S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1		Intro to C++		
2		Variable & op, ip in C++		
3		Operators in C++		
4		If else & Switch statements | Conditionals in C++	RECAP-1	
5		Loops in C++ | For, while, do-while		
6		Pattern printing | Problems on loops		
7		Problems on loops ~ Part 2	RECAP-2	
8		Number system | Decimal & Binary		
9		Functions in C++		
10		Functions - 2 | Pass by value & pass by reference	RECAP-3	
11		Array - 1 | Array creation types & operations		
12		Vectors in C++ | Looping & problem solving	Arrays 2	
13		Target Sum | Prob. in Arrays - 1		
14		Two pointers | Problems in Arrays - 2		
15		Prefix sum problems | Problems in Arrays - 3	RECAP-4	
16		2D - Arrays - 1 | What, Why, & How?		
17		Imp: prob in 2D Arrays | 2D vectors, Pascal's Δ		
18		Advanced Problem Solving in 2D Arrays		
19		Advanced Problem Solving in 2D Arrays - 2		
20		Advanced Problem Solving in 2D Arrays - 3		
21		Time Space Complexity & Big O Notation | How to avoid TLE		
22		Calculating Time Complexity | Problem Solving		
23		Pointers - 1 | Introduction, Data Storage & Access using Pointers		
24		Pointers - 2 | Call by reference, Arithmatic & Array as a pointer		
25		Pointers - 3 | Types of Pointers, Null, Wild, Dangling, Void		
26		Recursion | Recursion concept and problems		

INTRODUCTION TO C++

- Machine code (0, 1) is any low-level programming language.
- Assembly lang. is a type of low-level programming lang that is intended to communicate directly with a computer's hardware
- Compiler / Interpreter converts high level lang to low level lang.
  (ex: C, C++, Java, Python)
- High Level lang. are of 3 types → Procedural
  - Object-oriented
  - Functional

VARIABLES AND OUTPUT/INPUT IN C++

- C++ output
- C++ variables
- C++ identifiers
- C++ Data types
- C++ user input

C++ output:

```
#include <iostream>      // (header file) (has basic funcs like i/p, o/p, etc.) → directive processed by pre-processor (code which tells compiler to run the code below #...)
using namespace std;

int main() {             // (func.)
    // prints hello world   (comments)
    cout << "Hello world";   (to give o/p)
    return 0;            (insertion operator) (tells code to stop executing)
}
```

```
returntype function_name() {
}
```

func. def. : it's lines of code written to do a specific task.

* If we want our o/p to enter into next line → use → endl (or) \n

```
cout<<"Hello"<<endl<<"World";
```

o/p: Hello
World

C++ variables:

Variables are containers where we store values (or) data when our code is executed.

Syntax: Type variable_name [= value];

ex: int a = 4;

(variable type)

We can also change values of variables;

ex: int a = 3; & int c;
a = 5; ✓ c = 4;

↳ Value changed from '3' to '5'.

Rules for naming variables in C++:

- A C++ keyword can't be used as a variable name.
- When creating variables, it's preffered to give them meaningful names
- Variable names should not begin with a number.
- Whitespace is not permitted in variable names.
- All lowercase letters should be used when creating one-word variable names.
- It's preffered to use variables with more than one word with all lowercase letters for the first word & capitalization of the first letter of each subsequent word (ex: int TeaContainer)

C++ identifiers:

are names of class, variables, methods

Imp Points:

1) All identifiers should begin with a letter (A to Z, a to z), currency character ($) or an underscore (_).
2) After the first characters, identifiers can have any combination of characters.
3) A keyword cannot be used as an identifier
4) The identifiers are case sensitive.
5) Whitespace is not permitted

C++ datatypes:

It's imp. to mention as it tells our OS how much memory space to allocate for that given value as diff. datatype occupy diff. memory

* Primary datatypes / Built-in datatypes: integer, character, boolean, void, floating point, wide character, double floating point
* Derived datatypes (der. from primary): function, array, pointer, reference
* User-defined datatypes: class, structure, union, enum, typedef

⇒ Integer Data Type: → takes 4 bytes memory in space → int a;

1 byte = 8 bits (smallest unit of storage) ∨(0, 1)

1 byte = 2^8 ; 4 bytes = $2^{8+8+8+8}$ = 2^{32} bits

(means, we can store 2^{32} integer whose size is 4 bytes)

⇒ Character Data Type: → char name = 'a' → size = 1 byte
→ can store special symbols, letters, digits.

⇒ Boolean Data Type: → has only 2 values → true / false → size = 1 byte

ex: bool flag = true / false;

⇒ Floating point Data Type: → used to store decimals upto 7 decimal places.

ex: float rate = 5.34; → size = 4 bytes

⇒ Double Floating point Data Type: → size = 8 bytes

→ Precision = upto 15 decimal places

⇒ Void Data Type: → can't define anywhere as no memory is allocated to it

→ null/nothing

⇒ Wide Character Data Type: → size = 2 bytes

⇒ String Data Type: → not a built-in datatype

→ stores sequence of characters → ex: string name = "Urvi";
→ Dynamic memory is allocated

* To get input from user ~ int apples;
cin >> apples;

* Terminal (in VScode) is basically a command line where you give commands & interact with code.

OPERATORS IN C++

- C++ operators
- C++ operators precedence & their associativity

C++ operators:

1) C++ Arithmetic operations:

(+): Addition, (-): Subtraction, (*): Multiplication, (/): Division

(%): Modulus → (gives us remainder) ex: 5%2 = 1

(++): Increment → a = 3, a++ → a = 4

(--): Decrement → a = 3, a-- → a = 2

2) C++ Relational operators:

(==): is equals to, (!=): not equals to, (>): greater than, (<): less than, (>=): greater than (or) equals to, (<=): lesser than (or) equal to

```
ex: int num1 = 6;
    int num2 = 7;
    cout << num1 == num2 << endl;
```
o/p: 0 (means false ; 1 → true)

3) C++ Logical operators:

(&&): Logical AND, (||): Logical OR, (!): Logical NOT

(exp1 && exp2 → true) (exp1: true, exp2: true)

(!true = false)

```
ex: bool exp1 = true;
    bool exp2 = false;
    cout << (exp1 && exp2) << endl;
    cout << (!exp1);
```
o/p: 0 0

4) C++ Assignment operators:

(=): used to assign values ex: int p = 4; int q = p;

(+=): ex: a+=1 → a = a+1

(-=): ex: a-=1 → a = a-1

(/=): ex: a/=1 → a = a/1

(%=): ex: a%=1 → a = a%1

```
ex: num1+=3;
    cout << num1;
```
o/p → 6+3 = 9

5) C++ Bitwise Operators:

(Little info on binary representation)

ex: (8)$_{10}$ → (1000)$_2$ (decimal form → binary)

How to convert binary to decimal: 1) Start from Left to right

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

(2^3: as it's binary; 3: no of bits on its right)

How to convert decimal to binary: (remainder) ex: 8 → keep dividing by 2

8	0
4	0
2	0
1	1

go bottom to top → 8 → 1000

ex:

11	1
5	1
2	0
1	

11 → binary → 1011

(~): Bitwise Complement → ~0 = 1, ~1 = 0 ex: ~(0101) = 1010

(<<): Left Shift → (position of bit gets left shifted) ex: 0101 << 1 (left shift by 1) = 1010 (no. gets multiplied by 2 if its left shifted by 1)

if: 0100<<2 → 10000 (gets left shifted by 2) & (it gets multiplied by 2^2.)

(>>): Right shift → (no. gets right shifted) → ex: 0100>>1 → 0010

a<<b → $a\times2^b$ & a>>b → $a/2^b$

(|): Bitwise OR → 1|1 = 1, 2|0 = 1, 0|0 = 0, 0|1 = 1

(&): Bitwise AND

```
  1010
& 0110
------
  0010
```

(^): Bitwise exclusive OR → 0^0 } 0 & 0^1 } 1, 1^1 } 0 & 1^0 } 1

Ex:
```
int num1 = 5;                          // 0101
cout << (num1 << 1) << endl;           // 10
cout << (num1 >> 1) << endl;           // 2
int num2 = 8;                          // 1000
cout << (num1 & num2) << endl;         // 0
cout << (num1 | num2) << endl;         // 1101 = 13
```
o/p → 10, 2, 0, 13

b) C++ Misc Operators:

* **Sizeof operator**: int(a) = 4;
  cout << sizeof(a); → o/p → 4 bytes

* **Ternary operator**: { condition ? exp1 : exp2 } → ex:
  (if false → exp2; if true → exp1)
  ```
  int a;
  4==5 ? a=4 : a=5;
  cout << a;
  ```
  ↳ o/p → 5

* **Comma Operator**:
  ```
  int y = 4, 3, 6;
  cout << y;
  ```
  o/p → 6
  (takes last value always)

* **Dot & Arrow Operator**: . ↳ used in classes , → ↳ used in pointers.

* **Casting Operator**: we can cast from 1 datatype to other datatype
  ex:
  ```
  float b = 3.44;
  cout << int(b);
  ```
  o/p → 3

* **& Address Operator**: it prints the address in memory.
  (ampersand) ex: &a;

* *** Pointer Operator**: Its used in pointing out a variable address.
  ex: *p = a;
  ↳ ex:
  ```
  int a = 3;
  int* b = &a;
  cout << b << endl;
  cout << &a << endl;
  cout << *b << endl;
  ```
  o/p → 0x61ff0d, 0x61ff0d, 3

c) C++ Unary Operators

(+): Unary plus, (-): Unary minus (ex: -4), (++): Increment operator (ex: a++, ++a), (--): Decrement operator (ex: a--, --a)

(!): Logical compliment operator → (ex: !true = false → ex: !100 = 0)
↳ (anything which is not null)

* Post-increment operator & Pre-increment operator

a++ → (returns 'a' & then increments)
++a → (increment & then returns 'a')

ex:
```
int c = 6, b = 5;
cout << (c++) << endl;
cout << (--b) << endl;
```
o/p → 6, 4
(no need to keep brackets)

ex:
```
int c = 6;
cout << (c++) << endl;
int b = c+5;
cout << (b) << endl;
```
o/p → 6, 12

C++ Operators Precedence & Associativity:

Category	Operators	Associativity / Priority
Postfix	++ --	(L) Left to Right (R)
Unary	+ - ! ~ ++ --	R to L
Multiplicative	* / %	L to R
Additive	+ -	L to R
Shift	<< >>	L to R
Relational	< <= > >=	L to R
Equality	== !=	L to R
Bitwise AND	&	L to R
Bitwise XOR	^	L to R
Bitwise OR	|	L to R
Logical AND	&&	L to R
Logical OR	||	L to R
Conditional	? :	R to L
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	R to L

a) int p, q, r, s;
S = p-++r-++q;
'S' is equivalent to? A) - < ++
∴ S = p-(1+r)-(1+q)

b) Result of following code fragment?
A) == → (Left to right)
```
bool p = false;
bool q = false;
bool r = true;
cout << (p==q==r);
```
(p==q) → F==F → true → r: true → value = 1

IF ELSE & SWITCH STATEMENT | CONDITIONALS IN C++

if statement

```
if (condition){
    //code;
}
```

if-else statement

```
if (condition){
    //code;
}
else{
    //code;
}
```

if-else if

```
if (cond-1){
    //code;
}else if (cond-2){
    //code;
}else{
    //code;
}
```

(we can use multiple else if)

nested if-else

```
if (cond-1){
    if (cond-2){
        //code;
    }else{
        //code;
    }
}else{
    //code;
}
```

Conditional operators

* Logical-and operator : && : (exp1 && exp2)
* Logical-or operator : || : (exp1 || exp2)
* Ternary operator : short version of if-else.

Common misconceptions

* "&" (Bitwise) (works on bits) v/s "&&" (Logical AND) (gives cumulative result on multiple conds)

ex: `if (false & 5/0 == 1)`

Bitwise operator checks both value & here it will give run-time error as '5/0' not possible

`if (false && 5/0 == 1)`

Logical AND checks 1st cond., if that is false, then it won't check further → o/p = 0

* "|" (Bitwise OR) v/s "||" (Bitwise AND)

ex: `if (true | 5/0 == 1)`

[Bitwise → give errors]

`if (true || 5/0 == 1)`

Logical OR → checks 1st cond., if that is only true, it's not going to check the next condn. → o/p → 1

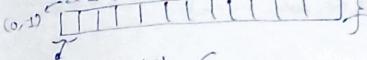
Switch Case (Syntax)

```
switch (expression){          // (value which should be compared with cases.)
    case x:
        //code;
        break;                // (helps to bring out of switch case once conditions are met.)
    case y:
        //code;
        break;
    default:                  // (runs when no case is matching with expression)
        //code;
}
```

(if expression is character → case 'a': ; case 'b': ;)

RECAP - 1

* C++ Datatypes : int → integer → (4 bytes space) → 4×8 = 32 bits



most significant bit used to tell value is +ve/-ve

Range → (for unsigned/non-negative int.) → 0 to $2^{32}-1$

→ (for signed/negative int.) → -2^{31} to $2^{31}-1$

* ex:

```
int a = 4;
cout << a++;      // 4
```

```
int a = 4;
cout << ++a;      // 5
```

```
#include <bits/stdc++.h>
int main(){
    float a = 3.14;
    cout << floor(a) << endl;   // 3
    cout << ceil(a) << endl;    // 4
```

floor(a) → largest integer ≤ a

ceil(a) → smallest integer ≥ a

* ex:

```
{ int a = 10;
  int b = a++;          // b = 10, a = 11
  cout << b << " ";     // 10
  b = ++a;              // a = 12, b = 12
  cout << a;            // 12
  return 0;
}
```

* use: #include <cmath> to use: sqrt func

LOOPS IN C++ | FOR, WHILE, DO-WHILE

While loop

```
initial statem.
while (condition){          // (keyword) (= true)
    //code → (consists updation)
}
```

ex. print sum of first 'n' natural no.'s, where n = i/p.

```
int n;
int sum = 0;
int i = 1;
while (i<=n){               // Loop variable
    sum += i;               // condition
    i++;                    // loop runs until condition becomes false
}                           // updating loop variable
cout << sum << endl;
```

For loop

```
for (init-statm; condn; updation){
    //code
}
```

(Starting point) (loop variable update)

ex: Sum of natural no.'s →

```
int n, sum=0;
cin>>n;
for(int i=1; i<=n; i++){
    sum += i;
}
cout << sum << endl;
```

Here int 'i' is defined only inside loop. Once loop is over, int 'i' gets destroyed.
Note: We can't → cout<<i; as 'i' is not declared outside loop.

Omitting Parts of for loop:

1) Omitting init. state^n

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

✓

2) Omitting condition

```
for(int i=1; ; i++){
    //code (should include my stopping cond^n.)
}
```

3) Omitting updation

```
for(int i=1; i<5; ){
    cout<<i;
    i++;   ——— ②
}
```

③ → If we don't write updation inside loop it will become ∞ loop

→ We can also use for loop like this(↓):

```
for(int i=0, j=4; i<4, j>0; i++, j--){
    //code
}
```

Break Keyword:

ex:
```
while(true){
    int n;
    cin>>n;
    if(n==-1){
        break;
    }
}
```

- means, if n=(-1), then it tells to come out of the loop.
- it comes out of nearest loop

ex:
```
for( ){ ——— (1)
    for( ){ ——— (2)
        //code
        //cond.
        break;
    }
}
```

→ {comes out of (2) loop but still stays in (1) loop.

Do-while loop:

```
do{
    //code
}while(cond^n);
```

→ this prog. will run atleast once, even if cond^n. is false as the cond^n. state^n is checked in last

Continue Keyword: → works on nearest loop.

↳ used to skip through an iteration.

ex: Suppose we dont want to print '3' from first 5 integers.

```
for(int i=1; i<6; i++){
    if(i==3){continue;}
    cout<<i<<endl;
}
```

o/p: 1, 2, 4, 5

PATTERN PRINTING / PROBLEMS ON LOOPS

Q) (Print) 5

A)
```
{ int r, c;
for(int r=1; r<4; r++){
    for(int c=1; c<7; c++){
        cout<<"*";
    }
    cout<<endl;
}
return 0;
}
```

* Always check :: 1) No. of rows
2) No of columns
3) what to print

Q) Print

A)
```
for(int i=1; i<=4; i++){
    for(int j=1; j<=6; j++){
        if(i==1||j==1||i==4||j==6){
            cout<<"*";
        }
        else{
            cout<<" ";
        }
    }cout<<endl;
}
```

Triangular Pattern

Q) Print

A) rows → 1 to 4
columns → 1 to 4
print → "*"

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

Q)

A)
```
for(int m=5; m>1; m--){
    for(int n=1; n<m; n++){
        cout<<"*";
    }cout<<endl;
}
```

Q)

A) rows → 4
columns → "*" → 2(row no.)-1
"_" → spaces → 1 to (4 - row no.) : 3 2 1 0

```
int n;
cin>>n;
for(int i=1; i<=n; i++){
    for(int j=1; j<=(n-i); j++){
        cout<<" ";
    }
    for(int j=1; j<=(2*i-1); j++){
        cout<<"*";
    }
    cout<<endl;
}
```

Numerical rectangular pattern:

Q)
1 2 3 4 5 6 7
2 3 4 5 6 7 1
3 4 5 6 7 1 2
4 5 6 7 1 2 3
5 6 7 1 2 3 4
6 7 1 2 3 4 5
7 1 2 3 4 5 6

(no. starting from row no.)
n = 7

A) rows → 1 to n
col. → 1 to n
print → i to n
+
1 to (i-1)

```
int n;
cin>>n;
for(int i=1; i<=n; i++){
    for(int j=i; j<=n; j++){
        cout<<j;
    }
    for(int j=1; j<=(i-1); j++){
        cout<<j;
    }
    cout<<endl;
}
```

Q) 1 2 1 2 1 2
2 1 2 1 2 1
1 2 1 2 1 2
2 1 2 1 2 1

A) n → 4, m → 6

e.g. 1st row

1 2 1 2 1 2

i=1, j=1 → j+1 = 2 → even = 1
i=1, j=2 → j+1 = 3 → odd = 2

```
int n,m;
cin>>n>>m;
for(int i=1; i<=n; i++){
    for(int j=1; j<=m; j++){
        if((i+j)%2==0){
            cout<<"1";
        }else{
            cout<<"2";
        }
    }cout<<endl;
}
```

Numerical triangular pattern

Q)
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1

A) rows → 1 to n, columns

u : 1 to (n-i) (row-no.)

num : (1 to i) (i-1) to 1

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

Q) 1 2 3 4 5 6
1 6
1 6
1 2 3 4 5 6

```
int r, c;
cin>>r>>c;
for(int i=1; i<=r; i++){
    for(int j=1; j<=c; j++){
        if(i==1||j==1||i==r||j==c){
            cout<<j;
        }else{
            cout<<" ";
        }
    }cout<<endl;
}
```

PROBLEMS ON LOOPS ~PART 2

Q) Count the no. of digits for a given no. 'n'?

A)
```
int n;
cin>>n;
int digits=0;          // eg. 610
while(n>0){
    digits++;          // 1, 2, 3
    n=n/10;            // 61, 6, 0
}
return 0;
```

Q) Find the sum of digits in a given no. 'n'?

A)
```
int n;
cin>>n;
int sum=0;
while(n>0){
    int lastdigit = n%10;
    sum+=lastdigit;
    n/=10;
}
cout<<sum<<endl;
```

Q) Reverse the digits of a no.?

A①
```
int n;
cin>>n;
int dig=0;
while(n>0){
    dig=n%10;
    cout<<dig<<"";
    dig=0;
    n=n/10;
}
```

②
```
int n;
cin>>n;
int reverse=0;
while(n>0){
    int lastdigit = n%10;
    reverse = reverse*10+lastdigit;
    n/=10;
}
```

Q) Find sum of following series: S = 1 - 2 + 3 - 4 ... n (input)

A)
```
int n;
cin>>n;
int result=0;
for(int i=1; i<=n; i++){
    if(i%2==0){
        result-=i;
    }
    else{
        result+=i;
    }
}
```

Q) Print the first 'n' factorial no.'s? {n! = n×(n-1)×(n-2)×...×1} {n! = n×(n-1)!}

A)
```
int n;
cin>>n;
int fact=1;
for(int i=0; i<n; i++){
    fact*=i;
    cout<<fact<<endl;
}
```

Q) Given 2 no.'s a & b. Find a raise to the power b?

A) (1) #include <cmath>

```
int main(){
    int a, b;
    cin>>a>>b;
    cout<<pow(a,b);
    return 0;
}
```

(2)

```
int a, b, res = 1;
cin>>a>>b;
for(int i = 0; i<b; i++){
    res = res * a;
}
cout<<res;
```

RECAP - 2

Q) WAP to print alphabet diamond pattern? ⇒

```
    A
   ABC
  ABCDE
 ABCDEFG
ABCDEFGHI
 ABCDEFG
  ABCDE
   ABC
    A
```

A)

```
1 - - - - A
2 - - - A B C
3 - - A B C D E
4 - A B C D E F G
5 A B C D E F G H I
6 - A B C D E F G
7 - - A B C D E
8 - - - A B C
9 - - - - A
```

n=5 → we have to print 9 lines

ex: char c = 'a';
then ASCII value of 'a' is stored in c++

ASCII: 65 - A, 66 - B, 67 - C ... ; 97 - a, 98 - b, 99 - c ...

no. of char. → 2*(line) - 1

(Δ) (line) → spaces: 1 → 4, 2 → 3, 3 → 2, 4 → 1, 5 → 0

▽

line	no. of char
6	1
7	3
8	5
9	7

Think of formula* ↳ 2*(2n-line)-1

▽

line	space
6	1
7	2
8	3
9	4

↳ (line-n)

Logic: line → i, space → (n-i)

⇒

```
int n = 5;  // given i/p
// loop to print upper Δ
for(int line = 1; line <= n; line++){
    int spaces = (n-line);
    for(int k=0; k<spaces; k++){
        cout<<" ";
    }
    int chars = 2*line - 1;
    for(int j=0; j<chars; j++){
        char ch = (char)('A'+j);
        cout<<ch;
    }
    cout<<endl;
}
// loop to print lower Δ
for(int line = n+1; line<=2*n-1; line++){
    int spaces = (line-n);
    for(int k=0; k<spaces; k++){
        cout<<" ";
    }
    int chars = 2*(2*n-line)-1;
    for(int j=0; j<chars; j++){
        cout<<(char)('A'+j);
    }
    cout<<endl;
}
```

Q) WAP to print * pattern :

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

A) Input → always odd

```
    0 1 2 3 4
0   - - * - -
1   - * * * -
2   * * * * *
3   - * * * -
4   - - * - -
```

n/2

```
int n = 5;
for(int line = 0; line<n; line++){
    for(int i=0; i<n; i++){
        if(i == n/2){cout<<"*";}
        else if(line == n/2){cout<<"*";}
        else {cout<<" ";}
    }
    cout<<endl;
}
```

Q) WAP to print rectangle of * ?

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

A)

```
int r, c;
cin>>r>>c;
for(int line = 1; line <= r; line++){
    for(int j = 1; j<=c; j++){
        cout<<"*";
    }
    cout<<"\n";
}
```

NUMBER SYSTEM | DECIMAL & BINARY

Decimal no. sys. ↳ base value = 10, 0-9 digits & powers of 10

ex: $6743 = 6\times10^3 + 7\times10^2 + 4\times10^1 + 3\times10^0$

Binary no. sys. ↳ 0, 1 digits, powers of 2 ; base value = 2

ex: $1010_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

Hexadecimal: ↳ base value = 16, 0-15 digits & powers of 16

ex: $4A5C_{16} = 4\times16^3 + 10\times16^2 + 5\times16^1 + 12\times16^0$

Conversion of binary to decimal:

ex: $1011_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 8+0+2+1 = 11_{10}$ (in decimal)

In code:

```
int n;
cin>>n;
int ans = 0;
int power = 1;
while(n>0){
    int lastdig = n%10;
    ans += lastdigit * power;
    power *= 2;
    n/=10;
}
cout<<ans;
```

Conversion of Decimal to binary:

* Parity digit → n/2 → (remainder)
  - 1 → parity odd of n
  - 0 → parity even of n

ex: 14₁₀

1	7
1	3
1	1
	0

$1110_2 = 1\times2^3 + 1\times2^2 + 1\times2 + 0\times2^0$

Incode:

```
int n;
cin >> n;
int ans = 0;
int pow = 1;
while(n>0){
    int paritydigit = n%2;
    ans += paritydigit * pow;
    pow *= 10;
    n /= 2;
}
cout << ans;
```

- 1 nibble contains 4 no. of bits

FUNCTIONS IN C++

- Functions are logical set of codes which performs a specific task (like a shortcut)
- I/p we pass in func. → we call it "parameter".

ex: int main() → is a func. called 'main'.

- We use func.'s as it ⇒ ↑'s the reusability.
  - ⇒ DRY principle (Don't Repeat Yourself)
  - ⇒ becomes easy to read.

Types of functions:

1) User-defined func.'s
2) Standard Library func.'s (built-in func.'s already)

How to declare func.:

```
return_type function_name (parameter1, parameter2 ...) {
    statements;
}
```

* func. name → should depict functionality
* return type → ex: for addition → int add( ) { ... return sum; } → returns 'sum' & code ends

if → (tells func. returns nthg.)

```
void print(int num){
    cout << num;
    return;
}
```

→ (tells that implementation of func. is over)

* parameters → we have mention their datatype too.
  ex: int add(int num1, int num2) { }

reason: so that we can use func. name again
ex: int add(int num1, int num2)
int add(int num1, int num2, int num3)
float add(float num1, float num2)
Now func. will be called as per input type

* func. body:

```
____ {
    // code
    return ___;
}
```

Call a func.:

To call a func. in C++, we have to write the func.'s name followed by 2 parentheses ( ) & a semicolon ;

* func. should be called via main. → (as main func. → starting point of C++ prog.)
* func. must always be written at top of main.

```
void welcome(){
    cout << "Hi";
}
int main(){
    welcome();
    return 0;
}
```

→ o/p ⇒ Hi

→ if we try to store value of welcome() in some variable inside main. (run) then it will give us compilation error as 'void' → returns nothing & variable is storing 'nothing' → so error. (void value can't be stored)

ex:

```
int add(int num1, int num2){
    int sum = num1 + num2;
    return sum;
}
float add(float num1, float num2){
    float sum = num1 + num2;
    return sum;
}
int main(){
    int a=5, b=4;
    float c=3.4, d=4.4;
    cout << add(a,b) << endl << add(c,d);
}
```

→ o/p → 9
7.8

→ if we write another `int add(int num3, int num4)` it will give us an error, as both are doing same func.

Func. prototype: if we want to define a func. after the func. call, we need to use the func. prototype.

```
ex: int add (int , int)
    int main(){
        int a=3, b=7;
        cout<< add(a,b);
    }
    int add (int a, int b){
        int sum= a+b;
        return sum;
    }
```

o/p → 10

→ gives main func. an assurity that 'add' func. is present in code. So it's just there to make its presence/prototype

Standard Library Functions:

#include <cmath> : must be included

ex: pow(a,b) → a^b & sqrt(a) → $\sqrt{a}$

* C++ double is a versatile datatype that can represent any numerical value in compiler, including decimal values

```
ex: int main(){
    cout << pow(2,3) <<endl<< sqrt(24)<<endl;
    int ans = sqrt(24);
    cout<<ans <<endl;
    double doubleans = sqrt(24);
    cout<< doubleans <<endl;
    return 0;
}
```

o/p: 8
4.89898
4
4.89898

→ (n+g mentioned so o/p came as float)

FUNCTIONS - 2 | PASS BY VALUE & PASS BY REFERENCE

Scope of variables in C++ : is part of program from where variable is accessible. (visibility of a variable)

(data containers storing values)

→ If we declare a variable inside a func., it's accessible everywhere →

```
ex: int main(){
    int apples=5;
    cout<<apples;
}
```

→ But if we declare a variable in a func. & call it from other func., it will show me an error.

```
ex: int main(){
    int apples=5;
    print();
    cout<<a;
}
void print(){
    int a=50;
    cout<<apples;
}
```

(error) → (error) ↳ 'a' is not declared in func.

(error)

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

→ 'i' variable is limited to part inside for loop. So it's scope is limited inside for loop

```
ex: int p=5;
    float p=5.7;
    cout<<p;
```

→ error, as it has same scope with 2 variables with same name.

```
ex: int p=5;
    while(){
        float p=5.7
        cout<<p;
    }
    cout<<p;
```

o/p: 5.7, 5 → No error

Local Variables

These are variables that are declared inside the function (or) block.

Global Variables:

These are the variables that exist throughout the whole program & can be used in any part of program.

```
ex: int apple=5;
    int main(){
        cout<<apple;
    }
```

```
void print(){
    cout<<apple;
}
```

o/p → 5 5 as scope of 'apple' is global.

* What if there exists a local variable with same name as that of the global variable inside a function?

A) Local will be given more precedance than global → Local > global

```
ex: int p=5;
    void print(){
        cout<<p;
    }
```

```
int main(){
    int p=7;
    cout<<p;
    print();
    cout<<::p;
}
```

o/p → 7 → (local > global)
5 → (global)
5 → (scope resolution)

Scope resolution operator → (::) → we use this operator when we want to use global variable inside a func.

Formal parameters & Actual parameters

```
void print( int param1, int param2){
    //code
}
int main(){
    int a=1, b=2;
    print(a,b);
}
```

// param1, param2 → are called formal parameters

// a, b → these are called = actual parameters

Call by Value & Call by Reference

(parameter values are copied to another variables)

(actual parameter values are passed. ↳ & → ampersand operator)

Suppose: `int p=5`, `int &q=p;`

both point towards same memory location ↳ p → 5 ← q

Ex:

```
void val(int z){
    z = 100;
}
int main(){
    int a = 5;
    val(a);   //actual
    cout<<a;
}
```

//formal

('z' scope is limited)

a=5 (pass by value) → z=100, a=5 (copied)

coming to main func. → 'z' get removed → a=5

(as main doesn't identify 'z')

∴ ans → 5

o/p : 5

* & → allows us to have alias names for the same memory location.

Call by reference ex:

```
int p = 5;
int &q = p;
q++;
cout<<p;
cout<<&p;
cout<<&q;
```

6 (as p==q)

0x1f

0x1f } (address)

Ex:

```
void val(int &z){
    z = 100;
}
int main(){
    int a = 5;
    val(a);
    cout<<a;
}
```

o/p = 100

as (z==a)

(alias name & same memory location)

Default Parameter Value :

It's value in the func. declaration automatically assigned by the compiler if the calling func. does not pass any value to that argument

Ex:

```
int add(int a, int b=1, int c=2){
    return a+b+c;
}
int main(){
    cout<<add(2)<<endl;
    cout<<add(2,2)<<endl;
    cout<<add(2,2,3);
    return 0;
}
```

// 5 → (only 1 value then default value used)

6

7 ↳ (all 3 values present, so default case is discarded)

Op of below code ?

```
void makeTwice(int p){
    p = p*2;
}
int main(){
    int p = 24;
    makeTwice(p);
    cout<<p;
}
```

//formal → 48

actual → 24

(p) is different in both as both are diff. parameters.

```
int add(int a, int b=0, int c){
}
```

→ error

{default value order: right to left(←)}

if 'b' is declared as default parameter, then all parameters next to it should also be declared as default else it would give me an error.

RECAP - 3

return → keyword

Ex:

```
void fun(string name){
    cout<<" Have fun "<<name<<" ?";
}
int main(){
    fun("Sanket");
    return 0;
}
```

here we can't return anything. If we write return 7 → error (as void func.)

if cout<<fun("Sanket"); ↳ error → as void func. doesn't return anything

we can't store fun("Sanket") in a variable also

so we directly call the func.

- If we want to check whether 'n' is a prime no. or not, then we check all the no.'s from 2 to $\sqrt{n}$, if they're divisible with 'n' or not.
  - if divisible → not prime
  - if not divisible → prime

- We can also call func. using pointers.

ex:
```
void fun(int* x){
    cout<<x;
}
int main(){
    fun(9);
    return 0;
}
```
→ error

{pointer concept}
as fun stores address & we are passing integer, so error.
Hence, we have to '&' operator

ex:
```
void fun (int* x) {
    cout<<x;
    cout<<*x;
}
int main(){
    int x=9;
    fun(&x);
    return 0;
}
```
→ op: 0x1664
9

(pointer → stores address)

known as → dereference operator.

'address of' operator

→ So if we change anything in address, our actual value will also get changed

```
void fun(int* x){
    cout<<x<<endl;
    cout<<*x<<endl;
    *x=29;
    cout<<*x<<endl;
}
int main(){
    int x=9;
    fun(&x);
    cout<<x;
    return 0;
}
```
→ op: 0x61ffd0c
9
29
29

ex: of default parameter

```
void fun(int x, int y=100, int z){
    cout<<x<<" "<<y<<" "<<z;
}
int main(){
    fun(10,20);
    return 0;
}
```
→ error (as all default values are not mentioned.)

- An inline func. is a func. which is expanded at each call during execution.

```
inline void copy(int&a, int& b, int& c){
    a*=2;
    b*=2;
    c*=2;
}
int main(){
    int x=2,y=5,z=7;
    copy(x,y,z);
    cout<<"x="<<x<<"y="<<y<<"z="<<z;
    return 0;
}
```
→ reference variables so actual value changes
a = a*2 → 4
b = b*2 → 10
c = c*2 → 14
→ op → x=4, y=10, z=14

```
long factorial(long p){
    if(p>1){
        return (p*factorial(p-1));
    }
    else { return (1); }
}
int main(){
    long q=3;
    cout<<q<<"!="<<factorial(q);
    return 0;
}
```
→ op → stack overflow → segmentation error

p=3 → 3*f(2) (again goes in long fact. func.)
p=4 → 4*f(3)
p=5 → 5*f(4)
∞
so error

ARRAYS - 1 | ARRAY CREATION, TYPES & OPERATIONS

Array → data structure which stores a collection of similar items
- They have contiguous memory
- representation:

ex: (length = 5)

| 1 | 2 | 3 | 4 | 5 | ← elements

index: 0 1 2 3 4
- always integer datatype
- starts with 0 always
- location of every element in array.

Syntax: → datatype arrayname [arraysize]
ex: int arr[5];

Array literal:

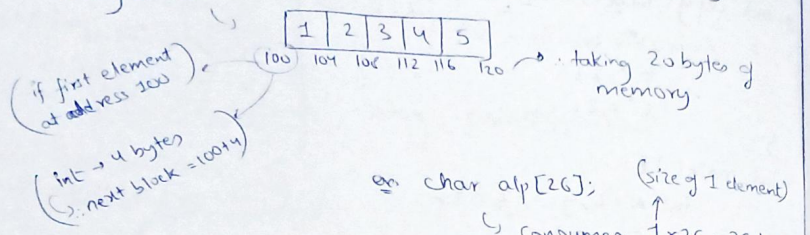
int array[] = {1,2,3,4,5};
// (elements written inside {})

- [] → no value written as we have initialized our array
- if not initialized, then we have to mention it's size, else it would throw an error.

Memory Allocation in arrays:

↳ it's contiguous → means consecutive blocks of memory.

eg: int array[5];



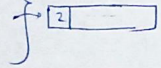
eg: char alp[26]; (size of 1 element)
↳ consumes 1 × 26 = 26 byte.
(char) (total element)

Array Types → Single dimensional (or 1-D array)
↳ Multidimensional array

1-D array:

- Accessing element: * 0-based indexing:

eg: int array[5];
array[0] = 2;



↳ int array[5] = {1,2,3,4,5}
cout << array[0]; → o/p → 1
// array[1]; → 2
array[index]

If → eg: int array[3];
cout << array[0] << endl;
cout << array[1] << endl;
cout << array[2] << endl;
o/p → 43023472
1
43023477
garbage values printed, as we didn't initialize our array.

- Size operations:

eg: int array[] = {1,2,3,4}
cout << sizeof(array);
o/p → 4 × 4 = 16
(space by 1 value) (total values)

// if we want total no. of
// elements in an array, then
cout << sizeof(array) / sizeof(array[0]); → o/p → 16/4 = 4 elements

- Traversing through the Array:

1) For loop
2) For each loop
3) While loop

1) For loop: eg: int arr[4] = {1,2,3,4}; // size of array

```
for(int index=0 ; index<4 ; index++){
    cout<<arr[index]<<endl;
}
```

o/p: 1 2 3 4

2) For each loop: syntax: for(datatype variable : arrayname){

}

↳ traverses each element in array, so we can't skip any element (unlike for loop)

eg:
```
int arr[4]={1,2,3,4};
for(int ele: arr){
    cout<<ele<<endl;
}
```

o/p: 1 2 3 4

3) While loop:
```
int arr[4] = {1,2,3,4};
int index=0;                    // size
while(index<4){
    cout<<arr[index]<<endl;
    index++;
}
```

o/p: 1 2 3 4

- Taking inputs:

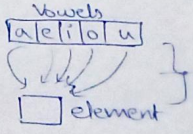
1) For loop: eg:
```
char vowels[5];
for(int idx=0 ; idx<5 ; idx++){
    cin >> vowels[idx];
}
for(int idx=0; idx<5; idx++){
    cout << vowels[idx]<<" ";
}
```

i/p: a e i o u
o/p: a e i o u

2) For each loop:
```
char vowels[5];
① for(char element : vowels){
    cin >> element;
}
for(int idx=0; idx<5; idx++){
    cout << vowels[idx]<<" ";
}
```

i/p: a e i o u
o/p
So this is wrong code ✗

* reason → Line ① {
cout<<element;
}



in every traversal array element gets copied in "element".

If: cin >> element.
↳ then element variable lies at diff. location than vowels variable.

So all values are getting stored in element & not vowel.

- Now if ~~user~~ want to store values in 'vowels', we need to use & operator.
  ↳ (alias name to same memory location)

∴ (correct code:)

```
char vowels[5];
for(char &element: vowels){
        cin>>element;
}
for(int i=0; i<5; i++){
        cout<<vowel[i]<<" ";
}
```

Q) Calculate sum of all elements in a given array?

A)
```
int arr[] = {3,4,10,11}, sum=0, size = sizeof(arr)/sizeof(arr[0]);
for(int i= 0; i<size; i++){
        sum+=array[i];
}
cout<<sum;
```

Q) Find the max. value out of elements in an array?

A) ex: | 3 | 7 | 18 | 9 | 11 |

compare, compare, compare

```
int arr[] = {3,5,18,9,11};
int max = arr[0]; // assume
for(int i=1; i<5; i++){
        if(arr[i]> max){
                max=arr[i];
        }
}
cout<<max;
```

- Linear Search:
  Search if a given element is present in the array or not. If it's not present then return -1 else return the index.

ex:
```
int array[] = {3,9,18,11,7};
int key= 11, ans=-1;
for(int i=0; i<5; i++){
        if(array[i]==key){
                ans=i;
        }
}
cout<<ans<<endl;
```

VECTORS IN C++ | LOOPING & PROBLEM SOLVING

ARRAYS - 2

- Vectors are dynamic arrays where you can resize when you want to insert (or) delete elements
  Arrays are static i.e. we can't resize → ex: int ar[3] (we can't make '3' as '4')
- Contiguous memory allocation
- Basic Operations in Vectors:

1) Declaration: #include <vector>
   (Syntax) vector <datatype> vector_name; (initially it is empty with size=0)
   ex: vector <int> v;
   (If we want to mention size) → vector <datatype> vector_name (size);
   ex: vector <int> v(5)

2) Size: v.size() → length → (vector_name.size())
   (How to check)
   ex: | 1 | 2 | 3 | 4 | → v.size() = 4 (vector name)

3) Resize: vector_name.resize(new size);
   ex: v.resize(8);
   v.size() = 8

4) Capacity: → v.capacity(), (or) vector_name.capacity()
   capacity ≥ size → (if size changes, capacity changes accordingly.)

ex: vector <int> v; size = 0, capacity = 0

| 1 | size = 1, capacity = 1
| 1 | 2 | size = 2, capacity = 2
| 1 | 2 | 3 | size = 3, capacity = 4
| 1 | 2 | 3 | 4 | 5 | size = 5, capacity = 8

(We can see capacity increases in 2^n order → when we are adding elements one by one.)

(now) v.resize(10); → capacity = 20 → (doesn't mean capacity must become 16, it can become something else also. So its compiler dependent.)

[1|2|3] size = 3, capacity = 4

v.resize(5); // size > capacity → so capacity changes

[1|2|3| | | | |] size = 5, capacity = 8 → (so now a new memory space is allocated & all values of vector are copied in this new vector.)

∴ Capacity is the reason, why we don't need to worry about size.

5) Add elements:

* vector_name.push_back(element) → value gets added at last position.

ex: [2|3|4|] v.push_back(5) → [2|3|4|5|]

* vector_name.insert(position, element) → (if we want to insert element at specific position)

(relative to position of 1st element)

(ending position) → vector_name.end()

(we can find by (position)) → vector_name.begin()

or: (5) [2|3|4] → v.insert(v.begin()+2, 5) → [2|3|5|4|]

6) Delete elements:

* vector_name.pop_back → removes last element

ex: [2|3|4] v.pop_back() → [2|3]

* vector_name.erase(position) → (relative to start/end position)

ex: [2|3|4|5|] v.erase(v.end()-2) → (if v.end()-1 → removes 5) → [2|3|5|]

* vector_name.clear() → removes all elements from vector

(in code) { boilerplate code not written here }

```
#include<vector>
using namespace std;
int main(){
    vector<int> v;
    cout<<"size: "<<v.size()<<endl;
    cout<<"capacity: "<<v.capacity()<<endl;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.resize(5);
    v.pop_back;
    v.pop_back;
```

o/p:
size: 0
capacity: 0
size: 3
capacity: 4
size: 5
capacity: 8
size: 3
capacity: 8

• Looping in Vectors: → for → for each → while

```
int main(){
    vector<int> v;
    //for loop
    for(int i=0;i<5;i++){
        int element;
        cin>>element;
        v.push_back(element);
    }
    for(int i=0; i<v.size(); i++){
        cout<<v[i]<<" ";
    }
    cout<<endl;
    v.insert(v.begin()+2,6);
    //for each loop
    for(int ele:v){
        cout<<ele<<" ";
    }
    cout<<endl;
    v.erase(v.end()-2);
    //while loop
    int idx=0;
    while(idx<v.size()){
        cout<<v[idx]<<" ";
        idx++;
    }
}
return 0;
```

in arrays → in for loop → cin>>array[i];

Here to use this, we need to define size of vector, then only we can use:

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

o/p:
1
2
3
4
5

o/p: 1 2 3 4 5
1 2 6 3 4 5
1 2 6 3 5

Q) Find the last occurrence of an element 'x' in given array

A) (x = 1)

arr → | 1 | 2 | 3 | 2 | 1 | 3 | 1 |

↑ ↑ ↑ (6 ✓)

```
(*) {
vector<int> v(6);
for(int i=0; i<6; i++){
    cin >> v[i];
}
cout<<"Enter x:";
int x;
cin >>x;
int occurence = -1;    (if 'x' not there in array, then it will return -1)
```

(when we are searching from last)

```
        last index
for(int i=v.size()-1; i>=0; i--){
    if(v[i]==x){
        occurence = i;
        break;
    }
}
cout<< occurence <<endl;
```

(OR)

(when we are searching from starting)

```
for(int i=0; i< v.size(); i++){
    if (v[i]==x){
        occurence=i;
    }
}
cout<< occurence <<endl;
```

(from last → 1st occurence = from start → last occurence)

Q) Count the number of elements strictly greater than value 'x' ?

A) (*)

```
int occurence = 0;
for (int i=0; i <v.size(); i++){
    if(v[i]>x) {
        occurence ++;
    }
}
cout<< "No. of elements greater than" <<x <<"are" << occurence;
```

Q) Check if the given array is sorted or not ?

A)

```
int array[]={1,2,3,4,5,6};
bool sorted = true;
for(int i=1; i<6; i++){
    if (array[i] <= array[i-1]){
        sorted = false;
    }
}
cout<< sorted ;
```

→ o/p: 1

Q) Find the difference btw the sum of elements at even indices to the sum of elements at odd indices?

A) ex: | | | | | | | | (+ − + − + − +, indices 0 1 2 3 4 5 6) → (even sum - odd sum)

```
int arr[]={1,2,1,2,1,2};
int sum=0;
for(int i=0;i<6; i++){
    if(i%2==0){
        sum+= arr[i];
    }
    else{
        sum -= arr[i];
    }
}
cout << sum;
```

→ o/p: -3

TARGET SUM | PROBLEMS IN ARRAYS - 1

Q) Pattern: Target Sum

a) Find the total number of pairs in the array whose sum is equal to the given value 'x' ?

A)

```
int arr[5]= {3,4,6,7,1};
int target = 5, pairs=0;
for(int i=0; i<5; i++){          (5 = size of array)
    for(int j=i+1; j<5; j++){
        if(arr[i]+arr[j]==target){
            pairs ++;
        }
    }
}
cout<< pairs;
```

→ explanation:

ex: (3), 4, 6, 7, 1

const: 3+4, 3+6, 3+7, 3+1 → we check

Next: 3, (4), 6, 7, 1

4+6, 4+7, 4+1

(no need to check '3+4' again as we did it previously.)

b) Count the number of triplets whose sum is equal to the given value 'x' ?

A)

```
int arr[6]={3,1,2,4,0,6};
int target = 6; triplet = 0;
for(int i=0; i<6; i++){          (6 = size of array)
    for(int j=i+1; j<6; j++){
        for(int k=j+1; k<6; k++){
            if (arr[i]+ arr[j] + arr[k] == target){
                triplet ++;
            }
        }
    }
}
cout<< triplet;
```

→ explanation:

(3) (1) 2, 4, 0, 6

const. const. 3+1+(...) ↑ check

3 → const

↳ 1 - const. ↳ 3+1+

↳ 2 - const. ↳ 3+2+

Like that

Q) Pattern: Array Manipulation

a) Find the unique number in a given array where all the elements are being repeated twice with one value unique?

A)
```
int arr[7] = {1,4,2,3,2,1,4};
for(int i=0; i<7 ; i++){
    for(int j=i+1; j<7 ;j++){
        if(arr[i] == arr[j]){
            arr[i] = arr[j] = -1;
        }
    }
}

for(int i=0; i<7; i++){
    if (arr[i]>0){
        cout<<arr[i];
    }
}
```

→ op: 3

→ explanation:
we first find the pairs & then change its value to -1.
So that the number which is unique doesn't change.

→ Here we print the unique number.

b) Find the second largest element in the given array?

A) ✱ 2 cond.'s possible → all array elements are unique
↳ array elements are repeated. (more accurate)

Method - 1

```
int largeEleIndex(int array[], int size){
    int max = INT_MIN;   (keyword)
    int maxindex = -1;
    for(int i=0; i<size; i++){
        if(array[i] > max){
            max = array[i];
            maxindex = i;
        }
    }
    return maxindex;
}

int main(){
    int array[] = {2,3,5,7,6,1,7};
    int n = 7;
    int indexoflargest = largeEleIndex(array,n);
    cout<< array[indexoflargest] <<endl;

    int largestele = array[indexoflargest];
    for(int i=0; i<n; i++){
        if(array[i] == largestele){
            array[i] = -1;
        }
    }
    int indexofseclarge = largeEleIndex(array,n);
    cout<<array[indexofseclarge]<<endl;
}
```

explanation:
① largeEleIndex → gives index of largest no.
② We make new variable & store value of array at largest index.
③ We change all values of max no. to -1.
④ We find large index & no. & print it

Above method runs loop → 3 times, little lengthy.

Method - 2:

```
int secondLargeEle(int array[], int size){
    int max = INT_MIN;
    int second_max = INT_MIN;
    for(int i=0; i<size; i++){
        if(array[i] > max){
            max = array[i];
        }
    }
    for(int i=0; i<size; i++){
        if(array[i] > second_max && array[i] != max){
            second_max = array[i];
        }
    }
    return second_max;
}

int main(){
    int array[] = {2,3,5,7,6,1,7};
    int n = 7;
    cout<<secondLargeEle(array,n);
}
```

Q) Rotate the given array 'a' by k steps, where k is non-(-)ve.
Note: 'k' can be greater than 'n' as well where 'n' is size of array 'a'?

A) eg: a=[1,2,3,4,5] , k=2

→ step1: [5 1 2 3 4]
→ step2: [4 5 1 2 3]

but if k>n → eg: k=24
↳ then 20 + 4 times
same 'a' 'a' shift by '4'.

k = k%n;

→ Method - 1:

```
int array[]={1,2,3,4,5};
int n=5, k=23;
k = k%n;
int ansarray[5];
int j=0;
// inserting last k elements in ans.arr
for(int i=n-k; i<n; i++){
    ansarray[j++] = array[i];
}
// inserting first n-k elements in ans arr.
for(int i=0; i<=k; i++){
    ansarray[j++] = array[i];
}
for(int i=0; i<n; i++){
    cout << ansarray[i]<<" ";
}
```

3 4 5 1 2

explanation:
[1, 2, 3, 4, 5] , k=3
(n-k) , k
last k elements will come to front & first (n-k) elements will come to end.
[3,4,5,1,2]

Here we declared 'j' outside, so j value is increased only after first j++.

In above soln; we are using extra memory space by creating ans array.

In order to avoid that, we can operate on same array.

Method-2:

↳ Now to reverse a vector: `reverse(v.begin(), v.end());`

`#include<algorithm>` needed

ex: [1 2 3 4 5] → [5, 4, 3, 2, 1]

(reverse) (reverse)

(k=2) → [4, 5, 3, 2, 1]

```
vector<int>v = {1,2,3,4,5};
int k=2;
k = k%v.size();
reverse(v.begin(), v.end());
reverse(v.begin(), v.begin()+k);
reverse(v.begin+k, v.end());
for(int a:v){
    cout<<a<<" ";
} cout<<endl;
```

→ incase doesn't work (use push_back property)

→ ~~code not working in my vs code but working in mam's vs code~~

Q) Given Q queries, check if the given number is present in the array (or) not.

Note: Value of all the elements in the array is less than 10 to the power 5. → (10^5)

A) ex: [2, 3, 5, 6, 7, 8] → 4 queries: 3, 6, 7, 11

```
int n; cin>>n;
vector<int> v(n);
for(int i=0; i<n; i++){
    cin>>v[i];
}
const int N = 1e5+10;
vector<int> freq(N,0);
for(int i=0; i<n; i++){
    freq[v[i]]++;
}
cout<<"Enter queries: "; int q; cin>>q;
while(q--){
    int queele;
    cin>>queele;
    cout<<freq[queele]<<endl;
}
```

→ scientific notation of 10^5.

initial value, size

// Enter queries: 3, 6, 7, 11

// 1 1 0 1 0 1

// 0 0 1 1 1

Enter queries: 5

2 → repeated twice

3 → repeated once

8 → 0 not present

Explanation: → Now Q queries can be very big no. (one crore) we need to check the query & traverse in array always.

2nd better soln:

Now lets create a 10^5 size → freq-array

& now wherever elements are present in original array, in those indices we can do +1.

ex: [2, 3, 5, 6, 7, 8]

freq array: | | 1 | 1 | | 1 | 1 | 1 | 1 | 0 |

2 3 4 5 6 7 8

Basically Pre-processing

Now, we need to only check, whether that query (or) element in that query, 1 is present (or) 0 is present.

Ex: we can get 400 queries also, so this is better, like we don't need to search in whole array but just it's freq.

TWO POINTERS | PROBLEMS IN ARRAY - 2

Q) Pattern: Two Pointers

a) Sort an array consisting of only 0's & 1's?

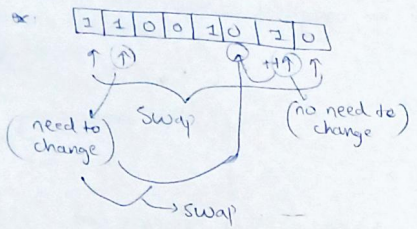
A) ex: 1 1 0 0 1 0 1 0 → 0 0 0 0 1 1 1 1

Now we get count of 0 & then we keep '0' until it's equal to count of 0 & then 1's at rest of the position.

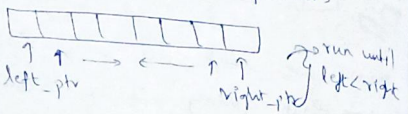
```
void SortZeroAndOne(vector<int> &v){ (pass by reference)
    int zero_count=0;
    for(int ele:v){
        if(ele==0){
            zero_count++;
        }
    }                                  } counting zeroes
    for(int i=0; i<v.size(); i++){
        if(i<zero_count){
            v[i]=0;
        }
        else{
            v[i]=1;
        }
    }                                  } sorting array
}
int main(){
    int n;
    cin>>n;
    vector<int> v(n);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    SortZeroAndOne(v);
    for(int i=0; i<n; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

Now in above soln., we need to traverse array 2 times → for finding zeroes
↳ re-sorting array
(Think of other soln.)

METHOD-2



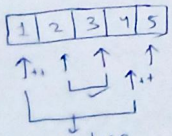
Means we are traversing our array from front & back & swapping when needed.
So we create 2 pointers




```
void sort2and0(vector<int>&v){
    int left = 0;
    int right = v.size()-1;
    while(left<right){
        if(v[left]==1 && v[right]==0){
            v[left++]=0;
            v[right--]=1;
        }
        if(v[left]==0){
            left++;
        }
        if(v[right]==1){
            right--;
        }
    }
    return;
}
```

```
int main(){
    int n;
    cin>>n;
    vector<int> v;
    for(int i=0;i<n;i++){
        int ele; cin>>ele;
        v.push_back(ele);
    }
    sort2and0(v);
    for(int i=0; i<n; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

b) Given an array of integers 'a', move all the even integers at the beginning of the array followed by all the odd integers. The relative order of odd (or) even integers does not matter. Return any array that satisfies the conditions?

A) 

: Here we can again run 2 pointers, one from start & other from end

. We get our answer by parity of odd & even

{decimal → binary chapter}

```
void sortByParity(vector<int> &v){
    int left = 0;
    int right = v.size()-1;
    while(left<right){
        if(v[left]%2==1 && v[right]%2==0){
            swap(v[left], v[right]);   (inbuilt func.)
            left++; right--;
        }
        if(v[left]%2==0){left++;}
        if(v[right]%2==1){right--;}
    }
    return;
}
```

```
int main(){
    int n; cin>>n;
    vector<int> v;
    for(int i=0; i<n; i++){
        int ele; cin>>ele;
        v.push_back(ele);
    }
    sortByParity(v);
    for(int i=0; i<n; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

c) Given an integer array 'a' sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order?

A) for all positive inputs, que → easy

eg: [-10, -3, 2, 5, 6] —take absolute value→ [|-10|, |-3|, |2|, |5|, |6|] → [100, 36, 25, 9, 4]

left ... right ← for 2 pointer approach

check sorting of absolute values & then returning its square

```
void sortSquareArr(vector<int> &v){
    vector<int> ans; //(create an ans array to store our values)
    int left=0;
    int right = v.size()-1;
    while(left<=right){
        if(abs(v[left])<abs(v[right])){   (inbuilt func. for absolute)
            ans.push_back(v[right]*v[right]);
            right--;
        }
        else{
            ans.push_back(v[left]*v[left]);
            left++;
        }
    }
    reverse(ans.begin(), ans.end());
    for(int i=0; i<ans.size(); i++){
        cout<<ans[i]<<" ";
    }cout<<endl;
}
```

```
int main(){
    int n;
    cin>>n;
    vector<int> v;
    for(int i=0; i<n; i++){
        int ele;
        cin>>ele;
        v.push_back(ele);
    }
    sortSquareArr(v);
    return 0;
}
```

PREFIX SUM PROBLEMS | PROBLEMS IN ARRAY - 3

Q) Pattern : Prefix Sums

a) Given an integers array 'a', return the prefix sum / running sum in the same array without creating a new array.

A) eg: [5, 4, 1, 2, 3] → [5, 9, 10, 12, 15]

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

(that ele. position + prev. sum) → we can't make new arr.

∴ prefix sum$_{i+1}$ = prefix sum$_i$ + a_{i+1}

```
int main(){
    int n;
    cin>>n;
    vector<int> v(n);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    runningSum(v);
    for(int i=0;i<n;i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

```
void runningSum(vector<int> &v){
    for(int i=1; i<v.size(); i++){
        v[i]=v[i]+v[i-1];
    }
    return;
}
```

(as first element won't change so we start from 2nd element)

b) Check if we can partition the array into 2 sub-arrays with equal sum. More formally, check that the prefix sum of a part of the array is equal to the suffix sum of rest of the array?

A) Simple way to do is → calculate prefix sum & Suffix sum = total sum − prefix sum; (check at each step)

```
int main(){
    int n; cin>>n;
    vector<int> v(n);
    for(int i=0;i<n;i++){
        cin>>v[i];
    }
    cout<<checkPreSufSum(v);
    return 0;
}
```

```
bool checkPreSufSum(vector<int> v){
    int total=0;
    for(int i=0; i<v.size(); i++){
        total+=v[i];
    }
    int pre_sum=0;
    for(int i=0; i<v.size(); i++){
        pre_sum+=v[i];
        int suf_sum=total-pre_sum;
        if(suf_sum==pre_sum){
            return true;
        }
    }
    return false
}
```

Dry run n=5 → 6 2 4 3 1 → total = 16

pre_sum = 6 → pre_sum = 6+2 = 8 → if cond. satisfied return true

suf_sum = 16−6 = 10 → suf_sum = 16−8 = 8

c) Given an array of integers of size 'n'. Answer 'q' queries where you need to print the sum of values in a given range of indices from l to r (both included).

Note: The values of l & r queries follow 1-based indexing

A) eg: (1 based indexing)

	a_1	a_2	a_3	a_4	a_5	a_6
	1	2	3	4	5	6

l=2, r=6 → $(a_2+a_3+a_4+a_5+a_6)$ → we can do a for loop from 'l' to 'r'.

But since queries can be many & large in size, so we should think of a better way

	5	1	2	3	4
	1	2	3	4	5

l=2 → Prefix sum$_l$; r=4 → Prefix sum$_r$ → we can write "ans = Prefix sum$_r$ − Prefix sum$_{l-1}$"

```
int main(){
    int n; cin>>n;
    vector<int> v(n+1,0);
    for(int i=1; i<=n; i++){
        cin>>v[i];
    }
    //Calculate prefix sum array
    for(int i=1; i<=n; i++){
        v[i]=v[i]+v[i-1];
    }
    int q;
    cout<<"Enter queries"<<endl;
    cin>>q;
    while(q>0){
        int l,r;
        cin>>l>>r;
        int ans=0;
        //ans = pref.sum(r) - pref.sumarray(l-1)
        ans=v[r]-v[l-1];
        cout<<ans;
        --q;
    }
    return 0;
}
```

index → 0 1 2 3 4 5 6

vector → 0 x1 x2 ...

0 based indexing → 0 ... n−1

1 based indexing → 1 ... n

initial value is 0.

Dry run

i/p → 5 1 2 3 4

vector → 0 5 1 2 3 4

pref.sum array → 0 5 6 8 11 15 (0 (+5) (+1) (+2) (+3))

Let q=3 → {means our prog. will run 3 times}

l=1, r=3 → ans = v[3] − v[0] = 8−0 = 8

l=2, r=5 → ans = v[5] − v[1] = 15−5 = 10

l=4, r=4 → ans = v[4] − v[3] = 3 (1st index element)

RECAP - 4

Q) Given 2 vectors arr1[] & arr2[] of size 'm' & 'n' sorted in increasing order. Merge the 2 arrays into a single sorted array of size m+n?

A) ex: arr1 = [1,3,5], arr2 = [2,4,6] → arr = [1,2,3,4,5,6]

ex: arr1 = [1,6,7,10] (m=4), arr2 = [0,2,3,8,11] (n=5)

result = [, , , , , , , , ,] (m+n = 9) → inc.

On 0th index we will store the smallest element btw 2 arrays

To fill result[0], we will compare arr1[0] & arr2[0].

→ ex: 1, 0 → 0<1 (j++) ; 2>1 (i++) → like that we compare 'i' & 'j' & do k++.

result[0] → k++ ; result[1] → k++

code:

```
int main(){
    int arr1[] = {1,6,7,10};
    int arr2[] = {0,1,3,8,11,12,15,18};
    int n=4, m=8;
    //code to merge 2 sorted arrays
    int result[m+n];
    int i=0, j=0, k=0;
    while(i<m && j<n){
        if(arr1[i]<arr2[j]){
            result[k] = arr1[i];
            k++; i++;
        } else {
            result[k] = arr2[j];
            k++; j++;
        }
    }
    while(i<m){
        result[k] = arr1[i];
        i++; k++;
    }
    while(j<n){
        result[k] = arr2[j];
        k++; j++;
    }
    for(int i=0; i<(m+n); i++){cout<<result[i]<<" ";}
    return 0;
}
```

iterate on result

iterate on arr2

will help us interate on arr1

→ arr2 was exhausted & we still got elements left in arr1

→ arr1 was exhausted & we still got elements left in arr2

Method 2: We can take 2 sorted arrays → merge in 1 → then use sorting

```
for(int i=0; i<m+n; i++){
    arr[i] = arr1[i];
    if(i>=m){
        arr[i] = arr2[m+n-i-1];
    }
}
```

Q) Given a vector arr[] sorted in increasing order of 'n' size & an integer 'x', find if there exists a pair in the array whose sum is exactly x.

A) Brute force method: → [, , , , ,] (i j)

Calculating all possible pairs & checking sum = x

If i/p is large this method not useful.

Method 2

ex: [-2, -1, 0, 3, 6, 8, 11, 12] (n=8) ; x = 14

(i) ; (j)

arr[i] + arr[j] > x → we need to ↓ sum → j--

11 + 3 ✓

12 + (-2) = 10 (means we need more sum)

if arr[7] → arr[6] → sum ↓s → we do i++

arr[i] + arr[j] < x → we need to ↑ sum

Code:

```
int main(){
    int arr[] = {-2,-1,0,3,6,8,11,12};
    int x=14, n=8;
    int i=0, j=n-1;
    bool found = false;
    while(i<j){
        if(arr[i]+arr[j]==x){
            //we found a pair
            found = true;
            break;
        }
        else if(arr[i]+arr[j]<x){
            i++;
        }else{
            j--;
        }
    }
    if(found==true){cout<<"YES";}
    else{cout<<"NO";}
    return 0;
}
```

o/p → YES

Q) Given a vector arr[] sorted in ↑ order of 'n' size & an integer x, find if there exists a pair in the array whose absolute difference is exactly x?

A) We *may* be able to do with prev. method but let's figure out new method:

ex: [1, 6, 8, 12, 15, 19] ; x = 7

i → 1, j → 6

if → (|arr[i] − arr[j]| < x) → we need to increase the abs diff.

(arr[i] − arr[j] > x)

we need to decrease the absolute diff. → i++

j++ ← so ↑j

since i, j moving in forward direction, so we have to make sure both are less than 'n'

Code:

```
int main(){
    int arr[] = {9,23,45,69,78};
    int n = 5 , x = 10;
    //code to check whether there is any pair with the
    //given abs. diff.
    int i=0, j=1;
    bool found = false;
    while(i<n && j<n){
        if(abs(arr[i] - arr[j]) == x){
            found = true;
            break;
        }else if(abs(arr[i] - arr[j]) < x){
            j++;
        }else{
            i++;
        }
    }
    return 0;
}
```

Q) Given a vector arr[] sorted in increasing order. Return an array of squares of each no. sorted in increasing order. Where size of vector : 1 < size < 101?

ex: [-4, -3, -1, 0, 2, 10] → [0, 1, 4, 9, 16, 100]

A) [-4, -3, -1, 0, 2, 10] — given ip

i → -4, j → 10

4 > 2 ✓

[, , , , ,] → result

0 1 2 3 4 5

i++ k-- ; j-- k-- } like that comparing from (last & first) + k--

Largest element we can get by the input's last or first element

Code:

```
int arr[] = {-4,-3,-1,0,2,10};
int n = 6;
int result[n];
int i=0, j=n-1, k=n-1;
while(i<=j && k>=0){
    if(abs(arr[i]) > abs(arr[j])){
        result[k] = arr[i] * arr[i];
        i++; k--;
    }
    else{
        result[k] = arr[j] * arr[j];
        j--; k--;
    }
}
for(int i=0; i<n; i++){cout<<result[i]<<" ";}
```

o/p: 0 1 4 9 16 100

Q) Given a vector arr[] sorted in increasing order of 'n' size & an integer x, find the no. of unique pairs that exist in the array whose absolute sum is exactly 'x'?

ex: [1,2,3,4,6] x = 7 → o/p = 2 ; {1,6 & 3,4}

ex: [2,2,2] x = 4 → o/p = 1 ; {2,2 is only unique pair}

A) Variation 1 → [2,2,2,2] x = 4

if we calculate based on index → o/p = 2

(like in any pair, we should have different indexes element)

Variation 2 → [1, 3, 3, 3] x = 4

(1,3) → now you can't take another '1' & '3', so we skip it

This que is similar to que. 2, but here we are not going to break loop, instead count++. We use → target Sum concept

```
int arr[] = {-2, -1, 0, 3, 6, 8, 11, 12};
int x = 10, n = 8, i = 0, answer = 0, j = n-1;
while (i<j){
    if (arr[i] + arr[j] == x){
        cout<<"Sum is "<<arr[i]<<" and "<<arr[j]<<endl;
        answer++;
        i++; j--;
    }
    else if (arr[i] + arr[j] < x){
        i++;
    }
    else{
        j--;
    }
}
cout<<answer;
```

2D-ARRAYS-1 | WHAT, WHY, AND HOW?

Multidimensional Array: array of arrays

↳ syn: datatype array_name [size 1] [size 2] ... [size N];

2D array: → 2 dimensions {like a table}

↳ syn: datatype array_name [row][col];

if row = 2, column = 3 ↳ representation :

(total elements = 2×3 = 6)

	col 1	col 2	col 3
row 1	a[0][0]	a[0][1]	a[0][2]
row 2	a[1][0]	a[1][1]	a[1][2]

Initializing 2D array: ex: int array[2][3] = {1,2,3,4,5,6};

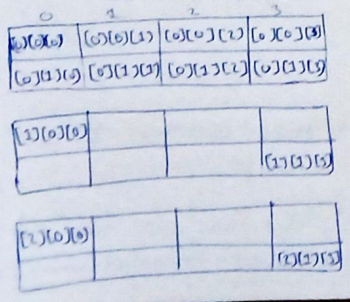
(or) = {{1,2,3}, {4,5,6}};

(in row1) (in row2)

↳ ex: initialising 3D array

ex: 3×2×4 → total elements: 3×2×4 = 24

↳ 3 → 2D arrays of [2×4]



Taking input:

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

Q) Why Multidimensional Arrays?

A) (1) representing grids (2) faster access of any element

(3) pre-defined size (4) ex: Day 1, Day 2, ... Hour 1, Hour 2

↳ 2D array represent

(if year added)

↳ (3D array)

Q) WAP to display multiplication of 2 matrices entered by the user?

A) ex: $\begin{bmatrix}1&2\\3&4\end{bmatrix}_{r_1\times c_1} \cdot \begin{bmatrix}5&6\\7&8\end{bmatrix}_{r_2\times c_2} = \begin{bmatrix}1\times5+2\times7 & 1\times6+2\times8\\ 3\times5+4\times7 & 3\times6+4\times8\end{bmatrix} = \begin{bmatrix}19&22\\43&50\end{bmatrix}$

(Both matrices should have $c_1 = r_2$)

↳ o/p → $r_1 \times c_2$ size

ex: $a = \begin{bmatrix}0&5&2\\-1&1&0\\7&4&3\end{bmatrix}_{3\times3}$ $b = \begin{bmatrix}1&2\\0&4\\3&5\end{bmatrix}_{3\times2}$ → $c = \begin{bmatrix}\\\\\end{bmatrix}_{3\times2}$

(for this → 0,1,2 we create another loop with 'k')

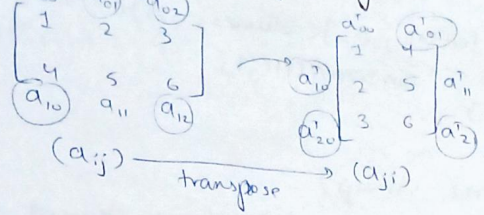
↳ = (row no. of b) = r_2

```
for(i=0; i<3; i++){
    for(j=0; j<2; j++){
        sum=0;
        for(k=0; k<3; k++)
        {
            sum=sum+a[i][k]
                    *b[k][j]
        c[i][j]=sum;
    }
}
```

```
int main(){
    int r1,c1,r2,c2;
    cin>>r1>>c1;
    int A[r1][c1];
    for(int i=0; i<r1; i++){
        for(int j=0; j<c1; j++){
            cin>>A[i][j];
        }
    }
    cin>>r2>>c2;
    int B[r2][c2];
    for(int i=0; i<r2; i++){
        for(int j=0; j<c2; j++){
            cin>>B[i][j];
        }
    }
    if(c1==r2){
        int C[r1][c2];
        for(int i=0; i<r1; i++){
            for(int j=0; j<c2; j++){
                int value=0;
                for(int k=0; k<r2; k++){
                    value+=A[i][k]*B[k][j];
                }
                C[i][j]=value;
            }
        }
        (printing matrix)
        for(int i=0; i<r1; i++){
            for(int j=0; j<c2; j++){cout<<C[i][j]<<" ";}
            cout<<endl;}
    }
    return 0;
}
```

Q) WAP to display transpose of matrix entered by the user.

A) ex:



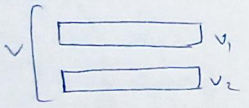

```
int n,m;
cin>>n>>m;
int arr[n][m];
for(int i=0; i<n; i++){
  for(int j=0; j<m; j++){
    cin>>arr[i][j];
}}
```

```
int transpose[m][n];
for(int i=0; i<m; i++){
  for(int j=0; j<n; j++){
    transpose[i][j] = arr[j][i];
}}
for(int i=0; i<m; i++){
  for(int j=0; j<n; j++){
    cout<<transpose[i][j]<<" ";
  }cout<<endl;
}
```

IMP. PROBLEMS IN 2D ARRAY | 2D VECTORS, PASCAL'S Δ

2D vectors: `vector<vector<datatype>> vector_name;`

eg.




```
vector<int> V1 = {1,2,3};
vector<int> V2 = {4,5};
vector<vector<int>> V = {V1,V2};
```

2D vector gets created → {{1,2,3},{4,5}}

Initializing with size:

2D → `vector<vector<int>> V(n, vector<int>(m));`

(rows) (size of column) = m

↳ size (n×m)

(If "vector<int>(m,0)" is written, then a vector of size n×m is made with elements = 0)

Q) Given an integer n, return the first 'n' rows of Pascal's triangle

```
      1
    1   1
  1   2   1
1   3   3   1
1 4   6   4 1
```

A) (1) Let us imagine our Δ as: $\{ {}^{n}C_{r} = \frac{n!}{(n-r)!\,r!} \}$

```
      1
    1   1
  1   2   1
1   3   3   1
1 4   6   4 1
```

→ By checking rows & columns:

↳ (a) Columns = row_number

(b) a[i][j] = ${}^{i}C_{j}$

(row no.) (column no.)

Now we need → 2 nested for loops (row, column)

(so that we get value of each element)

code:

```
long int factorial(int x){
  int fact = 1;
  for(int i=2; i<=x; i++){
    fact = fact*i;
  }
  return fact;
}
```

```
int main(){
int n;
cin>>n;
for(int i=0; i<n; i++){
  for(int j=0; j<=i; j++){
    cout<< factorial(i)/(factorial(j)*factorial(i-j)) <<" ";
  }cout<<endl;
}
return 0;}
```

(2) In above method, we have to find factorial of each position. Let's search for new soln. where we get our ans. maybe from prev. rows.

```
      1
    1   1
  1   2   1
1   3   3   1
1 4   6   4 1
```

(Pattern.)

(a) Columns = row_number = no. of elements in row

(b) a[i][j] = a[i-1][j] + a[i-1][j-1]

(c) n^{th} row → 0^{th} column → j = 0 → ${}^{n}C_{0}$

↳ j = n → ${}^{n}C_{n}$ → (element value)

code →

```
vector<vector<int>> pascalTri(int n){
    vector<vector<int>> pascal(n);
    for(int i=0; i<n; i++){
        pascal[i].resize(i+1);
        for(int j=0; j<i+1; j++){
            if(j==0 || j==i){
                pascal[i][j]=1;
            }
            else{
                pascal[i][j]=pascal[i-1][j]
                            +pascal[i-1][j-1];
            }
        }
    }
    return pascal;
}
```

```
int main(){
    int n;
    cin>>n;
    vector<vector<int>> ans;
    ans = pascalTri(n);
    for(int i=0; i<ans.size(); i++){
        for(int j=0; j<ans[i].size(); j++)
        {
            cout<<ans[i][j]<<" ";
        }cout<<endl;
    }
    return 0;
}

//#include<vector> library is
// included.
```

Dry run:

Let n = 4 → 2D vector of size '4' created. → [— — — —]

→ 0th element of vector is converted to vector of size (1)

[[1] — — —] } if else condn.

Now i = 2, resize → 3

{if-else condn.}

[[1], [1,1], — —] ← if else condn. ← Now to 2nd row, 'i' value changes to 2 {i = 1 → resize = 2}

[[1], [1,1], [1,2,1], — —] → (similarly ——)

ADVANCED PROBLEM SOLVING IN 2D ARRAYS

(1) Given a boolean 2D array, where each row is sorted. find the row with maximum number of 1's.?

ex: ip → row = 3, col = 4 op → 0

matrix[] = {{0 1 1 1},
{0 0 0 1},
{0 0 0 1}}

	0	1	2	3
0	0	1	1	1
1	0	0	0	1
2	0	0	1	1

As ex: ① We can make count of 1 in each row & compare (Better approach ↓)

② * SORTED * mention in que. → means if we find any '1' at 'x' col the rest all to right of it are 1's

ex: 2nd row → first occurence of 1 = 2

No. of ones = Columns - index = 4 - 2 = 2

```
int maxOnesRow(vector<vector<int>>&V){
    int maxOnes = INT16_MIN;
    int maxOnesRow = -1;
    int columns = V[0].size();
    for(int i=0; i<V.size(); i++){
        for(int j=0; j<V[i].size(); j++){
            if(V[i][j]==1){
                int numberOfOnes = columns-j;
                if(numberOfOnes > maxOnes){
                    maxOnes = numberOfOnes;
                    maxOnesRow = i;
                }
                break; ——— (1)
            }
        }
    }
    return maxOnesRow;
}
```

Dry run:

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

maxOnes = min_value
maxOnesRow = -1
columns = 4
{traversing in loop}
numberOfOnes = 4 - 1 = 3

(next row)

maxOnes = 3
maxOnesRow = 0
columns = 4
numberOfOnes = 4 - 3 = 1
↳ (1 > 3) → false

(next row)

maxOnes = 3
maxOnesRow = 0
columns = 4
numberOfOnes = 4 - 2 = 2

after checking condn
↓
we return maxOnesRow

```
int main(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> vec(n,vector<int>(m));
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>vec[i][j];
        }
    }
    int res = maxOnesRow(vec);
    cout<<"Row is"<<res<<endl;
    return 0;
}
```

(1) → to get out of columns loop, as we got our desired val. So we got to next row.

(3)

	0	1	2	3
0	0	1	1	1
1	0	0	0	1
2	0	0	1	1
3	1	1	1	1

→ row with leftmost one → row with max 1's ✓

i.e. row 0 → leftmost 1 at 3
row 1 → ∝
row 2 → ∝
row 3 → leftmost 1 at 4 ✓

[now we have to see if there are any 1's to the left of 3 in remaining rows]

Code:

```
int leftMostOneRow(vector<vector<int>>&V){
    int leftMostOne = -1, maxOnesRow = -1, j = V[0].size()-1;
    while(j>=0 && V[0][j]==1){
        leftMostOne = j;
        maxOnesRow = 0;
        j--; }
    for(int i=1; i<V.size(); i++){
        while(j>=0 && V[i][j]==1){
            leftMostOne = j;
            j--; maxOnesRow = i; }}
    return maxOnesRow;
}
```

(finding leftmost one in 0th row)

(checking in rest of the rows if we can find a one left to the leftMostone)

return maxOnesRow; }

(Here 'j' value is taken from (1))

"int main" → code you may edit it little from 2nd approach.

Q) Type: ROTATION OF MATRIX

a) Given a square matrix, turn it by 90 degrees in a clockwise direction without using any extra space?

1	2	3
4	5	6
7	8	9

⇓

7	4	1
8	5	2
9	6	3

ex: i/p → [[1,2,3], [4,5,6], [7,8,9]]

o/p → [[7,4,1], [8,5,2], [9,6,3]]

A) "Here we can't use any extra array")

```
1 2 3                  1 4 7     reverse      7 4 1
4 5 6   --transpose->  2 5 8    --every-->    8 5 2   ✓
7 8 9                  3 6 9       row        9 6 3
                         (i=j)
```

code

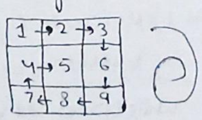
```
void rotateArray(vector<vector<int> > &vec){
    int n = vec.size();
    for(int i=0; i<n; i++){
        for(int j=0; j<i; j++){      → if j<n
                                       (vec 2 ↔ 4 & 4 ↔ 2
                                        we will get original)
                                       (so we change
                                        lower Δ matrix)
            swap(vec[i][j], vec[j][i]);   } transpose
        }
    }

    for(int i=0; i<n; i++){   ~ reverse
        reverse(vec[i].begin(), vec[i].end());
    }
    return;
```

```
int main(){
    int n;
    cin>>n;
    vector<vector<int>>vec(n,
                        vector<int>(n));
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            cin>>vec[i][j];
        }
    }
    rotateArray(vec);
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            cout<<vec[i][j]<<" ";
        }
    }
    return 0;
}
```

ADVANCED PROBLEM SOLVING IN 2D-ARRAYS ~2

Q) Given a n×m matrix 'a', return all elements of matrix in spiral order? {SPIRAL MATRIX TYPE}

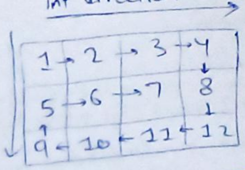
ex.



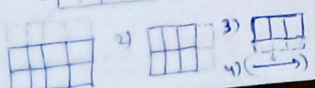
o/p → 1 2 3 6 9 8 7 4 5

int direction=0 →

A)



1) Left → right = 0
   top++ → (removing 1st row)
2) Top → bottom = 1
   right-- → (removing rightmost column)
3) Right → left = 2
   bottom-- → (removing last row)
4) bottom → top = 3   left++ → (removing 1st column)



```
void spiralOrder(vector<vector<int> >&matrix){
    int left=0;
    int right = matrix[0].size()-1;
    int top=0;
    int bottom=matrix.size()-1;
    int direction = 0;
    while(left<=right && top<=bottom){
        //left → right
        if(direction==0){
            for(int col=left; col<=right; col++){
                cout<<matrix[top][col]<<" ";
            } top++;
        }
        // top → bottom
        else if(direction==1){
            for(int row=top; row<=bottom; row++){
                cout<<matrix[row][right]<<" ";
            } right--;
        }
        //right → left
        else if (direction==2){
            for(int col=right; col>=left; col--){
                cout<<matrix[bottom][col]<<" ";
            } bottom--;
        }
        //bottom → top
        else{
            for(int row=bottom; row>=top; row--){
                cout<<matrix[row][left]<<" ";
            } left++;
        }
        direction= (direction+1)%4;
        //direction val must be 0,1,2,3 only
    }
}
```

```
int main(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> matrix(n,
                        vector<int>(m));

    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            cin>>matrix[i][j];
    }}
    spiralOrder(matrix);
```

Q) Given a (+)ve integer 'n', generate an n×n matrix filled with elements from 1 to n^2 in spiral order

e. 3 → 3×3 → 1 to 9 elements →

```
1 → 2 → 3
        ↓
8 → 9   4
↑       ↓
7 ← 6 ← 5
```

A) Logic is similar to prev. sum. So lets write code directly

```
vector<vector<int>> spiralMat(int n){
    vector<vector<int>> matrix(n, vector<int> n);
    int left = 0, right = n-1, top = 0, bottom = n-1;
    int direction = 0;
    int value = 1;
    while(left<=right && top<=bottom){
        if(direction==0){
            for(int i = left; i<=right; i++){
                matrix[top][i] = value++;
            } top++;
        }
        else if(direction==1){
            for(int j = top; j<=bottom; j++){
                matrix[j][right] = value++;
            } right--;
        }
        else if(direction==2){
            for(int i = right; i>=left; i--){
                matrix[bottom][i] = value++;
            } bottom--;
        }
        else{
            for(int j = bottom; j>=top; j--){
                matrix[j][left] = value++;
            } left++;
        }
        direction = (direction+1)%4;
    }
    return 0;
}
```

```
int main(){
    int n;
    cin>>n;
    vector<vector<int>> matrix(n, vector<int>);
    matrix = spiralMat(n);
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            cout<<matrix[i][j]<<
        } cout<<endl;
    }
    return 0;
}
```

ADVANCED PROBLEM SOLVING IN 2D-ARRAYS ~3

Q) Pattern: Prefix Sums in 2D Arrays

Given a matrix 'a' of dimension n×m & 2 co-ordinates (l1,r1) & (l2,r2). Return the sum of the rectangle from (l1,r1) to (l2,r2)

Ans: 1 2 3 4 / 5 6 7 8 / 9 10 11 12 — $(l_1, r_1) = (1,1)$, $(l_2, r_2) = (2,2)$

o/p → 6+7+10+11 = 34

{3×4}

Method-1: Brute Force

$l_1 \le l_2$, $r_1 \le r_2$ → we run 2 loops each with boundary conds → $i \to l_1$ to l_2; $j \to r_1$ to r_2; sum += array[i][j]

Code:

```
int rectangleSum(vector<vector<int>> &matrix, int l1, int r1, int l2, int r2){
    int sum=0;
    for(int i=l1; i<=l2; i++){
        for(int j=r1; j<=r2; j++){
            sum+=matrix[i][j];
        }
    }
    return sum;
}
```

} returning sum

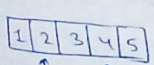
```
int main(){
    int n, m;
    cin>>n>>m;
    vector<vector<int>> matrix(n, vector<int>(m));
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            cin>>matrix[i][j];
        }
    }
    int l1, r1, l2, r2;
    cin>>l1>>r1>>l2>>r2;
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            cout<<matrix[i][j]<<" ";
        } cout<<endl;
    }
    int sum= rectangleSum(matrix, l1, r1, l2, r2);
    cout<<sum<<endl;
    return 0;
}
```

} input

} printing matrix

Method-2: Pre-Calculating the horizontal sum for each row in the Matrix

Prefix sum method we did in 1D array. →



arr[l] = arr[0] ... arr[l]

arr[r] = arr[0] ... arr[l] ... arr[r]

Sum = arr[r] − arr[l−1]

Similarly in 2D array,

1 2 3 4 / 5 6 7 8 / 9 10 11 12 → 1 3 6 10 / 5 11 18 26 / 9 19 30 42

Prefix sum matrix → arr[l1][r2] − arr[l2][r1−1]

13 − 5 = 13

30 − 9 = 21

34

Code:

```
int rectangleSum(vector<vector<int>>&matrix, int l1, int r1, int l2, int r2){
    int sum = 0;
    for(int i=0; i<matrix.size(); i++){
        for(int j=1; j<matrix[0].size(); j++){
            matrix[i][j] += matrix[i][j-1];
        }
    }
    for(int i=0; i<matrix.size(); i++){
        for(int j=1; j<matrix[i].size(); j++){
            cout<<matrix[i][j]<<" ";
        } cout<<endl;
    }
    for(int i=l1; i<=l2; i++){
        if(r1!=0){
            sum += matrix[i][r2] - matrix[i][r1-1];
        }
        else{
            sum += matrix[i][r2];
        }
    }
    return sum;
}
```

Prefix sum array row wise

Printing prefix sum array row wise

Applying logic

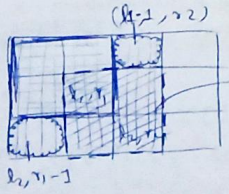
→ when r1=0

("int main" part is same)

Method-3: Prefix Sum Over columns and Rows Both

1 2 3 4	(row-wise)	(column-wise)
5 6 7 8	1 3 6 10	1 3 | 6 10
9 10 11 12	5 11 18 26	6 14 | 24 36
	9 19 30 42	15 33 54 78

∴ arr[l1][r1] → rectangle sum from (0,0) to (l1, r1)



∴ sum = □ $l_2 r_2$ − □ l_1-1, r_2 − □ l_2, r_1-1 + □ l_1-1, r_1-1

→ Need to remove.

{arr[l][r] → rectangle sum from (0,0) to (l,r)}

Code:

```
int rectangleSum(vector<vector<int>>&matrix, int l1, int r1, int l2, int r2){
    int sum = 0;
    for(int i=0; i<matrix.size(); i++){
        for(int j=1; j<matrix[0].size(); j++){
            matrix[i][j] += matrix[i][j-1];
        }
    }
    for(int i=1; i<matrix.size(); i++){
        for(int j=0; j<matrix[0].size(); j++){
            matrix[i][j] += matrix[i-1][j];
        }
    }
    for(int i=0; i<matrix.size(); i++){
        for(int j=0; j<matrix[0].size(); j++){
            cout<<matrix[i][j]<<" ";
        } cout<<endl;
    }
    int top_sum=0, left_sum=0, topleft_sum=0;
    if(l1!=0){top_sum = matrix[l1-1][r2];}
    if(r1!=0){left_sum = matrix[l2][r1-1];}
    if(l1!=0 && r1!=0){topleft_sum = matrix[l1-1][r1-1];}
    sum = matrix[l2][r2] - top_sum - left_sum + topleft_sum;
    return sum;
}
```

prefix sum array column wise

printing prefix sum 2D array

if l1, r1 = 0, no need to remove any block.

→ ("int main" part is same)

TIME SPACE COMPLEXITY & BIG O NOTATION | HOW TO AVOID TLE

Time Complexity:

ex. You're given a no. 'x' & y ($1 \le x \le 10^5$, $x \le y \le 10^5$). Calc. sum of all the numbers in the range [x, y]?

Ans {Soln. 1}

```
int ans=0;
for(int i=x; i<=y; i++){
    ans += i;
}
cout<<ans;
```

This code is good for small values of 'x' & 'y', but if x=2 & $y = 10^8$ → then loop has to run many times. So, it's not a feasible soln. Let's optimize it.

{Soln. 2}

Sum of 'n' terms in an A.P $= \frac{n}{2}[2a+(n-1)d]$

n → no. of terms = y − x + 1

a → first term = x

d → difference = 1

```
int n = y-x+1;
int a = x;
int result = n/2 * (2*a + (n-1)*1);
cout<<result;
```

Now as we can see in soln. 2, we need not run loop for each increment in 'x'. We just used one formula & got the ans.

- **Experimental time analysis:** is actual time taken by the code (analysis) { to get executed i.e. time taken by code from starting of prog. to end of prog.

→ If we run a given code; we can get different exp. time (maybe high or low)
(we can check in VS code)

this analysis is not a fruitful one as it takes diff. time for same code to execute.
{ reason: many things run in background, so it depends }

So, we talk based on: No. of operations.

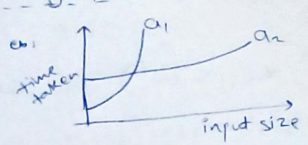
eg: a_1 → 10^5 operations/instructions
a_2 → 10 operations/instructions

Now → if comp. takes 1 ns for 1 operation
$t_1 = x \times 10^5$ ns (nanoseconds)
$t_2 = x \times 10$ ns
$(t_2 < t_1)$ (✓)

∴ t_2 has better working condn.

Similar to above example.
In soln. ① → if $x=1, y=100000$ → 10^5 operations
Soln. ② → $x=1, y=100000$ → within few operations we are getting our ans. ✓
↳ so soln. 2 is better soln.

- **Asymptotic analysis:** No. of operations w.r.t. change in input.

eg:


→ a_2 ✓ better than a_1

In this analysis, we always consider big inputs & also talk w.r.t. input size.

eg: (M1)
```
int n = (y-x+1);          —①
int a = x;                —②
int result = (n*(2a + (n-1)*1))/2;   (4 operations)
return result;            —③
```
for n iterations → total 7 instructions only (const. instr.)

(M2)
```
int result = 0;           ④
for(int i = x; i<=y; i++){ result += i; }
return result;            ⑤
```
n iterations → check cond^n, increment 'i', result add
∴ Total 3n instructions

If $x=1, y=10^8$ → in (M1) → only 7 operations
(in M2) instructions → $3 \times 10^8 + 3 \approx 3n + 3 \approx n$
(constant ops) → (neglected)

∴ M1 is better soln. wrt M2 as: input size ↑ ⇒ time taken ↑ (in M2)

→ Growth = $\frac{\Delta \text{no. of instru.}}{\Delta \text{input size}}$ → {More growth, not feasible code / Less growth, optimised code.

Time order:
- constant (best cases)
- log(n)
- $\sqrt{n}$
- n
- n log(n)
- n^2
- n^3
- 2^n (worst cases)

(growth ↑)

- **Types of Time Complexity Analysis & their Notations**
1) Worst Case Time Complexity —repre.→ Big O → O( )
2) Best Case Time Complexity —"→ Big omega → Ω( )
3) Average Case Time Complexity —"→ Big theta → Θ( )

→ whenever we get const. no. of instructions → write 1
↳ ie. O(1), Ω(1), Θ(1)

Ex1: Time complexity when traversing 2 individual arrays of length M & N respectively.

Code:
```
int main(){
    int arr1[] = {1,2,3,4,5,6,7,8,9};
    int arr2[] = {1,2,4,12,7,11,0,9};
    int n = 9, m = 8, sum1 = 0, sum2 = 0;
    for(int i=0; i<n; i++){
        sum1 += arr1[i];
    }
    for(int i=0; i<m; i++){
        sum2 += arr2[i];
    }
    cout<<sum1<<" "<<sum2;
    return 0;
}
```
→ prep. i/p & const.'s (ignored)
(3 instruc.) (i times running) → 3n times executed ≈ n (const. ignored)
→ 3m times executed ≈ m
∴ Time Complexity = O(m+n) (we are adding)

Ex 2(a): Time Complexity for Nested loops

```
int main(){
    int n=5;
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            cout<<"*";
        }
        cout<<endl;
    }
    return 0;
}
```

i:0 i:1 i:2 ... i:n-1
n + n + n ... + n = n*n = n^2

'n' iterations → O(n)

{Ignoring all const. terms}

∴ Total time complexity = O(n^2)

2(b):

```
for(int i=0; i<n; i++){
    for(int j=0; j<i; j++){
        cout<<"*";
    }cout<<endl;
}
```

i=0 → j=0 instruction
i=1 → j=1 "
i=2 → j=2 "
...
i=n-1 → j=n-1 "

0+1+2+3+...+n-1
= n(n-1)/2
= n^2/2 - n/2

ex: $\frac{10^{10}}{2} - \frac{10^5}{2}$

(small value wrt 10^{10}) (so ignored)

$\approx \frac{10^{10}}{2}$ ≈ n^2

ignored. Worst Case: O(n^2)

* #include <bits/stdc++.h> = consists all header files of c++

2(c):

```
for(int i=0; i<n; i++){
    for(int j=0; j<sqrt(n); j++){
        cout<<"*";
    }cout<<endl;
}
```

i=0, j=sqrt(n) instruction
i=1, j=sqrt(n) "
i=2, j=sqrt(n) "
i=3, j=sqrt(n) "
...
i=n-1, j=sqrt(n) "

∴ Total = n * sqrt(n)

(total no. of terms) (& in every term, we are doing sqrt(n) operations)

→ Worst Case → O(n sqrt(n))

→ Avoiding all const.

Space Complexity:

It's an extra memory space requirement of an algorithm, using asymptotic analysis.

→ i/p & o/p are not considered in SC

ex: fibonacci no.'s: 0, 1, 1, 2, 3, 5, 8, 13, ...
(default) (any i^th term is the sum of prev. 2 terms)
0th (Fib. no.)

if que. is: → Print the n^th fibonacci number?

Soln. 1)

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

Let n = 81;
Then with change in input, the space we took also changed.
↳ we created an array of 82 elements.
∴ Input ↑ = space ↑

∴ Worst Case → Space: O(n)
↳ Time: O(n) (n-2 ex)

(Soln. 2)

Now, for optimised soln, let think that for n^th F.No., we need only (n-1)^th & (n-2)^th (F.No) & rest all array is waste. So how about instead of maintaining whole array, we are just maintaining the prev. 2 elements.

```
int a=0;
int b=1;
int c=1;
for(int i=2; i<=n; i++){
    c=a+b;
    a=b;
    b=c;
}
return c;
```

→ n^th fib. no

dry run {let n=8}

a = 0 1 1 2 3 5 8 13
b = 1 1 2 3 5 8 13 21
c = 1 1 2 3 5 8 13 21
i = 2 3 4 5 6 7 8 (9)

∴ return c → we get 21

We can also put initial condn.
↳ like → if(n==0 || n==1){
return n;
}

If n = 8 (or 80), variables a, b, c are independent with change in input size. → Space → O(1) (const.)
Time → O(n) (n-2 ex)

CALCULATING TIME COMPLEXITY | PROBLEM SOLVING

Q) Find Time complexity:

a) int value = 0;
for(int i = 1; i ≤ n; i *= 2){
value++;
}

A) for(int i = 1; i ≤ n; i++){
value++;
}

1 → 2 → 3 → 4 ---- → n-1 → n → n+1 (terminate)

iterations → n times
comparision → (n+1) times
↳ O(n)

Now;

1 → 2 → 4 → 8 ---------- 2^k

(1st iteration) (2nd iteration) (3rd iteration) (4th iteration) ---- {let $2^k \leq n$}

~ 2^0 2^1 2^2 2^3 2^4 ---- 2^{k-1} 2^k

1st 2nd 3rd 4th ---------- $(k+1)^{th}$ iteration

Now: $2^k \leq n$ ⇝ (taking log on both sides)

↳ $\log_2 2^k \leq \log_2 n$

⇒ $k \leq \log_2 n$

∴ Max to max ⇝ total iterations = $k+1 = \log_2 n + 1$ {worst case}

∴ $T_c = O(\log n)$

b) int val = 0;
for(int i = 1; i ≤ N; i += i){
val++;
}

A)

Value of (i)	iteration #
2^0 ← 1	1
2^1 ← 2	2
2^2 ← 4	3
2^3 ← 8	4
⋮	⋮
2^{m-1} ← k	m

Now we are writing $(\log_2 n)$ as $(\log n)$ because if we want to write $\log_2 n$ as $\log_3 n$, we just need to multiply it with a const. So ignored

& condn. ⇝ $i = 2 \times i$

*This prob. is same as prev. prob. ↳ ∴ $T_c = O(\log n)$

c) int val = 0;
for(int i = 1; i ≤ N; i *= 2){
for(int j = 1; j ≤ i; j++){
val++;
}
}

A)

i	j	iteration #
1	1	1 } 1
2	1 2	2 3 } 2
4	1 2 3 4	4 5 6 7 } 4
8	1 2 3 4 ⋮ 8	8 9 10 11 12 13 14 15 } 8
⋮	⋮	⋮
k	2^k	2^k

total iterations = 1 + 2 + 4 + 8 + ---
↓ ↓ ↓ ↓
2^0 2^1 2^2 2^3

(k → no. of times 'i' is multiplied)

GP → $a \times \frac{(2^n - 1)}{(2-1)}$ → $1 \times (2^k - 1) = 2^k - 1$ (writing k in terms of N)

$2^k \leq N$ → taking log on both sides

↳ $k \leq \log_2 N$

∴ Total iterations = $2^k = 2^{\log_2 N} = O(N)$

*We have to figure out how many iterations are going.

Q) int val = 0;
for(int i = 1; i ≤ N; i *= 2){
for(int j = N; j > i; j--){
val++;
}
}

a) O(log N) b) O(N) c) O(N log N) d) None

A) $i = 2^0$ → $j \to [N, 2]$ (N-1) times
$i = 2^1$ → $j \to [N, 3]$
$i = 2^2$ → $j \to [N, 5]$
$i = 2^3$ → $j \to [N, 9]$
(k iterations);
$i = 2^{k-1}$ → $j \to [N, 2^k + 1]$

Total iterations = $(N-1) + (N-2) + (N-4) + \cdots - (N - 2^{k-1})$
(k times)
$= (N + N + \cdots - N) - (2^0 + 2^1 + \cdots 2^{k-1})$
$= Nk - 2^k + 1$ ---- ①

(writing k in terms of N)
↳ $2^{k-1} \leq N$ ⇝ $k \leq \log_2 N + 1$ ---- ②

∴ ② in ①

$N \log_2 N + 1 - 2^{\log_2 N} = N \log N + \underbrace{1 - N}_{\text{(smaller term)}} \approx O(N \log N)$

Q) int val = 0;
for(int i = N; i > 0; i /= 2){
for(int j = 0; j < i; j++){
val++;
}
}

a) O(log N) b) O(N) c) O(N log N) d) None

A) $i = N$ (1st ite.) → $j : [0, N-1]$ — N iterations

$i = N/2$ (2nd ite.) → $j : [0, N/2 - 1]$ — $N/2$ iterations

$i = N/4$ (3rd ite.) → $j = [0, N/4 - 1]$ — $N/4$ iterations

$i = \frac{N}{2^{k-1}}$ (k^{th} ite.) → $j : [0, 0]$ — 1 iteration

$\therefore$ Total $= N + N/2 + N/4 + \dots$

$= \left\{ a \times \frac{(r^n - 1)}{r - 1} \right\} \rightarrow$ GP

$= \frac{N\left(1 - \left(\frac{1}{2}\right)^{\log N}\right)}{1 - \frac{1}{2}}$ (k = total no. of terms)

$= \frac{N\left(1 - \frac{1}{n}\right)}{1 - \frac{1}{2}} = 2(N-1) \approx N$

$\therefore T_c = O(N)$

(final value) $\frac{N}{2^{k-1}} = 1 \rightarrow N = 2^{k-1} = \left(\frac{2^k}{2}\right)$

$\log(2N) = k$

$k \approx \log N$

Q) int val = 0;
for(int i = 2; i ≤ N; i *= i){
val++;
}

A) 2 → 4 → 16 → 256 → ...

2^1 2^2 2^4 2^8 ... 2^k (final value) $\therefore 2^k \le N$

$\log_2 2^k \le \log_2 N$

$k \le \log_2 N$

Powers are also in the form of 2^x.

→ 2^{2^0} 2^{2^1} 2^{2^2} 2^{2^3} ... $2^{2^x} \le k$

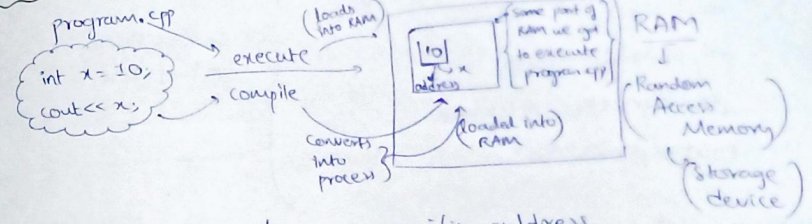
$\therefore 2^x \le k$ (log)

$x \le \log_2 k$

$\therefore T_c = O(\log k)$

$= O(\log(\log(N)))$

POINTERS - 1 | INTRODUCTION, DATA STORAGE & ACCESS USING POINTERS



Each memory location has a specific address.
Now we can access the variable value using its address also.

syntax: datatype *pointer_name;
(type of whose address we want to store)

Storing the address of data in a pointer:
If we want to access the address of a memory bucket we can use: Address of operator → "&"

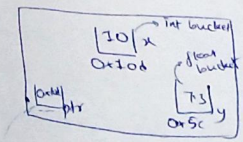
{usage → &<var-name>}

ex: int x = 10; → what's address in memory?
cout << &x; → 0x16b543368

Q) Can we store this address?
A) Pointers can store address.

{but pointers also create another bucket [above analogy] to store address. ↳ ⊔}

ex: int x = 10; → &x → 0x10d
float y = 7.3; → &y → 0x5c



Now: int z = (&x) → address → here char. + no. there → not possible

int *ptr → using this we can use to store address of an "integer" variable

float *ptr → using this we can store address of a float variable.

ex: int *ptr = &x;
cout << ptr;
Now ptr is also bucket, so it also has some address

{int* p / int * p / int *p}

if int *ptr;
cout << ptr → (garbage value)

Also,

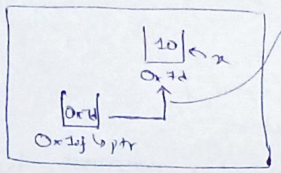
```
int *ptr ; // only declaring the pointer
cout<<ptr<<"\n";
int marks = 90;
ptr = &marks; // update
cout<<ptr<<"\n";
```

→ 0x104e4b05c
→ 0x16af5f33c

Accessing data through a pointer :

Dereference operator → *

ex:



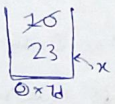
Now we want to access our data using the address of variable.

∴ cout<<ptr; → o/p: 0x7d

cout<<*ptr; → o/p: 10

We can use *ptr to actually access value stored on the address pointed by ptr.

Now, if: int x = 23; →



Now here bucket is same, just content is changed.

↳ so: cout<<*ptr; → o/p: 23

AND

*ptr → means it tells to go to this address

0x7d

So: if: *ptr = 50;

```
cout<<x;
cout<<*ptr;
```

→ o/p: 50 50

Also we can,

```
int valueAtX = *ptr;
cout<<valueAtX;
```

→ o/p: 50

Common errors while using pointers :

```
int x = 9, y = 2;
int *ptr = &x;
```

(ptr = 5;) → (error) ← (we can't do this as ptr stores only address & nothing else)

(*ptr = &y;) → (error) ← (*ptr → means 'x' : this won't work as we cannot store address in an int bucket)

Now, pointers are also basically bucket storing address. It itself also has an address.

```
int x = 5;
int *ptr = &x;
cout<<&x<<"\n"<<ptr<<"\n"<<&ptr;
```

→ o/p: 0x16d32 (address of 'x')
0x16d32
0x16d28 → (address of pointer itself)

Adding 2 no.'s using pointers

```
int x = 10, y = 20;
int *ptrx = &x;
int *ptry = &y;
int result;
int *ptr_result = &result;
*ptr_result = *ptrx + *ptry;
cout<<result<<" "<<*ptr_result;
```

*ptrx → 10

*ptry → 20

*ptrx + *ptry = 30

*ptr_result = 30

(we store 30 on the address stored in *ptr_result)

∴ result → 30

o/p: 30 30

POINTERS-2 | CALL BY REFERENCE, ARITHMATIC AND ARRAY AS A POINTER

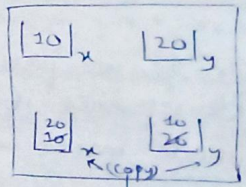
Call by reference using pointers :

ex1.

```
void swap(int x, int y){
    int temp = x;
    x = y;
    y = temp;
}
```

(copies)

```
int main(){
    int x = 10, y = 20;
    swap(x,y);
    cout<<x<<" "<<y;
}
```



(happening in new memory location)

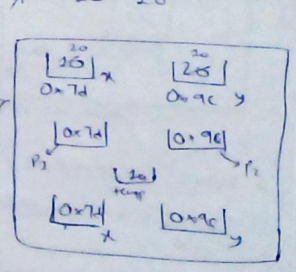
So the 'x' & 'y' inside main fun. are still the same, so we get o/p as 10 20.

To swap values, lets use pointers concept.

```
void swap(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main(){
    int x = 10, y = 20;
    int *p1 = &x;
    int *p2 = &y;
    swap(p1,p2);
    cout<<x<<" "<<y;
}
```

→ o/p 20 10



Now *x (of void fun.) means we go to the original value present at '0x7d'
(dereference operator)

In new copy, 'x' & 'y' → pointer copy is made but its not any

problem, because in copy also address of original 'x' & 'y' is stored.

Since address is stored, so we can easily de-reference it. After de-referencing it, we will get back to org. val. of 'x' & 'y'

ex-2.

We want to find index of first occuring & last occuring of 'a' in the string → aaacad

Soln

```
void compute(string s, char ch, int *first, int *last){
    for(int i=0; i<s.size(); i++){
        if(s[i]==ch){
            *first = i;
            break;
        }
    }
    for(int i=s.size()-1; i>=0; i--){
        if(s[i]==ch){
            *last = i;
            break;
        }
    }
}
```

① ②

```
int main(){
    String s="aaacad";
    char ch='a';
    int first=-1;
    int last=-1;
    int *pf = &first;
    int *pl = &last;
    compute(s, ch, pf, pl);
    cout<<*pf<<" "<<*pl<<"\n";
    cout<<first<<" "<<last;
}
```

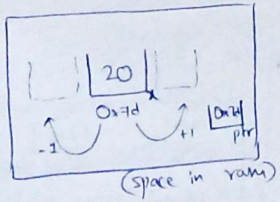
o/p: 0 4
0 4

In ⓪, we pass address & de-reference it, so any changes made inside →*← affects the whole

Pointer Arithmetic

(increment) (decrement) } different from integer arithmatic

ex: int x = 20;
int *ptr = &x;



∴ If we do ptr+1 → it will go to the next memory location

∴ Here increment/decrement of a pointer value refers to shift in memory location that pointer is pointing to!

• Shift is dependent on size of the type of variable

```
int x = 20;
int *ptr = &x;
cout<<ptr<<" "<<(ptr+1);
```

o/p: 0x16bab734 0x16bab738

in hexadecimal: 8 a a a b c

8+4 → as int. datatype takes 4 bytes size

(∴ ptr+1 ≡ adding 1 unit equivalent of the size of the type.)

ex:
```
double dec = 9.8;
double *ptrd = &dec;
cout<<ptrd<<" "<<(ptrd+2);
```

o/p: 0x16bab340 0x16bab350

= 40 →(+8) 48 → 49 → a → b → c → d → e → f → 50 (+8)

(double)

∴ 'ptrd+2' means: ptrd + 2 × size of (double)

2×8 = 16

→ We can also write: ptr = ptr+1 as ptr += 1

ex:
```
int arr[2] = {1,19};
int *ptr = &arr[0];
cout<<ptr<<" "<<*ptr<<"\n";
cout<<*ptr++<<"\n";
cout<<arr[0]<<" "<<arr[1]<<"\n";
cout<<ptr<<" "<<*ptr;
```

o/p:
0x16d98b320 1
1
1 19
0x16d98b324 19

(ptr ~, then ++)

this means ptr++ occurs then ptr. gets *, means its now pointing to next element of array

ex:
```
int arr[2] = {5,4};
int *ptr = &arr[0];
cout<<(*ptr)++<<"\n";
cout<<arr[0]<<" "<<arr[1]<<"\n";
```

o/p: 5
6 4

this means pointer is getting de referenced & now the value is increasing by 1

ex:
```
int arr[2] = {7,5};
int *ptr = &arr[0];
cout<<*++ptr<<"\n";
cout<<arr[0]<<" "<<arr[1]<<"\n";
```

o/p: 5
7 5

ex:
```
int arr[2] = {8,23};
int *ptr = &arr[0];
cout<<++*ptr<<"\n";
cout<<arr[0]<<" "<<arr[1];
```

o/p: 9
9 2

*ptr = 8
now ++ = 9
9 printed, as we read from right to left

∴ Total cases: *ptr++

(*ptr)++

*++ptr → first move pointer by 4 bytes, then de-reg.

++*ptr → first deref. increment the deref. value

Arrays as Pointers

ex: int arr[3] = {15, 12, 6};
int *ptr = &arr[0];
cout<<ptr<<" "<<arr<<" "<<*ptr<<" "<<*arr;

o/p: 0x16d338 0x16d338 15 15

How?

because name of array acts as pointer whose address is of 0th index. {Array ke naam mae 0th index ka address hota hai}

∴ If: *(arr+0) → 0th index value
*(arr+1) → 1st index value
*(arr+2) → 2nd index value

for(int i=0; i<3; i++){
cout<<*(arr+i)<<"\n";
}

o/p: 15
12
6

Now we can also call by reference.

```
void process(int *arr, int n){
    for(int i=0; i<n; i++){
        cout<<*(arr+i)<<" ";      (short hand of above one)
        (OR)
        cout<<arr[i]<<" ";
    }
    *(arr+1) = 33;
}

int main(){
    int arr[3] = {5,1,2};
    process(arr, 3);
    for(int i=0; i<3; i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

o/p: 5 1 2
5 33 2

Now we can use pointer to point an entire array.

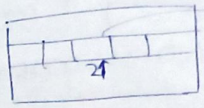
ex: int a[3] = {11,12,13};
int (*p)[3] = &a;
cout<<p<<" "<<a<<" "<<*p<<" "<<*a;

o/p: 0x165728 0x165728 0x165728 11

(even if we de-reg 'p', we get address of array only)

*a → already points to 0th index.
This means 'p' variable points completely to array.

Also, arr[3][4]



can write as.
((p+i)+2)
↑ column

POINTERS-3 | TYPES OF POINTERS, NULL, WILD, DANGLING, VOID

Wild Pointer

- A wild pointer is a type of a pointer where the user declares the pointer but not initialise it. Due to this it ends up pointing to some random memory location.

ex: int x → declaration of variable
int *ptr → declaration of a pointer variable (garbage value)

- Due to this we might get some undefined behaviour & crashes. In some cases we might get segmentation fault

Null Pointer:

- If we want to have a pointer variable, which is just declared but will get address later to store, we can use Null pointer.

ex: int *ptr = NULL; (null pointer)

ex: int *ptr = NULL;
int *p1 = 0;
int *p2 = '\0';
cout<<ptr<<" "<<p1<<" "<<p2;

o/p: 0x0 0x0 0x0
(All 3 have same address)
{0 in ASCII form/Null in ASCII form}

- '0' is mostly a special reserved memory address in many OS.
- Runtime error might occur on de-referencing null pointer.

Dangling Pointer:

It's a type of pointer that points to a memory location that is not valid now. (initially it might be valid)

ex:
```
int *ptr = NULL;
{
  int x = 10;
  ptr = &x;
}
cout<<ptr;
```
→ O/P: 0x7b67c764 (i.e. it points to that variable which doesn't exist)

Void Pointer:

- These are special pointers that can point to any datatype value.
- Void pointers cannot be de-referenced, but using typecasting we can!

ex:
```
float f = 11.3;
int x = 10;
void *ptr = &f;
ptr = &x;
int *intPtr = (int *)ptr;
cout<< *intPtr;
```
O/P: 10

RECURSION | RECURSION CONCEPT AND PROBLEMS

In recursion, we try to solve a bigger problem by finding out solutions to smaller sub-problems.

We represent these problems in the form of functions & these func. calls itself to solve smaller sub-problems.

ex: Write a func which calcs n!

→ $5! = 5\times4\times3\times2\times1 = 5\times4! = 5\times4\times3! = 5\times4\times3\times2! = 5\times4\times3\times2\times1!$

$f(n) = n \times f(n-1)$: $\{n! = n\times(n-1)!\}$

(func. to calc. n!) (Recursion → function calls itself)

simple soln. {Iterative soln}
```
ans=1;
for(int i=1; i<=n; i++){
  ans=ans*i;
}
```

Now, how can we think about this? So we should know; PMI → Principal of Mathematical Induction

Here we do 3 works:
- Base Case
- Assumption
- Self-work

ex: Prove sum of first 'N' natural no.s is $\frac{n(n+1)}{2}$.

Soln: # Base Case → for N=1 → ans will be = 1

Assumption → Lets say formula works for N=k → $\frac{k(k+1)}{2}$

Self-Work → $(1+2+3+\dots+k+k+1)$

$\frac{(k)(k+1)}{2} + (k+1) = \frac{(k+1)(k+2)}{2}$ → Satisfies for sum upto k+1. Similarly upto, k+2, k+3, ...

Hence Proved

In recursion also we use these 3 cases.

Now lets go back to factorial problem,

$f(n) = n \times f(n-1)$ → (recurrence reln.)

(assump + self-work)

Base Case = Halting condn.
Assumption = f(n-1)
Self-Work = multiply n & f(n-1)

we wrote a func. f (that returns n!) & assume func. f works correctly for 'n-1'

⇒ $5! = 5\times4!$ → $4\times3!$ → $3\times2!$ → $2\times1!$ → base case {1! = 1}

(120, 24, 6, 3×2, 2×1)

n==1 → f(n) = 1

∴ in code.

```
1) int f(int n){
2)   if(n==1){return 1;}
4)   int ans = n*f(n-1);
5)   return ans;
6) }
   int main(){
     int result = f(4);
     cout<<result;
     return 0;
   }
```

Now this prog. converts into process which is loaded into RAM. Now memory to this allocated in call stack way.

We can also write this as, return n*f(n-1);

{Better explanation next page}

O/P = 120