DOMS | Page No. 1
Date 26 / 4 / 24

INTRODUCTION TO C++ (RECORDED)

#= What is C++ ?

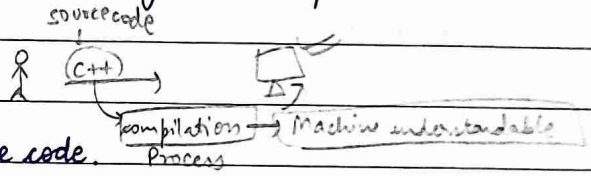
- Programming language (PL): is a computer language that is used by programmers to communicate with computers. It is a set of instructions written in any specific language to perform a specific task.

- C++ is a high-level, general-purpose PL.

Why do we need it?
→ Object oriented Programming.
→ Control over system resources and memory.
→ Performance: efficiency and speed.

- Uses of C++: system / software development, game development, real time systems

- Compilation Process



Turning source code into executable code.
Steps in compilation process: Preprocessing, compilation, assembly & linking

(Q) HW: Cpp is low-level or high-level language?
low level language: machine-friendly. Thus, very difficult to understand and learn by human.
High level language: human-friendly. Thus, very easy to understand & learn by any programmer.

C++ can perform both low-level and high level programming. ~~Hence, it is a mid level language~~. C++ is a high-level language. (low-level features for efficient system programming and high-level features for building complex applications)

Write your first program

int main () → starting point of program

{ define
← scope

}

What is return 0? ⇒ represents successful execution

⇒ if any other non-zero value is returned ⇒ unsuccessful execution.

let's print something

```
#include <iostream>
using namespace std;
int main () {
    cout << "Namaste Duniya" << endl;
    return 0;
}
```

HW Q1. Alternative of endl?

⇒ \n

Q2. if not using namespace std; then?

⇒ std:: cout << "Hi";

Q3 return -1?

⇒ unsuccessful execution

Q4. # include ⇒ preprocessor directive.

⇒ they are not C++ statements, so doesn't end with (;) semicolon.

⇒ They are lines of the source file where the first non-whitespace character is #, which distinguishes them from other lines of text.

.cpp → source code
.exe → executable file

Variables and Datatypes

Variables : named storage memory location.

eg: int marks = 50;

Syntax: datatype variable_name = value;

Datatypes : it specifies the size and type of information the variable will store.

//declaration

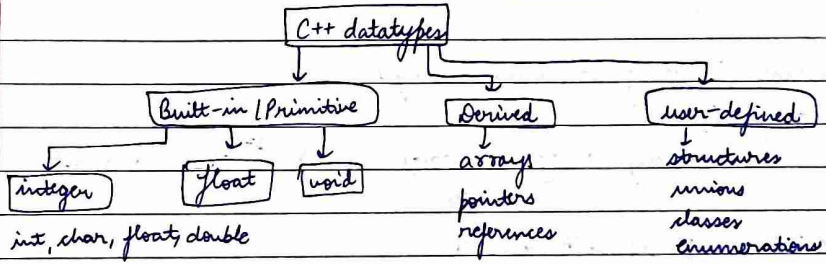
eg: int age; ⇒ it assigns any random value to variable age.
garbage value

age = 101;

//definition

eg: int age = 25;

* Same name ka variable 2 bar declare nhi kr skte ek code block mai. error aayega
⇒ dosre codeblock me dubara declare kr skte hai (even in nested {})



eg: double weight = 55.69887; ⇒ in o/p it is rounded off upto 6 places only.

bool isMale = false; ⇒ 0 in o/p.

bool isFemale = 1;

(HW) Q. How data is stored in memory?

Detailed info:

1 byte = 8 bits

Type	bits	range
int	32	-2147483648 to 2147483647
unsigned int	32	0 to 4294967295
signed int	32	-2147483648 to 2147483647
short int	16 (2 byte)	-32768 to 32767

int = 4 byte

Type	Bits	range
unsigned short int	16	0 to 65535
signed short int	16	-32768 to +32767
long int	32 or 64	-2147483648 to 2147483647 (32 bits)
unsigned long int	32 or 64	
signed long int	32	0 to 4294967295
float	32	±1.75E-38 to 3.4E+38
double	64	±2.2E-308 to ±1.7E+308
long double	96	±3.5E-4932 to ±1.18E+4932
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
bool	8	true or false

eg: int age=12;
cout << sizeof(age) << endl; → o/p → 4

* How data is getting stored in memory?
int age=5;
→ binary → 000---00101 (32 bits) → $2^{32}-1$ (max'm no.)

eg: 3bits [1|1|1] → 7 → 2^3-1
n bits → 2^n-1

* int
→ signed → +ve/-ve → Range $2^{n-1} \to 2^{n-1}-1$
→ unsigned → 0 to +ve → 0 to 2^n-1 if it takes n bits (n=32 here)

* bool
→ T → 00000001
→ F → 00000000
→ takes up 8 bits, i.e. 1 byte.

1 byte: smallest addressable space 1 bit: smallest memory location

HW → decimal N.S → binary N.S conversion and vice-versa.

* neno. — msB (most significant bit) [0 ... ... 1 1] → least SB (LSB)
-ve no. — first bit [1 ... ... 1]

How -ve no. are stored in memory?
→ 2's compliment = 1's complement +1

eg: 00101011
1's compliment = 11010100
2's compliment = add 1
= 11010101

decimal: 1+1 = 2, binary: 1+1=10
eg: 00000111+1 → 00001000

User input in C++ : cin
eg: int age;
cin >> age;

eg: bool marks; i/p → true
cin >> marks; o/p → 0
{give input as 0 or 1 in bool not as true or false}

(HW) Q. cin.ignore(), cin.fail(), getline(cin, 10)
Ans → ignore() is a method inside istream class.
cin.ignore() is used to ignore or clear one or more characters from input buffer. eg: cin.ignore(256,' '); → ignore 256 characters unless a space comes. This space is called delimiter.
→ cin.fail(): returns true when an input failure occurs.
→ cin doesn't allow to take input in multiple lines. input: Lakshay Kumar
To accept multiple lines, we use getline().
eg: cin.ignore(2^4, '\n'); → o/p → LS
eg: string name;
getline(cin, name);
cout << name << endl;
eg: cin.ignore(5, 'a'); → o/p → LK

→ cin.get : get one character.

Control flow

Decision making :

- if statement
- if-else
- if-else if
- if-else if-else
- nested if.

if-statement

syntax:
```
if (condition) {
    //statement
}
```

if-else block

```
if (cond") {
}
else {
}
```

if-else if block

```
if (cond") {
}
else if ( ) {
}
else if ( ) {
}
```

if-else if-else block

```
if ( ) {
}
else if ( ) {
}
else {
}
```

nested-if

```
if ( ) {
    if ( ) { }
    }
    else {
    }
}
else {
}
```

Switch statements → control flow.

syntax:
```
Switch (expression) {
    case val1:
        //statement
        break;
    case val 2;
        //statement
        break;
    default;
} //statement
```


eg :
```
switch(grade) {
    case 'A':
        cout << "marks btw 90 to 100" << endl;
        break;
    default:
        cout << "less than 60" << endl;
}
```

Conditions pertaining to 'switch'

- Expression type → grade/day → int / char / enum ⇒ float, string nhi nikal skte
- Unique Case Values → case value cannot be repeated
- No Range checking → ~~case (age > 18) :~~ ✗
- Fall through Behaviour → use break after every case
- Execution order.

Ternary Operator

syntax: condition ? expression_if_true : expression_if_false;

~~eg : (age > 18) ? cout << "can vote" : cout << "cannot vote";~~

eg :
```
int age = 12;
(age > 18) ? cout << "can vote" : cout << "cannot vote";
```

Loops - for, while, do-while

For loop

none of these are mandatory (;;)

syntax:
```
for (initialization; condition; updation) {
    //statement
}
```

Ⓐ Ⓑ Ⓓ Ⓒ

eg:
```
for (int i = 1; i <= 10; i++) {
    cout << "saumya" << endl;
}
```

Break keyword : ~~skip loop~~ ~~if used~~ skip loop

continue keyword : skip iteration.

while loop :

syntax:
```
initialization;
while (condition) {
    //updation;
}
```

Do-while loop

syntax:
```
initialization;
do { //statement
// updation
} while (c);
```
cond"

Q.
```
for (int i=1; i<=5; i++); {
}
```

Q.
```
if (cin >> i) {
    cout << "Saumya";
}
```
Ans: takes input and gives o/p as Saumya.

Q.
```
if (cout << "hi") {
    cout << "Saumya";
}
```
Ans: o/p → hiSaumya

Nested loops

syntax:
```
for ( ) {
    for ( ) {
    //statement
    }
}
```

eg:
```
for (int i=1; i<=2; i++) {
    for (int j=1; j<=2; j=j+1) {
        cout << i*j << " ";
    }
}
```
o/p → 1 2 2 4

Operators in C++ : is a symbol that operates on a value to perform specific mathematical or logical computations. They form the foundation of any programming language.

Unary Operator

- ++ --
  - → pre-increment and post-increment.

eg: ++a (operand) a++

++a: file increment, fir use

a++: file use, fir increment

- similarly for —
  - → post decrement and pre-decrement.

a-- : file use, fir decrement

--a : file decrement, fir use

Binary Operator

- Arithmetic
- Assignment.
- Relational
- Logical
- Bitwise

Arithmetic Operator : +, −, *, /, %. (perform on int only (try to avoid))

(Bonus concept)

* $\frac{int}{int}$ = int eg: $\frac{5}{2}$ = 5   * ; $\frac{int}{float}$ = float

* $\frac{float}{int}$ = float   * $\frac{double}{int}$ = double

Relational Operators : >, <, >=, <=, ==, !=
↳ check relation.
↳ gives o/p as 1 or 0.

Logical Operators : &&, ||, ! → ulta kardega, T ko F & false ka True
↳ for multiple conditions.
eg : age>18 && income ≥12

Bitwise Operators : &, |, ~ (tilda ☺), <<, >>, ^
↳ bit level pe lagega

Assignment Operators : =, +=, −=, *=, /=, %=
eg : int a=5;
a=a+5;] same thing.
a+=5;
a = a*b;] same
a* =b;
a=a/10.0] same
a/= 10.0

→ & →

a	b	o/p
0	0	0
0	1	0
1	0	0
1	1	1

eg : 5 → 00 — — 00101
4 → 00 — — 00100
5&4 ⇒ 000 - - - 100
5&4 ⇒ o/p ⇒ 4

'|' →

a	b	o/p
0	0	0
0	1	1
1	0	1
1	1	1

Eg: 5 | 4 ⇒ o/p ⇒ 4

'~' →

a	o/p
0	1
1	0

5 → 000 — — 000101
~5 → 1111 - - - 111010

'<<' → left shift   5<<1 ⇒ 000 - - - 001010   o/p ⇒ 10
↑ no. of shift
eg : 100<<5 → 100 × 2^5
5<<2 ⇒ 20   ★ 5<<n ⇒ 5 * 2^n

'>>' → right shift   1>>1 ⇒ 0000 - - 001   o/p ⇒ 0
↳ right shift by 1   ⇒ 00 — — 00
↳ can say we are dividing by 2.   eg : 10>>1 ⇒ o/p ⇒ 5
eg : 100>>3 ⇒ $\frac{100}{2^3}$ (in majority cases)   ★ A >>n ⇒ $\frac{A}{2^n}$

'^' → XOR

a	b	o/p	
0	0	0	same pe zero
0	1	1	→ opp pe one
1	0	1	
1	1	0	

⇒ a^a = 0
b^b = 0
5^4 ⇒ 5 → 00101
4 → 00 — 0100
5^4 ⇒ 1 → 001

HW Q. Bitwise operator can be applied on which type of datatype?
⇒ integer (byte, short, int, long) → cannot be used on floating (float, double)

HW Q. % operator on float?
⇒ NO.

Number System (N·S)

What is number system?

- method to represent numeric values or quantities using different digits.

Decimal Number System

- The decimal N·S has base 10.
- It uses digits from 0 to 9.
- Base: it is the number of symbols/digits a N·S uses.

Binary N·S

- It has base 2.
- It uses only two digits, i.e. 0 and 1.

(CPU works in binary, memory is stored in binary)

Counting in Binary N·S

Decimal	Binary	Decimal	Binary
0	0	12	1100
1	1	13	1101
2	10	14	1110
3	11	15	1111
4	100	16	10000
5	101	17	10001
6	110	18	10010
7	111	19	10011
8	1000	20	10100
9	1001	21	10101
10	1010	22	10110
11	1011	23	10111

Decimal to Binary Conversion

Division method

1. Divide number by 2.
2. Store remainder (that will be a bit in binary number)
3. Repeat above steps with the quotient until quotient is less than 2.
4. Reverse the bits so obtained.

eg: N=10 → 10/2 → 5

	Remainder
	0
5/2 → 2	1
2/2 → 1	0
1/2 → 0	1

N=10 (decimal)
1010 (binary)
10 → $(1010)_2$

for code → 0 → 0*10^0 + 0 = 0
1 → 1*10^1 + 0 = 10
0 → 0×10^2 + 10 = 10
1 → 1×10^3 + 10 = 1000
+ 10
1010 (store in int)

arrange

code → ans = 0
ans = (digit * 10^i) + ans

(for probable viva)

Bitwise method

1. Obtain bit with bitwise AND operation, i.e., N&1
2. Right shift N by 1. (N = N>>1)
3. Repeat above steps till N>0
4. Reverse the bits so obtained.

⇒ N&1 ① 10 → 1010
& 0001
0000 → 0 (1st bit)

② right shift → 101
001
001 (2nd bit)

③ right shift → 10
01
00 (3rd bit)
→ Reverse it

④ right shift → 000-01
001
001 (4th bit)

Binary To Decimal Conversion

1. Multiply each digit with its place value. (base)^i
2. Add up all place values.
3. Sum is the decimal number.

eg: 1010 → 10

1 0 1 0

$1\times2^3+0\times2^2+1\times2^1+0\times2^0$ = 8+0+2+0 = 10

eg: 10111 → 23

$1\times2^4+0\times2^3+1\times2^2+1\times2^1+1\times2^0$ = 16+0+4+2+1 = 23

code eg: binary = $(1010)_2$

① bit = 1010 % 10 ⇒ 0

dec = $0+0\times2^0=0$

N = N/10 ⇒ 101

② bit = 101 % 10 = 1

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

N = N/10 ⇒ 10

③ bit = 10 % 10 = 0

dec = $2+0\times2^2=2$

N = N/10 = 1

④ dec = $2+1\times2^3=2+8=10$

Typecasting

- Allows you to change the datatype of a variable from one type to another.
- Crucial when you need to perform operations involving variables of different data types, ensuring that the data is handled correctly.

Implicit Type Casting, aKa, Automatic Type Conversion.

- Compiler automatically converts one datatype to another during an operation.
- This happens when you perform operations involving variables of different datatypes, and the compiler promotes one type to a larger type to maintain precision.

eg:
```
int n1 = 10;
float n2 = 5.5;
float result = n1 + n2;
cout<<result;
```

Explicit Type Casting, aKa, Manual Type Conversion

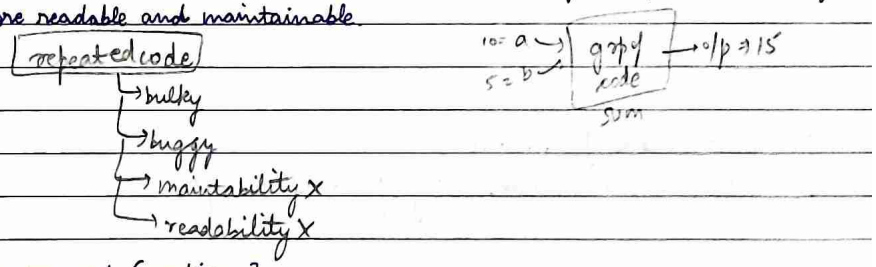
- Allows you to explicitly specify the desired datatype during an assignment or operation.
- You use the casting operator, which is represented by parenthesis containing the target datatype.

eg: `float result = n1 + (int)n2;`

Functions

What is a function?

A function is a way to group code into a single unit. It can take inputs, process them, and return a result. Functions helps in organising code, making it more readable and maintainable.



Why we need Function?

Fx^n's reduce repetition, simplify complex tasks, and helps in debugging. They also make it easier to test individual parts of your program.

Function Anatomy

- Declaration
- Definition
- Does order matter? ⇒ Yes

syntax: returntype functionName (input parameters); *optional

```
returntype functionName (input parameters)
{
    //body
}
```

returntype: void / non-void (int, char, float, etc)

eg: int sum(int a, int b) {
    return a+b;
}

Calling a function
eg: int ans = sum(5,10);
cout << ans;
(5, 10 ← arguments)

* Function should be above int main(). ⇒ if you want to Keep fx" below int main, then it should be atleast declared before int main.
(Jya fx jaisa bhi usse Kya)

declaration :
eg: int getsum(int x, int y); ← signature

definition : → fx" logic
eg: ~~int getsum~~ {
    //body
}

(Void wale fx" me return value nhi likhte but Khali return likh skte hai.)

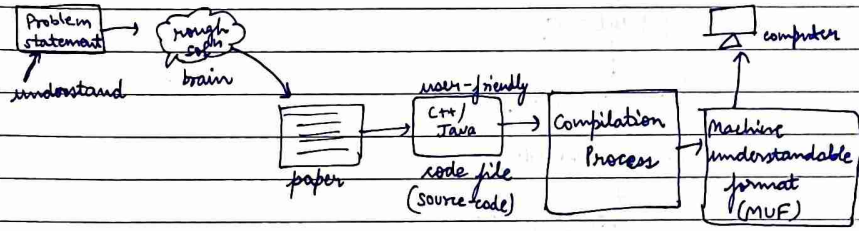
(HW)
Q. print Numbers from 1 to 100.
Q. Write a SIP calculator using function's concept.

BASICS OF PROGRAMMING - Level 1

[LIVE CLASS]
INTRODUCTION TO THE PROGRAMMING WORLD 29/4/24

* How To Approach a problem?

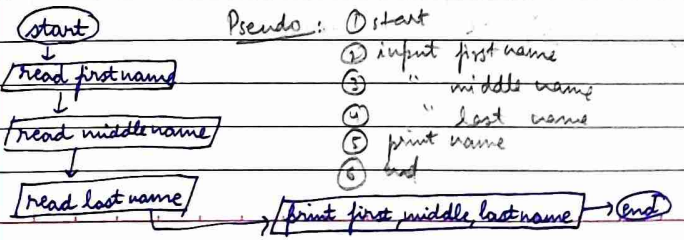
Thought process
→ let's first understand the problem
→ analyse problem → check input values / constraints / requirements.
→ logic building / approach / algorithm.



Series of steps: algorithm.

Flowchart & its components
Flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to arrive at a solution to a problem. It uses various symbols such as arrows, rectangles, diamonds to represent different types of actions or steps in a process.

① ⬭ : start / end : terminator
② ▭ : input / output
③ ▭ : process block (eg: a = a+b, etc other calculations)
④ ↓ : arrows : flow of execution.
⑤ ◇ : decision block

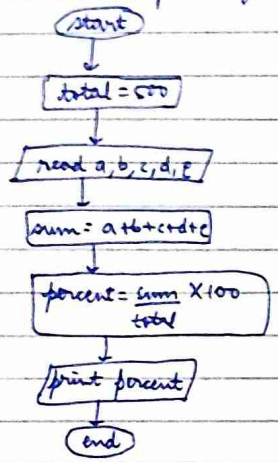
eg:


Pseudo:
① start
② input first name
③ " middle name
④ " last name
⑤ print name
⑥ end

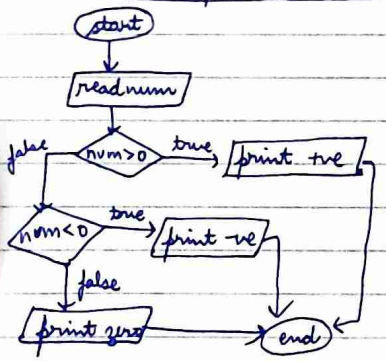
Pseudocode

It is a notational system that resembles a simplified version of a programming language, used for writing the algorithmic steps of a program in a human-readable form. It helps programmers plan and discuss algorithms without worrying about syntax details.

eg: Calculate B percentage



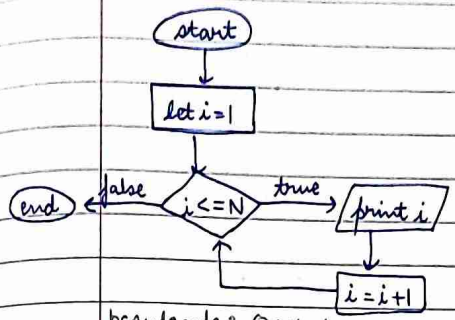
Check positive, -ve, or ZERO



Pseudocode

→ start
→ take input for num
→ if num > 0
  print +ve
  else if num < 0
    print -ve
  else print zero
→ end

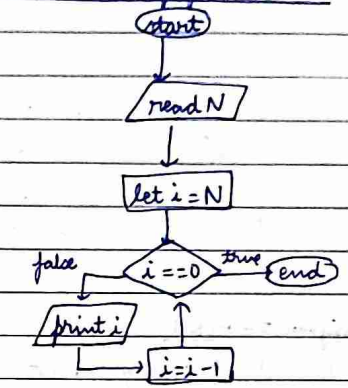
Print counting from 1 to N



Pseudocode:
① → start
② → take i/p for N
③ → let i = 1
④ → if i <= N → ⑤ print i ; i++
⑥ goto ④
⑦ else end

N = 4

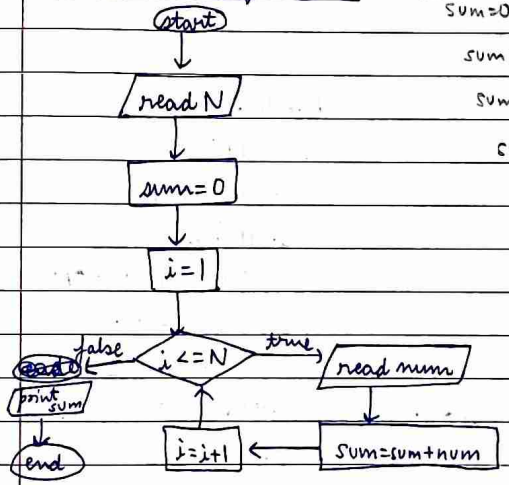
Print counting from N to 1



Add N Numbers from user

→ 5 number → intinos

sum = 0
a = 3
sum = 0 + 3
b = 4
sum = 0 + 3 + 4
c = 2
sum = 0 + 3 + 4 + 2 - - -

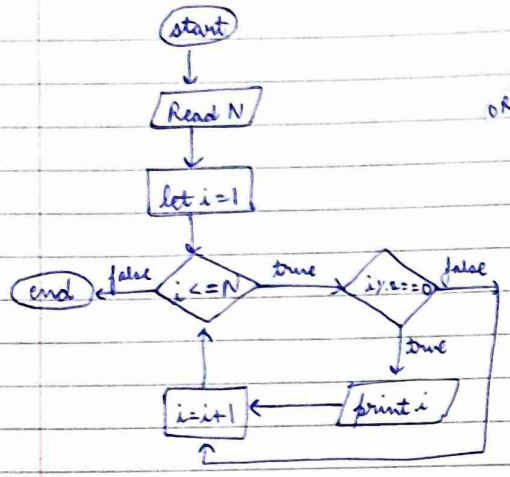


Print 1 to N numbers, only Even numbers

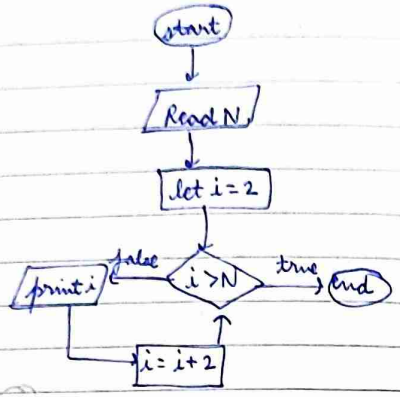
→ n % 2 == 0

(acha bacha ye tareeka abi use Krega
desire method badme
bayenge)

P.T.O



OR



OR

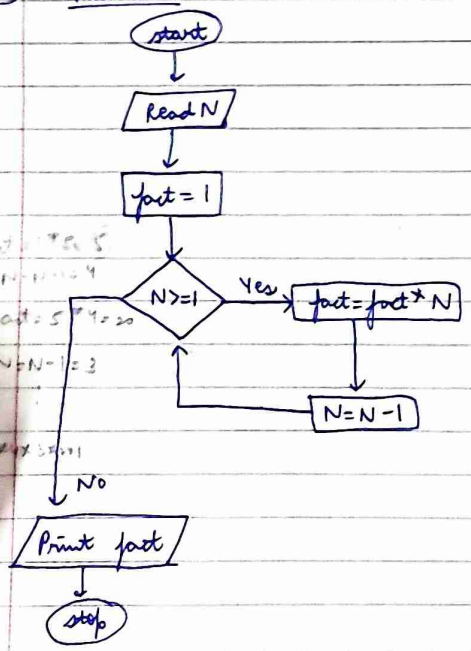
* last bit of even no ⇒ hold zero
odd no ⇒ hold one

N&1 → bitwise AND

Homework/Assignment-week-1

1. Check valid ▲ triangle or not
2. Check if a no is prime or not
3. Find factorial of a number

3. Flowchart



Pseudocode

1. Read N
2. Set fact=1
3. If N>=1
   A. fact=fact*N
   B. N=N-1
   C. Go to 3
4. Else print fact.
5. End

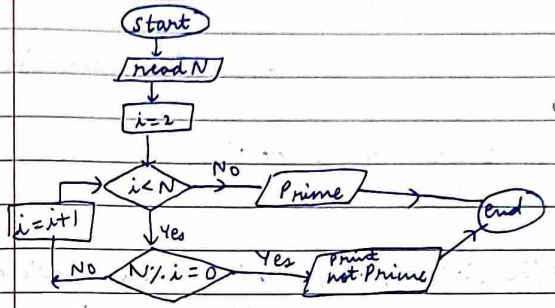
2. eg: 1 2 5 → 5(N)

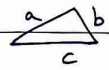
i=1	2	3	4	5
5/1 ✓	5/2 ✗	5/3 ✗	5/4 ✗	5/5=1 ✓

Pseudocode

1. Read N
2. Set I=2
3. If I<N
   (a) If N%.I=0
   → Print not Prime
   → Goto 5
   (B) else i=i+1
   (C) Go to 3
4. Else print "Prime"
5. End.

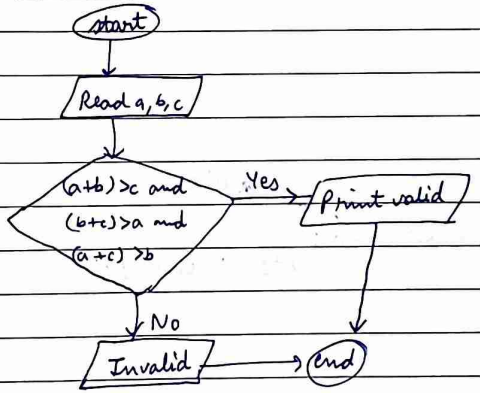
Flowchart



1. 

triangle: two side(sum) > other side

Flowchart



Pseudocode

1. Read a,b,c
2. If (a+b)>c and (b+c)>a and (a+c)>b.
   → Print valid triangle
   → Go to 4
3. Else print invalid Δ
4. End

BASICS OF C++ and FIRST PROGRAM (CLIFF CURES) (1 May 24)

* Let's write down the 1st code

Starting point : int main () {

return type, function name

.obj file

} → scope or code-block

* in main fn" → always return 0 for successful execution.

* #include <iostream>
  preprocessor directive → file (i: input, o: output → inse related functionality ki definition yaha hai)

* using namespace std;
  ↳ standard

eg: std::cout << "Hi";

⇒
```
#include <iostream>
using namespace std;
int main() {
    cout << "Namaste Duniya" << endl;
    return 0;
}
```
endl → termination of line

To print

Input / Output in C++

cin >> cout <<

give value to address of memory location.

eg: int age;
cin >> age;

Datatypes and variables

named memory location

→ type of data (int, float, string etc)
→ size of data (4 byte)

1 byte → 8 bits

double: can store more precision

char: a, ... z ; A ... Z ; @, ...

* use single quotes for character.

eg: char ch = 'saumya';
cout << ch;
⇒ o/p gives: a (diff behaviour in diff devices)
but keep only one character in char.
last character

eg: int age = 2;
by default it is signed
unsigned int age = 5;

⇒ sizeof(var-name) to get size in bytes.

* int → 4 byte → 32 bits

show an integer

eg: int a = 3

00000000	00000000	00000000	00000011
1st byte	2nd byte	3rd byte	4th byte

32 dabbe ⇒ Total combination : 2^{32} combination.

$1 \longrightarrow 2^{32}$

normal maths: $\frac{2^a}{2^b} = 2^{a-b}$

unsigned → $0 \rightarrow 2^{32}-1$ (range)

unsigned = $0 \rightarrow 2^n - 1$

signed

signed = $-2^{n-1} \rightarrow 2^{n-1} - 1$

eg: short

$-32768 \longrightarrow 32767$

cout << 32768 ⇒ o/p ⇒ -32768
cout << 32770 ⇒ o/p ⇒ -32766

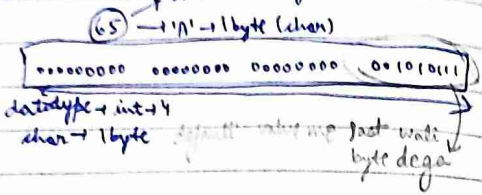
⇒ cyclic wapas jake print krdega. (cyclic nature)

Variable naming convention

- begin with alphabet
- no space
- digits may be used but only after alphabet
- no special symbol except underscore(_).

- cannot create ~~keyword~~ variable with reserved keyword or command.
- case sensitive (A & a are different variables)
- camel & case eg: myName

~~Operator is stored~~ Brain Teaser:



How data is stored?
(+ve or -ve integers)

OPERATORS, CONDITIONALS and loops [LIVECLASS] (6 May 2024)

Operators: symbols which carry special meaning.

- Arithmetic: +, −, *, /, % (remainder)
- Relational: >, <, >=, <=, ==, != (gives output as T or F)
- Assignment: =, +=, −=, *=, /=, %=.
- Logical: &&, ||, !
- Bitwise

- use brackets, there is no need of operator precedence.
- operand: value on which operation is performed.

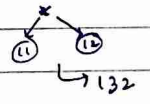
* Unary: single operand binary: 2 operands

* Assignment operator (shorthand)
  a=a+10 or a+=10 (similar for others)

* short circuit
  eg: (false || true || ~~false~~ || ~~true~~)
  ↑ dp

* ~~of~~ unary operator: ++, --
  ++ → a++ (post increment) → while use krne pe badme increment
  → ++a (pre-increment) → while increment krne ke liye use
  -- → a-- (post-decrement)
  → --a (pre-decrement)

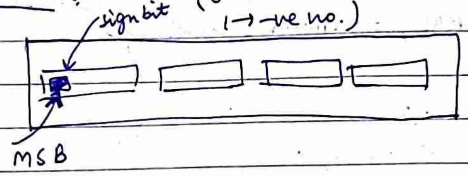
eg: ans = ++a * ++a → use brackets always.



compiler dependent ans: 132 or 144

How +/-ve no. are stored?

(0 → +ve no.
1 → -ve no.)



-ve no : → ① ignore -ve sign
② 2's complement is stored.

eg: a=-10

-10 → -ve X

a=10 ⇒ 00 — 001010

1's complement 1111 --- 0101

+1

111- - - -10110

memory : 1111 - - - - 11110110

(-ve) → 1's : 0000 - - - - 00001001

↓ 2's complement

+ 1

000 - - - 0001010

10

Conditionals : if, if-else, nested if, ternary operator, switch.

① Basic if

Syntax : if (condⁿ) {
—
}

② if-else

syntax : if (condⁿ) {
—
}
else { —
}

if-else if

syntax : if (condⁿ) {
—
}
else if (-) {
}
else {
}

③ nested-if

if (cond1) {
if (cond2) {
=
}
else { =
}
}
else { =
}

[Focus in intro to c/c++ recordings]

Q. if (min > age) {
—
}

Q. if (cont << 5) {
—
}

Q. if (condⁿ); {
—
}

* Ternary operator : (condⁿ) ? true : false

* Switch case :

Rules? → HW

Syntax : switch (expression) {
case:
▭
break;
case: ▭
break;
default: ▭
}

Loops : for, while, do-while

- **for**

```
for (initialization; cond"; updation){

}
```

- for(;;) → ∞ loop
- break : skip loop
- continue : skip iteration.

LET'S SOLVE PATTERNS (LIVE CLASS) (8 May 2024)

* outer loop ki har ek single value ke liye inner loop pura chalta hai.

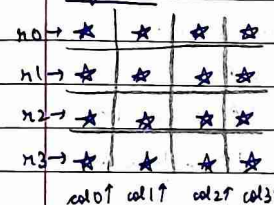
eg

```
for(int i =1; i<=3; i++) {
    cout << i << "→ ";
    for(int j=1; j<=2; j++) {
        cout << "*";
    }
    cout<<endl;
}
```

output:
1→ * *
2→ * *
3→ * *

Patterns : helps in logic building.

① **Square**



Rules :
① divide in rows and columns.
② find no. of rows.
③ row 0 : 4 *
row 1 : 4 *
row 2 : 4 *
row 3 : 4 *
(find formula by analysing rows)

⇒ no. of rows : 4
↓
outer loop
cols → inner loop

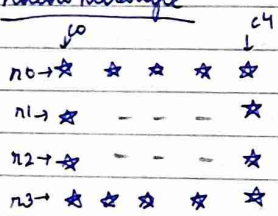
code:
```
// outer loop
for ( int row 0 to row < 4 ) {
    // inner loop
    for ( int col 0 to col < 4 ){
        cout << "*";
    }
    // after printing 4 stars, go to next line
    cout << endl;
}
```

② Rectangle

① no. of rows → 3
② r0 → 5★
r1 → 5★
r2 → 5★

```
//outer loop
for(r=0 to r<3){
    //inner loop
    for( c=0 to c<5){
        cout<<"★";
    }
    cout<<endl;
}
```

③ Hollow Rectangle



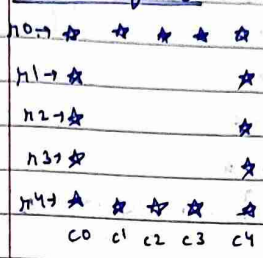
① no. of rows → n=4
② r0 → 5★
r1 → 1★ 3 space 1★
r2 → 1★ 3 space 1★
r3 → 5★

```
//outer loop
for(int r=0 to r<4) {
    for(int c=0 to c<5) {
        //logic
    }
}
```

→ logic : if(r=0 || row=last){
↳ cout << "★"
middle rows → else {
if(col=0 || col=last)
cout << "★"
else → space
}

(or) r=0 || row=last || col=0 || col=last
cout << "★";

④ Hollow square



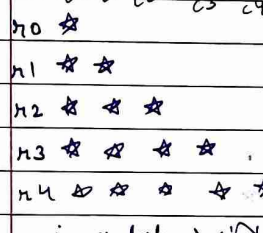
① ~~total~~ rows → n=5
② r0 → 5★
r1 → 1★ 3 space 1★
r2 " " "
r3 " " "
r4 → 5★

way 1 observation → first / last row / ~~col~~ ~~all~~ → ★
→ middle row → ~~space~~ last and first col → ★
else → space

OR

way 2 → r=0, r=n-1, c=0, c=n-1 → print ★
else ~~space~~ space

⑤ Half pyramid



inner loop → ~~single~~

① total rows → n=5
② r0 → 1★
r1 → 2★
r2 → 3★
r3 → 4★
r4 → 5★
∴ row+1 " stars

```
//outer loop
for( r=0 → r<n ) {
    for( c=0 → c<row+1){
        cout << "★";
    }
}
```

⑥ Inverted half pyramid

```
r0  ★ ★ ★ ★
r1  ★ ★ ★
r2  ★ ★
r3  ★
```

① total rows → n=4

② r0 → 4★
r1 → 3★
r2 → 2★
r3 → 1★
↳ n-row → print stars

(inputs available (n, r, star))

```
// outer loop
for(r=0 to r<n) {
    for(c=0 to c<n-r) {
        cout<<"★";
    }
    cout<<endl;
}
```

★★ hit & trial :

r	row	target
		5
r=5, r=0	5	
r=5, r=1	4	
r=5, r=4	1 ✓	

(n-r)

⑦ Hollow half pyramid

```
r0  ★
r1  ★ ★
r2  ★ _ ★
r3  ★ _ _ ★
r4  ★ _ _ _ ★
r5  ★ ★ ★ ★ ★ ★
```

① r0 → ★
r1 → ★ ★
r2 → ★ _ ★
---
r5 → ★ ★ ★ ★ ★ ★

if r==0 || r==1 || row==n-1 ⇒ ★
col==0 || col==(row+1) → ★
else space

⑧ Inverted hollow half pyramid

```
★ ★ ★ ★ ★
★   ★
★   ★
★ ★
★
```

① r=0, r=n-1, c=0, diagonal } star
② else space

diagonal → row==col
diagonal → row+col==n-1

⑨ Numeric half pyramid

```
r0  1
r1  1 2
r2  1 2 3
r3  1 2 3 4
```

c0, c1 (+1), c2 (+1), c3 (+1)

n=4
for(r=0 → r<4) {
  for(col=0 → col<row+1) {
    cout<<col+1;
  }
  cout<<endl;
}

⑩ Inverted numeric half pyramid

```
r0  1 2 3 4 5
r1  1 2 3 4
r2  1 2 3
r3  1 2
r4  1
```

c0 c1 c2 c3 c4 (+1)

n=5

	i/p	target
r0 → 5 no.	r=5, r=0	5
r1 → 4 no.	n-r ✓	
r2 → 3 no.		
r3 → 2 no.		
r4 → 1 no.		

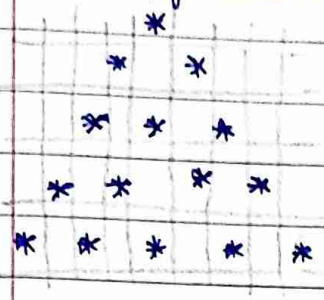
```
for(r=0 to r<n) {
    for(c=0 to c<n-row) {
        cout<<col+1;
    }
}
```

Homework: revise all patterns.

BASICS OF PROGRAMMING - level 2

Patterns (Mega class) (live class) 12 May 2024

① Full pyramid



n=5

① Outer loop : rows : 0 → 4 or 0 → N-1

② column wise

r_0 → 4 sp, 1 ★ → row+1

r_1 → 3 sp, 2 ★

r_2 → 2 sp, 3 ★

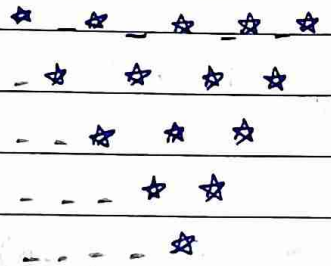
r_3 → 1 sp, 4 ★

r_4 → 0 sp, 5 ★

N=5

N-r-1 ✓

② Inverted Full pyramid



n=5

① Outer loop : rows : 0 → N-1

② inner loop : r_0 → 0 sp, 5 ★

r_1 → 1 sp, 4 ★

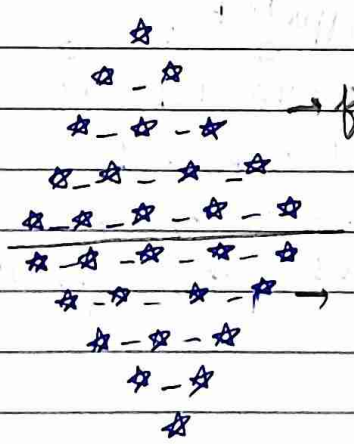
r_2 → 2 sp, 3 ★

r_3 → 3 sp, 2 ★

r_4 → 4 sp, 1 ★

r ↙ ↳ N - row

③ Diamond



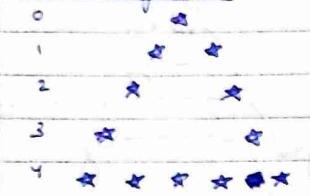
→ full pyramid

+

→ inverted full pyramid

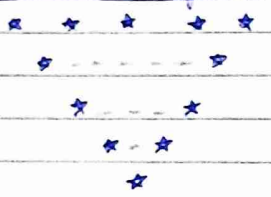
don't touch spaces wala loop

④ Hollow Pyramid



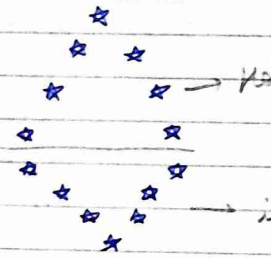
① $i==0$ or $i==N-1$ or $j==0$ or $j==i$
⇒ star

⑤ Inverted Hollow Pyramid



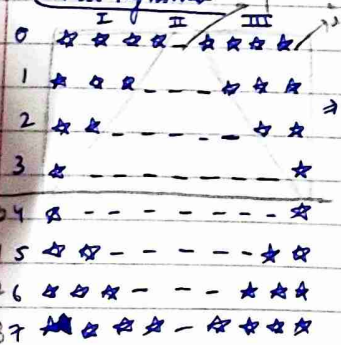
① stars: $i==0$ or $i==n-1$ or $j==n-i-1$ or $j==0$

⑥ Hollow diamond



→ Hollow Pyramid (remove $i==n-1$)
+
→ inverted Hollow Pyramid (remove $i==0$)

⑦ Mixed Pyramid



full pyramid
inverted half pyramid
⇒ n=4

upper half
I II III
n0 → 4★, 1 sp, 4★
n1 → 3★, 3 sp, 3★
n2 → 2★, 5sp, 2★
n3 → 1★, 7 sp, 1★

- (I) stars : $j: 0 \to n-i$
  (III) stars : $j: 0 \to n-i$
  (II) spaces : $0 \to 2i+1$

lower half : I II III
$i=0$ | $n=0$: 1★, 7sp, 1★
$n=1$: 2★, 5sp, 2★
$n=2$ → 3★, 3 sp, 3★
$n=3$ → 4★, 1sp, 4★

(I) : $j = 0 \to i+1$
(III) : $j: 0 \to i+1$
(II) : $j: 0 \to 2(n-i)-1$

⑧ Fancy 12 Pattern

1
2 * 2
3 * 3 * 3
4★4★4★4
5★5★5★5★5
0 1 2 3 4 5 6 7 8

n=5
outer loop → 0 → n
inner loop : n0 → 1 times
n1 → 3 times
n2 → 5 times
n3 → 7 times
n4 → 9 times
$j: 0 \to 2i+1$

what to print?
check if $j\%2$ ★ even → no. $(i+1)$
odd → ★

⑨ Full fancy 12

1
2 * 2
3★3★3
4★4★4★4
$i=0$ 3★3★3
$i=1$ 2★2
$i=2$ 1

fancy n=4 → n=3

lower half : outer $i = 0 \to n$
inner : $i=0$ → 5 times
$i=1$ → 3 times
$i=2$ → 1 time
$j: 0 \to 2(n-i)-1$
even → n-i
odd → ★

(10) ABCBA pattern

```
A
ABA
ABCBA
ABCDCBA
ABCDEDCBA
```


(Live class)

Bitwise Operators and Functions — 13 May 2024

Bitwise Operators

- & (and)
- | (or)
- ^ (xor)
- ~ (not)
- << (left shift)
- >> (right shift)

★ avoid using % → computation intensive operation
instead use "&" → faster
foreg: n & 1 → 0 or 1 (eg: 4 & 1 → 0 ; 5 & 1 → 1) if (n & 1) → odd else → even
↓ even ↳ odd

★ XOR (^): very important application (asked many times): find unique number. eg: 3, 8, 7, 4, 4, 12, 3, 12, 8.
diff(1) same(0)

3 ^ 8 ^ 7 ^ 4 ^ 4 ^ 12 ^ 3 ^ 12 ^ 8
↳ ans ✓

set bit : 1
eg: 0000 --- 00010[1] : 5

n & 1
1 → LSB: 0 → 0
↓ LSB: set bit

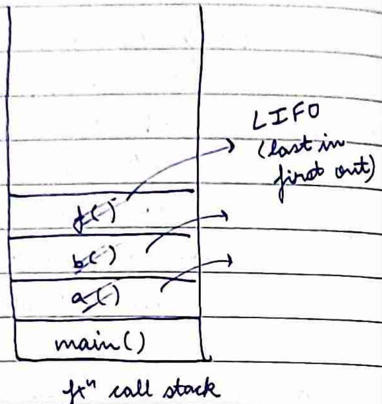
n = n & (n-1);
↳ set bit
n ^ (n & (n-1)) → return rightmost 1

⇒ 5: 0000 --- 00101
& 1 → 1st set bit
>>1 ⇒ 000 --- 0010 & 1 → 0
>>1 ⇒ 00--0001 & 1 → 1 → 2nd set bit

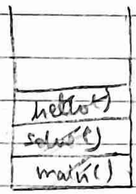
Homework: Bitwise Operators questions will be added (SHORTCUTS)

Functions

eg:
```
int main() {
    a();
    return 0;
}
int a() {
    b()
    return 1;
}
int b() {
    f()
    return 2;
}
int f() {
    return 1;
}
```



eg: (code)
```
int main() {
    cout<<1;
    solve();
    cout<<2;
    return 0;
}
void solve() {
    cout<<3;
    hello();
    cout<<4;
}
void hello() {
    cout<<4;
    return;
}
```



o/p → 13442

eg: sum of AP = $\frac{n}{2}(a+l)$

eg: Prime no. check (n=7)

- 2 → 7%2 = 1
- 3 → 7%3 = 1
- 4 → 7%4 → 3
- 5 → 7%5 = 2
- 6 → 7%6 = 1

} non-zero remainders → prime ✓

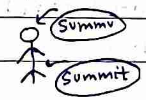
o) Pass by value | Pass by reference

- Pass by value → normally used; copy banti hai
- Pass by reference → used jab apne ko track rakhna ho; actual value me change hota hai

eg: Pass by value
```
int main() {
    int a=15;      // 104 address
    a++;           // →16
    solve(a);
    a++;           // →17
    cout<<a;       // →17
}
void solve(int a) {    // 216 address (eg.)
    a++;           // →17
    cout<<a;
    a++;           // →18
}
```

eg: Pass by reference → `void solve(int &a) { a=a+10; }` ← jo ni bhi same name ho

Reference variable: `int age=15;` (104, age → love)



`int &love = age;` (memory location ko ek or naam dedia)

Homework

Week-2 assignments

① Numeric hollow half pyramid

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

```
n=5
for (i=0→n) {
  for(j=0→ j<=i ) {          or j<i+1
    if (j==0 || i==n-1 || j==i) {
      cout<<j+1; }
    else { space }
  }
endl;
}
```

② Numeric hollow inverted half pyramid

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

① $j == i+1$

② $j == n$

③ $i == 0$

```
n=5
for(i=0→ i<n) {
  for( j=i+1 →j<=n) {
    if( i==0 || j==i+1 || j==n){
      cout<<j;
    }
    else { " "; }
  }
endl;
}
```

③ Numeric Palindrome Equilateral pyramid (left and right dono side se barabar)

n=5

		nos. K ∈ N → ++	
0	_ _ _ _ 1	1	< 5
1	_ _ _ 1 2 1	3	< 6
2	_ _ 1 2 3 2 1	5	< 7
3	_ 1 2 3 4 3 2 1	7	< 8
4	1 2 3 4 5 4 3 2 1	9	< 9
	0 1 2 3 4 5 6 7 8		

int c=1;

numbers ⇒ cout<<c;

row: i=0 → i<n

col: j=0 → j<K (K++)

spaces: i=0 → 4
i=1 → 3
i=2 → 2
i=3 → 1
i=4 → 0
j < N-i-1

eg: i=3 c=1

1 2 3 4 3 2 1

c=5 → c=c-2

Pseudocode
① n=5; K=n;
② for(int i=0 → <n){
int c=1;
③ for(int j=0; j<K){
④ if(j<n-i-1){ Print space}
else if(j<=n-1){ Print c; c++}
else if(j==n){c=c-2;
Print c; c--;}}
else { cout<<c; c--;}
K++;
cout endl;}

④ solid half diamond

n=5

0	☆	1
1	☆ ☆	2
2	☆ ☆ ☆	3
3	☆ ☆ ☆ ☆	4
4	☆ ☆ ☆ ☆ ☆	5
5	☆ ☆ ☆ ☆	4
6	☆ ☆ ☆	3
7	☆ ☆	2
8	☆	1

0 1 2 3 4

rows ⇒ i=0 → <9 (2n-1)

columns: cond^n ? ① if (i<n) → cond=i

② else ⇒ cond = n-(i%n)-2

j=0 → j<=cond {
cout<<" ☆ ";
}

i=5 → 4 ☆ (i%N) ⇒ 0 (i%5)
i=6 → 3 ☆ 1
i=7 → 2 ☆ 2
i=8 → 1 ☆ 3

N-(i%5) 5-2 = 3
4-2 = 2
3-2 = 1
2-2 = 0

(5) Fancy Pattern 1 (This pattern can go from N=1 to N=9 only)

col → 17 items

N = 5

row → i=0 → i<N

col → j=0 → j<17

i=0 → 8th index ⇒ Printing ✓

i=1 → 8-1 ⇒ 7th index

i=2 → 8-2 ⇒ 6th index ...

start ⇒ 8-i

digit Printing

	times
i=0 → i+1=1	1
i=1 → i+2=2	2

(6) Fancy Pattern 2

N=4

1) growing phase

row ⇒ 0 → <n

col ⇒ 0 → i

int c=1 ⇒ c++

c=11 ⇒ start = c-N

2) shrinking phase

row ⇒ 0 → <n

col ⇒ N-i-1

i=1 ⇒ start=4

update ⇒ start -(n-i-1)

(7) Fancy Pattern 3

N=5

row : 0 → <n

col : i=0 → j=0 (index)

growing: i=0 → N/2 ; i=1 → j=2 ; i=2 → j=4

i → j = 2*i

growing

cond ⇒ if (i<= N/2) ⇒ cond = 2*i ;

col ⇒ i=2 ⇒ 1 2 3 2 1 ⇒ cond=4 ⇒ if(j<cond){ cout << cond-j+1; }

shrinking : row: i=3 j(index) 2 ⇒ cond = 2(N-i-1)

i=4 0

(8) Floyd's Triangle

N=7

row : 0 → <7

col : 0 → i

int count=1 ⇒ cout << count.

count++

(9) Pascal's Triangle : binomial coefficient

c = c * (i-j)/j ;

i = [1,N]

j = [1,i]

(10) Butterfly Pattern

N=5

spaces: 8 6 4 2 0 0 2 4 6 8

2*(5-i-1) ⇒ 2*(N-i-1)

cond" (for *) N+(n-i-1)

cond" (for space) i-cond"-1

Function

① KM to miles → 1Km = 0.621371 miles

② Print all digits of an Integer (split order not anyega)

eg: N=1234

N%10 ⇒ reminder = 4

N/10 ⇒ get next digit ⇒ 123

while(n>0){

① ones place = N%10

② cout << ones place;

③ N=N/10;

}

③ Display area of circle : A = π * r * r = (3.14)(r)(r)

④ no. is even or odd?

m-1 : N%2==0 : even

m-2: Bitwise method eg: 5 → 000101

(faster) ↓

even: LSB: 0 always

odd: LSB: 1 always

if (N&1)==0 → even else odd

⑤ Find factorial of a number : eg: 5! i=1 ⇒ fact = 1×1 = 1

for(i=1; i<=5; i++){ i=2 ⇒ fact = 1×2 = 2

fact = fact*i; i=3 ⇒ fact = 2×3 = 6

} i=4 ⇒ fact = 6×4 = 24 --

⑥ No. is Prime or not? : N is prime when it has only 2 factors: 1 & N.

eg: N=5 5%1 = 0 | 5%2 = 1 | 5%3 = 2 | 5%4 = 1 | 5%5 = 0

Iterate

N ⇒ i ∈ [2 -- (N-1)].

↳ if (N%i==0){ optimised ⇒ for(i=2; i<=sqrt(n); i++)

//not prime

}

⑦ Print all prime from 1 to N

for(i=2 → <=n){ ...created in ⑥ ago

bool isPrime = checkPrime(i);

if (isPrime){ i };

}

⑧ Reverse integer

Leetcode 7: algorithm. i/p ⇒ N=123

o/p ⇒ 3 2 1 (but we don't know if it's 3 digit no.)

300

20

1 ⇒ 3×10² + 2×10¹ + 1×10⁰

⇒ remainder = N%10 → 3

ans = 0

(while n>0)

① ans = ans×10 + rem → 0×10+3 = 3 ⇒ N=N/10 ⇒ N=12

② ans = 3×10 + 2 = 32 ⇒ N=N/10 ⇒ 1

③ ans = 32×10 + 1 = 321

if n is -ve ⇒ |-123| → 123

↳ return → -ve

⇒ -123 ⇒ x = -(-123) ⇒ +123

↳ reverse

→ isNeg == 1

↳ -321

• Gives runtime error (signed integer overflow)

if (x<INT_MIN){

return 0;

}

while(-){ ---

if (ans>INT_MAX/10){

return 0;

}

⑨ Set the k^{th} bit

eg: N=10 → 1010
eg: K=2 → 1110 → o/p → 14

eg:
```
        1 0 1 0
Bitwise ← | 0 1 0 0
OR      ---------
          1 1 1 0
```

① N = 1010
② value ← 1<<K   eg: K = 2 → 1<<2 → 100
③ N | value

⑩ convert the temperature

Leetcode 2469. i/p → celsius

K = C + 273.15

F = C* 1.80 + 32
(OR 9/5)

⑪ ~~Count~~ ~~Convert~~ count all set Bits

→ Given an integer 'N', count no. of set bits (1) in it.

eg: N = 34   Binary = 100010

eg: N = 7 → '111' → set bit = 3

→ convert into binary and find no. of 1s in it.

M-1
① bit = N%2;
② if (bit==1) {count ++;}
③ N = N/2;

M-2 bitwise method ← faster

7 = "111"   no. ko bitwise AND karogenton us no. ki 1se

7&1 = 001   LSB mil jayegi then right shift. again AND with 1.

⑫ create number using digits

eg: 1,2,3,4,5,6 → 123456

① → int num=0
→ "1" → 0×10+1 = 1 = num

num=1 → num*10 → 1×10 → 10+2 = 12 ✓   ② num = num×10 + digit;

→ 12×10 → 120+3 = 123 ...

ARRAYS, TIME COMPLEXITY & SPACE COMPLEXITY

Let's learn Arrays (live class) 15 May 2024

Arrays: packet/container which store SAME type items.

★ Creation of an array

eg: int arr[10000];   declaration of array (garbage value allocated by default)

syntax: datatype arrname[size];

eg

★ array elements must have continuous memory.

eg: int a[5];

104 108 112 116 120

&(a) address

int size;
>>size;
int arr[size];
BAD practice

→ memory allocation: symbol table
(done by address and not by any name)   a → 104 (address)

★ Initialization of array

Method 1: eg: int arr[5] = {10, 20, 30, 40, 50}

★ if array size is large and elements given is less then
eg: int arr[5] = {10, 20}

| 10 | 20 | 0 | 0 | 0 |

syntax: datatype arrname[size] = {elements}

★ fill (...)

→ Indexing
counting: 1 → n (human numbers)
: 0 → (n-1) (")

eg: int marks[6]; → starting/base address
104 → array size: 6, index: 0-5
marks 1 2 3 4 5

•) 0 based numbering / indexing is followed in arrays.

* (Short ...) ... array size
... : 0 → n-1

* fill :
eg: int arr[5] = {10, 20, ...};
syntax: fill (starting address, ending address, value)
→ fill (arr, arr+3, 23)

104 → arr+3
arr ... arr+2

* Taking input
eg: int arr[3]
→ 3 block
arr[0] → cin >> arr[0]
arr[1] → cin >> arr[1]   } loop
arr[2] → cin >> arr[2]

```
for (int i=0; i<5; i++){
    cin >> arr[i]
}
```

* arr → [10|20| | | |]
sizeof(arr) = 24 (6×4)

⇒ array ke andar number of elements kitne hai ye hum kisi formula se find out nhi kr skte.
iske liye, alag se ek variable create krna padega jo carry krta hua chlega kitne element array ke andar hai. (int n=2)

•) Why indexing start from 0?
* arr[i] → value present at [base address + size × i]
                                  of datatype (int = 4)
or i[arr] → ... likhte hai
eg: ...
104 base address
104 108 112 116
arr 0 1 2 3 4
arr[0] → value present at (104 + 4*0)
: value at 104
arr[3] : ... (104 + (4*3))
: value at 116

* arr[i] ⇒ i[arr]
compiler → *(arr + i)   *(i + arr)
value present at address
eg: arr
→ 104
→ value present at address (104 + ...)   size of datatype ke acc. add krna hai
⇒ " " (104 + 12)   10 ... 107
" " " " 116   int me +1 ... bool me +1 ... se +4 hoga, ...
depends on Datatype (byte)

Functions with Arrays
eg: int main(){   int solve(int arr[], int size) {
  int arr[5] = ...;   }
  int size = 5;
  solve(arr, size);
}

* arr → pass by value
pass by reference ✓ → actual value pe kaam hoga
size → pass by value

① Algorithm → linear search

0th 1st 2nd 3rd 4th

array

int arr[5] = {10, 20, 30, 40, 50}

arr | 10 | 20 | 30 | 40 | 50 |
0 1 2 3 4

target → find (53) → element not found

target (40) → element found

for (int i = size-1; i>=0; i--){ // for" also

}

Q. Find max no. present in array.

| 13 | 21 | 14 | 17 |
0 1 2 3

To find max no. initialize ans = INT_MIN ; 21 → final ans

int → range $-2^{31} \to 2^{31}-1$ → INT_MIN, INT_MAX

To find min no. initialize ans = INT_MAX

Q. count 0's and 1's in array

| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

zeroCount = 0 +1 +1 +1 → 3

oneCount = 0 +1 +1 +1 +1 → 4

2 pointer technique

★ i/p → arr | 10 | 20 | 30 | 40 | 50 | 60 |
0 1 2 3 4 5

i=0, j=n-1 ; i++, j-- → j<i → stop

o/p → 10 60 20 50 30 40 (alternate printing)

odd no. of elements → | 10 | 20 | 30 | 40 | 50 | 60 | 70 |
0 1 2 3 4 5 6

while (i<=j) → ✓ also = sign lagadena i=j wale case ke liye.

(HW)Q ① Swap (using temp var..) → swap() ; temp ; (interview) } code github

XOR swapping: a=3, b=6
- a=a^b ; 011 ⊕ 110 = 101 → a=5 ; b=6
- b=a^b ; b=5 ; a=3
- a=a^b ; a=6 ; b=5 ✓

② reverse array

③ Watch Time & Space complexity recording

An introduction to Time and Space complexity (recording)

What is Time Complexity?

- amount of time taken by an algorithm to run as a function of length of input ↳ CPU operations (not actual time)

eg: for (int i=0; i<N; i++){
cout<<"Hi";
}

f ∝ N
N↑ ⇒ time taken ↑ or no. of operations ↑

⇒ TC : O(N)

Why to study Time and Space complexity?

- good computer engineer always thinks about the complexity of code written.
- resources are limited.
- measure algorithm to make efficient programs.
- Asked by interviewer after every solution you give.

What is space complexity?

- amount of space taken by an algorithm to run as a function of length of input.

eg: int a=1; // variable
int b[5]; // array
for(int i=0→n){ — } O(1) → constant space
(variable not being used)

eg: int n;
int *b = new int[n];
for → print array b ⇒ for(int i=0→n; i++){ cout<<b[i]; } eg: n=2 b[2] eg: n=2000 b[2000]
}
SC: O(n)

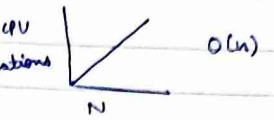
units to represent complexity,

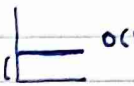
1) * Big O: upper bound (jyada se jyada kitna time lega) → worst case
2) Theta θ: average case
3) Omega Ω: lower bound (kam se kam kitna time lega)

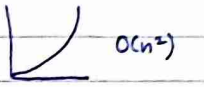
eg: search: [1|2|3|4|5|6]
→ item find → ① → omega : O(1)
⑥ → Big O : O(n)
worst case

Big O : complexities

1. constant time: O(1) → int a=5;
2. linear time: O(n) → for(int i=0 to N){ cout<<i; }
3. logarithmic time: O(log N) → binary search
4. quadratic time: O(N^2) → for(i=0→N){ for(j=0 to N){ -- } }
5. cubic time: O(N^3) → 3 level nesting







eg. f(n) = 2n² + 3n ⇒ O(2n²) ⇒ O(n²)
upper bound dekho

eg. N² + log N ⇒ O(N²)

eg: 200 ⇒ O(1)

* least complexity → most complexity
O(1), O(log N), O(√N), O(N), O(n log n), O(n²), O(n³), O(2ⁿ), O(N!), O(Nⁿ)

eg: [1|2|3|4|5|6|7]
① linear search ⇒ O(n)
② binary search ⇒ O(log₂N)

[1 2 3] [4] [5 6 7]
x middle

N → N/2 → N/4 → N/8 → ---- 1

5 6 7
↑ ans

N / 2^k = 1

log₂N = log₂k ⇒ K = log₂N (no. of operations)

* for(){ } O(N) + O(M) ⇒ O(N+M)
for(){ }
(not nested loop)

→ space

eg: ① int a=5;
O(1)

② int* a = new int[N] → O(N)

Arrays- Class-2 (live class) 19 May 2024

Leetcode 136: → hashing method → Mega array class
XOR (most optimized)
repeat in O/A
loop for(i=0→N) → O(N) : TC
int ans=0; → O(1) : SC

Sort 0 & 1s

Q. | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

Q. | 0 | 0 | 2 | 1 | 1 | 2 | 0 | 0 | 0 | → Dutch National Flag or sort 0, 1 and 2's
HW

HW → Q. | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
flip 0→1, 1→0 } 1s complement

Q. 1 → -1   2's complement

Sort 0s and 1's
i/p | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
o/p | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

algorithms
① counting
zerocount=3; onecount=5;
3zero insert, 5one insert
TC: O(N)+O(N) = O(N)
SC: O(1)

② sorting : inbuilt : sort(arr, arr+n)
ascending
→ TC: O(nlogn)

③ 2 pointer → l=0, h=n-1
eg: | 0 | 1 | ... | 0 | 1 | 1 |
jab l=1 ho h=0 ho toh swap krdo nhi toh ...

Q. | 10 | 20 | 30 | 40 |
i/p → 0 1 2 3
print all pairs

(10,10) (20,10) (30,10) (40,10)
(10,20) (20,20) ...
(40,40)

i=0: 10 → 10, 20, 30, 40
i=1: 20 → 10, 20, 30, 40

Sol:
M-1
for(i=0 → <n){
for(j=0 → n){
all pairs
}
}

M-2 (variations)
for(i=0 → n){
for(j=1 → n){
}

M-3 (variation)

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

M-4 (variation) upper Δ with diagonal

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

M-5

```
for(i=0→n){
    for(j=n-1; j>=i; j--){
    }
}
```

M-6 (variation) lower Δ excluding diagonal

```
for(i=0→n){
    for(j=n-1; j>i; j--){
    }
}
```

Two Sum

i/p → | 10 | 5 | 20 | 15 | 30 | (indices 0 1 2 3 4)

target → 35

check if a pair that sums equal to target exists or not

pairs:

10,10	5,10	20,10	15,10	30,10
10,5	5,5	20,5	15,5	30,5
10,20	5,20	20,20	15,20	30,20
10,15	5,15	20,15	15,15	30,15
10,30	5,30	20,30	15,30	30,30

logic: i=0→n
j=0→n
→ arr[i] + arr[j] == target

TC: O(N^2)
SC: O(1)

Pair

function → int — int, int (pair)
→ bool
→ char
→ string

pair<int, int> p
[|] int, int

Toolbox
p.first
p.second

Pair

declare
→ pair<int, int> p;
↳ var_name

initialize
↳ pair<int, int> p = make_pair(.., ..)

access
p.first = 100
p.second = 200

Print all triplets | 10 | 20 | 30 | 40 | (indices 0 1 2 3)

10,10,10   10,20,10
10,10,20
10,10,30
10,10,40

[10|20|30|40] [10|20|30|40] [10|20|30|40]
4 × 4 × 4 = 64

i	j	k

i=0→n
j=0→n
k=0→n

TC: O(N^3)
SC: O(1)

Leetcode 1 — **Two sum**: duplicate pair X

| 10 | 20 | 30 | 40 |

(M-1)

```
for(i=0→n){
    for(j=i+1→n){
        //logic  nums[i]+nums[j]==target
        return
    }
}
```

10,20   20,30   30,40
10,30   20,40
10,40
→ unique and relevant pairs

approach → find all pairs
→ check if their sum = target

TC: O(n^2)
SC: O(1)

(M-2) Two Pointer approach (Pg 70)

HW → watch vector STL recording

Three sum

i/p → | 10 | 20 | 30 | 40 |

target → 70

find a triplet ↳ sum = target

non-duplicate

10 20 30 → 60

10 20 40 → 70

20 30 40 → 90

TC: $O(n^3)$

SC: $O(1)$

Leetcode 15.

Leetcode 189.

↳ Shift array → 1 element, 2 element, n element

eg: | 10 | 20 | 30 | 40 | 50 |

1 element right shift → | 50 | 10 | 20 | 30 | 40 |

eg: | 10 | 20 | 30 | 40 | 50 | → | 40 | 50 | 10 | 20 | 30 |

n = 2

Method 1: temp method → SC: O(n)

Step 1 ↳ store last n element in a temp array | 50 | 60 | (Size − n → n)

Method 2: reverse method (better) SC: O(1)

1. Reverse
2. first part reverse
3. last part reverse → 5 6 7 1 2 3 4

Step 2 ↳ shift all elements by n places arr[i] = arr[i − n]

Step 3 ↳ copy temp array in original array

* 6 places se shift Krna = 0 place shift Krna
7 " " " = 1 " " "
and so on ...

modulus
whenever shift > size → shift = shift % size

Golden rule: dry run every ques.

Vector STL (revision)

- The standard Template Library (STL) provides a collection of template classes and functions that offer common data structures and algorithms to make programming more efficient and convenient.
- A vector in C++ is a dynamic array that can grow or shrink in size, make it a versatile and efficient data structure for storing and manipulating sequences of elements.

Features

~~Contiguous memory~~

eg: int arr[5]; ↳ static array (static memory allocation)

int n; → dynamic array
cin>>n;
int *arr = new int [n];
(Dynamic memory allocation)

- in vector, there is no need to tell size of vector; just keep inserting values, STL will manage the allocation.

syntax: vector<int> v; (v → variable name)

To access size: v.size();

To insert value: v.push_back(5); (5 → value)
v.push_back(10);

capacity (not needed)	size (size se nahi hai bas)	
1	1	[1] size ↑ Kr rhai hai tab capacity double Krdo
2	2	[1|2]
4	3	[1|2|3]
4	4	[1|2|3|4] (value add Kardia isliye capacity double)
8	5	[1|2|3|4|5| | |]

v.capacity()

Delete element: v.pop_back(); //delete from end

* v.at(i) → print element at i^{th} index.
* v.clear() → clear the vector

diff. ways to create vector

```
vector<int> arr;
vector<int> arr2(5);        // 0 0 0 0 0
vector<int> arr3{1,2,3,4,5};
vector<int> arr4{1,2,3,4,5};
```

(can push and pop as per requirement)

copy vector (vector can be char, float, other datatype too)

Eg: `vector<int> arr = {1,2,3,4,5};`
`vector<int> arr1(arr);`

* Front element: v[0] OR v.front()
  Last element: v[v.size()-1] OR v.back()
  eg: 3-1 (2nd element) v[2]

another method to print elements of vector

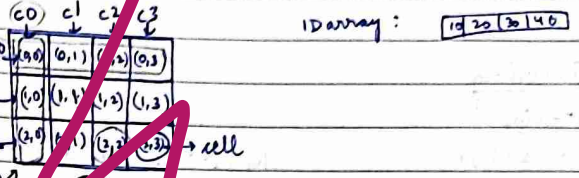
```
for(auto it : vector_name){
    cout<<it<<" ";
}
```

Features of vector:

1. contiguous memory
2. dynamic sizing
3. Automatic resizing
4. size and capacity
5. array-like access

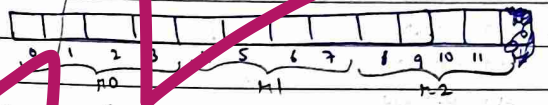
Array class-3 (Live class) 20/May/2024

2D arrays



1D array: 10 | 20 | 30 | 40

memory



* formula: i * c + j (convert 2D indexing to 1D)
  c → total number of columns
  i: row index ; j: col index
  eg: 4 × 2 + 2 = 10

creation

* declare: int arr[ ][ ]; (rowsize, colsize)
  → garbage value (only in case of static array)
  → ... (in case of dynamic array)
* initialization: int arr[4][3] { {10,20,30}, {11,12,13}, {...,16,17}, {20,21,22} }
* access: arr[i][j] (i: rowIndex, j: colIndex)
* print array:

```
for(row0 -> <3) {
    for(col=0 -> <3){
        cout<<arr[row][col];
    }
}
```

row0 → col0 col1 col2
row1 → c0 c1 c2
row2 → c0 c1 c2

TC: $O(n^2)$

Q. fill 2D array? ⇒ fill(&a[0][0], &a_add + row*col), value)

Traversing

① row-wise

10	20	30
110	120	130
210	220	230

```
for (i=0 → <rowSize) {
    for (j=0 → colSize) {
        arr[i][j]
    }
}
```

TC: $O(n^2)$ or $O(m \times n)$ (if m!=n)

o/p ⇒ 10 20 30
110 120 130
210 220 230

② col-wise

```
for (col=0 → <m) {
    for (row=0 → <n)      → square matrix
        arr[row][col]
}
```

o/p ⇒ 10 110 210
20 120 220
30 130 230

③ diagonal traversal

```
for (row=0 → <m) {
    for (col=0 → <n) {
        if (row==col) {
            cout<<arr[row][col]
        }
    }
}
```

or

```
for(i=0→ i<rowSize){   (or colsize)
    arr[i][i];
}
```

④ ↗ diagonal

```
int j=ncols-1;
for(i=0; i<rows; i++){
    cout<<a[i][j];
    j--;
}
```


★ Input

```
eg: int arr[3][3] — for (i=0 → rowSize) {
                        for (j=0 → colSize) {
                            cin>>arr[i][j]
                        }
                    }
```

→ row wise

Q. Search an element (if it's present or not)

linear search

```
if arr[i][j] == target
    ⇒ return true
```

10	20	30	40
50	55	57	59
69	7[illegible]	79	89

target → 79

2D array with function

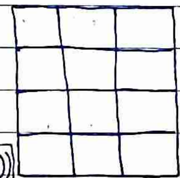
```
int main(){
    int arr[2][2] = {0}
    int rowSize=2, colSize=2
    solve(arr, rowSize, colSize);
}

void solve(int arr[][2], int rowSize,
           int colSize) {
    —
}
```

↓ gives error

(row nhi doge toh chlega but col dena padega)

2D array using vectors

int arr[4][3]



vector<vector<int>> arr(4, vector<int>(3,0))

- vector<vector<int>> → 2D array
- arr → name
- 4 → no. of rows
- 3 → no. of columns
- 0 → value initialize

```
int rowSize = arr.size()
int colSize = arr[0].size();
```

★ array jab bhi aap pass krte ho wo by default pass by reference hota hai, vector jab bhi aap pass krte ho wo pass by value hota hai.

Q. find minimum value

int minVal = INT_MAX

↳ find ...

70	23	41
2	69	42
56	1	18

```
for(i→0 ⇒ rS){
    for(j→0 ⇒ cS){
        minVal = min(minVal, arr[i][j])
    }
}
```

rS : rowSize
cS : colSize

TC: O(n×m)

SC: O(1)

Q. find maximum value

maxValue = INT_MIN

Q. find sum.

① row-wise sum.

	20	30	→ 60
	50	60	→ 150
70	80	90	→ 240

row 0 int sum=0
↳ col0 + col1 + col2 → print sum

row 1 int sum=0
↳ col0 + col1 + col2 → print sum

row 2
↳ int sum=0
col0 + col1 + col2 → print sum

② col-wise sum

10	20	30
0	80	80
20	40	60
↓ 70	↓ 140	↓ 170

col 0
↳ int sum=0
row0 + row1 + row2 → print sum

col 1
↳ int sum=0
row0 + row1 + row2 → print sum

col 2
↳ int sum=0
row0 + row1 + row2 → print sum

③ Diagonal sum

```
for (0 → <3) {
    sum = sum + arr[i][i];
}
```

10	20	30
15	20	50
25	10	10

MN 40

Q. Transpose of a matrix

M-① new empty array
↳ row loop
↳ col loop
↳ arr[j][i]

10	11	12
20	21	22
30	31	32

arr[i][j] → transpose (rows, col) → arr[j][i]

10	20	30
11	21	31
12	22	32

M-② without making new array

⇒ swap(arr[i][j], arr[j][i]) → original array hi aajayegi

dry run:

(0,0) 10	(0,1) 11	(0,2) 12
(1,0) 20	(1,1)	(1,2) 22
(2,0) 30	(2,1) 31	(2,2) 32

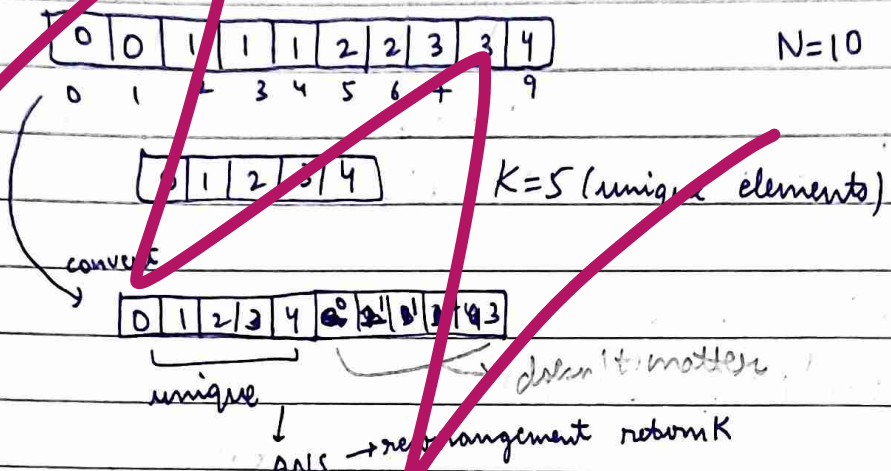
do baar swap ho nhi hai
↳ ek baar swap karo (upper triangle)

```
for (i=0 ; i<row ; i++){
    for(j=i ; j<col ; j++){
        swap
    }
}
```

leetcode 26.

Week-3 assignment

① ★ Remove Duplicates from sorted array.



Two pointer

0 0 1 1 1 2 2 3 3 4

i =

j =

j i

position of unique elements

if (a[i] == a[j]) → duplicate

i++

else → non-duplicate

{ j++ ; a[j] = a[i]; i++ }

↓ i ko j me store krdo (for rearrangement)

⇒ K = ??

K = j+1

TC: O(n)

SC: O(1)

Leetcode 724.

② Find Pivot index

nums → 1 7 3 6 5 6

sum: 11 11

Incase of multiple pivot index, jo left me hoga wo wala return karna hai

(M1) → Brute force

loop → i → [0→5]

(jis index pe hum hai use include nit krna)

```
for(int j=0; j<i; j++){
    leftsum += a[j]
}
for(int j = i+1; j<n; j++){
    rightsum += a[j]
}
```

i=0, lsum=0 ; rsum = 27 ×

i=1, lsum= ; rsum = 20 ×

i=2, lsum= 8 ; rsum = 17 ×

i=3, lsum=11 ; rsum=11 ✓ ⇒ return i

TC: O(n) (outer loop)

O(n) (inner loop) · O(n)

TC ⇒ $O(n^2)$

SC: O(1)

(M2) →

	0	1	2	3	4	5
	1	7	3	6	5	6
lsum	0	1	8	11	17	22
rsum	27	20	17	11	6	0

TC: O(n)

```
for (i = ; i < nums.size(); i++){
    lsum[i] = lsum[i-1] + num[i-1];
}
for (i = n-2; i >= 0; i--){
    rsum[i] = rsum[i+1] + num[i+1];
}
```

③ Key Pair // Two sum

(M-1) Page 59

(M-2) Two pointer approach

| 1 | 4 | 45 | 6 | 10 | 8 |

sort ⇒

| 1 | 4 | 6 | 8 | 10 | 45 |
0 1 2 3 4 5

l → 0, h → 5

```
l = 0
h = n-1
while (l < h) {
    csum = a[l] + a[h]
    if (csum == target)
        return true;
    else if (csum > target) {
        h--;
    }
    else (csum < target) {
        l++;
    }
}
```

① l=0, h=5
a[l] + a[h] = 46
csum = 46 ⇒ h--
target = 16

② l = 0, h = 4
sum = 11 ⇒ l++

sort : O(nlogn)
loop : O(n)
TC: O(nlogn)
SC: O(1)

④ leetcode 268.
Missing Number (Given in sorting & missing num-1)

another method : sort(arr.begin(), arr.end())

```
for (i = 0 ; i < n.size ) {
    if ( i == arr[i] continue
    else
        return i
}
return arr.size();
```

TC: O(nlogn)
SC: O(1)

XOR method : 9^6^4^2^3^5^7^0^1 (A^A = 0)

(XOR with index) ^0^1^2 - - ^7^8^9

TC: O(n) → optimise
SC: O(1)

leetcode 643.

⑤ maximum average subarray

| 1 | 22 | -5 | -6 | 50 | 3 |
0 1 2 3 4 5

K = 4 (length of subarray)

(M-1) Brute force

find all sub-array sum of length K

| 1 | 12 | -5 | -6 | 50 | 3 |
i j

① i=0; j=3; sum = 2
② i=1; j=4; sum = 51
③ i=2; j=5; sum = 42

loop ↑ when index gets out of bound

⇒ average = max sum (max) = 12.75

TC: O(N²)

(M-2) Sliding Window Method

| 1 | 12 | -5 | -6 | 50 | 3 |

① i=0, j=k-1 sum=2 1+12-5-6 = 2 (window 1)
② i=1, j= 12--6+50 = 5
② sum=2-1+ sum = sum-num[i];
sum=1+50=51 sum = sum+num[j];
i++; j++;
③ sum=51-12=39
sum = 39+3=42

sum = sum - num[i++];
sum = sum + num[j++];

leetcode 75.

⑥ Sort colors (sort an array of 0,1,2)

| 2 | 0 | 2 | 1 | 1 | 0 |
→ | 0 | 1 | 1 | 2 |

(M-1) sorting (using STL) TC: O(nlogn) SC: O(n) worst case

(M-2) counting method → not working inplace (ignore this method)

eg: override

zero → 2 → ①
ones → 2 → ②
two → 1 → ③

step1 (count)

Tc: O(n)
sc: O(1)

step2 → put zero then ones then two in array

(m-3) (in-place) 3-pointer approach

2 | 0 | 2 | 1 | 1 | 0

l h

(m should point at 1)

if (nums[m] == 0) swap(nums[l], nums[m]) l++, m++; }

else if (nums[m] == 1) { m++; }

else { swap(nums[m], nums[h]); h--; }

0 | 0 | 2 | 1 | 1 | 2

l m h

0 | 0 | 2 | 1 | 1 | 2

l m h

0 | 0 | 1 | 1 | 2 | 2 → 0 | 0 | 1 | 1 | 2 | 2 stop loop when (m > h)

l h m

(7) Move all -ve nos to left side of an array

eg: 1 | 2 | -3 | 4 | -5 | 6 → order maintenance not required

(1) sort : -3 | -5 | 1 | 2 | 6 → STL wala TC: O(nlogn)

(2) Dutch National Flag algorithm (2 pointer approach)

l h

maintain +ve nos. -ve nos.

1 | 2 | -3 | 4 | -5 | 6

l h

(0) (size-1)

while (l < h) {

if (a[l] < 0) { l++; }

else if (a[h] > 0) { h--; }

else { swap(a[l], a[h]) }

}

TC: O(n)

SC: O(1)

→ 1 | 2 | -3 | 4 | -5 | 6 (swap)

l h

→ -5 | 2 | -3 | 4 | 1 | 6 (swap)

l h

→ -5 | -3 | 2 | 4 | 1 | 6 (stop)

l h

Leetcode 287.

(8) Find duplicate number

eg: 1 | 3 | 4 | 2 | 2

0 1 2 3 4

N+1 = 5 ⇒ N = 4

1 | 3 | 4 | 2

nums[i] ∈ [1, n] ⇒ [1, 4]

m (1) sort → 1 | 2 | 2 | 3 | 4

0 1 2 3 4

TC: O(nlogn)

SC: O(n)

a[i] == a[i+1] → same: duplicate

m (2) Negative marking method

1 | 3 | 4 | 2 | 2 → N+1 ⇒ N = 4

0 1 2 3 4

nums[i] → index

(elements behave as index)

① iterate index

② mark visited

③ already visited ↓ return duplicate

→ 1 | 3 | 4 | 2 | 2

a 1 2 3 4

i

(1) nums[nums[i]]

mark -ve

1 | -3 | 4 | 2 | 2

i

(-3) abs → 3 ; i++

1 | -3 | 4 | -2 | 2

i

; i++

1 | -3 | 4 | -2 | -2 (visited)

1 | -3 | -4 | -2 | -2 → return ✓

i i

m (3) Positioning method

1 | 3 | 4 | 2

0 1 2 3 4

(element ko uske index pe le jao)

1 | 3 | 4 | 2 | 2 → 3 | 1 | 4 | 2 | 2 → 2 | 1 | 4 | 3 | 2 → 4 | 1 | 2 | 3 | 2

i ↓

3 | 1 | 2 | 3 | 4

extra element

TC: O(n)

SC: O(1)

(9) Missing elements from an array with duplicates

→ 2 | 3 | 5 | 3 | 4

1 2 3 4 5

i

N → size of array

a[i] ∈ [1, N]

(M-1) marking method

TC: O(n)

SC: O(1)

-1 | 3 | 5 | 3 | 4 ⇒ -1 | 3 | -5 | 3 | 4 ⇒ -1 | 3 | -5 | 3 | -4

i i i

⇒ -1 | 3 | -5 | -3 | -4

elements not visited → duplicate missing element

(M-2) sorting + swapping method

1 | 3 | 3 | 5 | 3 | 4

1 2 3 4 5

↳ sort 1 | 3 | 3 | 4 | 5

1 2 3 4 5

TC: O(n)

SC: O(1)

a[i] → index

1 | 3 | 5 | 3 | 4 ⇒ 1 | 5 | 3 | 3 | 4 ⇒ 1 | 4 | 3 | 3 | 5

1 2 3 4 5

i i i

⇒ 1 | 3 | 3 | 4 | 5

(10) Find first repeating element

1 | 5 | 3 | 4 | 3 | 5 | 6

0 1 2 3 4 5 6

(M-1) Iterate and check if element repeats

~~Recursion~~ TC: O(n²)

SC: O(1)

(M-2) optimised solution

hashing

1 → 1

5 → 2 ✓ (return)

3 → 2

4 → 1

6 → 1

① Traverse: O(n) store

② Iterate → check each element if it has occurrence later on.

TC: O(n)

SC: O(n)

(11) Common elements in 3 sorted arrays

A → 1 | 5 | 10 | 20 | 40 | 80

B → 6 | 7 | 20 | 80 | 100

C → 3 | 4 | 15 | 20 | 30 | 70 | 80 | 120

① if (A[i] == B[j] == C[k]) → common found

i++; j++; k++;

② else if (A[i] < B[j]) → i++

③ else if (B[j] < C[k]) → j++

④ else k++

TC: O(n1 + n2 + n3)

SC: O(min(n))

* if common elements are duplicate → use set

eg: A → 3 3 3

B → 3 3 3

C → 3 3 3

~~Q~~ * without using another data structure

A, B, C → remove duplicates.

(12) Wave Print a Matrix

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

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

col → 0 1 2 3

TB BT TB BT

Top to Bottom (even col)

Bottom to Top (odd col)

Leetcode 54.

(13) Sprint Print A matrix

0 1 2 3

0 1 2 3

1 4 5 6

2 7 8 9

1 2 3 6 9 8 7 4 5

update

• Printing: ① starting row ; ++

② ending col ; --

③ ending row ; --

④ starting col ; ++

DOMS | Page No. 76
Date / /

(14) Add two numbers represented by two arrays ** If factorial of a large number

- store no. in array

Q. add 2 no represented by array

A → 9 5 4 9 (i)
B → 2 1 4 (j)

carry = 0
int x = a[i] + b[j] + carry
int digit = x%10;
carry = x/10;
i--; j--

A 9 5 4 9
B 0 2 1 4 → j
output array 9 7 6 3

Q. factorial

7! → 1 x 2 x 3 x 4 x 5 x 6 x 7
↳ 5040

vector/array → array → 2 1

carry = 0
i = 2
int x = ans[j]*i + carry;
ans[j] = x%10;
carry = x/10;
if (carry) {
push carry }

① ans. size = 1, x = 1*2 + c
j=0 a[j] = x%10 → 2
carry = 2/10 = 0

② i = 3, int x = 2*3 + 0 = 6
a[j] = 6%10 = 6, c = 0

③ i = 4, x = 6*4 + 0 = 24
a[j] = x%10 = 4, c = 24/10 = 2

④ i = 5, x = 4*5 + 0 = 20
a[j] = x%10 = 0, c = 2

⑤ i = 6, ~~x = 7*6 + 2 = 44~~ x = 7*20

⑥ i = 7, 7*20 = 5040

→ reverse

SEARCHING AND SORTING

Searching and sorting class-1 (live class) 22 May 2024

linear search → T.C: O(n) (LS) → S.C: O(1)

binary search (BS) → T.C: O(

element

comparison

(agar monotonic fn ho toh binary search se ho skta log a wo sochna ek baar)

binary search

(monotonic function (element in ↑ or ↓ order))

i/p → 2 8 10 20 30 34 46 78 (n size)
0 1 2 3 4 5 6 7
start, left, mid, right, end

target/Key → 46

mid = (s+e)/2 = 7/2 = 3

step: divide in 2 parts [mid = (s+e)/2]

② if (arr[mid] == target) {
return true
}

30 34 46 78 (n/2 size)
s=4 5 6 7 e=7
mid = (4+7)/2 = 5
L mid R

③ → if mid value is not target then decide which part to choose (left or right) and go to step 1 again. (start = mid+1 / end = mid-1)

46 78 (n/4 size)
s 6 mid 7 e
mid = (6+7)/2 = 6

o) size: n → n/2 → n/4 → ... → 1

if (target > arr[mid]) {
s = mid+1
}

assuming ↑ order

if (target < arr[mid]) {
e = mid-1
}

invalid array ⇒ while(s<=e) → isme chalna hai baas (invalid array me nahi jao)
s>e

Leetcode 704.

* int mid = (s+e)/2 ⇒ out of range (INT_MAX & INT_MIN)

error free : int m... s + (e-s)/2

Time complexity (a) = O(log n)

* size: n → n/2^1 → n/2^2 → ..... n/2^(a-1) = 1 (a^th iteration)

$n = 2^{a-1}$

$n = 2^a$

apply log ⇒ $\log_2 n = \log_2(2^a)$

$\log n = a$

eg: decreasing order

100	90	80	70	60	55	50	40	30	20
0	1	2	3	4	5	6	7	8	9

s, mid, e target = 30

mid = 9/2 = 4

* if (target < arr[mid]) {
    s = mid+1
  }

* if (target > arr[mid]) {
    e = mid-1
  }

STL (Standard template library)

binary_search(starting address, ending address, target)

binary_search(arr, arr+n, target);

Q. find first occurrence (return index)

20	20	20	30	...	50	60	70
0	1	2	3		5	6	7

s, e target = 20

* Store and comp... (Bhayankar technique) → reference variable (ans wala index store krlia (it can be final ans or nhi bhi) index load bhi operation jaari rkho)

20	20	20
	1	2

mid, e

ans found → first occurrence / not first occurrence

e = m-1

⇒ store index : ansIndex = ~~-1~~ 0

left | 20 | right

0

s, mid, e

TC: O(log n)

SC: O(1)

Q. find last occurrence

10	20	20	20	20	20	20	20	20	30
0	1	2	3	4	5	6	7		

s, mid, e

ansIndex = ~~-1~~ ~~3~~ 8

s = mid+1

20	20	20	20	30
5	6	7	8	9

s, mid, e

20	30
8	9

s, mid, e

~~30~~

s, mid, e

Leetcode 34.

Q. find total occurrence

target → 20

10	20	20	20	20	20	30	40	50
0	1	2	3	4	5	6	7	8

O(log n) → first occ. → ind

O(log n) → last occ. → index → 5

occurrence = last occ − first occ + 1

= 5 − 1 + 1

= 5

Q. find Missing

Leetcode 268.

n elements

[0 → n]

0 → 4

sum of AP: $\frac{n}{2}(a+l)$ ← {0, 1, 3, 4} ← ans

ip: {0, 1, 3, 4} sum = 10

compare → tell missing

(Page 70)

difference = 10 − 8 = 2 (ans)

→ (mathematical meth → HW)

using binary search,

→ TC : O(nlogn)

eg:

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

sort(arr, arr+n)

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

* if (number > index) {
// left

* if (number == index) {
// right

discard this method

* if (arr[mid] + 1 != arr[mid + 1]) → arr[mid] + 1 → missing no.

(or) if (arr[mid + 1] − arr[mid] != 1)

↳ ans : arr[mid] + 1

if (arr[mid] − 1 != arr[mid − 1]) → arr[mid] − 1 → missing no. (or) if (arr[mid] − arr[mid − 1] != 1)

↳ return : arr[mid] − 1

(extreme pe wala dekha ye)

eg:

0	1	3	4	5	6
s 0	1	mid 2	3	4	5 e

observation : if diff → 0 ⇒ ans in right : s = mid + 1

else if diff → 1 ⇒ ans → store → mid

e = mid − 1

eg: extreme 0 wala case

1	2	3	4	5	6
s 0	1	mid 2	3	4	5 e

ans = ~~X~~ 0

1	2
s 0	mid 1 e

1
s 0 e

TC: O(nlogn)

SC: O(1)

eg: extreme n wala case

0	1	2	3	4	5
s 0	1	mid 2	3	4	5 e

ans = −1

(return n) alag se handle this case

3	4	5
s	mid	e

5
s mid e

~~another method (page 80)~~.

(live class)

Mega class - Arrays 26/May/2024

Hash maps
↳ tables

Key (unique, non-duplicating)	value
1	53
2	~~54~~ 57
3	55
4	56

(jisko dhundhna hai)

To implement:
- unordered_map <int, string> table_name;
  " <int, int> " ;

insertion: table_name[Key] = value;
eg: table[1] = 53;

iterator: unordered_map <int, int>:: iterator i; (iterator name)

~~For each loop~~: for loop:

```
for(it = desk_map.begin(); it != desk_map.end(); it++){
    int Key = it->first;
    int value = it->second;
}
```

for-each loop:

```
for(auto it : deskmap){
    int Key = it.first;
    int value = it.second;
}
```

very fast in table/map → TC: O(1) in average case

Find an entry:

```
if (tablename.find(2) != tablename.end()){
    //found
    int val = table_name[2];
}
else{
    //not found
}
```

(default int map, garbage se nhi)

deletion: table_name.erase(2); (key)

Questions

① 2's complement : 1's complement + 1

TC: O(n)
OSC: O(n)

Q. array given having binary number.

eg: | 0 | 1 | 0 | 1 | 1 | 0 |
0 1 2 3 4 5

0 0 0 0 1 1

step 1: (flip step)

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

step 2: (add 1)

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

i = n-1 → 0 (flipped array)
① sum = a[i] + c (carry)
② a[i] = sum % 2 (2%2 = 0)
③ c = sum/2 (2/2 = 1)

☆ ip ⇒ Binary array
o/p ⇒ complement array ⇒ size: Binary array size + 1 (if arr = 0,0,0,0 → o/p 1,0,0,0,0)

→ after all iterations, if (carry == 1) ⇒ a[0] = 1

leetcode 136
② Single number : hashing method

eg: | 4 | 1 | 2 | 1 | 2 |

no.	freq
4	1
1	✗2
2	✗2

leetcode 48 (imp. ques of 2D arrays)
↳ ③ ☆ rotate image

i/p

1	2	3
4	5	6
7	8	9

→ 90° →

o/p

7	4	1
8	5	2
9	6	3

↓ transpose

1	4	7
2	5	8
3	6	9

→ row wise reverse

(kaise transpose karte me for lagta hai)

```
vector<vector<int>> matrix () {
    int nrows = matrix.size()
    int ncols = matrix[0].size()
}
```

→ matrix 0

TC: $O(n^2)$

Leetcode 53. (bhot imp. ques) → [illegible]

(4) Maximum subarray

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

(1) Brute force: find sum of all sub-arrays. → TLE ⇒ TC: $O(n^2)$

- -2   -2 1 -3 4   -2 1 -3 4 -1 2 1   1
- -2 1   -2 1 -3 4 -1   -2 1 -3 4 -1 2 1 -5   1 -3
- -2 1 -3   -2 1 -3 4 -1 2   -2 1 -3 4 -1 2 1 -5 -4   1 -3 4 ....

1 → sum = 1
1, -3 ⇒ sum = sum + (-3) = -2
1, -3, 4 ⇒ sum = sum + 4 = 2 ....

```
for( int i=0; i<n; i++){        → single element bhi subarray hoga (isliye i+1 nhi)
    for(int j = i; j<n; j++){
    }
}
```

(2) divide and conquer →

(bhot imp famous) (3) ★★ famous algo

Kadane Algo   TC: $O(n)$

(1) iterate Kro
  ↳ sum Kro
  ↳ ans Ko update Kro
  ↳ sum < 0
    ↳ reset sum = 0

ans = ...

sum = 0

-2	1	-3	4	-1	2	1	-5	4

step (1) sum = ...
step (2) update ans
step (3) sum reset if sum < 0

TC: $O(n)$

SC:

eg: if all elements +ve ⇒ whole sum [illegible] ans.
if all elements -ve ⇒ sabse bada element jo hoga wo ans ayega.

week-3 assignment: ★ Q. missing element from an array with duplicates.

eg:

1	3	5	3	4
1	2	3	4	5

N = 5
$a[i] \in [1, N]$

step (1) a[i] index maanlo
step (2) waha jakr -ve Krdo
  ⇒ a[a[i]] = -ve (agar >0 hai toh)
  eg: a[1], a[3], a[5]

★ jo index +ve bach jayega wo ans. (jo -ve hai usko dubara -ve nhi Krna)

★ zero index pe random value daldo eg: v.insert(v.begin(), 737);

Q. (left in class)
- single no. 2
- single no. 3
- row with max-1
- majority element 1] ⇒ Moore's voting algorithm
- " " 2]

C++ STL containers (recording) 27/May/2024

↳ standard template library

What is STL?

- It implement many popular and commonly used algorithms and data structures.

Why we need STL?

- efficiency, productivity, safety, performance optimization, consistency, maintainability.

Common components in STL

- Containers: vector, list, queue, stack, set, map etc.
- Algorithms: sort(), binary_search(), reverse(), etc.
- Iterators
- Functors

Containers

- are classes or data structures that are designed to store and manage collection of objects. They provide a standardized way to store, retrieve and manipulate data in various ways.

VECTOR

- Dynamic array that can grow or shrink in size.
- Allows fast random access to elements.
- Insertion and removal of elements at the end is efficient.
- Suitable for most scenarios when elements need to be stored in a linear sequence.

Member Functions

- **begin()**: returns an iterator pointing to the first element in the vector.

  ↳ standard way to traverse over container

  *it → value at it ; it++

- **end()**: returns an iterator pointing to the position just after the last element in the vector.
- **size()**: returns the number of elements in the vector.
- **push_back()**: adds an element to the end of the vector.
- **pop_back()**: removes the last element from the vector.
- **empty()**: checks if the vector is empty (i.e whether its size is 0).
- **front()**: accesses the first element in the vector.
- **back()**: accesses the last element in the vector.
- **operator[]**: accesses the element at the specified index without bounds checking.

  eg: marks[0]

  (agar ki value access karni hai toh size defined kr usme diff no. with error aajayega)
- **at()**: accesses the element at the specified index with bounds checking. eg: marks.at(0)
- **capacity()**: returns the number of elements that the vector can hold before needing to allocate more space.
- **reserve(n)**: requests that the vector capacity be increased to atleast n elements, potentially reducing the no. of reallocations.
- **max_size()**: returns the maximum no. of elements that the vector can hold due to system or library limitations.
- **clear()**: removes all elements from the vector, which are destroyed, and leaves it with a size of 0.

- insert(iterator position, value): insert a new element before the specified pos"
- erase(iterator position) or erase(iterator first, iterator last): removes one or more elements from the vector starting at the specified position.
- swap(vector2nd): swaps the contents of the vector with those of another vector of same type, including their sizes and capacities.

For-each loop

```
for(int i:v) {
    cout<<i;
}
```

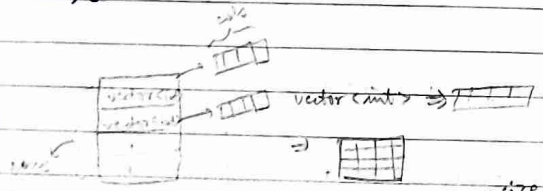
Iterator: vector<int>::iterator iterator_name = vectorname.begin();

eg: vector<int>::iterator it = distances.begin();

```
while (it != distances.end()) {
    cout<<*it; }
```

2D array using vector

vector<vector<int>>



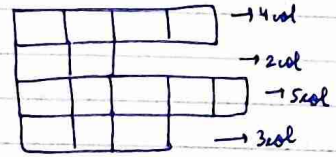
syntax: vector<vector<int>> arr (no. of rows, vector<int>(4,0)) → value

size

↓ har ek row me kya hoga?

```
int totalRows = arr.size();
int totalCols = arr[0].size();
```

Jagged array : 4 rows



⇒ vector<vector<int>> arr(4)

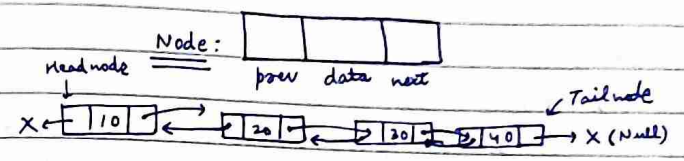
arr[0] = vector<int>(4)

arr[1] = vector<int>(2)

LIST

: linked-list → collection of nodes

- Doubly-linked list
- Allows fast insertions and removals anywhere in the list.
- No random access like vectors.

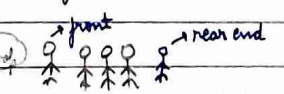


Member functions

- same as vector.
- begin(), end(), size(), empty(), front(), back(), push_back(), pop_back(), erase(), clear(), insert(), swap().
- pop_front(): removes the first element from list.
- push_front(e): adds an element to the beginning of list
- remove(n): removes all elements from the list that are equal to the specified value.

QUEUE: → cannot iterate

- Adaptor class that provides a First-In, First-Out (FIFO) data structure.
- Implemented using other containers (eg: deque, list) as the underlying storage.



addition: rear end

removal: front

Member functions (clear ki koto isme)

- empty(): checks if queue is empty.
- size(), swap().
- front(): accesses the first element in queue, which is next element to be removed
- back(): " " last element, which is most recently added element.
- push(n): push to end of queue, pop(): removes first element.

O(1)

STACK :

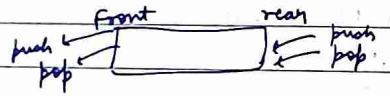
- Adapter class that provides a Last-In, First-Out (LIFO) data structure.
- Implemented using other containers (eg: vector, deque, list) as the underlying storage.

Member functions

- empty(), size(), swap()
- top(): accesses the top element of stack, which is most recently added.
- push(x): adds an element to top of stack. → O(1)
- pop(): removes the top element from stack.

DEQUE:

- Double-ended queue
- Similar to vectors but allows efficient insertion and removal at both ends.
- Suitable when elements need to be inserted or removed frequently from the front or back.

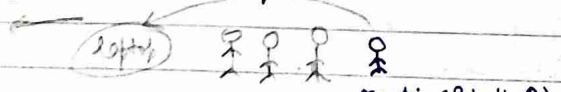


Member functions

- begin(), end(), size(), empty(), front(), back(), operator[], at(), push_back(), push_front(), pop_back(), insert(), erase(), clear(), swap().

PRIORITY QUEUE

- Adapter class that provides a priority queue (heap).
- Elements are stored in a way that allows the retrieval of the highest-priority element efficiency.



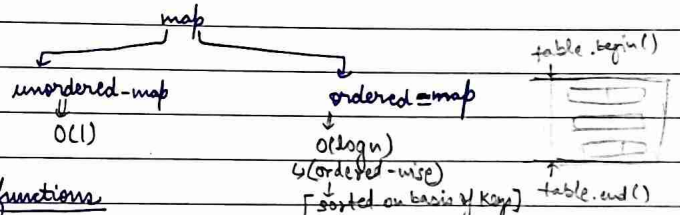
Member functions

- empty(), size(), swap().
- top(): accesses the element at the top of priority queue, which is largest or highest priority element, depending upon comparator used. O(1)
- push(): adds an element to priority queue and reorders it to maintain the heap property. → O(log n)
- pop(): removes the top element.

* max-heap → max no. is highest priority.
* min-heap → min no. is highest priority.
  - ↳ syntax: priority_queue<int, vector<int>, greater<int>> pq;
  - ↓ boolean challenge

MAP

- Associative container that stores key-value pairs.
- Allows efficient retrieval and modifications of value based on keys.
- Keys are unique.

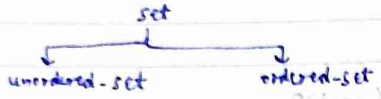


Member functions

- begin(), end(), empty(), size(), operator[], at(), insert(), erase(), clear()
- * find(k): returns an iterator to the element with the given key, or end() if key not found.
- * count(k): returns the number of elements with the specified key (1 or 0 since std::map does not allow duplicate keys)

SET

- Sorted collection of unique elements.
- Elements are sorted in sorted order and duplicates are automatically removed.
- Provides efficient insertion, deletion and search operations.



Member functions

:) begin(), end(), empty(), size(), insert(), erase(), clear(), find(), count()

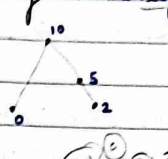
SEARCHING And SORTING (LIVE) class-2 29/May/24

Q) Find Binary search (contd..) peak → first → rotated array

** (katai imp)

Leetcode 852. Peak index in a Mountain array in O(logN) → BS

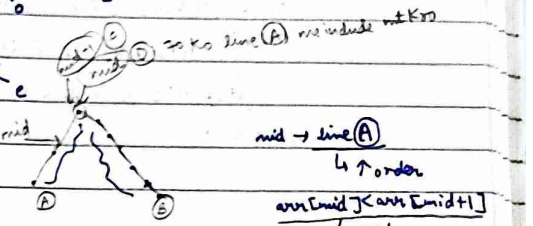
eg: 0 10 5 2



eg: 10 20 70 60 50 30 15 5

0 1 2 3 4 5 6 7

s mid e



s=0, e=7, mid=3

mid → line (A) → ↑ order

arr[mid] < arr[mid+1] → right

mid → line (B) → ↓ order

arr[mid] > arr[mid+1] → left

⇒ if (arr[mid] < arr[mid+1]) → line (A)

{ s = mid+1; }

else { mid line (B)

ans

⇒ e = mid

e = mid-1

e = mid; ✓

}

dbt class:

```
arr[mid] < arr[mid+1] {
    // line A
    right
    s=mid+1; }
else if arr[mid] < arr[mid-1] {
    //left    // line B
    e=mid-1; }
else { ans = mid; }
```

* method-2: store and compute ✓

HW → multiple peaks / mountains.

Leetcode 33. Pivot index. (Search in rotated sorted array)

↳ jaha order change ho jaye.

arr = 10, 20, 30, 40, 50, 60, 7 → sorted

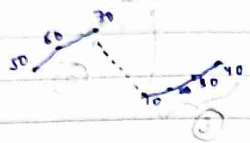
70, 10, 20, 30, 40, 50, 60

60, 70, 10, 20, 30, 40, 50

50, 60, 70, 10, 20, 30, 40 → sorted & rotated array

8

arr | 50 | 60 | 70 | 10 | 20 | 30 | 40 |
0 1 2 3 4 5 6



(A) → right.
(B) → left

* if(arr[mid] > arr[mid+1]) → pivot ✓ → return mid
if (arr[mid] < arr[mid-1]) → mid-1 ✓

→ par yahan tak done line (A) & (B) ↑ order me hai
if (arr[start] ≥ arr[mid]) → line (B) → left jao.
obs: line B element < line A element

eg: target = 20

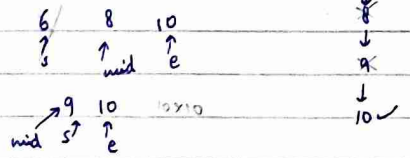
* Search space pattern
↳ answer space → B.S → store and compute

Q sqrt(x): 0 ——— 100
√100 = 10

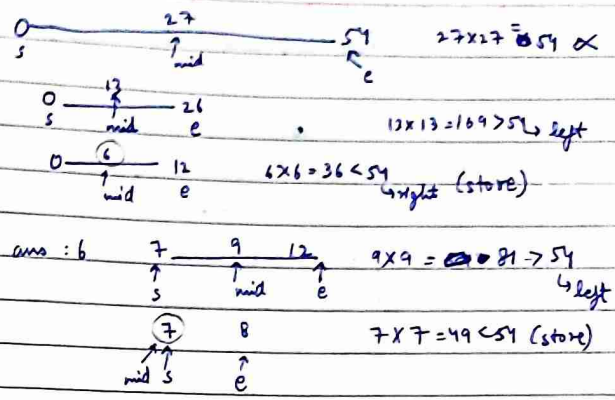
while() {
if (predicate()) → return ans
if → left
else → right
}

Leetcode 69

① √100 = 10

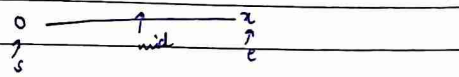


② √54



ans : 6

⇒ int x → √x



while (s<=e) {
if (mid * mid == x)
↳ return mid
if (mid * mid > x)
↳ left → e = mid-1
else ans = mid
↳ right → s = mid+1

* ans in float → code

2D array searching
* 2D → 1D → [c*i+j]
* 1D → 2D
i = mid / c
j = mid % c

Leetcode 74

O(logn)
⇒ O(log(row x col))

[redacted]

Searching and sorting class-3 (Live) 30 May 24

Q. ~~iff~~ Divisor and dividend → given
find quotient without using "/" or "%"

divisor × quotient + remainder = dividend

eg : dividend = 10, divisor = 2

-10 (s) ← 0 (mid) → +10 (e)

mid → 0

divisor × q + r = dividend

2 × 0 <= dividend

0 <= dividend

x + y = dividend

if y = 0 ⇒ x = dividend

if y = +x ⇒ x < dividend

store and compute ans → 0 → 10 → 5

2 × 8 <= 10

16 <= 10

→ false

eg : dividend = 17, divisor = 4

max-ans = +17

min-ans = -17

search space → ans → quotient ↓ mid

-17 (s) ← 0 (mid) → +17 (e)

divisor × quotient <= dividend

4 × 0 <= 17

0 <= 17

ans
↓
0 (OK: liya true hai toh
~~aage~~ piche wale ke liye bhi true hoga)
↓
4

1 (s) ← 9 (mid) → 17 (e)

4 × 9 <= 17

36 <= 17

→ False → left

4 (s), 6 (mid), 8 (e)

4 × 4 <= 17

16 <= 17

→ T

5 (s), 6 (mid), 8 (e)

4 × 6 <= 17

24 <= 17 → F

5 (s, e, mid)

4 × 5 <= 17

20 <= 17 → F

4 ans ✓

Q. * Search in a nearly sorted array.

arr[i] → arr[i-1], arr[i], arr[i+1]

sorted →

3	10	20	40	50	70	80

target = 40

nearly sorted :

10	3	40	20	50	80	70
0	1	2	3	4	5	6

s = 0, mid = 3, e = 6

left, ans, right

mid = 3

(40 is already checked as its mid-1)
(50 " " " " mid+1)

left ⇒ e = mid - 2

right ⇒ s = mid + 2

target = 70

Leetcode 540

Q. Odd occurring element.

(A) all element occurs even no. of times except one.

(B) element repeats itself in pairs.

(C) no pair repeats itself, no number can occur more than 2 times. a single stretch.

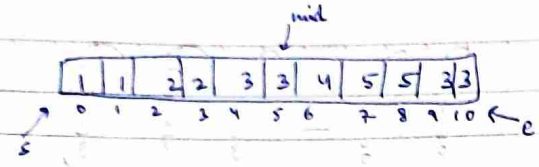
(D) find element that occur odd times.

eg :

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

Algo → XOR ⇒ TC: O(n)

→ map
1 → 2
2 → 2
3 → 4
5 → 2
} even ⇒ TC: O(n)
4 → 1 → odd

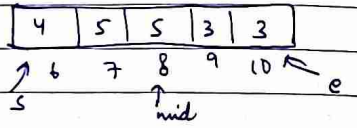


3 index → pair → ✓ → I^st element index : 2 (even)
↳ ans right me hai

3 element → I^st element index = 9 (odd)
↳ ans left me hai.

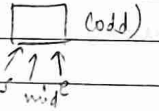
mid = 5;
↳ pairIndex = 4 (even)
↳ move right

TC: O(logn)
SC: O(1)

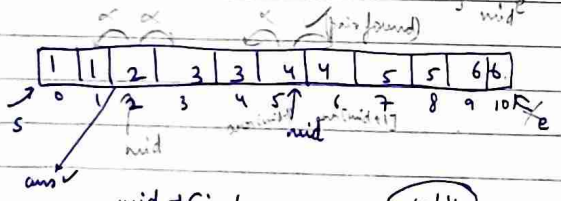


mid = 8
↳ pairIndex = 7 (odd)
↳ move left

mid = 6
↳ pair nhi hai
↳ ans milgya



eg:



mid → 5 index
↳ element = 4

4 / 4
5 6

↳ odd → go left

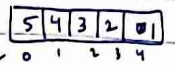
Case I → single element
Case II → no duplicate
Case III → left side duplicate
Case 4 → right side duplicate

SORTING ALGORITHMS

[recorded]

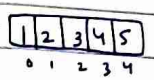
Bubble Sort → Order me sort kro

eg: N = 5



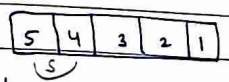
Order

↳ sort:

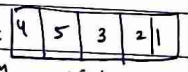


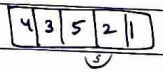
1) Swap the adjacents if needed till we get all the array sorted.

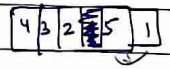
I^st iteration

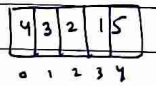


① → I^st largest element at its position









(4 comparisons)

② 2^nd largest at its position

4 3 2 1 5
3 4 2 1 5
3 2 4 1 5
3 2 1 4 5
(3 comparisons)

③ 3^rd largest at its position

3 2 1 4 5
2 3 1 4 5
2 1 3 4 5
(2 comparisons)

④ 4^th largest at its position

2 1 3 4 5
1 2 3 4 5 (1 comparison)

N = 5

outer loop: 0 ---- (n-1) ⇒ [0, 4) → n-i-1 (inner loop)

comparisons: (n-1), (n-2) --- 1

Sum = 1 + 2 + 3 --- (n-2) + (n-1)

$S_n = \frac{n(n-1)}{2}$

$O(n) = O\left(\frac{n^2}{2} - \frac{n}{2}\right) = O(n^2)$ → TC (not good)

SC: O(1)

Selection Sort

What if I select the minimum element & put it at left-most position.

eg: 44 33 55 22 11

① 11 33 55 22 44
② 11 22 55 33 44
③ 11 22 33 55 44
④ 11 22 33 44 55

→ for ith iteration, pick smallest element from i to (n-1) index & swap it with i^{th} element.

$i \Rightarrow 0 \rightarrow < n-1$

$i \in [0, n-1)$

step 1. [0, 4)
min index = 4

step 2. swap(v[i], v[min index])
swap(v[0], v[4])

repeat the steps....

N = 5

		comparison
I.	i = 0	4
II	i = 1	3
III.	i = 2	2
IV.	i = 3	1
	↓	↓
	outer loop (n-1) times	inner loop j = i+1 to n

TC: $O(n^2)$
SC: $O(1)$

Insertion Sort

eg: 4 5 X 3 2 1 (Left me dekho 4 se bada koun hai)

3 4 5 3 2 1

2 3 4 5 2 1

2 3 4 5 X 1

1 2 3 4 5 ✓

eg: 44 33 55 22 11

33 44 55 22 11

33 44 55 22 11

22 33 44 55 11

11 22 33 44 55 sorted ✓

→ i = 1, key = v[i] ⇒ key = 4 (i = 1)

j = i - 1

while (j >= 0 && v[j] > key) {
v[j+1] = v[j];
j--;
}
v[j+1] = key;

5 4 3 2 1
0 1 2 3 4

TC: $O(n^2)$
SC: $O(1)$

i = 2, key = 3 ⇒ 3 4 5 2 1

i = 3, key = 2 ⇒ 2 3 4 5 1

i = 4, key = 1 ⇒ 1 2 3 4 5

Custom Comparator

sort(v.begin(), v.end()); : ↑ order

arr ⇒ sort(a, a+n);

↓order ⇒ sort(v.begin(), v.end(), mycomp); (mycomp: apna fn lamayenge bool type ka)

vector of vector: [[1,44], [0,55], [0,22], [0,11], [2,33], ----]

[a,b]

sort by 1st index of vector ⇒ use comparator.

(by default sort 0th index ke basis pe hoga)

C++ STL Containers [cont'd] recorded

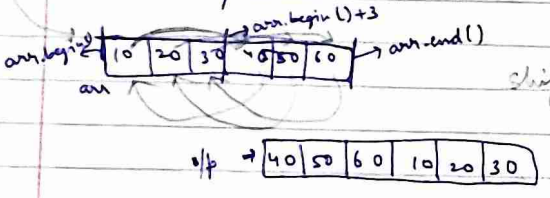
C++ STL Algorithms

Algorithms

The C++ STL includes a wide range of algorithms that operate on various container types and provide essential functionality for data manipulation. These algorithms are defined in the <algorithm> header and are part of the STL's core functionality.

Iterators and iterating algorithms

- std::for_each : applies a function to each element in a range.
- std::find : searches for a specific element in a range.
- std::find_if : " " the first " that satisfies a given predicate.
- std::count : counts the occurrences of a value in a range.
- std::count_if : " " element that satisfy a given predicate.
- std::sort : sorts the element in ascending order.
- std::reverse : reverses the order of elements in a range.
- std::rotate : rotates element in a range.
- std::unique : removes duplicate elements from a sorted range.
- std::partition : divides elements in a range into two groups based on a predicate.

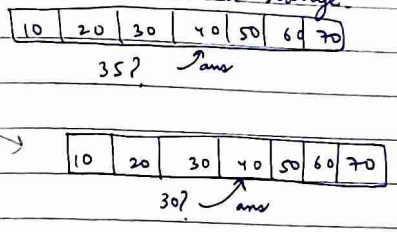


Numeric Algorithms:

- std::accumulate : computes the sum of elements in a range.
- std::inner_product : " " inner product of two ranges.
- std::partial_sum : " " partial sums of a range.
- std::iota : fills a range with incrementing values.

Searching and finding Algorithms

- std::binary_search : checks if a value exists in a sorted range.
- std::lower_bound : finds the first element greater or equal to a value in a sorted range.
- std::upper_bound : finds the first element greater than a value in sorted range.
- std::equal_range : finds a range of elements equal to a value in a sorted range.



Min and max algorithm

- min : returns smaller of 2 values
- max : " larger " "
- min_element : find smallest element in a range.
- max_element : " largest " " " "

Heap Algorithms

- make_heap : converts a range into a max-heap → O(n)
- push_heap : inserts an element into a max-heap → O(log n)
- pop_heap : removes the largest element from a max-heap → O(1)
- sort_heap : sorts a range that represents a max-heap. → O(n)

Set Algorithms

- set_union : computes union of two sorted ranges
- set_intersection : " intersection
- set_difference : " difference btw two sorted ranges.
- set_symmetric_difference : computes symmetric difference of 2 sorted ranges

C++STL: Iterators

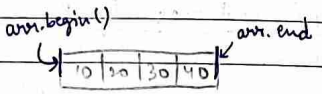
- An iterator is a ~~pointer~~ pointer-like object representing an element's position in a container. It is used to iterate over elements in a container.

- Suppose we have a vector named nums of size 4. Then, begin() and end() are member functions() that return iterators pointing to the beginning and end of the vector.
  - → nums.begin() points to the first element in the vector, i.e 0th index.
  - → nums.begin()+i points to the element at i^{th} index.
  - → nums.end() points to one element past the final element in vector.

Creating and Traversing Iterators

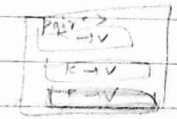
initialise Kadia arr.begin() se

```
vector<int>::iterator it;  (OR)  it = arr.begin();
while (it != arr.end()) {
    cout << *it;
    it++;
}
```



Iterator operations

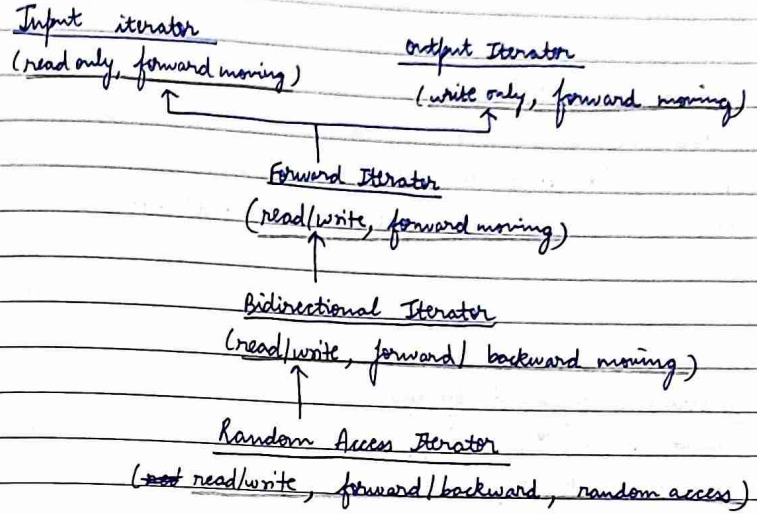
- ★ *itr : returns the element at the current position.
- ★ ++itr : moves iterator to next position. (itr++ or itr = itr+1)
- ★ --itr : " " " = previous".
- ★ itr + i : moves " by i position.
- ★ itr1 == itr2 : (comparison) returns true if the pos^n pointed by iterators are same.
- ★ itr1 != itr2 : return true if pos^n pointed by iterators are not same.
- ★ itr = itr1 : assigns the position pointed by itr1 to itr pointer.
- ★ itr→m or (*itr).m : returns the member value 'm' of the object pointed by the iterator.



★ itr→ pair<int,int> (OR) (*itr).first ⇒ itr→first

p.first, p.second (*itr).second ⇒ itr→second

Iterator Types (not needed as such)



Input iterator

→ Typically used for algorithms that need to read data from container, such as std::find or std::for_each.

eg: istream_iterator<int> input_itr(cin)

Output iterator

→ less used as compared to other iterator types.

eg: ostream_iterator

Forward Iterator

→ Have combined capabilities of i/p iterator and o/p iterator. For eg: lists support forward iterator.

eg: forward_list<int> nums = {1,2,3,4};

forward_list<int>::iterator its = nums.begin();

↳ singly linked list

Bidirectional Iterator

→ Supported by containers like lists, double-ended queues (deques).

eg: list<int>::iterator it = nums.begin();

Random Access Iterator

- Vectors, arrays & deques provide random access iterators.

eg: vector<int> :: iterator itr_first = vec.begin();

eg: " " itr_last = vec.end()-1;

random : arr.begin()+i;

Operations supported by iterator

input : ++, *, →, ==, !=

output : ++, *, =

forward : ++, *, →, ==, !=

Bidirectional : ++, --, *, →, ==, !=

Random access : ++, --, *, →, [], +, -, <, <=, >, >=, ==, !=

Why use Iterators?

- saves memory.
- uniform approach: code is more consistent and easier to manage.
- simpler code.
- working with algorithms

C++ STL: Functors (Function objects)

- Functors are objects that behave like functions and can be called with the same syntax as regular functions. They are implemented as classes or structs that overload the operator()
- In context of C++ STL, functors are often used as custom comparators or custom operations when working with various algorithms and containers.

Usage in algorithms

- Functors are commonly used as arguments to STL algorithms, providing custom behaviour for sorting, searching and other operations.
- For example: when sorting a container of custom objects, you can provide a functor that defines the sorting criteria based on specific object attributes.

Function call operator():

- Functors define the behaviour of their function call operator()
- When a ~~functor~~ functor object is called like a function, the operator() is executed, allowing you to encapsulate custom behaviour within functor.

eg: bool operator()(Student a, Student b) {

}

Custom Comparators

- Functors are frequently used to provide custom comparison logic when sorting or searching in STL containers.
- For example: std::sort algorithm, you can pass a custom functor to specify sorting order.

Advantages of Functors

- Provide flexible way to customize behaviour compared to fn pointers.
- type-safe, allowing the compiler to catch type-related errors at compile-time.
- It can carry additional data or state, making them versatile.

Mega Class (live) 2 June 2024

Q. Sqrt with precision

$\sqrt{63} = 7.937253319$ (Brute force approach)

① 0 — 31 — 63 → B.S search space

31 is my answer? Easily find: $31 \times 31 = 63$?

$961 \neq 63$

When can I apply search space pattern?

① jab apka ans kisi search-space/range me aata ho → Determine

② mid ko easily prove kr paa ki wo ans hai ya nhi.

② 0 — 15 — 30 $15 \times 15 \leq 63$

③ 0 — 7 — 14 $7 \times 7 \leq 63$ (maybe it's my answer) ans = 7

④ 8 — 11 — 14 $11 \times 11 \leq 63$

⑤ 8 — 9 — 10 $9 \times 9 \leq 63$

TC: O(logn) + O(precision)

⑥ 8 — 8 mid = 8 $8 \times 8 \leq 63$ $64 \leq 63$

⑦ 8 — 7 start > end ⇒ stop

end, start

ans = 7 → 7.9

precision

7 — 8

store and compute

7.0 → $7 \times 7 < 63$ ✓

7.1 → $7.1 \times 7.1 < 63$ ✓

7.2 → $7.2 \times 7.2 = 51.84 \leq 63$ ✓

7.3 → $7.3 \times 7.3 = 53.29 \leq 63$ ✓

7.4 → $54.76 \leq 63$ ✓

7.5 → $56.25 \leq 63$ ✓

7.6 → $57.76 \leq 63$ ✓

7.7 → $59.29 \leq 63$ ✓

7.8 → $60.84 \leq 63$ ✓

7.9 → $62.41 \leq 63$ ✓

7.9 → 1 precision (0.1 step)

7.90 → 2 precision (0.01 step) step = step/10

7.91 → $62.56 \leq 63$ ✓

7.92 → $62.72 \leq 63$ ✓

7.93 → $62.88 \leq 63$ ✓ ans = 7.93

7.94 → $63.04 \leq 63$

7.99

3 precision (0.001 step)

7.930 → $62.88 \leq 63$ ✓

7.931 → $62.90 \leq 63$ ✓

7.932 → $62.91 \leq 63$ ✓

ans = 7.937

7.936

7.9367 → $62.99 \leq 63$ ✓

7.938

7.938

Method - 2 start ≤ end ⇒ 0 ≤ end - start

while (end - start ≥ 0)

↳ Old B.S non-precision cond" ⇒ s = mid + 1, e = mid - 1, mid = (s+e)/2 } integers (starting se non-precision ka the)

⇓

new method

while (end - start ≥ 0.1) {

}

⇒ now consider double values.

(dry run)

① 0 — 31.5 — 63.0

$31.5 \times 31.5 = 992.25 \leq 63.0$

end = mid; end = mid - 1 ⇒ 0 → 30.5

start = mid;

TC: Same complexity as Method 1

Q. Precision Division (Div)

Method 1: BS (non-precision integer) + linear search (precision)

Method 2: fully Binary Search

Leetcode 29. Divide 2 integers. (without using *, /, %)

INT_MIN ⟶ INT_MAX

-2^{31} $2^{31}-1$

-2147483648 2147483647

$\frac{-INT\_MIN}{-1}$ ⇒ 2147483648 ⇒ INT_MAX se bada hai

4 is easy to do logic se handle karlo

(Bitwise multiplication : badme karenge)

(Imp)

Leetcode 875. Koko eating bananas

a) N piles N=4 ← bananas

b) h hours (given) : guard (eg: 8 hours) gone

[3|6|7|11] (0 1 2 3)

1 → 4 bananas eaten in 4 hours

2 → 7 bananas eaten total in 7 hours

c) each hour she can eat only K bananas.

eg: K=2, h=8

[3|6|7|11]

1 4 5 9 : first 4 hours (8 eaten)

0 2 3 7 : couldn't eat all of bananas.

eg: [3|6|7|11] K=11, h=8

hour: h1 h2 h3 h4

0 0 0 0

eg: K=7 [3|6|7|11]

0 0 0 4 → h5

0

(constraint)

a) piles.length <= h

search space: 1 ——— 11

K=3

how many hours koko will take to finish all bananas with K bananas eaten speed per hour?

[3|6|7|11]

3/3 6/3 7/3 11/3

hour: 1 2 2.3 3.6

hour: 1+2+2.3+3.6

eg: 7 → h=4 → h=1 → h=0

⇒ 1+2+ceil(2.3)+ceil(3.6)

⇒ 1+2+3+4 = 10 hr

[2.5 → ceil → 3.0
2.5 → floor → 2.0]

10 ≤ 8

① 1 — mid — 6 — 11 K=6

total hour = 3/6 + 6/6 + 7/6 + 11/6 = 1+1+2+2 = 6 hours

6 ≤ 8 ✓ ans = 6 → ans = 8

To minimise K, move left

② 1 — 3 — 5

K=3 hours = 10 hours 10 ≤ 8

③ 4 — mid — 5

K=4; hr = 3/4 + 6/4 + 7/4 + 11/4 = 1+2+2+3 = 8 hours

8 ≤ 8 ✓

④ 4 — 3 (s > e) ⇒ stop

s e

TC: O(log(max(piles)) * O(n))

TC: O(n * log(max(piles)))

Leetcode 1482. Minimum no. of days to Make m Bouquets.

[1|10|3|10|2] → Bloomday

m = 3

K = 2

adjacent

| 1 | 10 | 3 | 10 | 2 |

N=5

day 1: x 9 2 9 1 : flower left
day 2: x 8 1 8 x
day 3: x 7 x 7 x
day 4: x 6 x 6 x
day 5: x 5 x 5 x
⋮
day 9: x 1 x 1 x
day 10: x x x x x

① ② bouquet ⇒ return -1

$N \geq mK$ ⇒ ~~5≥6~~

| 7 | 7 | 7 | 7 | 12 | 7 | 7 |

N=7

m = 2 → bouquet, K=3 → flower

mK = 6

7 ≥ 6 ✓

| 7 | 7 | 7 | 7 | 11 | 12 | 7 | 7 |

N=8 m=2, K=3

8 ≥ 6 ✓

day 1:
⋮
day 7: x x x x 4 5 x x
⋮
day 11: x x x x x 1 x x
day 12: x x x x x x x x
① ②

monotonic ↑search space

1 ——— 30

eg: K=3, m=4

day D: x x x x x _ _ x x _ _ x x x

c++ → c: 1 2 3 0 1 0 0 1 2 0 0 1 2 3

~~Counter=0~~ c=0;

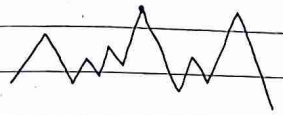
```
if (c==K){   → 1 bouquet
    c=0;
    m--;
}
if (non-bloomed) { c=0; }
```

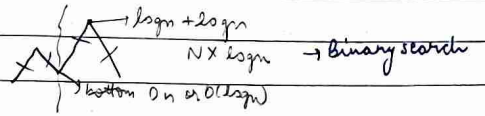

① Search space: days
② monotonic
③ given days: predicate fx
↓
m: bouquet

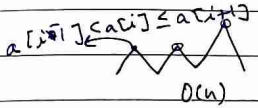
eg: | 2 | 3 | 4 | 7 | 7 | 7 | 8 | 2 | 1 | 10 | 11 |

① → end = max → element array
start = 1 → atleast 1 flower bloomed
→ start = min (array) ✓

Leetcode 162: Find peak element







→ linear search wala method best nhega isme

O(n)

Leetcode (Medium) Week-4 Assignments

① K-diff Pairs in an array

1	4	1	5
0		3	4

K = 2

M-1: Brute force consider each and every pair & take difference

```
for (i=0; i<n; i++) {
    for (j=i+1; j<n; j++) {
        if (abs(A[i] - A[j]) == K) { count++; }
    }
```

TC: O(n²)

M-2: 2 pointer approach

1	1	3	4	5
0	1	2	3	4
i	j			

K = 2

① diff = a[j] - a[i]

```
if (diff == K) {
    i++; j++;
}
else if (diff > K) { i++; }
else { j++; }
```

dry run: ① 1|1|3|4|5
i j

② diff = 0 ⇒ j++ (diff < K)

② 1|1|3|4|5
i j

diff = 3-1 = 2 ans ✓ ; i++ ; j++

③ diff = 4-1 = 3 > K ; i++

④ diff = 4-3 = 1 < K ; j++

⑤ diff = 5-3 = 2 ; j++; i++ ans ✓

set: to store unique pair.

~~-3 : binary search~~

~~1 | 1 | 3 | 4 | 5~~ K = 2

~~1 → K-1~~

~~a[i]~~

~~a[i] = K~~

~~a[j] = a[i]~~

~~TC: O(n logn)~~

TC: O(nlogn)

brute force
① i=0, a[i]=1, K+a[i]=3
search (3) → not present
Pair → (1,3)
② i=1, a[i]=3, K+a[i]=5
search (5) → ✓
Pair → (3,5)
③ i=3, a[i]=4, K+a[i]=6

leetcode 658. (Medium)

② Find K- closest elements

0	1	2	3	4
1	2	3	4	5

eg: K=4, x=3

diff : 2 1 0 1 2

close

→ 3 → 3, 2, 4, 1, ~~5~~

eg: 12 | 16 | 22 | 30 | 35 | 39 | 42 | 45 | 48 | 50 | 53 | 55 | 56 (0 1 2 3 4 5 6 7 8 9 10 11 12) K=4, x=35

diff : 23 19 ~~15~~ 13 5 0 4 7 10 13 15 18 20 21

|x-a[i]| :

M-① sol^n → sort wrt difference

35 | 39 | 30 | 42 | 45 | 22 | 48 ---

ans

Two Pointer

M-② 12 | 16 | 22 | 30 | 35 | 39 | 42 | 45 | 48 | 50 | 53 | 55 | 56

L ↓ → ↘ H

→ jo difference bada hai use age move karo

while(h-l+1>k){ if (x-a[l] > a[h]-x) { l++; } else { h--; } }

TC: O(n-K)

SC: O(1)

M-③ ① find smallest element in array which is >=x.

lower bound : closest element to x

② H → index of closest element (35)

L → H-1

③ [L,H] window form, size → K

↳ expand window to K.

if (x - arr[l] > arr[h] - x) {h++}
else {l--}

Hit 0 index → h++ not
Hit last index → l--

eg: 3|5|8|10 x=2, x=15
10 > 15 → binary search ✗ ans = -1 → ans = end ✓

③ Exponential Search and unbounded Binary Search

- exponential search aka doubling search, galloping ", struzik ".
- application in large array

→ sorted

3|4|5|6|11|13|14|15|56|70 x = 13 (find)
0 1 2 3 4 5 6 7 8 9

[log n → 4 steps]

→ step ✓

```
if (a[0] == x) return 0;
int i = 1;
while (i < n && a[i] <= x) {
    i = i * 2;
}
return BS(arr, i/2, min(i, n-1), x)
```

TC: ① O(log m) → while loop
② BS on subarray
↳ $2^4 - 2^3 = 8$
⇒ $2^{\log m} - 2^{\log m - 1}$
$= 2^{\log m}(1 - 2^{-1})$
$= \frac{2^{\log m}}{2}$
TC: O(log $(2^{\log m - 1})$)

Application: ① search in ∞ arrays
↳ unbounded array
② better than B.S (x is near in beginning)

- unbounded binary search
→ find element in an ∞ array (sorted)
(M-1) → use exp search ↳ optimised ✓
(M-2) → brute force

```
i = 0; while(1) { if (a[i] > x) break;
                  if (a[i] == x) ans = i;
                  i++
}
```


exp search x = 6

1|2|3|4|5|6|7|8|9|10|11 ---
0 1 2 3 4 5 6 7 8 9 10
i j

```
while (a[j] < x) { i = j;
    j = j * 2; }
```

algo: i = 0; j = 1;

```
while (a[j] < x) { i = j;
    j = j * 2;
}
start = i; end = j
```

B.S (with value x)

gfg: hard

④ Book Allocation Problem

| 12 | 34 | 67 | 90 | → array of books
0 1 2 3
→ A_i → no. of page in that book

M → no. of students M = 2

① 12 | 34 67 90 → max: 191
② 12 34 | 67 90 → max: 157 → min: 113
③ 12 34 67 | 90 → max: 113

- if no. of students > no. of books → return -1;

① Brute force
② better sol^n ⇒ define search space (12+34+67+90 = 203)

0 (min) ——— 203 (max)

① → i = 1 → sol^n?
i = 2 → "
!
i = 113 → " ✓

int mid = 0 + 203 / 2 = 101

Try to allocate each student atmost 101 pages (mid)
I → 12+34+67 = 113
II → 67 + 90 = 157
101 is not a possible sol^n

- if (not possible) {
start = mid + 1;
}
102 ——— 203
mid = 152
I → 12+34+67+90 = 203
II → 90 ✓
- if (possible sol^n) {
// minimise pages
end = mid - 1;
}

⑤ Imp Painters Partition Problem

5 | 10 | 30 | 20 | 15 → A[i] → length of i^{th} board ; N=5 ; K → painters

1 unit time to paint 1 unit board

eg : N=4, K=2

10 | 20 | 30 | 40

					Total time together
①	10	20 30	40	→	90
②	10 20	30	40	→	70
③	10 20	30	40	→	60

min → 60

(similar to book allocation problem)

⑥ Aggressive Cows

stalls: | 1 | 2 | 4 | 8 | 9 | N=5

(position) 0 1 2 3 4 K(cows)=3

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

1	2	4	8	9	K=3
C_1	C_2	C_3			→ 1 (min distance b/w cows) → maximise min distance
C_1		C_2		C_3	→ 3
C_1			C_2	C_3	→ 1
	C_1	C_2		C_3	→ 2
C_1		C_2	C_3		→ 3
		C_1	C_2	C_3	→ 1

→ max = 3

Suppose ans is 1

ans 1 → 1 | 2 | 4 | 8 | 9 ; C_1 C_2 C_3 (gaps 1, 2) ✓

ans 2 (max) → C_1 C_2 C_3 (gaps 3, 4) ✓

ans 3 → C_1 C_2 C_3 (gaps 3, 4) ✓

ans 4 → not possible to place 3 cows with min distance 4.

∴ Binary search ✓ ⇒ search space : min: 0 ; max: maxPos − minPos = 9 − 1 = 8

① 0 —— 4 —— 8

mid = 4 → at least 4 distance par cows place krna. → not possible

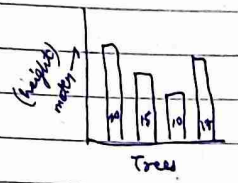
if (not possible) { end = mid − 1; }

② 0 — 3 ✓ if (possible sol^n) { start = mid + 1; }

③ 2 — 3 ✓ ④ 3 — 3 ✓ ⑤ start > end → stop

⑦ EKO SPOJ

20	15	10	17



sawblade height = ?

M = 7m

no. of trees = 4

let sawblade height = 15m ⇒ 10m wala tree bach jayega ⇒ 5 + 0 + 0 + 2 = 7m

(m-1) Brute force

sawblade height = 1m ✓ but minimise the sol^n

sawblade height = 2m ✓ " " " " ....

start : 0 (max wood) ; end : max heighted tree (min wood)

(m-2) binary search

① start = 0 end = 20

mid = 20+0/2 = 10 m ≥ 7m ? ✓

if (Possible sol^n) {

ans = mid;

start = mid + 1;

}

② 11 —— 20

mid = 15 ✓

③ 16 —— 20

mid = 18 ✗

if (not sol^n) {

end = mid − 1; }

⑧ PRATA SPOJ

P → prata <= 1000 ; C_i → 1st prata 2nd 3rd ---- 1000

L cooks <= 50 ; Rmins 1R 2R 3R 1000R

Rank → [1, 8]

(M-1) Brute force

nP = 10 , 4 cooks, [1|2|3|4] → R

```
for(i=0min; i<max; i++){
   —  }
```

C_4 → all 10 prata $1\times4 + 2\times4 + 3\times4 + 4\times4 \text{ ---- } 10\times4$ mins

$\Rightarrow 4[1+2+3 \text{ --- } 10] = 4\left[\frac{10\times(10+1)}{2}\right] = 220m$

(M-2) Binary search

① 0 (min) —— 110 (mid) —— 220 (max) → Time taken by cook with highest rank when all prata are given to him

can cooks cook order of 10 prata in mid = 110 min??

$C_1 = 55$ min $\leq$ 110 ✓

```
if(Possible soln) {
   ans = mid; end = mid-1;
}
```

② 0 —— 54 —— 109

cooks can work parallely

$C_1 = 1+2+ \text{ --- } 9+10 = 55$ min $= 45$ min ≤ 54 ✓

$C_2 = 1\times2 = 2$ min

③ 0 —— 26 —— 53

$C_1 = 1+2+ \text{ --- } 6 = 21$ min ✓ ≤ 26

$C_2 = 2+4+6+8 = 20$ min

④ 0 —— 12 —— 25 ✓

$C_1 = 1+2+3+4 = 10$ min

$C_2 = 2+4+6 = 12$ min

$C_3 = 3+6 = 9$ min

$C_4 = 4 = 4$ min

⑤ 0 —— 5 —— 11 X

$C_1 = 1+2 = 3$ min

$C_2 = 2$ min

$C_3 = 3$ min

$C_4 = 4$ min

```
if (not Possible){
   start = mid+1;
}
```

⑥ 6 —— 8 —— 11 X

$C_1 \rightarrow 1+2+3 = 6$ min $\rightarrow 3p$

$C_2 \rightarrow 2+4 \rightarrow 2p$

$C_3 \rightarrow 3 \rightarrow 1p$

$C_4 \rightarrow 4 \rightarrow 1p$

⑦ 9 —— 10 —— 11 X

ans = 12