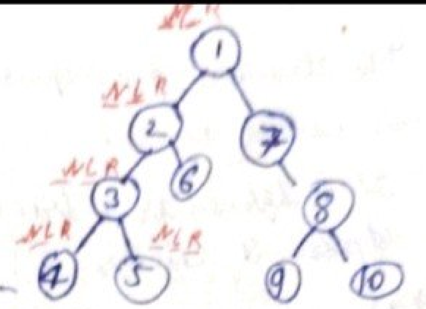
S.C. → Height of tree (O(h))





Traversal

① Pre-order Traversal (NLR)
② In-order Traversal (LNR)
③ Post-order Traversal (LRN)



Pre :- 1 2 3 4 5 6 7 8 9 10 (NLR)
In ⇒ 4 3 5 2 6 1 7 9 8 10. (LNR)
Post ⇒ 4 5 3 6 2 9 10 8 7 1 (LRN).

```
void Preorder(Node *root)
{
    if(root == NULL)
        return;
    cout<< root -> data;
    Preorder(root -> left);
    Preorder(root -> Right);
}
```

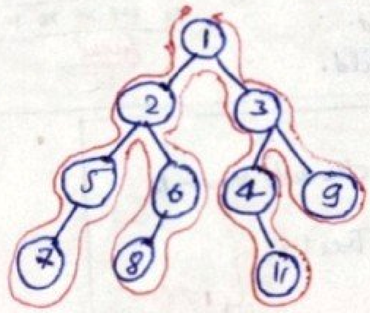
```
void Inorder(Node *root)
{
    if(root == NULL)
        return;
    Inorder(root -> left);
    cout<< root -> data;
    Inorder(root -> right);
}
```

```
void Postorder(Node *root)
{
    if(root == NULL)
        return;
    Postorder(root -> left);
    Post order(root -> Right);
    cout<< root -> data;
}
```

④ level-order Traversal (- 1 2 7 3 6 8 4 5 9 10.

```
vector <int> LevelOrder (Node *root)
{
    queue< Node *> q;
    q. push(root);
    vector<int> ans;
    Node * temp;
    while(! q empty())
    {
        temp = q.front();
        q. pop();
        ans.push_back(temp->data);
        if(temp -> left)
            q. push(temp->left);
        else
            q. push(temp -> Right);
    }
    return ans;
}
```

T.C. = O(n)
S.C ⇒ O(n) ⇒ worst case.



Pre-order: first time dekha to Print kara do.
1 2 5 7 6 8 3 4 11 9

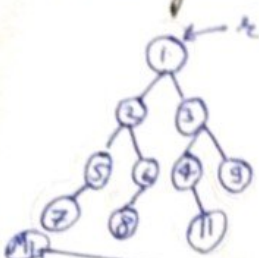
In Order: second time dekhne pe Print karo
7 5 2 8 6 1 4 11 3 9 1.

4 ko left se (-1) to 4 second time dekha

Post Order :- 3rd time visit pe Print karo
7 5 8 6 2 11 4 9 3 1.

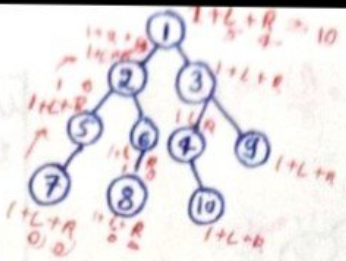
Jis node ke pass wapis kabhi nhi aa sakte usse Print karo

- Size of Binary tree :-



(M1)

```
void Total(Node *root, int& count)
{
    if(root == NULL)
        return;
    count++;
    Total(root->left, count);
    Total(root->right, count);
}
```



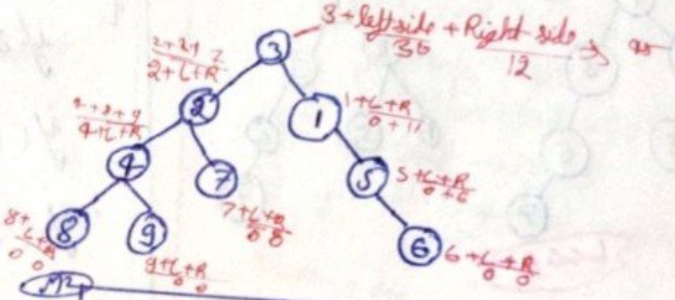
(M2)

```
int TotalNode (Node * root)
{
    if (root == NULL)
        return 0;
    return (1 + TotalNode(root->left) +
              Total Node(root->right));
}
```

- Sum of Binary Tree :-

(M1)

```
Void Total(Node*root, int& sum)
{
    if(root == NULL)
        return;
    sum += root->data;
    Total(root->left, sum);
    Total(root->right, sum);
}
```



(M2)

```
int Total(Node *root)
{
    if(root == NULL)
        return 0;
    return(root->data + Total(root->left)
              + Total(root->right));
}
```

→ Count leaves in Binary tree :-

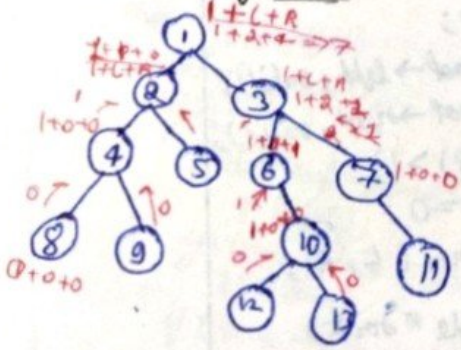
(M1)

```
void CountLeaf(Node*root, int& count)
{
    if(root == NULL)
        return;
    if(!root->left && !root->right)
    {
        count++;
        return;
    }
    CountLeaf(root->left, count);
    CountLeaf(root->right, count);
}
```

(M2)

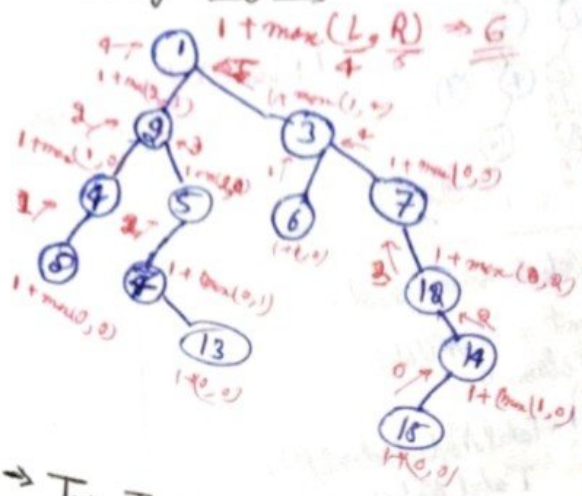
```
int Count leaf(Node*root)
{
    if(root == NULL)
        return 0;
    if(!root->left && !root->right)
        return 1;
    return(Count Leaf(root->left) +
              count Leaf(root->right));
}
```

→ Count Non leaf Nodes :-



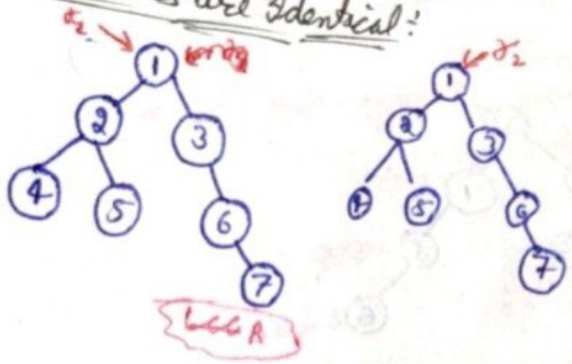

```
int NonLeaf(Node*root)
{
    if(root == NULL):
        return 0;
    if(!root->left && !root->right)
        return 0;
    return(1 + NonLeaf(root->left) +
              NonLeaf(root->right));
}
```

Height of Binary Tree



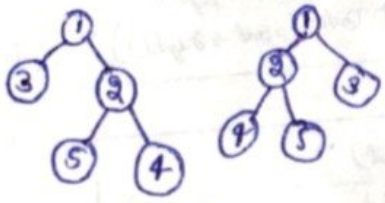

```
int height (Node *root)
{
    if (root == NULL)
        return 0;
    return (1 + max (height (root->left),
                     height (root->right));
}
```

→ Two Trees are Identical :-



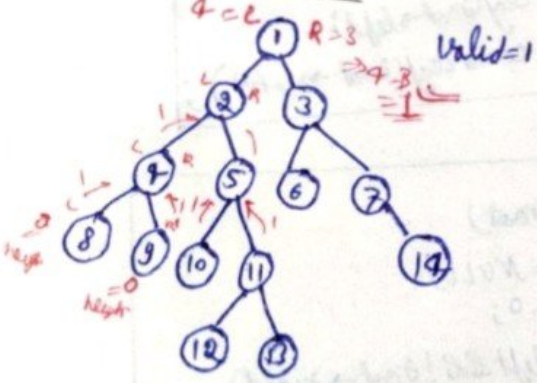

```
bool IsIdentical(Node* r1, Node *r2)
{
    if(r1==NULL && r2==NULL)
        return 1;
    if((!r1 && r2) || (r1 && !r2))
        return 0;
    if (r1->data != r2->data)
        return 0;
    return IsIdentical (r1->left, r2->left) &&
           IsIdentical (r2->right, r2->right);
}
```

→ Mirror Tree :-




```
void Mirror (Node *root)
{
    if(! root)
        return ;
    Node* temp = root->right;
    root->right = root->left;
    root->left = temp;
    Mirror (root->left);
}
    Mirror (root->Right);
```

→ Check for Balance Tree :-



cond* for balance

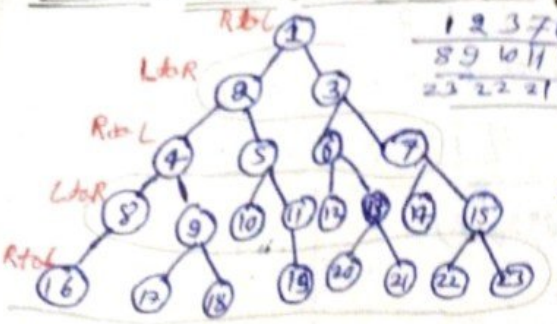
$-1 \le \text{height(left)} - \text{height(R)} \le 1$

(-1, 0, 1)

for each node

```
int height(Node* root, bool &valid)
{
    if (! root)
        return 0;
    int L = height (root->left, valid);
    int R = height (root->right, valid);
    if (abs(L - R) > 1)
        valid = 0;
    return 1 + max ( L, R);
}
bool Is balance (Node * root)
{
    bool valid = 1;
    Height (root, valid);
    return valid;
}
```

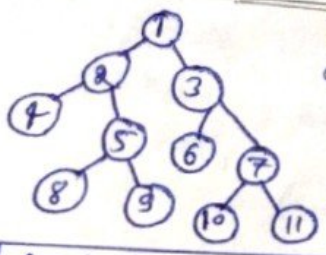
→ level order Traversal in spiral form :-



1 2 3 7 6 5 4
8 9 10 11 12 13 14 15
23 22 21 20 19 18 17 16

```
vector<int> Levelorder(Node *root)
{
    stack<Node*> S1;  // R to L
    stack<Node*> S2;  // L to R.
    S1.push(root);
    vector<int> ans;
    while(!S1.empty() || !S2.empty())
    {
        if(!S1.empty())  // R to L
        {
            while(!S1.empty())
            {
                Node *temp = S1.top();
                S1.pop();
                ans.push_back(temp->data);
                if(temp->right)
                    S2.push(temp->right);
                if(temp->left)
                    S2.push(temp->left);
            }
        }
        else  // L to R
        {
            while(!S2.empty())
            {
                Node *temp = S2.top();
                S2.pop();
                ans.push_back(temp->data);
                if(temp->left)
                    S1.push(temp->left);
                if(temp->right)
                    S1.push(temp->right);
            }
        }
    }
    return ans;
}
```

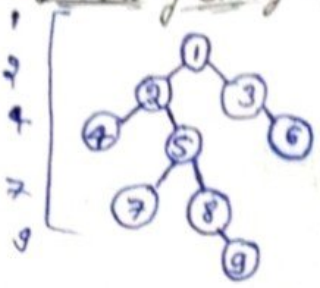
→ Check if 2 Node are cousin :-



① Same level
② Parent different

```
bool IsCousin(Node *root, int a, int b)
{
    queue<Node*> q;
    q.push(root);
    int l1 = -1, l2 = -1;
    int level = 0;
    while(!q.empty())
    {
        int n = q.size();
        while(n--)
        {
            Node *temp = q.front();
            q.pop();
            if(temp->data == a)
                l1 = level;
            if(temp->data == b)
                l2 = level;
            if(temp->left)
                q.push(temp->left);
            if(temp->right)
                q.push(temp->right);
        }
        level++;
        if(l1 != l2)
            return 0;
        if(l1 != -1)
            break;
    }
    return !Parent(root, a, b);
}

bool Parent(Node *root, int a, int b)
{
    if(!root)
        return 0;
    if(root->left && root->right)
        if(root->left->data == a && root->right->data == b)
            return 1;
        if(root->left->data == b && root->right->data == a)
            return 1;
    return Parent(root->left, a, b) ||
           Parent(root->right, a, b);
}
```

Left view of Binary Tree :-



→ Level order Traversal.

1 2 3 4 5 6

level = 0, 1, 2

n = 1, 2, 4

1 2 4 7 9

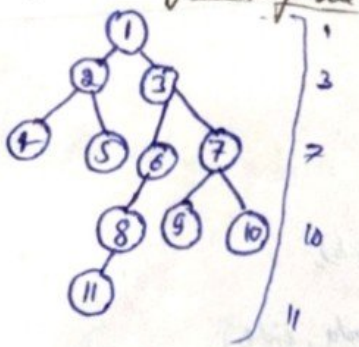
(M1)

```
vector<int> leftView(Node* root)
{
    queue<Node*> q;
    q.push(root);
    vector<int> ans;
    while(!q.empty())
    {
        int n = q.size();
        ans.push_back(q.front()->data);
        while(n--)
        {
            Node *temp = q.front();
            q.pop();
            if(temp->left)
                q.push(temp->left);
            if(temp->right)
                q.push(temp->right);
        }
    }
    return ans;
}
```

(M2)

```
void LView(Node *root, int level, vector<int> &ans)
{
    if(!root)
        return;
    if(level == ans.size())
        ans.push_back(root->data);
    LView(root->left, level+1, ans);
    LView(root->right, level+1, ans);
}
```

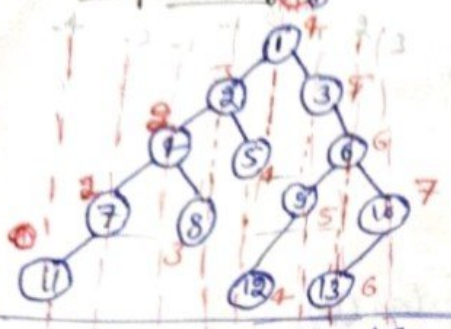
Right view of Binary Tree :-



1 3 7 10 11

[Above code same only if(temp->left) if(temp->Right) change for vice]

→ Top view of B tree



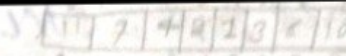
11 7 4 2 1 3 6 10

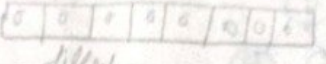
(1)

```
void find(Node* root, int hos, int &l, int &r)
{
    if (!root)
        return;
    l = min(l, hos);
    r = max(r, hos);
    find(root->left, hos-1, l, r);
    find(root->right, hos+1, l, r);
}
vector<int> Top view(Node* root)
{
    int l=0, r=0;
    find(root, 0, l, r);
    vector<int> ans(r-l+1);
    vector<bool> filled(r-l+1, 0);
    queue<Node*> q;
    queue<int> index;
    q.push(root);
    index.push(-1*l);
    while(!q.empty())
    {
        Node* temp = q.front();
        q.pop();
        int pos = index.front();
        index.pop();
        if(!filled[pos])
        {
            filled[pos] = 1;
            ans[pos] = temp->data;
        }
        if(temp->left)
        {
            q.push(temp->left);
            index.push(pos-1);
        }
        if(temp->right)
        {
            q.push(temp->right);
            index.push(pos+1);
        }
    }
    return ans;
}
```

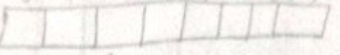
TC → O(n)

SC = O(n)





filled



ans

① left max
② Right most
③ ans = ans
R - L + 1

→ Preorder Traversal Iterative :- N L R

```
vector<int> Preorder(Node *root)
{
    stack<Node*> s;
    s.push(root);
    while(!s.empty())
    {
        Node *temp = s.top();
        s.pop();
        ans.push_back(temp->data);
        if(temp->Right)
            s.push(temp->right);
        if(temp->left)
            s.push(temp->left);
    }
    return ans;
}
```

→ Post order Traversal :- L R N

Left side →
Right side
Mid

⇒ N R L

1 3 7 6 2 5 8 4 ⇒ NRL

4 8 5 2 6 7 3 1 ⇒ LRN

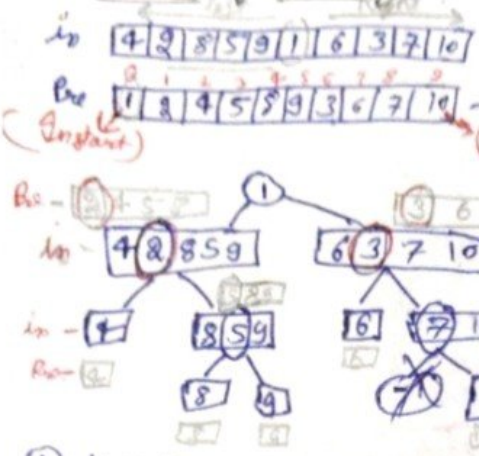
```
vector<int> PostOrder(Node *root)
{
    stack<Node*> s;
    s.push(root);
    vector<int> ans;
    while(!st.empty())
    {
        Node *temp = s.top();
        s.pop();
        ans.push_back(temp->data);
        if(temp->left)
            s.push(temp->left);
        if temp->right)
            s.push(temp->right);
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

→ In Order Traversal :- L N R

① I^st time visit → R N L
② II^nd time visit → print Data

```
vector<int> InOrder(Node *root)
{
    stack<Node*> s;
    stack<bool> visited;
    s.push(root);
    visited(0);
    vector<int> ans;
    while(!s.empty())
    {
        Node *temp = s.top();
        s.pop();
        bool flag = visited.top();
        visited.pop();
        if(!flag) // flag == 0 (first time)
        {
            if(temp->right)
            {
                s.push(temp->right);
                visited.push(0);
            }
            s.push(temp);
            visited.push(1);
            if(temp->left)
            {
                s.push(temp->left);
                visited.push(0);
            }
        }
        else // second time visited
            ans.push_back(temp->data);
    }
    return ans;
}
```

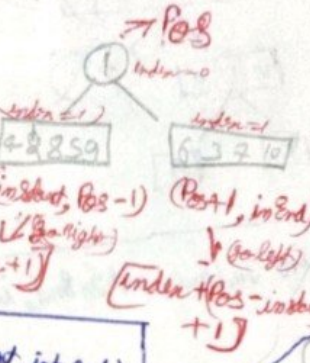
→ Construction of Tree using Inorder and Preorder :-



① find element in Pre order

② find it's Position in In order.

③ Divide in Left & Right.

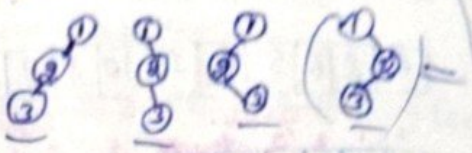



```
int find(int *in, int target, int start, int end)
{
    for(i=start; i<=end; i++)
    {
        if(in[i] == target)
            return i;
    }
    return -1;
}

Node *Tree(int *in, int *pre, int InStart,
           int InEnd, int index)
{
    if(InStart > InEnd)
        return NULL;
    Node *root = new Node(Pre[index]);
    int Pos = find(in, pre[index],
                   InStart, InEnd);
    root->left = Tree(in, pre, InStart,
                      Pos-1, index+1);
    root->right = Tree(in, pre, Pos+1, InEnd,
                       index+(Pos-InStart)+1);
    return root;
}
```

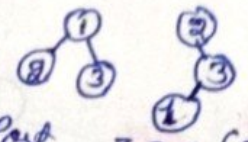
→ T.C. → O(n²)

S.C. → O(n).

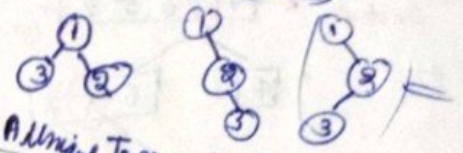
- Pre order - 1 2 3 (N L R)



In Order - 2 1 3 (L N R)



- Post - 3 2 1 (L R N)

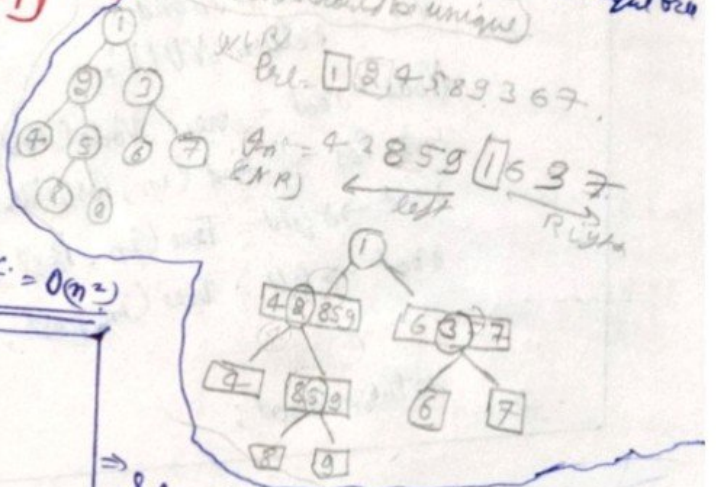


A Unique Tree can't be created using only Pre, Post & In Order Traversal

Pre - In ⇒ could create unique tree

Post - In ⇒ could create unique tree

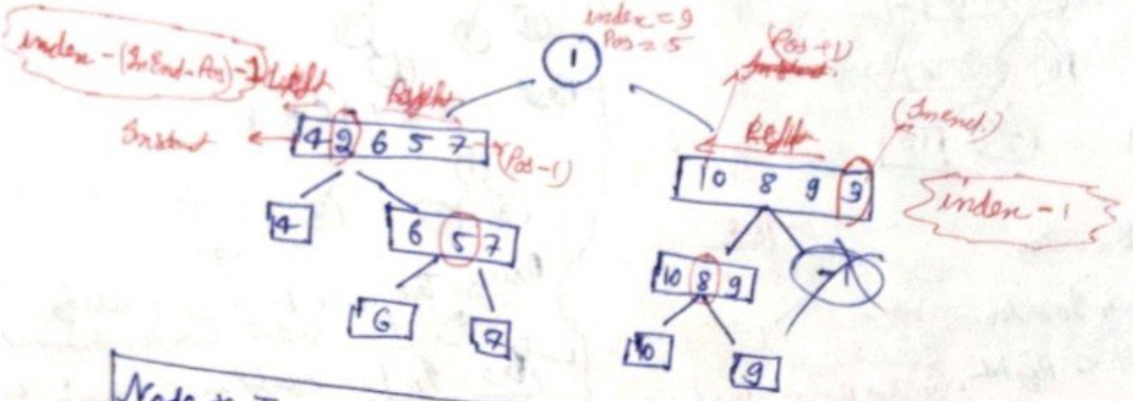
Post - Pre ⇒ can't create unique tree



- Construct a tree from inorder & postorder:

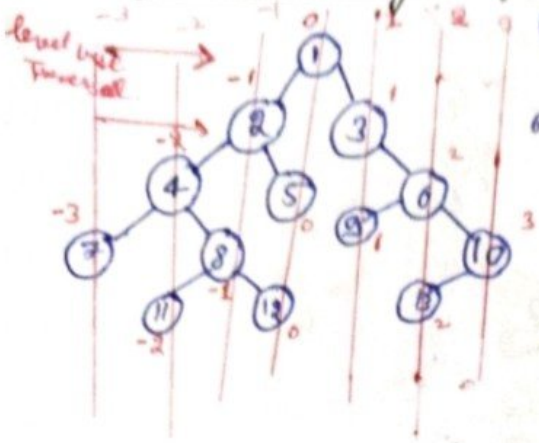
in - | 4 | 2 | 6 | 5 | 7 | 1 | 10 | 8 | 9 | 3 | LNR.

Post - | 4 | 6 | 7 | 5 | 2 | 10 | 9 | 8 | 3 | 1 | LRN.




```
Node * Tree (int *in, int * Post, int InStart, int InEnd, int index)
{
    if( InStart > InEnd)
        return NULL;
    Node *root = new Node (Post[index]);
    int Pos = find (in, Post[index], InStart, InEnd);
    root -> right = Tree (in, Post, Pos + 1, InEnd, index - 1);
    root -> left = Tree (in, Post, InStart, Pos - 1,
                         index - (InEnd - Pos) - 1);
    return root;
}
```

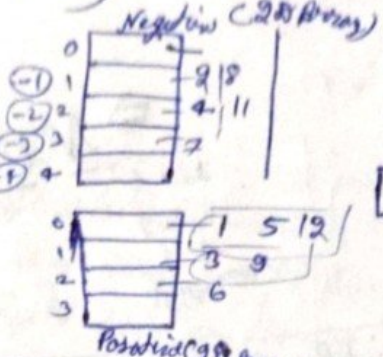
Vertical Traversal of Binary Tree



(7 4 11 2 8 1 5 12 3 9 6 13 10)

① Print lowest index

② if index is same, print low level.



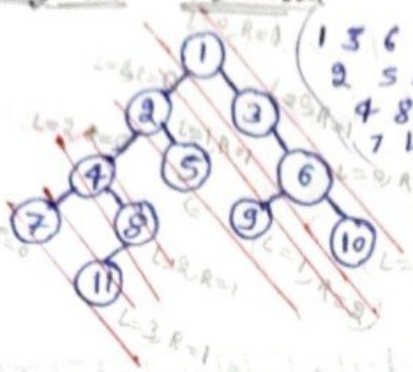
7	4	11	2	8	1	5	12	3	9	6	13	10

Positive (2D Array)

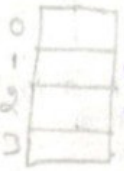
```
void find(Node * root, int Pos, int &l, int &r)
{
    if( ! root)
        return;
    l = min(l, Pos);
    r = max(r, Pos);
    find(root->left, Pos-1, l, r);
    find(root->right, Pos+1, l, r);
}

vector<int> VerticalOrder(Node* root)
{
    int l=0, r=0;
    find(root, 0, l, r);
    vector<vector<int>> Positive(r+1);
    vector<vector<int>> negative(abs(l)+1);
    queue<Node *> q;
    queue<int> index;
    q.push(root);
    index.push(0);
    while(! q.empty())
    {
        Node* temp = q.front();
        q.pop();
        int Pos = index.front();
        index.pop();
        if(Pos >= 0)
            Positive[Pos].push_back(temp->data);
        else
            Negative[abs(Pos)].push_back(temp->data);
        if(temp->left)
        {
            q.push(temp->left);
            index.push(Pos-1);
        }
        if(temp->right)
        {
            q.push(temp->right);
            index.push(Pos+1);
        }
    }
```

```
// store ans in 2D vector
vector<int> ans;
// store -ve vector
for(i = negative.size(); i>0; i++)
    for(j=0; j<negative[i].size(); j++)
        ans.push_back(negative[i][j]);
// store +ve vector
for(i=0; i<Positive.size(); i++)
    for(j=0; j<Positive[i].size(); j++)
        ans.push_back(Positive[i][j]);
return ans;
```

- Diagonal Traversal



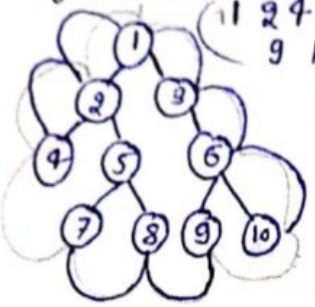
⇒ N L R

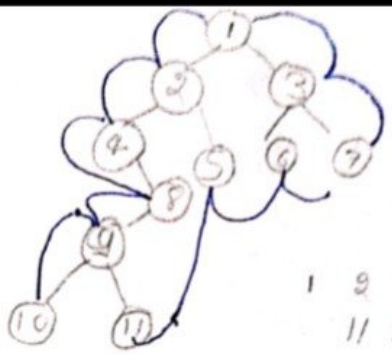



```
void find (Node * root, int Pos, int & l)
{
    if(! root)
        return;
    l = max (Pos, l);
    find (root -> left, Pos+1, l);
    find (root -> right, Pos, l);
}
void find Digo (Node * root, int Pos, vector<vector<int>> & ans)
{
    if( ! root)
        return;
    ans[Pos] . push_back (root -> data);
    find Digo (root -> left, Pos+1, ans);
    find Digo (root -> right, Pos, ans);
}
vector <int> Digonal (Node * root)
{
    int l=0;
    find (root, 0, l);
    vector < vector <int>> ans(l+1);
    find Digo (root, 0, ans);
    vector < int > temp;
    for (i=0; i < ans.size(); i++)
        for (j=0; j < ans[i].size() ; j++)
            temp . push_back (ans[i][j]);
    return temp;
}
```

→ T.C. → O(n)
S.C. → O(h) = O(n)

- Boundry Traversal ⇒

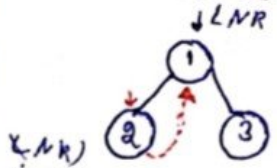


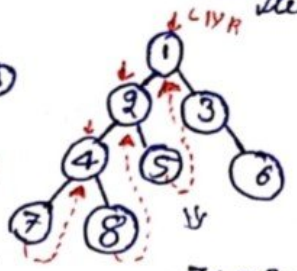


① Root node ko answer me add karo.
② left subtree ka Traverse karo, Node koadd karo, except leaf.
③ leaf node ke add karo.
④ Right subtree traverse, reverse order, except leaf node.

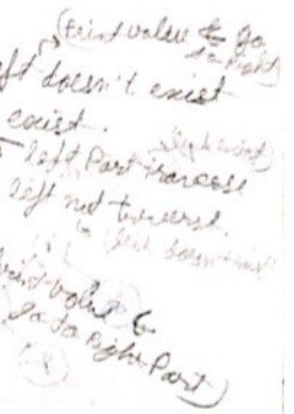
→ Morris Traversal ⇒

in-order ⊣ Recursion
Iterative (stack)] — T.C. ⇒ O(n)
S.C. ⇒ O(n)





① Node ke left me jao.
② left ke right most me jao ke us node se link karo.



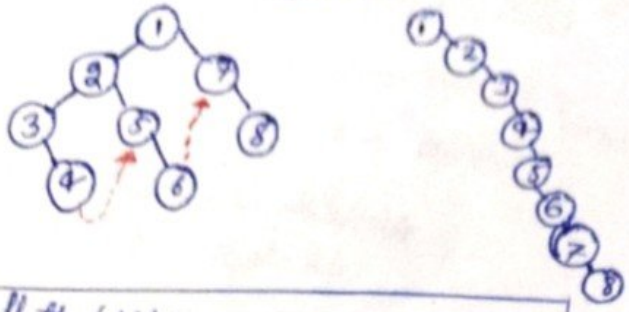
- Pseudo code:
① left doesn't Exist.
→ Note down the data
→ Move to right.
② left part exist:
→ if left subtree Not Traverse
↳ create the link.
↳ Move to left
→ else (Already traverse)
↳ Remove the link.
↳ Note down data.
↳ Move to right.

```
vector<int> inorder(Node* root)
{
    vector<int> ans;
    while (root)
    {
        if (!root->left) // left part doesn't exist.
        {
            ans.push_back(root->data)
            root = root->right;
        }
        else // left exist
        {
            Node* curr = root->left;
            while (curr->right && curr->right != root)
            {
                curr = curr->right;
            }
            // Not traverse
            if (curr->right == NULL)
            {
                curr->right = root;
                root = root->left;
            }
            else // Traverse
            {
                curr->right = NULL;
                ans.push_back(root->data);
                root = root->right;
            }
        }
    }
    return ans.
}
```

↳ T.C. ⇒ O(n).
S.C ⇒ O(1).

→ Flatten Binary to linked list :-

Preorder → 1, 2, 3, 4, 5, 6, 7, 8



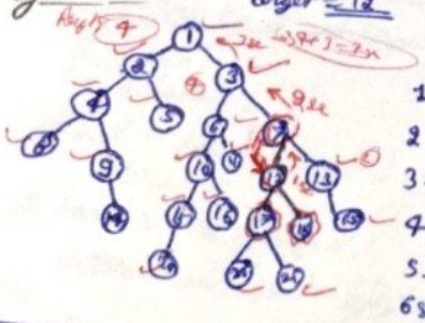

```
void flatten(Node* root)
{
    vector<int> ans;
    while(root)
    {
        if(!root->left)
            root = root->right;
        else
        {
            Node* curr = root->left;
            while (root->right)
                curr = curr->right;
            curr->right = root;
            root->left = root->right;
            root->left = NULL;
        } root = root->right;
    }
}
```

if left doesn't exist :
root = root → right

else (left exist) :
- curr pointer → root → left
- curr ka right most point.
- curr ka link, banao root ka right se
- root ke left ka root ka right bana do.
- root ke left ko null kar do.

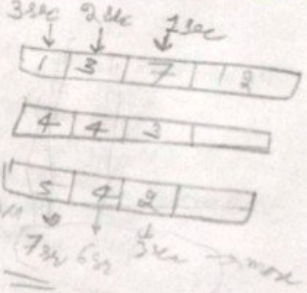
→ Burning Tree :-

Target = 18



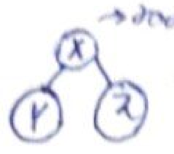
Time	Burned
1 sec	⇒ 7, 17, 18
2 sec	⇒ 3, 13, 20, 21
3 sec	⇒ 1, 6, 19
4 sec	⇒ 2, 10, 11
5 sec	⇒ 5, 4, 15, 16
6 sec	⇒ 8, 9, 20,
7 sec	⇒ 14

7 sec to burn whole tree.

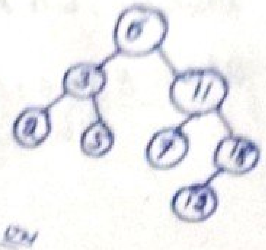


```
int Burn(Node *root, int &Timer, int Target)
{
    if(!root)
        return 0;
    if(root->node == target)
        return -1;
    int left = Burn(root->left, timer, target);
    int Right = Burn(root->right, timer, target);
    if(left < 0)
    {
        Timer = max(Timer, abs(left)+Right);
        return left-1;
    }
    if(Right < 0)
    {
        Timer = max(Timer, left+abs(Right));
        return Right-1;
    }
    return 1 + max(left, Right);
}
```

Binary Search tree



6 | 3 | 11 | 5 | 7 | 18 | 12 | 2 | 11 ⇒



① X = = Y ⇒ Right side
X = = Y ⇒ Left side } → choose any

Y < X < Z
↳ ① Root > value → left side.
② Else :- Right side.

```
Node * insert (Node * root, int target)
{
    if (! root)
    {
        Node * temp = new Node (target);
        return temp;
    }
    if (target < root → data) // left.
        root → left = insert(root → left, target);
    else
        root → right = insert (root → right, target);
    return root;
}
int main ()
{
    int arr [ ] = {3, 7, 4, 1, 6, 8};
    Node * root = NULL;
    for (i = 0; i < 6; i++)
        root = insert (root, arr[i]);
}
```

→ T.C. ⇒ O(h).
→ Worst case = O(n).

→ S.C. → O(n)

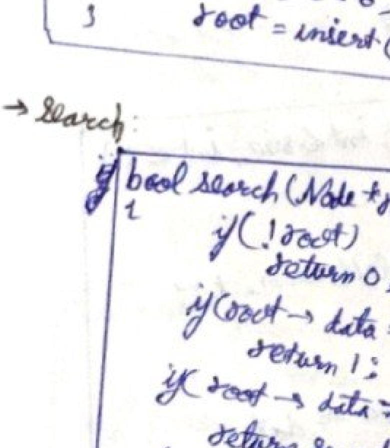
→ Search:

```
bool search (Node * root, int target)
{
    if (! root)
        return 0;
    if (root → data == target)
        return 1;
    if (root → data > target)
        return search (root → left, target);
    else
        return search (root → right, target);
}
```

↳ T.C. → O(h)

```
Node * deleteNode (Node * root, int target)
{
    if (! root)
        return NULL;
    if (root → data > target)
    {
        root → left = delNode (root → left, target);
        return root;
    }
    else if (root → data < target)
    {
        root → right = delNode (root → right, target);
        return root;
    }
    else
    {
        // leaf Node
        if (! root → left && ! root → right)
        {
            delete Node;
            return NULL;
        }
        if (! root → left) // Right child exist
        {
            Node * temp = root → right;
            delete root;
            return temp;
        }
        else if (! root → right) // left child exist
        {
            Node * temp = root → left;
            delete root;
            return temp;
        }
        else
        {
            // find greatest element from left
            Node * child = root → left;
            Node * Parent = root;
            while (child → right) // reach rightmost Node
            {
                Parent = child;
                child = child → right;
            }
            if (root ! = Parent)
            {
                Parent → right = child → left;
                child → left = root → left;
                child → right = root → right;
                delete root;
                return child;
            }
            else
            {
                child → right = root → right;
                delete root;
                return child;
            }
        }
    }
}
```

- Delete Node ⇒



① if leaf Node
↳ delete Node
↳ return NULL.

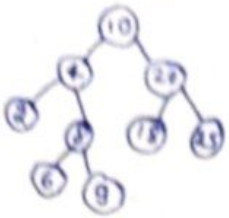
② if Only one child
↳ if left exist.
↳ delete node
↳ return left child.
↳ if Right exist.
↳ delete Node
↳ return Right child.

③ if both child exist.
↳ find greatest element in left (named child)
↳ Parent ke right ka Bolo child ke left ko Point kare.
↳ child → left = root → left.
↳ child → right = root → right.
↳ delete node;
↳ return child.

```
if (Parent == root)
{
    child → right = root → right;
    delete root
}
return child;
```

→ Check BST

inorder : L N R



2 4 6 8 9 10 15 20 25 → Assending order.

(prev = INT_MIN)

(M1)

```
void inorder(Node* root, Node<int>& ans)
{
    if (!root)
        return;
    inorder(root->left, ans);
    ans.push_back(root->data);
    inorder(root->right, ans);
}
bool isBST(Node *root)
{
    inorder(...)
    for(int i=1, i < ans.size(); i++)
        if (ans[i] <= ans[i-1])
            return 0;
    return 1;
}
```

T.C. -> O(n).

```
bool BST(Node* root, int& prev)
{
    if(!root)
        return 1;
    bool l = BST(root->left, prev);
    if(l == 0)
        return 0;
    if(root->data <= prev)
        return 0;
    prev = root->data;
    return BST(root->right, prev);
}
```

→ Minimum Distance b/w BST Nodes :-

inorder, prev = INT_MIN, ans = INT_MAX

```
void MinDist(Node *root, int& prev, int& ans)
{
    if (!root)
        return;
    MinDist(root->left, prev, ans);
    if(prev != INT_MIN)
        ans = min(ans, root->data - prev);
    prev = root->data;
    MinDist(root->right, prev, ans);
}
```

```
int minDistBST(Node* root)
{
    int prev = INT_MIN;
    int ans = INT_MAX;
    minDist(root, prev, ans);
    return ans;
}
```

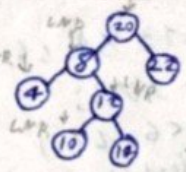
→ Kth Largest Element in BST :

K = 3rd.

L N R → assending order

R N L → decending order

→ Sum of K smallest Element in BST.



K = 3

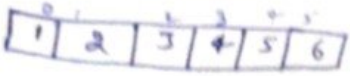
sum = 22

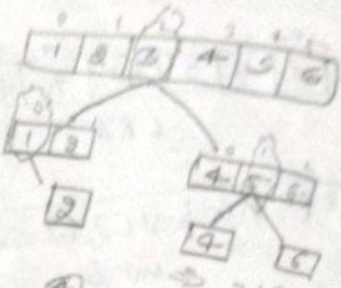
```
void KSum(Node *root, int& sum, int& K)
{
    if(!root)
        return;
    KSum(root->left, sum, K);
    K--;
    if(K >= 0)
        sum += root->data;
    if(K <= 0)
        return;
    KSum(root->right, sum, K);
}
```

```
void KLargest(Node *root, int& ans, int& K)
{
    if(!root)
        return;
    KLargest(root->right, ans, K);
    K--;
    if(K == 0)
        ans = root->data;
    if(K <= 0)
        return;
    KLargest(root->left, ans, K);
}
```

→ Array to BST

Balanced BST : $H_1 - H_2 \leq 1$.





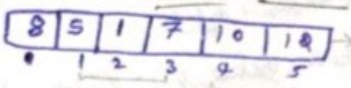
⇒ 3 1 2 5 4 6

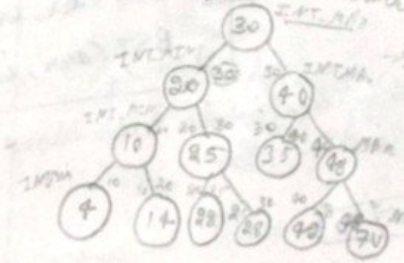
⇒ 4 2 1 3 5 6

find pre order
lexicograph asc order

```
void ArrayToBST(vector<int> arr, int start,
                int end, vector<int> &ans)
{
    if (start > end)
        return;
    int mid = (start + (end - start)/2);
    ans.push_back(arr[mid]);
    ArrayToBST(arr, start, mid-1, ans);
    ArrayToBST(arr, mid+1, end, ans);
}
int main()
{
    vector<int> ans;
    vector<int> nums;
    ArrayToBST(nums, 0, nums.size()-1, ans);
    return ans;
}
```

→ Construct BST from preorder Traversal :-






```
Node* BST(vector<int> &preorder, int &index,
          int lower, int upper)
{
    if (index == preorder.size() || preorder[index] < lower
                                 || preorder[index] > upper)
        return NULL;
    Node *root = new Node(preorder[index++]);
    root->left = BST(preorder, index, lower, root->data);
    root->right = BST(preorder, index, root->data, upper);
    return root;
}
```

→ Construct BST from Postorder :-

Post

0	1	2	3	4	5
1	7	5	50	40	10

LRN.

```
Node * BST(int post[], int &index,
           int lower, int upper)
{
    if(index < 0 || Post[index] < lower ||
                    Post[index] > upper)
        return NULL;
    Node *root = new Node(Post[index--]);
    root -> right = BST(Post, index, root -> data, upper);
    root -> left = BST(Post, index, lower, root -> data);
    return root;
}
```

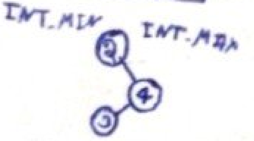
→ Preorder & BST :-

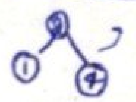
2	4	3

①

2	4	1

⓪





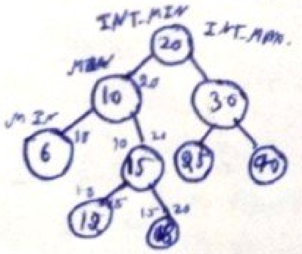
(M1)

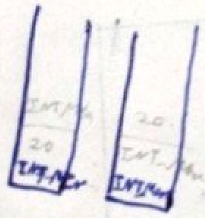
```
void BST(int arr[], int & index, int lower, int upper,
         int N)
{
    if (index == N || arr[index] < lower ||
                      arr[index] > upper)
        return;
    int value = arr[index++];
    BST(arr, index, lower, value, N); //left
    BST(arr, index, value, upper, N); //right.
}
```

- Segementation fault :- aaise memory location ko access karne ke koshish kar raha hu Jiska mujhe authority nahi hai

Preorder.

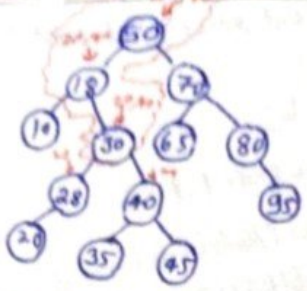
20	10	6	15	12	18	30	25	40





// Code Using stack (M2)

→ Lowest common Ancestor in BST



$n_1 = 20$

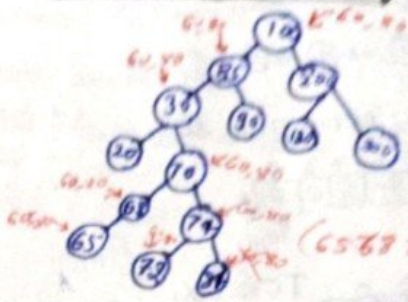
$n_2 = 40$

50 18 30 28 20

50 18 30 40

```
Node * LCA(Node *root, int n1, int n2)
{
    if(!root)
        return NULL;
    if(root->data > n1 && root->data > n2)
        return LCA(root->left, n1, n2);
    else if(root->data < n1 && root->data < n2)
        return LCA(root->right, n1, n2);
    else
        return root;
}
```

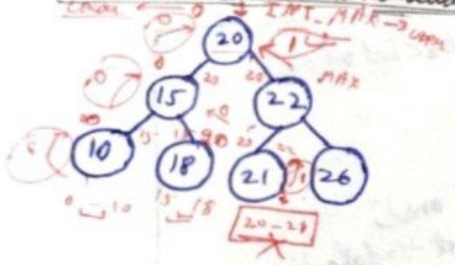
→ Print BST element in given range.



(65 67 70 73 74)

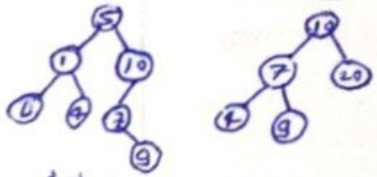
```
void find(Node *root, vector<int> &ans)
{
    if(!root)
        return;
    if(root->data > n1 && root->data > n2)
        find(root->left, ans);
    else if(root->data < n1 && root->data < n2)
        find(root->right, ans);
    else
    {
        find(root->left, ans);
        ans.push_back(root->data);
        find(root->right, ans);
    }
}
```

→ Check whether BST contains Dead end:



```
bool Dead(Node *root, int lower, int upper)
{
    if(!root)
        return 0;
    if(!root->left && !root->right)  // Leaf Node
    {
        if(root->data - lower == 1 && upper - root->data == 1)
            return 1;
        else
            return 0;
    }
    return Dead(root->left, lower, root->data) ||
           Dead(root->right, root->data, upper);
}
```

→ Common Nodes in two BST:



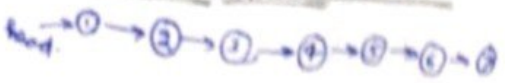
a1 - 0 1 4 5 7 9 10 → (4, 7, 9, 10)

a2 - 4 7 9 10 20

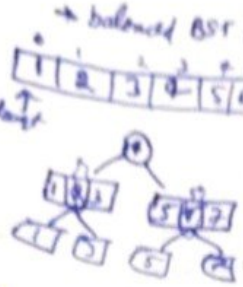
20 70 30 90 55 60 80 100 110 120 150 180 200 300

① Left most element ko stack me push karo.
② Pop the element & Print it.
③ Go to right side & push left elements into stack

Sorted linked list to BST :-



→ balanced BST ⇒ $O(n)$ $SC \Rightarrow O(n)$

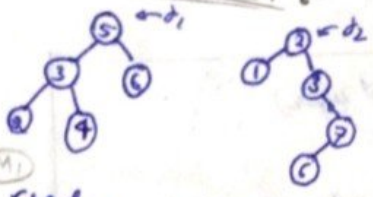


TC ⇒ $O(n)$

```
class LNode
{
    Public:
    int data;
} LNode *next;

class TNode
{
    Public
    int data;
} TNode *left, *right;
```

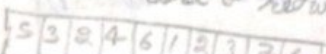
```
TNode * BuildBST(vector<int>& Tree,
                 int start, int end)
{
    if (start > end)
        return NULL;
    int mid = (start + end + 1) / 2;
    TNode *root = new TNode(Tree[mid]);
    root->left = BuildBST(Tree, start, mid-1);
    root->right = BuildBST(Tree, mid+1, end);
} return root;

TNode * createBST(LNode *head)
{
    vector<int> Tree;
    while(head)
    {
        Tree.push_back(head->data);
    } head = head->data;
    return BuildBST(Tree, 0, Tree.size()-1);
}
```

→ Merge two BST :-



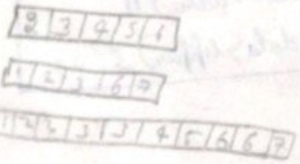
M1

create a vector store both tree elements & sort & return.



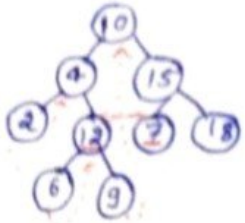
T.C ⇒ $O(n \log n)$

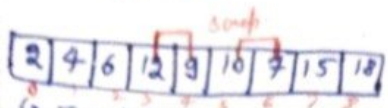
M2



```
void inorder(Node*root, vector<int>&ans)
{
    if (!root)
        return;
    inorder(root->left, ans);
    ans.push_back(root->data);
} inorder(root->right, ans);

vector<int> MergeTwoBST(Node*root1, Node*root2)
{
    vector<int> ans1;
    vector<int> ans2;
    inorder(root1, ans1);
    inorder(root2, ans2);
    vector<int> ans;
    int i=0, j=0;
    while(i<ans1.size() && j<ans2.size())
    {
        if(ans1[i] < ans2[j])
            ans.push_back(ans1[i++]);
        else
    } ans.push_back(ans2[j++]);
    while(i<ans1.size())
        ans.push_back(ans1[i++]);
    while(j<ans2.size())
        ans.push_back(ans2[j++]);
    return ans;
}
```

Fixing Two Nodes of a BST





T.C. $\Rightarrow O(n)$ $\Rightarrow O(n) < T$, $O(n) = S$

(M1) first = index/3, second = 4, third = 5, forth = 6

first = ~~*~~ (3), second = ~~*~~ (6)

(M2) Morris traversal $\Rightarrow$ S.C. $\Rightarrow O(1)$

curr = ~~4~~ 6 12 9 ~~10~~ ~~15~~

prev = ~~4~~ ~~6~~ 12 ~~9~~ ~~10~~ 7

first : 12

second : ~~9~~ 7

```
void correctBST(Node *root)
{
    Node* curr = NULL;
    Node *first = NULL, *second = NULL;
    Node *last = NULL, *present = NULL;
    while (root)
    {
        if(!root->left)
        {
            last = present;
            present = root;
            if(last && last->data > present->data)
            {
                if(!first)
                    first = last;
                second = present;
            }
            root = root->right;
        }
        else
        {
            curr = root->left;
            while(curr->right && curr->right
                                != root)
                curr = curr->right;
            if(!curr->right)
            {
                curr->right = root;
                root = root->left;
            }
            else
            {
                curr->right = NULL;
                last = present;
                present = root;
                if(last && last->data > present->data)
                {
                    if(!first)
                        first = last;
                    second = present;
                }
                root = root->right;
            }
        }
    int num = first->data;
    first->data = second->data;
    second->data = num;
}
```

Largest BST

① left BST
② Right BST
③ left side → max < root → data
④ Right side → min > root → data.
⑤ size : Left size + right side + 1.

BST:
size :
Min :
Max :

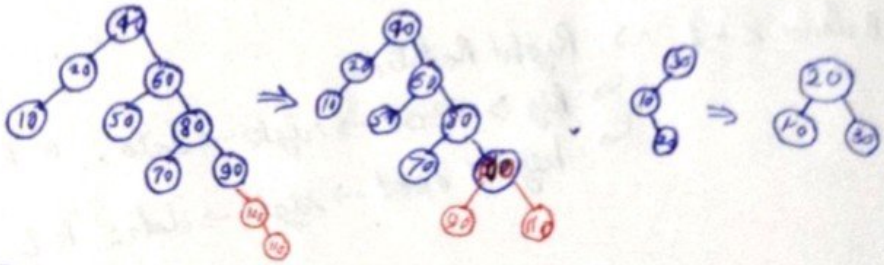
```
if (left head → BST && Right head → BST ,
    && left head → max < root → data
    && Right head → min > root → data)
{
    Box * head = new Box (root → data);
    head → size += Left head → size +
                   Right head → size;
    head → min = Left head → min;
    head → max = Right → max;
    Total size = max (Total size, head → size);
    return head;
}
else.
{
    left head → BST = 0;
    return left head;
}
}
```

```
class Box
{
    Public:
    bool BST;
    int size;
    int min, max;
    Box (int data)
    {
        BST = 1;
        size = 1;
        min = data;
        max = data;
    }
};
int total BST (Node * root)
{
    int Total size = 0;
    find (root, Total size);
    return Total size;
}
Box * find (Node * root, int & Total size)
{
    if (! root → left && ! root → right)
        return new Box (root → data);
    else if (! root → left && root → right)
    {
        Box * head = find (root → right, Total size);
        if (head → BST && head → min > root → data)
        {
            head → size ++;
            head → min = root → data;
            Total size = max (Total size, head → size);
            return head;
        }
        else
        {
            head → BST = 0;
            return head;
        }
    }
    else if (root → left && ! root → right)
    {
        Box * head = find (root → left, Total size);
        if (head → BST && head → max < root → data)
        {
            head → size ++;
            head → max = root → data;
            total size = max (Total size, head → size);
            return head;
        }
        else
        {
            head → BST = 0
            return head;
        }
    }
    else
    {
        Box * left head = find (root → left, Total size);
        Box * Right hed = find (root → right, Total size);
```

AVL Tree – Self Balance Binary Search Tree

BST ⇒ T.C.
Insertion : O(n).
Search : O(n).
Deletion : O(n)
creation : → Online Array : nlogn
→ Runtime : n²

Balance BST ⇒ $-1 \le h(LST) - h(RST) \le 1$

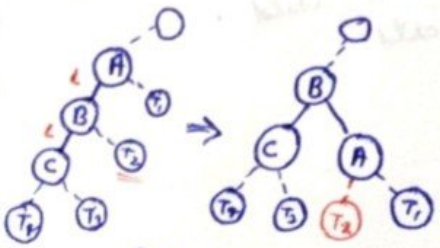


Left Rotation Right Rotation.

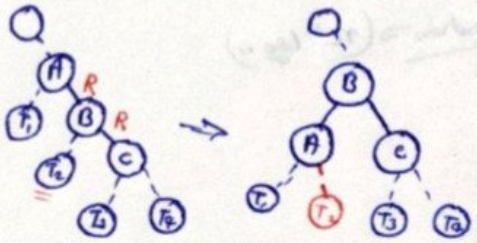
"Left rotate Top".
"Right Rotate Top"
"Left Right Rotate Middle"
"Right Rotate Top"
"Right Rotate Middle"
"Left Rotate Top".

① LL case : Right Rotation (Top).
② RR case : Left Rotation (Top).
③ LR Case : Left Rotation (Middle), Right Rotation (Top).
④ RL case : Right Rotation (Middle), Left Rotate (Top).

⊛ Edge cases.



Right Rotation
B → Right child → A → left child



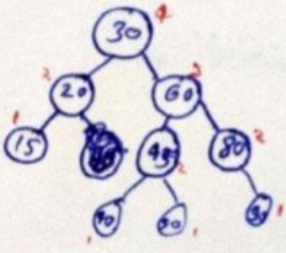
Left Rotation
B → left child ⇒ A → Right child.

⊛ Examples:

Balance > 1 ⇒ left side
Balance < -1 ⇒ Right side

→ h(LST) - h(RST)
3 - 1 ⇒ 2 > 1
left side

⇒



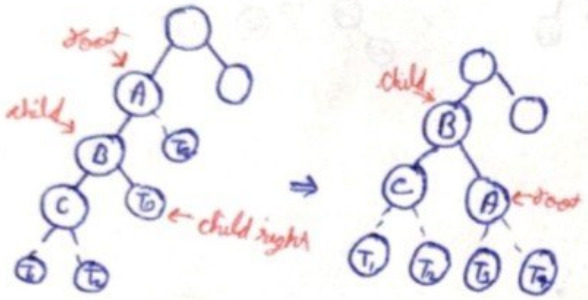
Node
→ data
→ left
→ Right
→ height.

height = 1 + max(L, R)

Balance > 1 ⇒ left Rotation
- key < root → left → data : L L.
- key > root → left → data : L R.

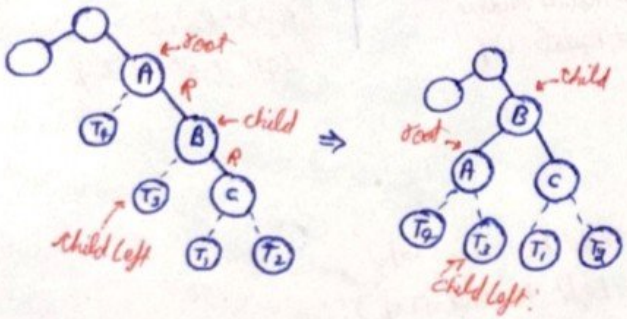
Balance < -1 ⇒ Right Rotation
- key > root → right → data : R R.
- key < root → right → data : R L.

→ Right Rotate



① child = root → left.
② childRight = child → right.
③ child → right = root.
④ root → left = childRight.
⑤ update root height.
⑥ update child height.
return child.

→ Left Rotation



① child = root → right.
② child Left = child → left.
③ child → left = root;
④ root → right = child Left;
⑤ update height - root
child.
return child.

T.C. → one rotation → $(n * \log n)$

STL

→ Pair :

Pair < Type 1, Type 2 > name_of_pair ;

Rohit, 30 → Pair < string, int > P ;

① P = make_pair ("Rohit", 30);

② P.first = "Rohit";
P.second = 30;

⇒ Pair < string, Pair < int, int > > P ;

① P = make_pair ("Harsh", make_pair (19, 64));

② P.first = "Harsh";
~~P.first~~ P.second.first = 19;
P.second.second = 64;

→ list : — (Doubly linked list):

list < Type > name_of_list ;

list < int > l1 ;

l1.push_back (30);
l2.push_back (50);
l1.push_front (70);

① push_back();
② push_front();

③ Pop_front();
④ Pop_back();

⑤ front();
⑥ back();
⑦ l1.size();
⑧ l1.empty();

70 ⇄ 30 ⇄ 50

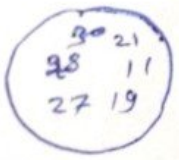
// iterator.

```
for (auto it = l.begin(); it != l.end(); it++)
{
    cout << *it << " ";
}
```

→ Set :

① It stores only unique element.
② It stores value in sorted order.

search, insert, delete
↳ log(n) ⇒ Time.



↳ AVL Tree

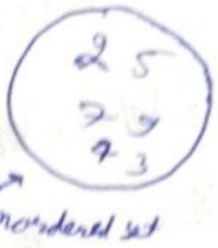
set < int > s ;

s.insert (20);
s.

- Multiset ⇒ could store non unique elements

→ Unordered Set : → implement by Hashing

- it store unique element.
- it store data in unordered way



unordered set

① Search, insert, delete → O(1).

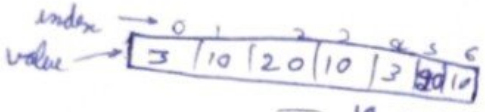
→ Map :-

Key – value

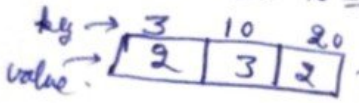
unique duplicate.

→ Sorted form on basis of Key.

→ Search, Insert, Delete → log(n)



= 10 ⇒ 3 times → O(n)



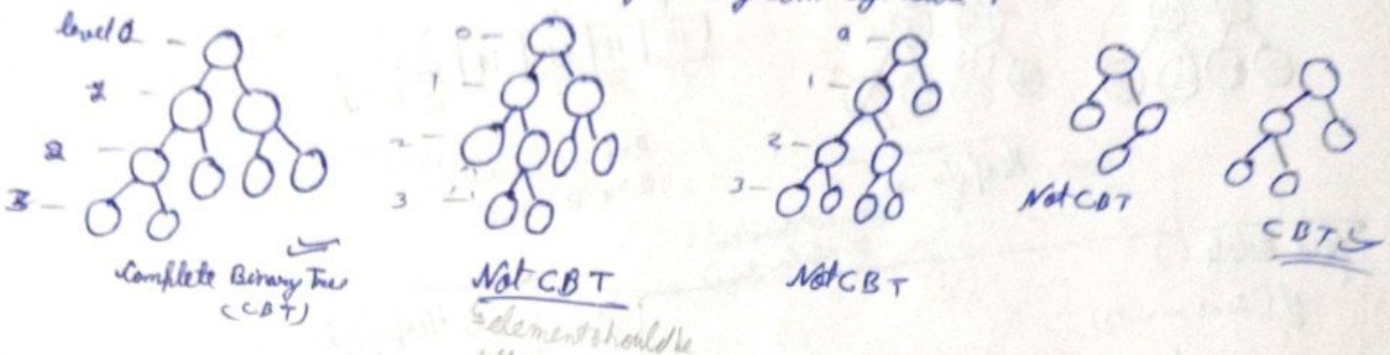
→ O(log n).

Password check

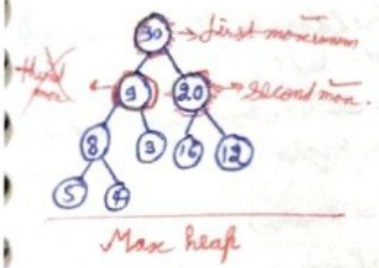
cout << m[10];

Heap

↳ It is a complete Binary Tree.
↳ All level are completely filled except the last level.
↳ At last level, nodes should be filled from left side.

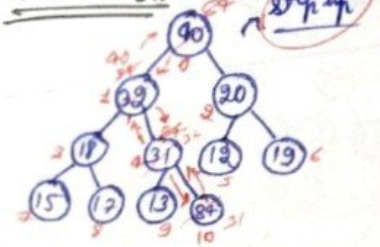


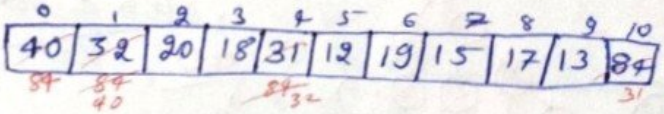
- Max heap :- ↳ CBT
  ↳ Parent node should be greater than or equal to child node.
  ↳ for all Node it should be valid.
- Min heap :- ↳ CBT
  ↳ Parent node should be less than or equal to it's child node.
  ↳ it should be valid for each node.



- Build max heap
  ① find the node where, the new node will be inserted → O(n)
  ② Maintain the heap properly (swap the nodes).

→ Insertion





T.C. → log n.

Parent_index - i
↳ 2*i + 1 (left child)
↳ 2*i + 2 (Right Child)

Child_index - i
↳ Parent: (i - 1)/2.

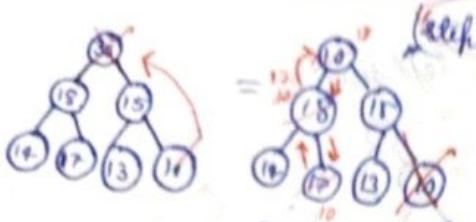
↳ T.C. (worst case) ⇒ n log n

```
class MaxHeap
{
    int *arr;
    int size; // Total element in heap
    int TotalSize; // Total element in array
    MaxHeap(int n) // n is size of array
    {
        arr = new int[n];
        size = 0;
        total_size = n;
    }
```

```
// insert into heap.
void insert(int value)
{
    if(size == TotalSize)
    {
        cout<< "Heap Overflow";
        return;
    }
    arr[size] = value;
    int index = size;
    size++;
    while(index > 0 && arr[(index-1)/2] < arr[index])
    {
        swap(arr[index], arr[(index-1)/2]);
        index = (index-1)/2;
    }
};
```

→ Delete in Heap :

Max heap → Delete → Top node → T.C. → O(log n)



0	1	2	3	4	5
10	18	15	14	17	13

⇒

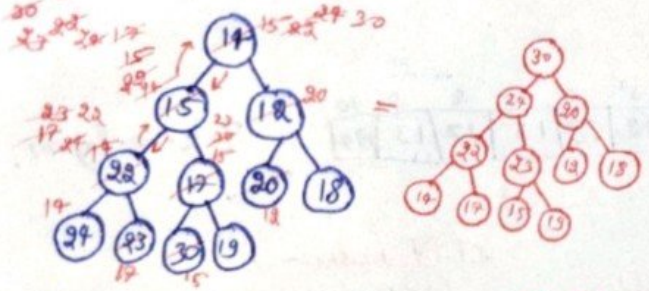
2×i + 1 = L.C.
2×i + 2 = R.C.

Remember - Heapify → (correct position Bi chala jaaa)

```
void Delete()
{
  if(size == 0)
  {
    cout << "Heap Overflow \n";
  }
  return;
  cout << arr[0] << " is deleted from heap";
  arr[0] = arr[size-1];
  size--;
  if(size == 0)
    return;
  Heapify(0); //
}
```

```
// Heapify function.
void Heapify(int index)
{
  int largest = index;
  int left = 2*index+1;
  int right = 2*index+2;
  // largest will be replaced by the index
     having greatest value among
     parent, left and right;
  if(left < size && arr[left] > arr[largest])
     largest = left;
  if(right < size && arr[right] > arr[largest])
     left = right;
  if(largest != index)
  {
    swap(arr[index], arr[largest]);
    Heapify(largest);
  }
}
```

→ Build Heap (Slip up)



child = i
Parent = $\frac{i-1}{2}$

0	1	2	3	4	5	6	7	8	9	10
14	15	12	23	17	20	18	24	23	30	19

→ T.C. → (n log n)

(height → $\log_2 n$)

$\Rightarrow \frac{n}{2}\log n + \frac{n}{4}(\log n - 1) + \frac{n}{8}(\log n - 2) + \cdots + 1$

$\frac{n}{2}\log n + \frac{n}{4}\log n - \frac{n}{8}\log n + \cdots - \left(\frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots\right)$

$n\log n\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \text{upto } \log n\right) - n\left(\frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots\right)$

GP = $\frac{a}{(1-r)}$

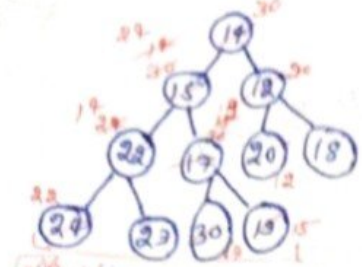
$n\log n\,(1) - n\,(A') \Rightarrow n\log n$

Heapify down

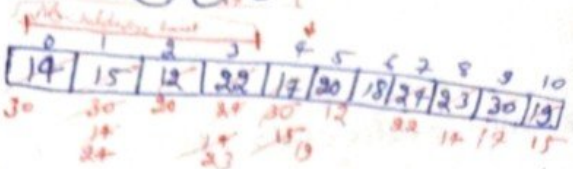
Parent = i

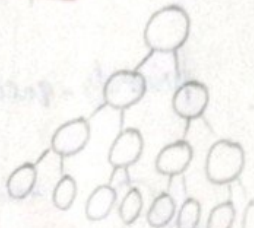
Left child = 2*i + 1;

Right child = 2*i + 2;



T.C. = O(n) ?

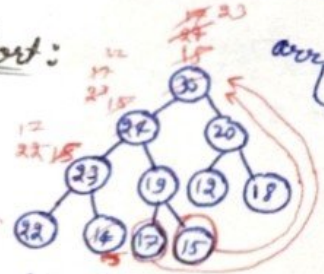




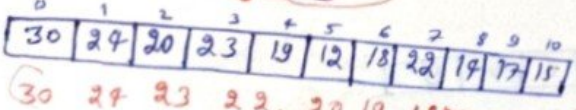
```
void Heapify(int arr[], int index, int n)
{
    int largest = index;
    int left = 2*index + 1;
    int Right = 2*index + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != index)
    {
        swap(arr[index], arr[largest]);
        Heapify(arr, largest, n);
    }
}
void BuildMaxHeap(arr[], int n)
{
    for(i = n/2 - 1; i >= 0; i--)
        Heapify(arr, i, n);
}
```

→ Heapify : T.C. ⇒ O(n)

→ Heap Sort :



arr → Max_heap O(n)

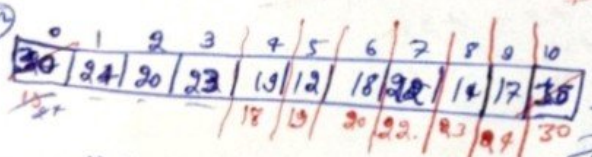


30 24 23 22 20 19 18 17 15 14 17 ⇒ S.C = O(n)

(M1)

① Top element ko Bahar nikalo.
② Top element replace by last level right most node
③ replace nodes to make Max_heap.

(M2)



⇒ S.C ⇒ O(1)

T.C. ⇒ O(n log n)

(M2)

① Replace first element with last element
② Decrease size by 1
③ Apply heapify.

// Max heap ka array ko Min Sort.

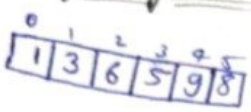
```
void sortArray(int arr[], int n)
{
    for(int i = n-1; i > 0; i--)
    {
        swap(arr[i], arr[0]);
        Heapify(arr, 0, i);
    }
}
```

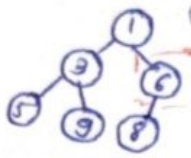
→ Priority Queue : → it is type of STL.
Part

priority_queue < int > P; //(Max heap)
P.pop();
P.top();
P.size();

priority_queue <int, vector<int>, greater<int> > P; //(Min Heap).

→ Height of a heap :

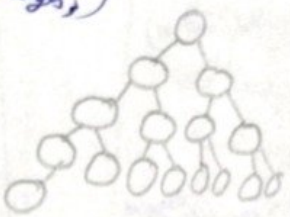




(Return no. of edges;)
edge

if height = 1 return 1

```
int heapHeight (int N, int arr[])
{
    if (N == 1)
        return 1;
    int height = 0;
    while (N > 1)
    {
        height ++;
        N /= 2;
    }
    return height;
}
```



Nodes — Height (no. of edges)
1 — 0
2 — 1
3 — 1
4 — 2
5 — 2
6 — 2
7 — 2
8 — 3
9 — 3
15 — 3
16 — 4

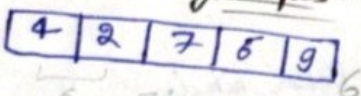
$\log_2 N$ = Height

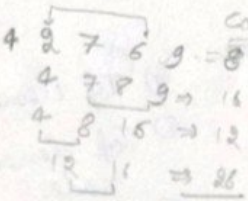
$2^{Height} = N$

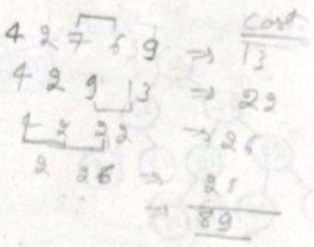
$2^{height} = 4 \Rightarrow 4$

T.C. = log N

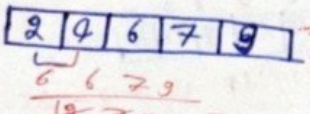
→ Minimum cost of Ropes :⇒







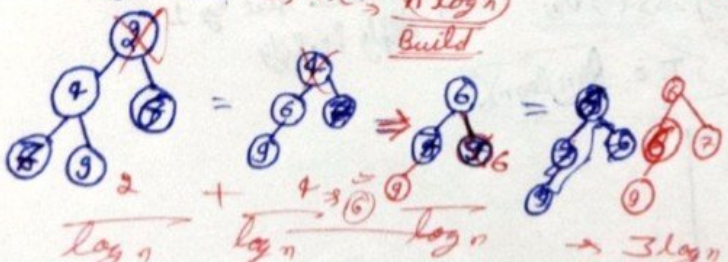
- Select 2 smallest nodes. → 'Sort the Array'
(n log n)



Sch down
O(n²)

6 6 7 9
12 7 9 → 7 9 12
16 12 → 12 16
28.

M2 ⇒ Make priority Queue ⇒ T.C → (n log n)
Build



→ 3 log n

⇒ (n-1) log n
→ (n log n) ⇒ T.C.

```
int minCost(int arr[], int n)
{
    priority_queue<int, vector<int>, greater<int>> P;
    for(int i=0; i<n; i++)
        P.push(arr[i]);
    int cost = 0;
    while(P.size()>1)
    {
        int rope = P.top();
        P.pop();
        rope += P.top();
        P.pop();
        cost += rope;
        P.push(rope);
    }
    return cost;
}
```

→ Magician & chocolates :-

2	4	8	6	10

A=5 arr

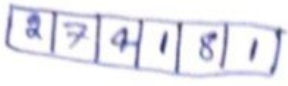
2 4 8 3 10
2 4 8 3 5
2 4 8 3 2
2 2 8 3 2
2 1 8 3 2

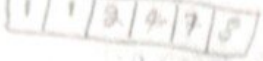
eat max chocolate.
if I eat a bag's chocklet it filled half of original chocklet chocolate again.

27

Delete + insert = heap

→ Last Stone Weight :-

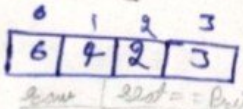





```
int lastStoneWeight(vector<int>& stones)
{
    priority_queue<int> P;
    for(int i=0; i<stone.size(); i++)
    P.push(stone[i]);
    while(P.size()>1)
    {
        int weight = P.top();
        P.pop();
        weight -= P.top();
        P.pop();
        if(weight)
        P.push(weight);
    }
    return P.empty();
}
```

→ Take gift from Richest Pile :-

→ Profit Maximization :-



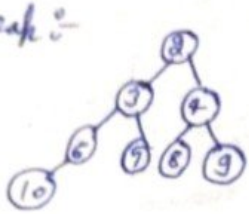
→ K^{th} Smallest Element

K = 4

10	3	7	4	8	9	2	6

① Sort → 2 3 4 6 7 8 9 10 ∴ (T.C. → n log n, S.C. → O(1)).

② Min_heap :-



T.C. → [O(n) + K log n] → (n log n) W.C.
S.C. → 1

③

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

Smallest Element → Max heap
Largest Element → Min heap.

Max heap → O(K) + O(K log K) → O(K) ⇒ Helps

n − K times

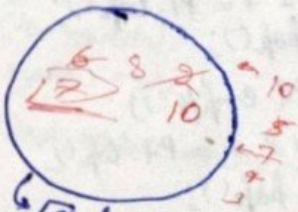
(n − k) * 2 log K + K → n log K (W.C.)

```
int KthSmallest(int arr[], int n, int K)   // (SIZE(arr)-1)
{
    // int n = arr.size() - 1;
    priority_queue<int> P;
    for(int i=0; i<K; i++)
        P.push(arr[i]);
    for(int i=k; i<=n; i++)
    {
        if(arr[i] < P.top())
        {
            P.pop();
            P.push(arr[i]);
        }
    }
    return P.top();
}
```

→ K^{th} largest Element ⇒

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

K = 3



(n log K) S.C. = K

```
int findKthLargest(vector<int>& nums, int K)
{
    priority_queue<int, vector<int>, greater<int>> P;
    for(int i=0; i<K; i++)
        P.push(nums[i]);
    for(int i=K; i<nums.size(); i++)
    {
        if(nums[i] > P.top())
        {
            P.pop();
            P.push(nums[i]);
        }
    }
    return P.top();
}
```

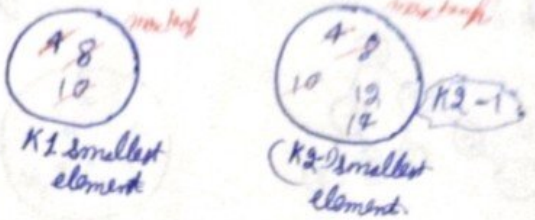
→ Sum of Element b/w K1 & K2 Smallest Element in

20	8	22	4	12	10	14

K1 = 3, K2 = 6

① sort : 4 8 10 12 14 20 22
(0 1 2 3 4 5 6)

② "K smallest ⇒ Max heap" ⇒



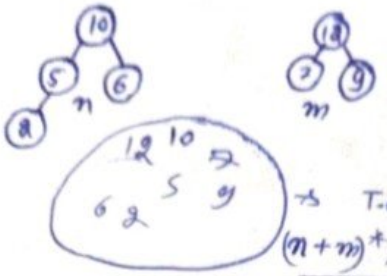

```
int sum(int A[], int n, int K1, int K2)
{
    priority_queue<int> P1;
    priority_queue<int> P2;
    for(int i=0; i<K1; i++)
    P1.push(A[i]);
    for(i=0; i<K2-1; i++)
    P2.push(A[i]);
    // K1 smallest Element.
    for(i=K1; i<N; i++)
    {
        if(A[i] < P1.top())
        {
            P1.pop();
        }
        P1.push(A[i]);
    }
    // K2 smallest element.
    for(i=K2-1; i<N; i++)
    {
        if(A[i] < P2.top())
        {
            P2.pop();
            P2.push(A[i]);
        }
    }
    int sum1=0, sum2=0;
    while(!P1.empty())
    {
        sum1 += P1.top();
        P1.pop();
    }
    while(!P2.empty())
    {
        sum2 += P2.top();
        P2.pop();
    }
    return sum2 - sum1;
}
```

→ Merge two binary Max heap:

10	5	6	2

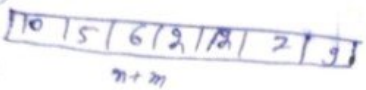
12	7	9

(M1)



→ T.C. $(n+m)*\log(n+m)$

(M2)

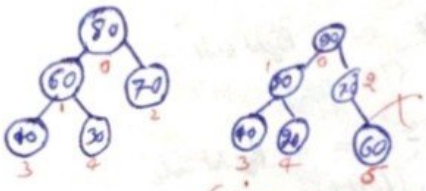


over. → $O(n+m)$

```
vector<int> mergeHeap(vector<int>& a,
        vector<int>& b, int n, int m)
{
    vector<int> ans;
    for(i=0; i<n; i++)
    ans.push_back(a[i]);
    for(int i=0; i<m; i++)
      ans.push_back(b[i]);
    n = ans.size();
    for(int i= n/2-1; i>=0; i--)
      Heapify(ans, i, n);
  return ans;
}
```

→ Is BT Max heap:

cond: ① Complete BT.
② Every Parent >= it's child.



(index >= Total Node)

① No of node = $O(n)$
② CBT → $O(n)$ → T.C. → $O(n)$.
③ Parent >= child → $O(1)$

→ BST to Max Heap:

① Every Parent > child
② left Subtree < Right subtree elements

(M1)

① Inorder Traversal to get values.
(1, 2, 3, 4, 5, 6, 7)
② Post order Traversal to fill values.

```
void inorder(Node* root, vector<int>& ans)
{ if(!root)
    return;
  inorder(root->left, ans);
  ans.push_back(root->data);
  inorder(root->right, ans);
void Postorder(Node* root, vector<int>& ans, int& index)
{   if(!root)
      return;
    Postorder(root->left, ans, index);
    Postorder(root->right, ans, index);
    root->data = ans[index];
    index++;
void BST to MaxHeap(Node* root)
{
    // Inorder Traversal.
    vector<int> ans;
    inorder(root, ans);
  // int Post order Traversal.
    int index=0;
    Postorder(root, ans, index);
}
```

→ Kth Element in a Matrix :

16	28	68	70
22	41	63	91
27	50	87	93
36	78	87	94

K = 6

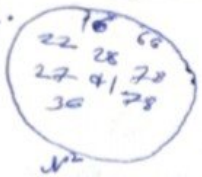
N × N

M1
① copy elements in 1D Array (N^2)
② Sort the array ($N^2 \log N^2$)
③ find/search Element $\in O(1)$

$N^2 + N^2 \log N^2 + 1$

$\Rightarrow (N^2 \log N) \Rightarrow$ T.C.

M2 ① Min heap.

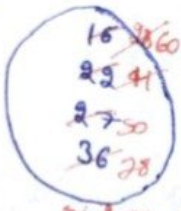


Remove Delete $\Rightarrow K \log N$

N^2

$\Rightarrow$ T.C. $\Rightarrow N^2 + K \log N$

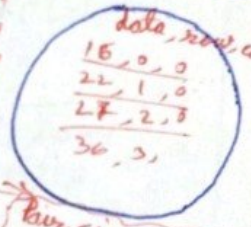
M3



Min heap (Col)

16 - 1st smallest
22 - 2nd smallest
27 - 3rd smallest
38 ⇒ 4th
36 ⇒ 5th
41 ⇒ 6th

T.C. ⇒ K log n + n

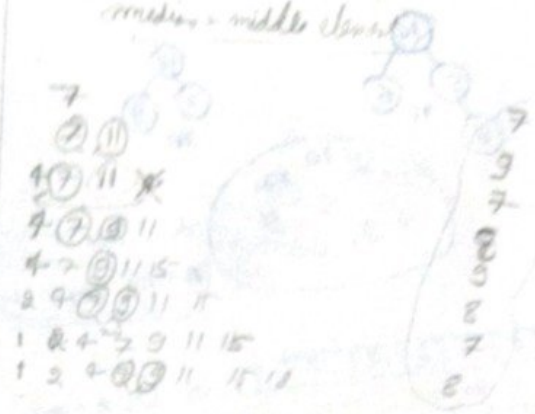


Pair<int, Pair<int, int>>

→ find Median in a Stream :

7	11	4	9	15	2	1	18

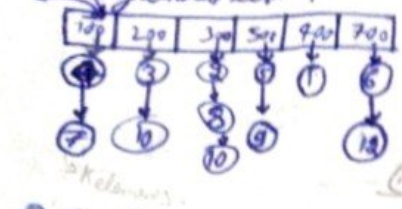
median = middle element



2	4	6	7	8	10	12	14

① left side == Right side → $\left(\frac{T_{ot} + T_{ok}}{2}\right)$
② left side − 1 == Right side → (left tok − one)
③ Right side > left side (1 element)
left side > Right side + 1 (1 element)

Merge K Sorted Linked List ⇒



① Brutforce ↙
② Merge sort ↘ → Linked list.
③ Heap.

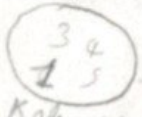
```
Class compare
{
    public:
    bool operator()(Node* a, Node* b)
    {
        return a->data > b->data;
    }
};

Node* MergeKlist(vector<Node*>& arr)
{
    int n = arr.size();
    Priority_queue<Node*, vector<Node*>,
        compare> P(arr.begin(), arr.end());
    Node* root = new Node(0);
    Node* tail = root;
    Node* temp;
    while(!P.empty())
    {
        temp = P.top();
        P.pop();
        tail->next = temp;
        tail = tail->next;
        if(temp->next)
        P.push(temp->next);
    }
    return root->next;
}
```

→ Merge K Sorted Array ⇒

3	4	5	11
1	6	7	14
4	5	8	13
6	9	10	12

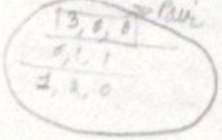
K=4

merge sort / $n \log n$ | Heap sort
$K^2 + K^2 \log K = T$ | $O(K^2 \log K)$
T.C → $O(K^2)$ | $O(K) = S$.



$K^2 \log K + K + (K^2 - K) \log K$

K elements → Pair



②

min heap

T.C ⇒ K elements

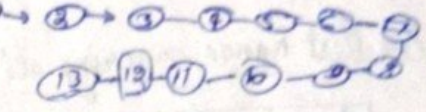
$nK * \log nK$ + $nK * \log nK$

Deletion

S.C ⇒ nK → heap.

④

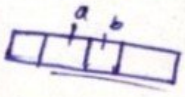
but store as on address



T.C → $nk \log k$
S.C ⇒ $O(K)$.

```
class compare.  // (customise compare)
{
    Public :-
    bool operator (Node* a, Node* b)
    {
        return a->data > b->data;
    }
};
```

a → data > b → data; → Min heap.
a → data < b → data; (Max heap)



in Heap it opposite with operator

```
vector<int> mergeArray(vector<vector<int>>& arr, int K)
{
    vector<Pair<int, Pair<int, int>>> temp;
    for(i=0; i<K; i++)
        temp.push_back(make_pair(arr[i][0], make_pair(i,0)));
    Priority_queue<Pair<int, Pair<int, int>>,
        vector<Pair<int, Pair<int,int>>>,
        greater<Pair<int, Pair<int, int>>>> P(temp.begin(),
                                              temp.end());
    vector<int> ans;
    Pair<int, Pair<int,int>> Element;
    int i, j;
    while(!P.empty())
    {
        Element = P.top();
        P.pop();
        ans.push_back(Element.first);
        i = Element.second.first;
        j = Element.second.second;
        if(j+1 < K)
        P.push(make_pair(arr[i][j+1], make_pair(i, j+1)));
    }
    return ans;
}
```

Sliding Window

→ Zero sum Subarray :-

0	1	2	3	4	5	6	7	8	9
6	-1	-3	4	-2	2	4	6	-12	-2

Total no of subarray
Sum = 0

brutt force :

↗ O(n³)

```
int total = 0;
for (i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        int sum=0
        for (k=i; k<=j; k++)
        {
            sum += arr[k];
        }
        if (sum == 0)
            total ++;
    }
}
```

↗ O(n²)

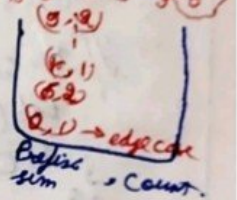
```
int total=0
for(i=0; i<n; i++)
{
    int sum=0
    for(j=i; j<n; j++)
    {
        sum += arr[j];
        if(sum == 0)
            Total ++;
    }
}
```

M3 Prefix Sum :

0	1	2	3	4	5	6	7	8	9
6	-1	-3	4	-2	2	4	6	-12	-2

Prefix sum → 6 5 2 6 4 6 10 16 4 2

Total = 1 + 2 + 1 + 1 ⇒



(map <int, int> m)

unorder_map → Key: Prefix, value: count

insert/search

↳ T.C. ⇒ O(1)

map → insert/search ⇒ T.C. ⇒ O(1).

```
int findSubarray(vector<int> &arr)
{
    int n = arr.size();
    int total = 0;
    unordered_map<int, int> m;
    int PrefixSum = 0;
    m[0] = 1;
    for(i=0; i<n; i++)
    {
        PrefixSum += arr[i];
        if(m.count(PrefixSum))
        {
            total += m[PrefixSum];
            m[PrefixSum]++;
        }
        else
            m[PrefixSum] = 1;
    }
    return total;
}
```

→ Subarray Sum equals K :-

K = 6

0	1	2	3	4	5	6	7	8	9
3	1	7	-7	2	1	3	4	-15	2	-3	6

2 3 10 6 8 9 12 16 1 3 0 6 ← Prefix sum.

Prefix - K

Total = 8

1+1+1+1+1+2+1

```
int SubarraySum(vector<int>& nums, int K)
{
    unordered_map<int, int> m;
    m[0] = 1
    int PrefixSum = 0;
    int total = 0;
    for(i=0; i<nums.size(); i++)
    {
        PrefixSum += nums[i];
        if(m.count(PrefixSum - K))
        {
            total += m[PrefixSum - K];
        }
        m[PrefixSum]++;
    }
    return total;
}
```

```
else
    m[PrefixSum]++;
}
return total;
}
```

→ Subarray Sums Divisible by K :

K = 7

3	5	6	3	9	4	6	9

3 8 14 17 26 30 36 45

%7 → 3 1 0 3 5 2 1 5

2	3	-1	-3	11	4	8	6	9	4

Prefix Sums → 2 5 -3 -6 5 9 17 23 32 36

%K(7) → 2 5 -3 -6 5 2 3 1 4 1

4 1

```
int SubDivK(vector<int>& nums, int k)
{
    unordered_map<int, int> m;
    m[0] = 1
    int PrefixSum = 0, rem, total = 0;
    for(i=0; i < nums.size(); i++)
    {
        PrefixSum += nums[i];
        rem = PrefixSum % k;
        if(rem < 0)
            rem += k;
        if(m.count(rem))
        {
            total += m[rem];
            m[rem]++;
        }
        else
            m[rem] = 1;
    }
    return total;
}
```

7 × 0 − 3 = −3
7 × −1 + 4 = −3

→ Subarray Product less than K :-

2	5	10	8	100	1000	5	15

↑ start, ↑ end

K = 999

```
Product = 1
while(end < n)
{
    Product *= nums[end];
    while( Product >= k && start <= end)
    {
        Product /= nums[start];
        start++;
    }
    count += end - start + 1;
    end++;
}
```

→ Minimum Size Subarray Sum :

2	3	1	2	4	3

Target = 7

start ↑ ↑ end.

```
while (end < n)
{
    sum += nums[end];
    while(sum >= target)
    {
        target = min(total, end-start+1);
        sum -= nums[start++];
    }
    end++;
}
return total == INT_MAX ? 0 : total;
```

T.C. ⇒ O(n)

→ Minimum Window Substring :

s = ADOBECODEBANC
t = ABC.

window length increase ⇒ t ke saare char present na ho
window length decrease ⇒ till t ka koi char miss na ho jaye

map

A	+0
B	+0
C	+0
D	-1
O	-1
E	-1

total = 3 : window length ↑ ⇒ total decrease (↓)
total 0
until char me -ve number nahi hona chaiye.
window length ↓ ⇒ total increase (↑)
when +ve number update hota hai

```
unordered_map <char, int> m;
    for (i=0; i<t.size(); i++)
        m[t[i]]++;
int start = 0, end = 0, ans = INT_MAX,
    index = -1, n = s.size();
while (end < n)
{
    m[s[end]]--;
    if (m[s[end]] >= 0)
        total--;
    while (!total && start <= end)
    {
        if (ans > end-start+1)
        {
            ans = end-start+1;
            index = start;
        }
        m[s[start]]++;
        if (m[s[start]] > 0)
            total++;
        start++;
    }
    end++;
}
if (index == -1)
    return "";
string str
for (i = index; i < index + ans; i++)
    str += s[i]
return str;
```

→ Length of longest subarray with at most K frequency :-

1	2	3	1	2	3	3	4	1	2

K = 2

```
int maxSubArr(vector<int>& nums, int k)
{
    int len = 0;
    unordered_map<int, int> count;
    int start = 0, end = 0, n = nums.size();
    while(end < n)
    {
        count[nums[end]]++;
        while(count[nums[end]] > k)
        {
            count[nums[start]]--;
            start++;
        }
        len = max(len, end - start + 1);
        end++;
    }
    return len;
}
```

→ Count Subarray where max Element appear at least K times :-

1	2	3	2	3	1	2	3	3	2

K = 2

```
start = 0, end = 0, maxEle, count = 0
total = 0
while (end < n)
{
    if (nums[end] == MaxEle)
        count++;
    while (count == k)
    {
        total += n - end;
        if(nums[start] == MaxEle)
            count--;
        start++;
    }
    end++;
}
return total.
```

→ Subarray with k different Integer :-

1	2	1	2	3

K = 2

Total no of subarray, atleast K different int present here

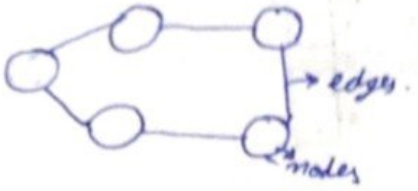
Atleast(2) : Sub(2) + sub(3) + sub(4) + ...

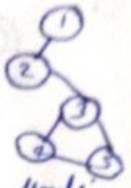
Atleast(1) ⇒ sub(3) + sub(4) sub(5).

Atleast(3) - Atleast(2) ⇒ sub(2)

Graph

→ Graph is a non-linear data structure consisting of vertices and edges.

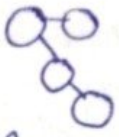




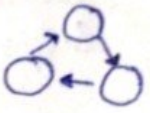
undirected graph

directed Graph.

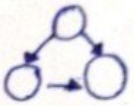
Cyclic graph



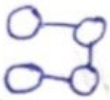
Acyclic graph.



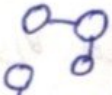
Directed cyclic graph



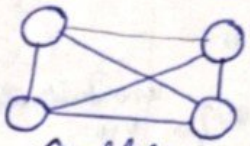
Directed Acyclic graph.



Connected graph

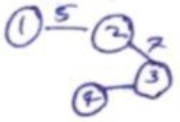


Disconnected graph



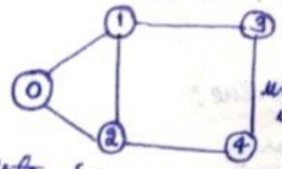
$^nC_2 = \frac{n(n-1)}{2}$ (n = nodes)

Complete graph ⇒ Every node connected with each other.



⇒ weighted graph

→ ~~Graph~~ Graph representation :- ① Adjacency Matrix
② Adjacency List.



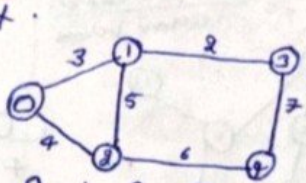
① undirected unweight graph.

Vertex { 0, 1, 2, 3, 4 }

Edge : { {0,1}, {1,2}, {0,2}, {2,3}, {3,4}, {2,4}, {1,3}, {1,4} }

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	1	1	0
2	1	1	0	0	1
3	0	1	0	0	1
4	0	0	1	1	0

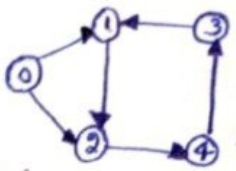
⇒ (2D Array)



② undirected weighted graph

	0	1	2	3	4
0	0	3	4	0	0
1	3	0	5	2	0
2	4	5	0	0	6
3	0	2	0	0	7
4	0	0	6	7	0

⇒ (2D Array)



Directed unweighted graph ③

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	1	0	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	0	1	0

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

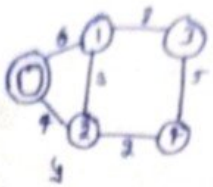
S.C. = $O(V^2)$

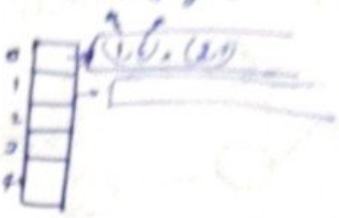
② Adjacency list :-

Adj List[0].push_back(1);
Adj List[1].push_back(0);

- Adj List[0][1] => 1
  Adj List[2][3] => 4

vector<int> AdjList[5]





vector<Pair<int, int>> Adj List[5];

T.C. => O(V+2E) => O(V+E)
S.C => O(V+E)

worst case O(V²)

O(V²) ≈ complete graph

Adj Matrix

Add Edge : O(1)
Remove Edge : O(1)
edge Exist : O(1)
Space Com : => O(V²)

Dense

Adj list

O(1)
O(V) => unordered map O(1).
O(E) => O(1).
O(V+E) => O(V²)

Edge = = complete graph

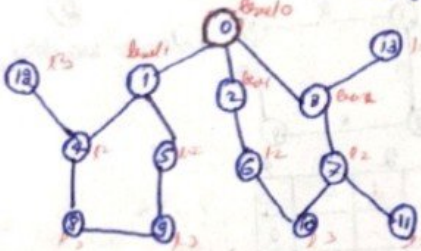
Sparse.

Tree V/s Graph

n-nodes
(n-1) Edges
loop ✗

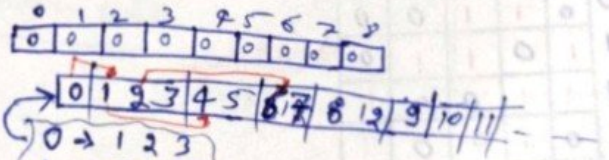
loop ✓

→ BFS Traversal :- Breath first search
Cover Verex



BFS
[0 1 2 3
4 5 6 7 13
12 8 9 10 11]

BFS → queue:
visited_array



① Pop a node from queue & print it and put it in answer
② Node will look into it's unvisited neighbour.
- Push them in queue.
- Mark visited them as 1

0 → 1 2 3
1 → 0 4 5
2 → 0 6
3 → 0 7
4 → 8 12
5 → 1 9
6 → 2 10
7 → 3 11 10
8 → 4 9
9 → 5 8
10 → 6 7
11 → 7
12 → 4
13 → 3

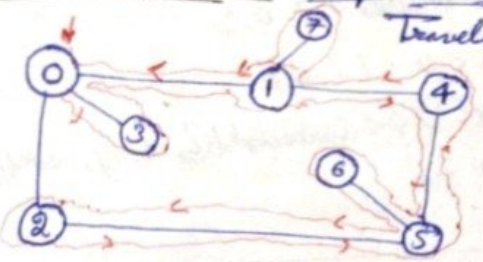
```
vector <int> BFS Graph(int V, vector<int> adj[])
{
    queue <int> q;
    vector <bool> visited(V, 0);
    q.push(0);
    visited[0] = 1;
    vector <int> ans;
    while(!q.empty())
    {
        node = q.front();
        q.pop();
        ans.push_back(node);
        for(j = 0; j < adj[node].size(); j++)
        {
            if(!visited[adj[node][j]])
                visited[adj[node][j]] = 1;
                q.push(adj[node][j]);
        }
    }
    return ans;
}
```

→ S.C. ⇒ O(V)
→ T.C. ⇒ O(V+2E) ⇒ O(V+E)

→ DFS Traversal ⇒ Depth first Search

Travelling.

DFS → Recursion
→ stack – HW



0 → 1 3 2
1 → 0 7 4
2 → 0 5
3 → 0
4 → 1 5
5 → 6 2
6 → 5
7 → 1

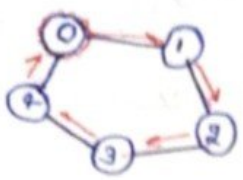
1 0 7 3 1 4 5 6 2 7

→ it will visit its all unvisited neighbour one by one.

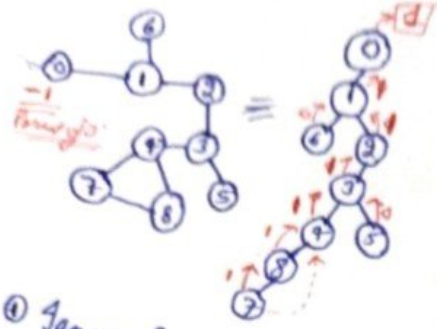
→ if visited all it's neighbour it will back from it has been called.

```
void DFS(int node, vector<vector<int>>& adj,
         vector<int>& ans, vector<bool>& visited)
{
    visited[node] = 1;
    ans.push_back(node);
    for(j = 0; j < adj[node].size(); j++)
    {
        if(!visited[adj[node][j]])
            DFS(visited[adj[node][j]], adj, ans, visited);
    }
}
vector<int> dfs(vector<vector<int>>& adj)
{
    int v = adj.size();
    vector<bool> visited(V, 0);
    vector<int> ans;
    DFS(0, adj, ans, visited);
    return ans;
}
```

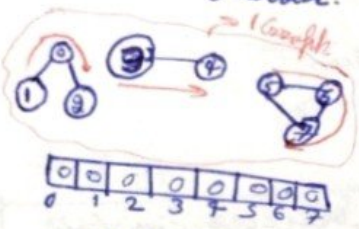
→ Cycle Detection ① DFS



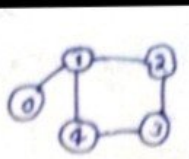
0 - 1 4
1 - 2 0
2 - 3 1
3 - 4 2
4 - 0 3



① Ignore, Parent node
② Neighbour node already visited
↳ cycle detected
③ visit the neighbour.



T.C. ⇒ O(V+2E) ≈ T.C. = O(V+E)

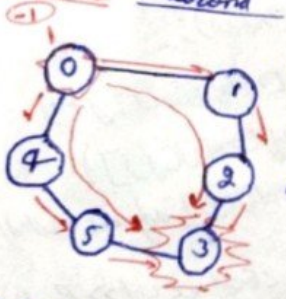


0 - 1
1 - 0 2
2 - 1 3
3 - 2 4
4 - 1 3

Exception: skip Parent node

```
bool CycleDetected(int Node, int Parent, vector<vector<int>>& adj,
                   vector<bool>& visited) {
    visited[node] = 1;
    for(i=0; i<adj[node].size(); i++)
    {
        if(Parent == adj[node][i])
            continue;
        if(visited[adj[node][i]])
            return 1;
        if(CycleDetected(adj[node][i], node, adj, visited))
            return 1;
    }
    return 0;
}
bool isCycle(int V, vector<vector<int>>& adj)
{
    vector<bool> visited(V, 0);
    for(i=0; i<V; i++) // for non connecting node
    {
        if(!visited[i] && CycleDetected(i, -1, adj, visited))
            return 1;
    }
    return 0;
}
```

② BFS - Leetcode



0 - 1, 4
1 - 2 0
2 - 3 1
3 - 4 2
4 - 0 3

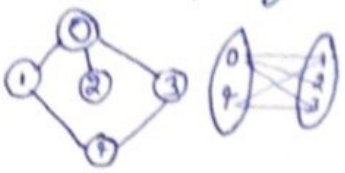
BFS: queue.

(node, Parent)

① Parent == neighbour (Ignore)
② if neighbour is already visited
↳ cycle detected.
③ neighbour ko queue ke under insert karo.
& visited mark karo.

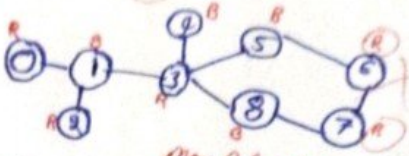
Bipartite Graph

it is a graph in which the vertices can be divided into two disjoint sets, such that no 2 vertices within the same set are adjacent. In other words, it is a graph in which every edge connected a vertex of one set to a vertex of other set.

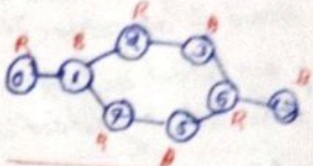


→ 2 colouring Algorithm.
Red Blue.

- BFS



Non Bipartite
↳ Odd length cycle



↳ Even length cycle
↳ Bipartite graph

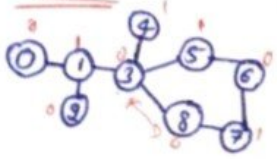
R = 0
B = 1

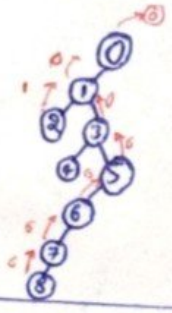
colour

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

2 [0 3 4 5 8]
1 [1 2 6 7]

- BFS



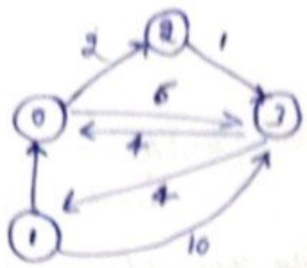


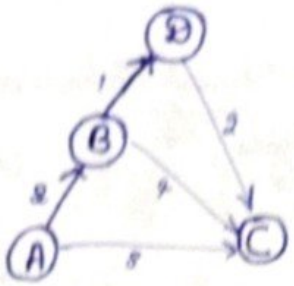
① it will look at it's all neighbour
↳ if colour is not assigned
⇒ assigned a colour to them as opposite.
↳ else (colour is assigned
⇒ if neighbour color is same as parent we will declare it as not a Bipartite graph.

```
bool IsBip (int node, vector<int> adj[],
            vector<int> &color)
{
    for (int j=0; j < adj[node].size(); j++)
    {
        if (color[adj[node][j]] == -1)
        {
            color[adj[node][j]] = (color[node]+1)%2;
            if (!IsBip (adj[node][i], adj, color))
                return 0;
        }
        else
        {
            if (color[node] == color[adj[node][j]])
                return 0;
        }
    }
    return 1;
}
```

→ Image Recommendation System.
② Stable Marriage Problem

→ Floyd Warshall Algorithm



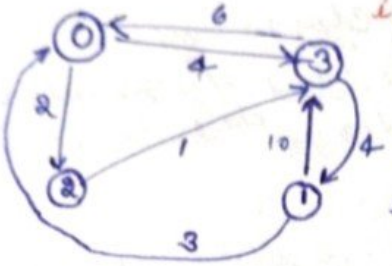


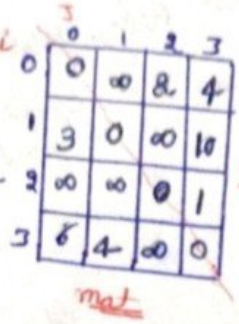
A → B
A → C
B → B
B → C
B → C

Warshall :- Transitivity
A → B, B → C
A → C

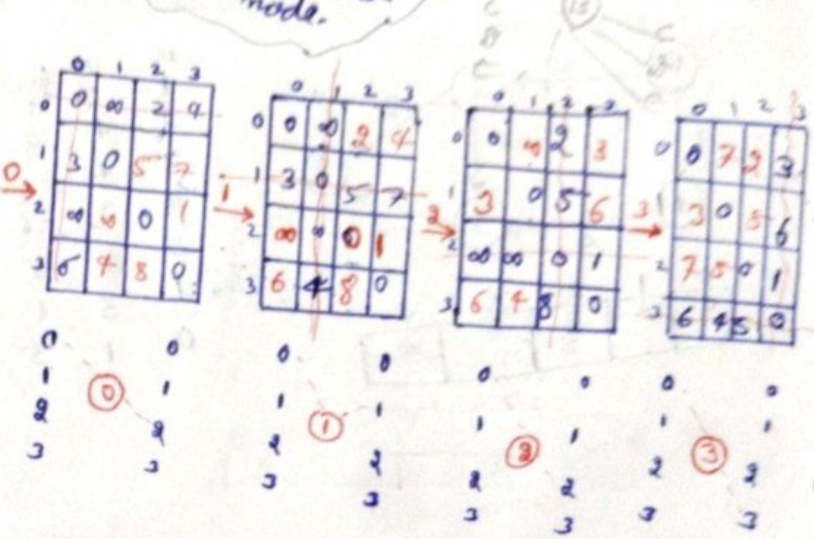
[A → B B → C]
A → C

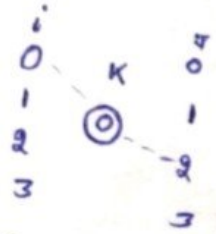
Intermediate node → select each node as intermediate node.





mat





Path(i, j) = min[Path(i, j), Path(i, k) + Path(k, j)];

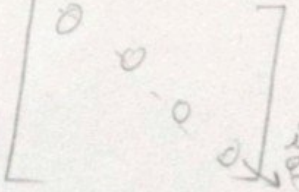
mat[i][j] = min(mat[i][j], mat[i][k] + mat[k][j]);

① order does not matter.

→
```
for (k=0 ; k<n ; k++)
    for (i=0; i<n ; i++)
        for (j=0 ; j<n ; j++)
            mat[i][j] = min (mat[i][j], mat[i][k] + mat[k][j]);
```

T.C. → O(n³) = O(V³)

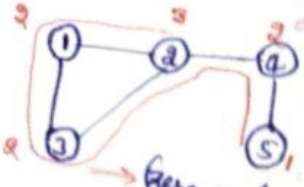
S.C. → O(1).

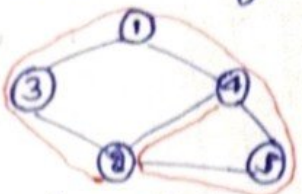


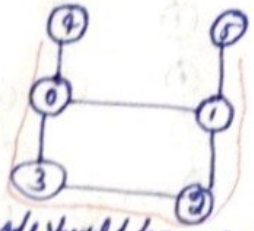
if (-ve) → negative cycle

→ Euler Path :-

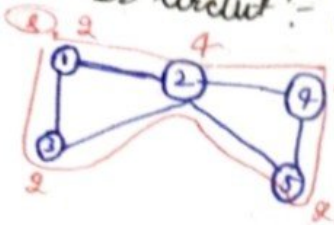
It is a Path in a graph that visit every edge exactly once:







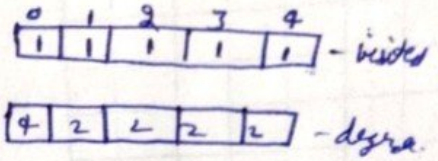
→ Euler Circuit :- every edge exactly once & starting Point should be end Point.



① All edges should be visited exactly once
② start == end.

① All nodes degree should be even.
② All edges should be part of single component (All non-zero degree node should be connected).

- ① find degree of each node.
- ② if degree of any node is odd, not E.C.
- ③ Even → E.C.
- ④ Apply DFS from any non-zero degree node



if EC → EP ✓

EP ✗ EC.

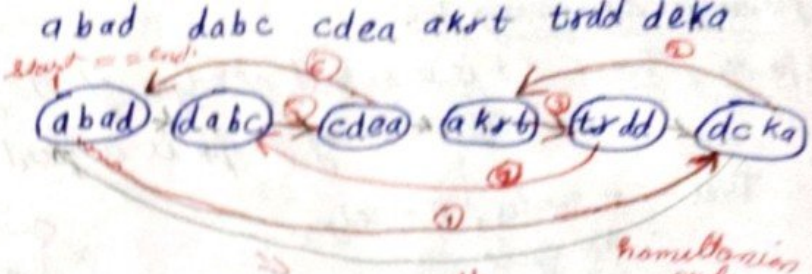
→ Circle of String :-

a b _ b c _ c d _ d a

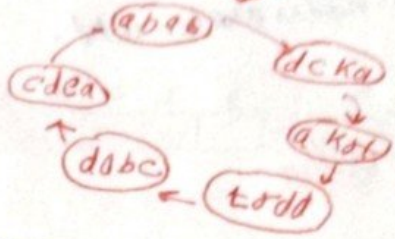
x ... y

last char of x = = first char of y

a b a d d a b c c d e a a k r t t r d d d e k a



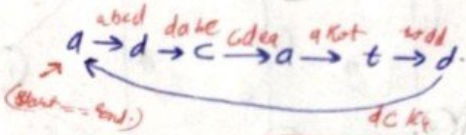
Hamiltonian cycle.



T.C. = N!

ad → dc → ca → at → td → dd.

a → d
d → c
c → a
a → t
t → d
d → a



Eulerian circuit

① create edges b/w first & last char of string.

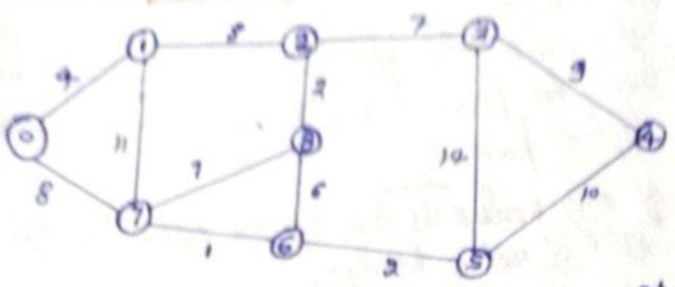
② Find Eulerian circuit

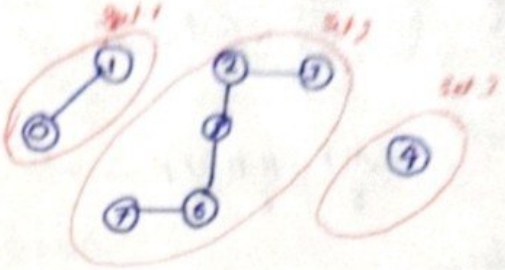
↳ [Indegree == outdegree]

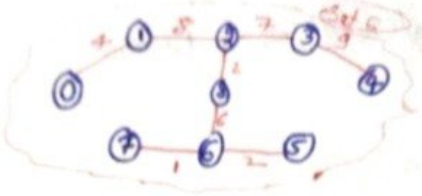
↳ DFS : All the edges are part of single connected component.

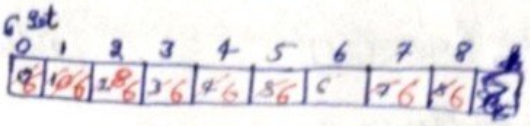
- Kruskal Algorithm :- Minimum Spanning Tree

Disjoint Set

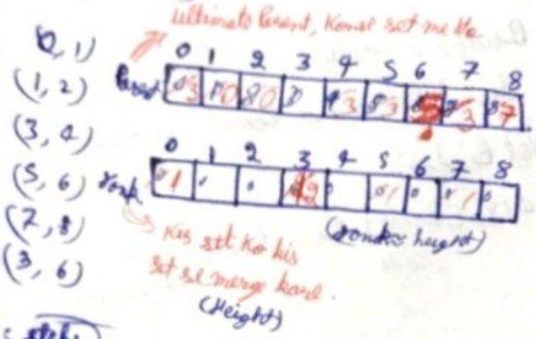


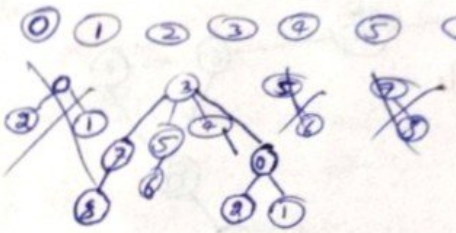






(Union by rank/size)





T.C. ⇒ O(1)

(0, 1)
(1, 2)
(3, 4)
(5, 6)
(7, 8)
(3, 6)

(0, 1).
(1, 2).
(3, 4).
(5, 6).
(7, 8).
(3, 6).
(4, 6).
(1, 4).
(2, 6).
(3, 5).

steps

① find the ultimate parent of u & v
② find the rank of ultimate parent. (PU, PV)
③ merge the smaller set into larger set. (rank)

int find

① Priority queue → sort all edges int P-Q.

Pair<int, Pair<int, int>>
wt u v

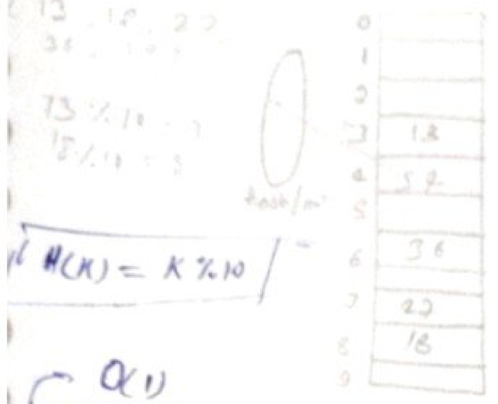
```
int findParent(int U, vector<int> &Parent)
{
    if(U == Parent[U])
        return U;
    return Parent[U] = findParent(Parent[U], Parent);   // Path compression
}
void UnionByRank(int U, int V, vector<int> &P,
                 vector<int> &rank)
{
    int PU = findParent(U, Parent);
    int PV = findParent(V, Parent);
    if(rank[PU] > rank[PV])
        Parent[PV] = PU;
    else if(rank[PU] < rank[PV])
        Parent[PU] = PV;
    else
        Parent[PV] = PU;
        rank[PV]++;
}
```

⇒ ultimate Parent.

* Hashing

it is a technique which is used in data structure to store & retrieve data effectively.

- Insertion, deletion, Search → O Array, L.L., Stack, Queue, Binary Tree, BST → Take $O(n)$ in worst case. & AVL Tree → $O(\log n)$



$H(K) = K \% 10$

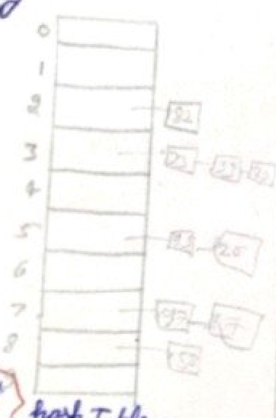
→ $O(1)$

hash Collision

① Separate chaining :-

num % 10

worst case → $O(n)$ search & delete



hash Table.

load factor = Total no. of element / size of Table

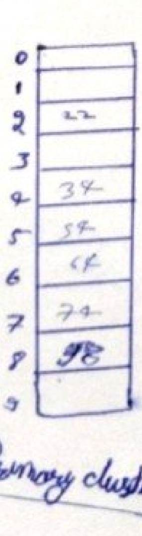
LF → 9/10 → 0.9

② linear Probing

122, 32, 54, 67, 98, 74, 99}.

$H(K) = K \% 10$

$H(K) = (K + i) \% 10$



Primary clustering.

③ Quadratic Probing:

172, 84, 96, 94, 58, 4}

$H(K) = (K + i^2) \% 10$

$(72+0) \% 10 =$

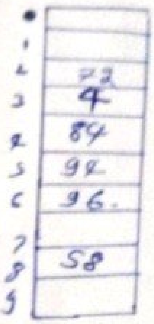
$(94+1) \% 10 = 4$

$(4+0^2) \% 10 = 4$

$(4+1^2) \% 10 = 5$

$(4+2^2) \% 10 = 8$

$(4+3^2) \% 10 = 3$



Secondary Clustering

④ Double Hashing : (computation time ↑)

132, 47, 52, 87, 2

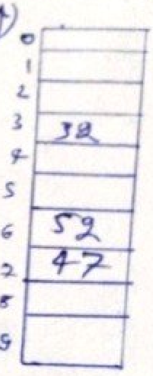
$H(K) = (H^1(K) + i * H^2(K)) \% 10;$

$(K \% 5 + i * K \% 6) \% 10;$

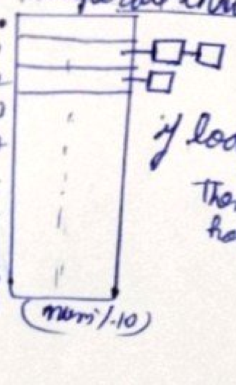
① $(32 \% 5 + 0 * (32 \% 6)) \% 10 = 2$

② $((47 \% 5) + 0 * (47 \% 6)) \% 10 = 2$ *

$((47 \% 5) + 1 * (47 \% 6)) \% 10 = 7$

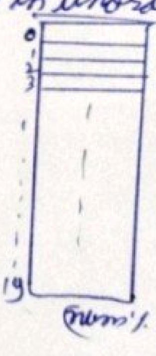


⑤ Seperate chaining using in Unordered map



(num % 10)

if load factor > 5 Then double the hash Table.



(num % 20).

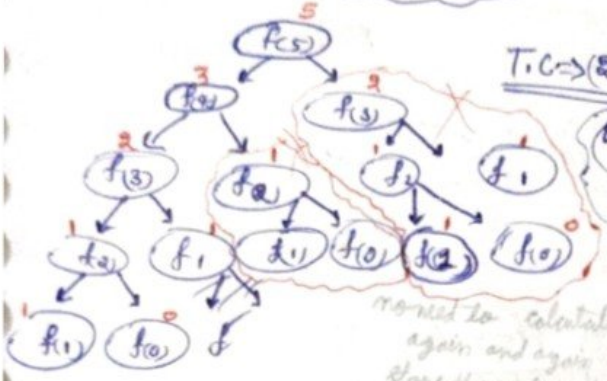
Hash fn → Hash code (32, "Rohit") ; → Compression fn (~~Has~~ Mapping fn) (num % 10)

Dynamic Programming

Recursion → Top down approach (Recursion + memoization) → Bottom up approach (Tabulation) (Loops) → Space optimization

→ Fib no: 0 1 1 2 3 5 8 13 21

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



T.C → (2^n)

Overlapping subproblem

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

memoization

$f(0) = 0$
$f(1) = 1$
$f(2) = 1$
$f(3) = 2$
$f(4) = 3$
$f(5) = 5$
$f(6) = 8$

0	0
1	1
1	2
2	3
3	4
5	5
8	6

dP

length of array → (n+1) (6+1=7)

```
int fibo(int n, vector<int>& dp)
{
    if (n <= 1)
        return n;
    if (dp[n] != -1)
        return dp[n];
    return dp[n] = fib(n-1, dp) + fib(n-2, dp);
}
```

T.C = O(n)

S.C → O(n) (array) + O(n) (rec. stack)

→ Bottom up: Base case → upper.

dP: 0 1 2 3 4 5

```
for (i = 2; i <= n; i++)
{
    dp[i] = dp[i-1] + dp[i-2];
}
```

① base case fill karo.

② function ka dp mein main concept kaha (Array).

⇒ T.C → O(n).
S.C → O(n) (dp array)

→ Space optimization:

dP: 0 1 1 2 3 5 8 13 21 34

0 1 1 / 1 1 2 / 1 2 3 ...

Prev2 Prev Curr

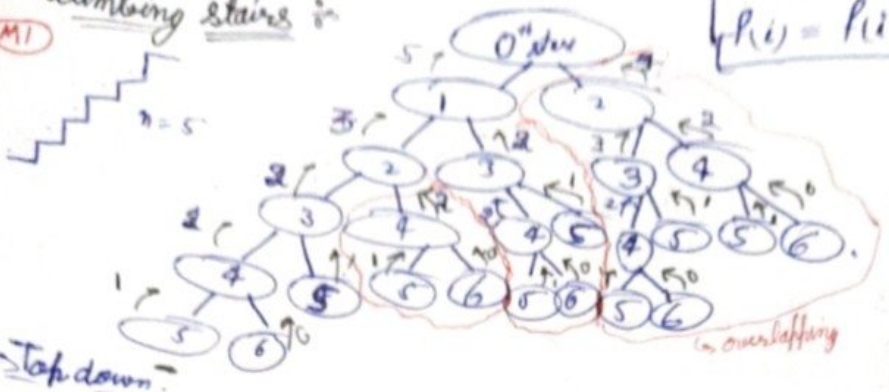
```
for (int i = 2; i <= n; i++)
{
    curr = prev1 + prev2;
    prev2 = prev1;
    prev1 = curr;
}
return curr;
```

T.C = O(n)

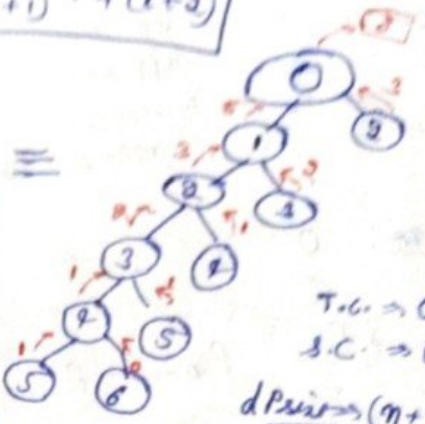
S.C → O(1)

Climbing stairs :-

M1



$$f(i) = f(i+1) + f(i+2)$$



T.C. → O(n)
S.C. → O(n) + O(n)

dP size → (n+2)

vector<int> dP(n+2, -1)

→ Top down :-

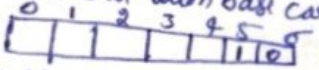
```
int count (int i, int n)
{
    if(i == n)
        return 1;
    if(i > n)
        return 0;
    if(dP[i] != -1)
        return dP[i];
    return dP[i] = count(i+1, n) + count(i+2, n);
}
```

→ TC → O(n)
→ S.C → O(n) + O(n)
dP Recursive tree.

→ Bottom up

① vector<int> dP(n+2, -1)

② Initialize it with base case.

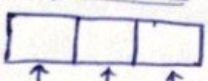


③ fill the remaining dP (reverse order of Top down)

```
for(i = n-1; i >= 0; i--)
{
    dP[i] = dP[i+1] + dP[i+2];
}
return dP[0];
```

→ TC = O(n)
→ S.C → O(n)

→ Space optimization




```
curr, next1 = 1, next2 = 0
for(i = 0; i >= 0; i--)
{
    curr = next1 + next2;
    next2 = next1;
    next1 = curr;
}
return curr;
```

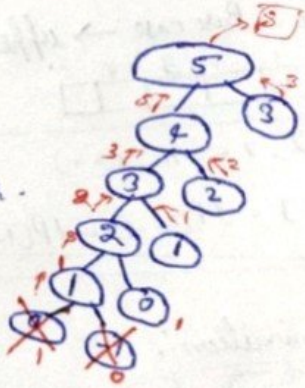
→ TC → O(n)
→ S.C → O(1)

M2

f(i) = f(i-1) + f(i-2)

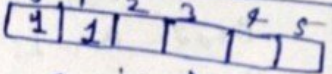
① Total no. of ways to go from 0th stair to 5th stair.
(nth)

② Total no of ways to reach 5th stair from 0th stair.



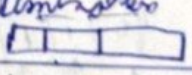
```
int count(int i, vector<int> dP)
{
    if(i <= 1)
        return 1;
    if(dP[i] != -1)
        return dP[i];
    return dP[i] = count(i-1, dP) + count(i-2, dP);
}
```

→ Bottom down :-




```
for(i = 2; i <= n; i++)
{
    dP[i] = dP[i-1] + dP[i-2];
}
return dP[n];
```

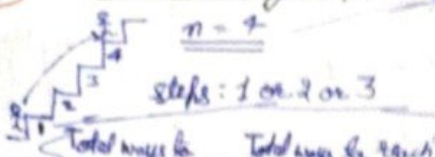
→ Space optimization



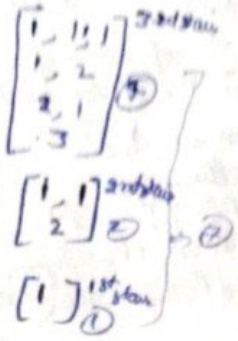
```
for(i = 2; i <= n; i++)
{
    curr = prev1 + prev2;
    prev2 = prev1;
    prev1 = curr;
}
return curr;
```

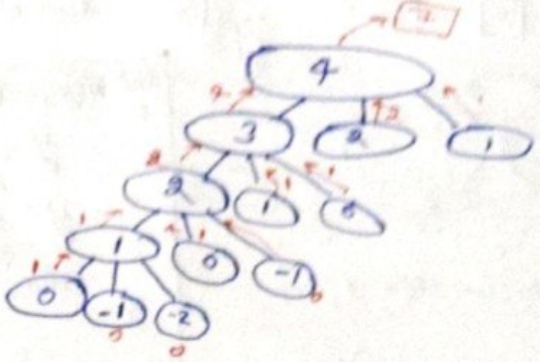
→ TC → O(n)
→ S.C → O(1)

Q1 Count number of hops: → {(2,2), (2,1,1), (1,1,2), (1,2,1), (1,1,1,1), (2,1,1), (1,2,1), (1,1,2)} → 7 ways.



Total ways to reach nth stair = Total ways to reach n-1 stair + Total ways to reach n-2 stair + Total ways to reach n-3 stair.





① Recursion

```
int ways(int n)
{
  if(n==0)
    return 1;
  if(n<0)
    return 0;
  return ways(n-1)+ways(n-2)
         + ways(n-3);
}
```

→ Top down. ⇒ vector<int> dP(n+1, -1)

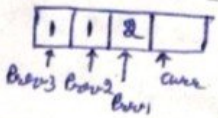
```
int ways(int n, vector<int> dP)
{
  if(n<=1)
    return 1;
  if(n==2)
    return 2;
  if(dP[n] != -1)
    return dP[n];
  return dP[n] = ways(n-1,dP)+ways(n-2,dP) + ways(n-3,dP);
}
```

↳ T.C. = O(n)
S.C. ⇒ O(n) + O(n).

→ Bottom up: ⇒ 0 → 4

```
for(i=3; i<=n; i++)          → O(n)
{
  dP[i] = dP[i-1]+dP[i-2]+   → O(n)
          dP[i-3];
}
return dP[n];
```

→ Space optimization




```
for(i=3; i<=n; i++)
{
  curr = Prev1 + Prev2 + Prev3;
  Prev3 = Prev2;
  Prev2 = Prev1;
  Prev1 = curr;
}
```

→ T.C ⇒ O(n)
→ S.C ⇒ O(1)

Q2 0th → nth stair