

## DATA STRUCTURE & ALGO

1.0 Binary (0,1) → Language of computer  
 $(0,1 \rightarrow \text{Low/High Voltage})$

Ques We have two hourglass of 7hr & 4hr.  
 How to measure 9mins?

In 4 hours, the hourglass II will be empty, further we will flip the hourglass II for 3 hr till hourglass I is empty. Now 7 hours have elapsed and we have 1 hr sand in hourglass II, let the sand flow & flip hourglass I. After eighth hour, hourglass II will be empty, again we will flip hourglass I & II. When hourglass I will be empty, 9 hours will be elapsed.

1.1 PROGRAM: Set of instructions that tell a computer what to do.

1.2 Compiler: Computer program that translates HLL to Machine Language. (Intermediate Code)

1.3 Interpreter: Executes the instructions directly & removes the process of converting to intermediate code

1.4 C++ → Cross-platform language, Developed by Bjarne Stroustrup.

Ques 3 light bulbs in a room, 3 button outside, can go inside once, how to find the light & its switch

First, turn on a light for 30 mins, turn it off & enter the room,

↳ WARM LIGHT → First Switch

↳ Glowing Light → Second Switch

↳ OFF LIGHT → Third Switch

Ques 3ltr jug + 5ltr jug. How to measure 4ltr

↳ Fill 3ltr jug (0, 3)

↳ Pour 3ltr to 5ltr & fill 3ltr again (3, 3)

↳ Pour 3ltr to 5ltr again (5, 1)

↳ Empty 5ltr & pour 3ltr to 5ltr (1, 0)

↳ Fill 3ltr (1, 3)

↳ Pour 3ltr to 5ltr (4, 0)

## 2.0

VARIABLES: Container used to hold data.

Each var should have a unique name

Data Type	Meaning	Size
int	Integer	4
float	Floating-point	4 → Upto 7 Decimal
double	Double floating pt.	8 → Upto 15 Decimal
char	Character	1 → Inside single quote
wchar_t	Wide Character	2
bool	Boolean	1 → false / true
void	Empty	0

## 2.1 DATA TYPE Modifiers

- ↳ Used to modify the fundamental Data Type.
- ↳ signed, unsigned, long, short

iostream: Input/Output stream in C++ used for input as well as output of a particular value.

- ↳ cout → Standard Output
- ↳ cin → Standard Input
- ↳ cerr → Standard Error

3.0 \* Comment: Displays important info about prog.  
Compiler skips the comment line

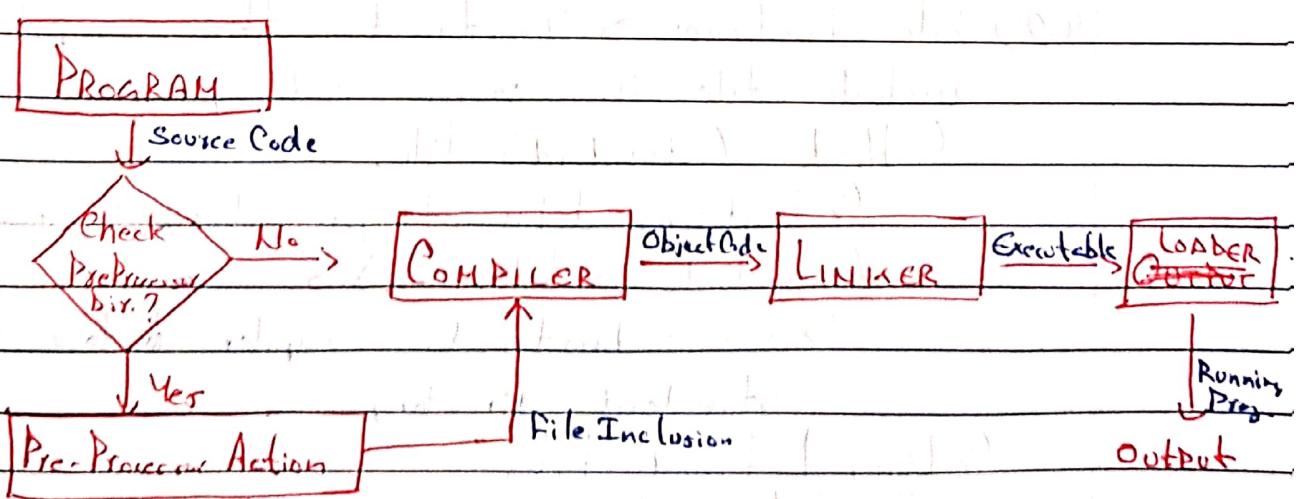
// Single line      /\* Multi-line Comment \*/

When & Why Comments?

↳ Make our code more readable

↳ Provide description to algorithm

4.0 \* Pre-Processor: Programs that process source code before compilation.



↳ Pre-processor begins with `#` symbol.

4.1) Macro → Piece of code which is given some name.

↳ Whenever the name is encountered, compiler replaces name with code.

`#include <iostream>`

`using namespace std;`

`#define LIMIT 5` ] → The limit var defined  
int main() { ... } ] will replace whenever  
`cout << LIMIT << endl;` ] called.  
`return 0;`

↳ Note: There is no semicolon (`;`) at the end of macro definition.

↳ We can also pass arguments to macro & make it as function.

`#define AREA(l, b) (l*b)` ] → Here, the macro

`AREA(5, 2)` ]

is sep by space b/w arguments &

return value, act like function.

4.2) File Inclusion → Instructs the compiler to include file in source code program.

↳ Header (Standard) files

↳ Contains defn of pre-defined functions

`#include <filename>` ]

↳ `<>` → Instructs compiler to look files in standard directory.

↳ User Defined Files

`#include "filename"`

↳ `" "` → Instruct compiler to look in current directory.

### 4.3) Header File Inclusion

↳ Used in large programs where a single user-defined header is in use by multiple files.

**#ifdef MACRO**

Used in case of Macro

**#endif**

**#ifndef -- HEADER--**

**#define -- HEADER--**

HEADER FILE CONTENT

**#endif**

→ Used in header

files so that

linker doesn't link

the content multiple

times.

(github.com/kshitizsaini13/L2-Emulator)

↳ Here, if the macro is defined, the compiler will skip block of code else will simply execute it.

### 4.4) #undef → Used to undefine a macro.

**#undef MACRO.**

5.0 Why not use "using namespace std"

↳ It is suggested to use scope resolution operator (::) to import specific functions rather than importing entire std namespace.

~~# using std::cout;~~

↳ std namespace is huge & has hundreds of defn so user may ~~redefine~~ redefine & result in compilation error.

6.0 main()

↳ main() → Entry point of program.

↳ Every C++ program must have a main() function.

## 7.0

### BASIC INPUT / OUTPUT

- ↳ C++ Input / Output are known as streams
- ↳ Input Stream: Keyboard → Main Memory
- ↳ Output Stream: Main Memory → Screen

iostream : (Standard input / output) Contains defn of cin, cout, cerr, etc.

iomanip : (Input / output manipulators) Contains methods for manipulating streams.

fstream : (File Stream) Handler data from file as input or output

↳ cout : Produce o/p on the stream (screen) using the insertion operator (<<)

↳ cin : Read i/p from the keyboard using the extraction operator (<>)

↳ cerr : Standard error stream used to o/p error message immediately. & don't store for later.

↳ clog : Error is first inserted to buffer and then displayed.

## 8.0

### bits/stdc++.h

↳ Using <bits/stdc++.h>

↳ Not available in all compilers

↳ Reduces time of writing include

↳ Increases compilation time.

## 3.0 NAME SPACE

↳ Allow us to group named entities & remove them from global namespace to reduced namespace.

namespace NAMESPACE { }

}

## 4.0 endl v/s \n

↳ endl → Inserts a new line & flushes the stream.

↳ \n → Only inserts a new line.

`cout << endl;` = `cout << '\n' << flush;`

\n is performance-wise better unless flushing of stream is reqd.

## 5.0 PreIncrement v/s postIncrement

↳ preIncrement can be used as left value but postIncrement can never be used as a left value.

$$t+a = 20 \quad \checkmark$$

$$a++ = 20 \rightarrow \text{ERROR}$$

↳ ++a returns a reference to variable to which we can further assign whereas a++ holds the value.

$$\text{int } b = ++a \rightarrow \checkmark$$

$$\text{int } b = a++ \rightarrow \text{ERROR}$$

$$a++ = 20 \quad \text{Illegal}$$

$$\text{G} \boxed{10 = 20}$$

$$a = a + 1$$

## 12.0 PRECISION FOR NUMBERS

( $\hookrightarrow$ ) 12.1) `floor()`

( $\hookrightarrow$ ) Rounds off to closest integer less than given value.

( $\hookrightarrow$ ) 12.2) `ceil()`

( $\hookrightarrow$ ) Rounds off to closest integer more than given value.

( $\hookrightarrow$ ) 12.3) `trunc()`

( $\hookrightarrow$ ) Removes digits after decimal points.

( $\hookrightarrow$ ) 12.4) `round()`

( $\hookrightarrow$ ) Rounds number to closest integers.

( $\hookrightarrow$ ) 12.5) `setprecision()`

( $\hookrightarrow$ ) Provides precision to floats when used with fixed keyword.

```
int x = 1.13113
cout << floor(x); → 1
cout << ceil(x); → 2
cout << trunc(x); → 1
cout << round(x); → 1
cout << fixed << setprecision(2) << x; → 1.13
```

## 13.0 if - elif - else

( $\hookrightarrow$ ) Used to specify the code to be executed according to some conditions.

`if (Condition) {`

$\hookrightarrow$  `else if (Condition) {`

$\hookrightarrow$  `else {`

$\hookrightarrow$  `}`

Que

Check if even odd number.

#include <iostream>

using namespace std;

int main() {

int n;

cin >> n;

if (n % 2 == 0) {

cout << "Even";

}

else {

cout << "Odd";

}

return 0;

}

Condition for checking  
if a number is  
even or odd by  
dividing it by 2 and  
checking remainder.

Que

Max-Min from two numbers.

void max\_min (int n1, int n2) {

if (n1 > n2) {

cout << "Max" << n1;

cout << "Min" << n2;

}

else (n2 > n1) {

cout << "Max" << n2;

cout << "Min" << n1;

}

y

Que

Max from 3 numbers

int max\_three (int n1, int n2, int n3) {

if (n1 > n2) {

if (n1 > n3) {

```

cout << return n1;
}
else {
    cout << return n3;
}
}
else if (n2 > n3) {
    cout << return n1;
}
else {
    cout << return n3;
}
}

```

Que Check the type of triangle.

```

if (s1 == s2 && s2 == s3) {
    cout << "Equilateral" << endl;
}
else if (s1 == s2 || s2 == s3 || s3 == s1) {
    cout << "Isosceles" << endl;
}
else {
    cout << "Scalene" << endl;
}

```

## 14.0 Loops

↳ Loops are used for executing a block of statement repeatedly until a condition is satisfied.

- ↳ ENTRY CONTROLLED : for, while
- ↳ EXIT CONTROLLED : do-while.

### 14.1) for Loop

```
for (INITIALIZATION; CONDITION; MODIFICATION)
```

// STATEMENTS

↳

### 14.2) while Loop

```
while (CONDITION)
```

{

// STATEMENT

↳

### 14.3) do-while loop

do

{

// STATEMENT

↳ while (CONDITION);

## 15.0 Jumps (Loops) (Control Flow)

↳ Jumps are used to control the flow of loop.

↳ break

↳ continue

15.1) continue

↳ used to skip to next iteration of loop.

15.2) break

↳ used to terminate the current loop.

Que

Calculate sum of first n natural.

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

```
    sum += i;
```

}

```
cout << sum;
```

Here the sum must be initialized to 0 and n must be declared first.

Que

Print table of a number

```
for (int i=0; i<=10; i++) {
```

```
    cout << "n << " * " << i << " = " << n*i;
```

}

Que

Add numbers till +ve are encountered.

```
int sum () {
```

```
    int n=0;
```

```
    int sum=0;
```

```
    while (n >= 0) {
```

```
        sum += n;
```

```
        cin >> n;
```

}

```
    return sum;
```

}

Ques Print odd till n

```
#include <iostream>
using namespace std;
void odd ( int n ) {
    for ( int i = 1; i <= n; i++ ) {
        if ( i % 2 == 0 ) {
            continue;
        }
    }
}
```

Ques Check prime number

```
bool prime_check ( int n ) {
    for ( int i = 2; i <= sqrt ( n ); i++ ) {
        if ( n % i == 0 ) {
            return false;
        }
    }
}
```

math.h needs to be included  
for the function to work

```
return true;
```

```
}
```

## 16.0 Switch-Case

↳ Substitute for long if that compare a variable to multiple values.

↳ Switch is a multiway branch statements which provides an easy way to dispatch execution to different part of code, based on value.

→ No duplicate case allowed.

→ Optional break & default statement.

→ Two cases can't be same.

→ Constant expression must be used instead of variables in case.

Some interesting programs about switch-case,

16.1) Using var as case,

```
const int i = 0;
```

```
switch (in)
```

```
{
```

```
case i:
```

→ Here case will

```
//STATEMENT
```

work as it is

```
break;
```

provided constant

value.

```
...
```

```
}
```

16.2) Find largest b/w two no using switch case

```
switch (n1 < n2)
```

```
{
```

```
case 0:
```

→ We use case 0 to

```
//STATEMENT
```

denote false case

```
break;
```

```
default:
```

→ We use default because

```
//STATEMENT everything except 0
```

break; is true.

```
}
```

Ques

Simple Calculator

```
int calc (int n1, int n2, char op) {
```

```
switch (op) {
```

```
case '+':
```

```
return n1 + n2;
```

```
case '-':
```

```
return n1 - n2;
```

```
case '*':
```

```
return n1 * n2;
```

```
case '/':
```

```
return n1 / n2;
```

## 17.0 OPERATORS

### 17.1) Arithmetic Operators

↳ +, -, \*, /, %, ++, --

Performs some arithmetic operations.

### 17.2) Relational Operators

↳ ==, !=, >, <, >=, <=

Define relation b/w entities, result in true/false.

### 17.3) Logical Operators

↳ &, ||, !

Connect multiple expressions / conditions.

### 17.4) Bitwise Operators

↳ &, |, ^, ~, <<, >>

Operate on bits & perform bit-by-bit operation.

Postfix → Unary → Multiplicative → Additive → Shift →  
 Relational → Equality → Bitwise AND → Bitwise XOR → Bitwise OR →  
 logical AND → logical OR → Ternary → Assignment

Que Reverse a number

```
int reverse (int num)
```

{

    int revNum = 0;

    for (; num > 0; num /= 10)

    {

        revNum \*= 10;

        revNum += num % 10;

    }

    return revNum;

}

Que

Armstrong Number Check  
bool isArmstrong ( int num )

{

int count = 0;

for ( int temp = num; temp > 0; temp /= 10 )

{

count ++;

}

int arm = 0;

for ( int temp = num; temp > 0; temp /= 10 )

{

arm += pow ( temp % 10, count );

}

return num == arm;

}

math.h needs to be imported.

18.0

## FUNCTIONS

↳ Function is a block of code that perform a specific task.

→ Easy maintenance, more readability, easy to understand.

returnType functionName ( param1, param2, ... )

{

// Function Body

}

→ returnType: Datatype of var that function returns.

→ functionName: Unique name of function.

→ parameters: The values that function accept, along with datatype.

Note: It is recommended to declare a function before it is being used.

Que Calculate factorial

```
int factorial (int num)
{
```

```
    int fact = 1;
```

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

```
        fact *= i;
```

```
}
```

```
    if (n >= 0) <
```

```
        return fact;
```

```
    } else {
```

```
        return -1;
```

```
}
```

```
y
```

Que Print fibonacci

```
void fibonacci (int n1, int n2), int n)
```

```
{
```

```
cout << n1 << " " << n2 << " ",
```

```
for (int i=1; i<n-1; i++)
{
```

```
    int temp = n1;
```

```
    n1 = n2;
```

```
    n2 = n2 + temp;
```

```
    cout << n2 << " ",
```

```
y
```

```
cout << endl;
```

```
y
```

Que

Calculate sum of first n naturals

int sumOfNaturals (int n)

{

    int sum = (n) \* (n - 1) / 2;

    return sum;

}

Que

Check pythagorean triplet

bool calculatePythagorean (int s1, int s2, int s3)

{

    return (s1 \* s1 == s2 \* s2 + s3 \* s3);

}

bool checkPythagorean (int s1, int s2, int s3)

{

    int max\_side = max (s1, max (s2, s3));

    bool isPythagorean = false;

    if (max\_side == s1)

        isPythagorean = calculatePythagorean (s1, s2, s3);

    else if (max\_side == s2)

        isPythagorean = calculatePythagorean (s2, s3, s1);

    else

        isPythagorean = calculatePythagorean (s3, s1, s2);

    return isPythagorean;

}

Here, checkPythagorean () function will be called  
which in-turn will call calculatePythagorean ()  
function.

Que Binary to Decimal

```
int binaryToDecimal() { int binary; }
```

```
int decimal = 0;
```

```
for (int i=0; binary>0; binary/=10, i++)
```

```
if (binary % 10)
```

```
decimal += pow(2, i); Needs to import  
math.h
```

```
y
```

```
return decimal;
```

```
y
```

Que Octal to Decimal

```
int octalToDecimal(int octal);
```

```
y
```

```
int decimal = 0;
```

```
for (int i=0; octal>0; octal/=10, i++)
```

```
decimal += octal % 10 * pow(8, i);
```

```
y
```

```
return decimal;
```

```
y
```

Que Hexadecimal to Decimal

```
int hexadecimalToDecimal(string hexadecimal);
```

```
y
```

```
int decimalans = 0;
```

```
int x = 1;
```

```
for (int i = hexadecimal.size()-1; i>=0; i--)
```

```

if (hexadecimal[i] >= '0' && hexadecimal[i] <= '9')
    ans += x * (hexadecimal[i] - '0');
else if (hexadecimal[i] >= 'A' && hexadecimal[i] <= 'F')
    ans += x * (hexadecimal[i] - 'A' + 10);
x *= 16;
return ans;
}

```

Que Decimal to Binary

```
int decimalToBinary (int decimal)
```

```

int binary = 0;
int x = 1;
while (x <= decimal)
{
    x *= 2
}
```

```

    x /= 2;
    while (x > 0)
    {

```

```

        int last = decimal / x;
        decimal -= last * x;
        x /= 2;
        binary *= 10;
        binary += last;
    }
```

```
    return binary;
```

Que Decimal to Octal

int decimalToOctal (int decimal)

{

    int octal = 0;

    int x = 1;

    while (x <= decimal)

    {

        x \* = 8;

    }

    x /= 8;

    while (x > 0)

    {

        int last = decimal / x;

        decimal -= last \* x;

        x /= 8;

        octal \* = 10;

        octal += last;

    }

    return octal;

}

Que Decimal to Hexadecimal

string decimalToHexadecimal (int decimal)

{

    string hexadecimal = " ";

    int x = 1;

    while (x <= decimal)

    {

        x \* = 16;

    }

    x /= 16;

while ( $x > 0$ )

{

    int last = decimal /  $x$ ;

    decimal -= last \*  $x$ ;

$x /= 16$ ;

    if (last < 10)

    {

        hexadecimal += to\_string (last);

    }

    else

    {

        char ch = 'A' + last - 10;

        hexadecimal += ch;

    }

}

return hexadecimal;

}

## 19.0 Space & Time Complexity

### 19.1) Time Complexity

↳ Amount of time taken by algorithm to run as function of length of input

$O \rightarrow O$ -notation  $\rightarrow$  Worst Case

$\Omega \rightarrow \Omega$ -notation  $\rightarrow$  Best Case

$\Theta \rightarrow \Theta$ -notation  $\rightarrow$  Average Case.

$O(n^n) > O(n!) > O(n^3) > O(n^2) > O(n \log n) > O(n \log(\log(\log n)))$   
 $> O(n) > O(\sqrt[n]{n}) > O(\log n) > O(1)$

## 19.2) Space Complexity

↳ Quantifies the amount of ~~time~~ <sup>Space</sup> taken by prog. to run as function of length of input. Directly prop. to largest mem. acquired during the running time.

Ques

Add binary numbers,

```
int addBinary (int n1, int n2) {
```

```
    int ans=0, carry=0;
```

```
    while (n1>0 & n2>0) {
```

```
        if (n1%2==0 && n2%2==0) {
```

```
            ans = ans*10 + carry;
```

```
            carry = 0;
```

}

```
        else if ((n1%2==0 && n2%2==1) || (n1%2==1 && n2%2==0)) {
```

}

```
        if (carry == 1) {
```

```
            ans = ans*10 + 1;
```

```
            carry = 0;
```

}

```
        else {
```

```
            ans = ans*10 + 1;
```

```
            carry = 0;
```

}

```
    else {
```

```
        ans = ans*10 + carry;
```

```
        carry = 1;
```

}

```
    num1 /= 10;
```

```
    num2 /= 10;
```

}

```

while (n1 > 0) {
    if (carry = 1) {
        if (n1 % 2 == 1)
            ans = ans * 10 + 0;
        carry = 1;
    }
}

```

```

else {
    ans = ans * 10 + 1;
    carry = 0
}

```

```

} else {
    ans = ans * 10 + (n1 % 2);
}

```

```

num1 /= 10;

```

```

while (n2 > 0) {
    if (carry == 1) {
        if (n2 % 2 == 1)
            ans = ans * 10 + 0;
        carry = 1;
    }
}

```

```

ans = ans * 10 + 1;
carry = 0;

```

```

} else {
    ans = ans * 10 + (n2 % 2);
}

```

```

num2 /= 10;

```

```

if (carry == 1)
    ans = ans * 10 + 1;

```

```

return reverse(ans);

```

## 2.0.0 ARRAYS IN C++

↳ Array is a collection of similar data items stored at contiguous memory locations whose elements can be accessed randomly using indices of an array.

(In C++, Array can be of primitive as well as of derived datatype).

Array is used to represent many instances of variable in a single variable.

## 2.0.1 Declaring Array in C++

↳ `int arr[3];`

↳ Will create an array of size 3 with garbage elements.

↳ `int arr[Size] = {v1, v2, ..., vsize};`

↳ Will create an array of the desired size & will initialize the elements.

↳ `int arr[Size] = {};`

↳ Will create an array of desired size & will initialize all elements with zero.

↳ `int arr[Size] = {Val};`

↳ Will initialize all the elements of array with specified value.

## 2.0.2 ADVANTAGES (Array)

↳ Random access using index.

↳ Single array can access multiple elements.

↳ Easy access of elements.

↳ Traversal through a single loop.

20.3

### DISADVANTAGES (Array)

- ↳ Allows fixed no. of elements to be decided at time of declaration.
- ↳ Insertion & deletion can be costly as elements are needed to be maintained in accordance with new memory allocation.

20.4

### FACTS ABOUT ARRAY

- ↳ Array are accessed using indexes and array is pointer to first element.
- ↳ Index Out Of Bound performs compilation error

20.5

### Array v/s Pointers

- ↳ Array and pointers are two different things, but arrays use the concept of pointer as address of first element is accessed using pointers.

### Vector (Short Note)

- ↳ Vector is an C++ STL class which represent an array.
- ↳ Vectors supports dynamic size as well

Array has better cache locality.

20.6

### Applications of Array

- ↳ Store data elements of same datatype
- ↳ Can be used for CPU scheduling
- ↳ Used to implement other DS.

## Q1. Passing 2D ARRAY As An ARGUMENT To FUNCTION

↳ C++ doesn't allow to passing an array as an argument to a function.

### Passing 1D- Array.

void fun ( int arr[] ) ] → Accepting argument  
 { }

y

Passing Argument → [ int arr[3] = { 0, 1, 2 }  
 fun (arr);

int size = sizeof (arr) / sizeof (arr[0]);

### Passing 2D- Array.

void fun ( int arr[][size] )  
 { }

y

### Que Min Max from Array.

int max = INT\_MIN; ] → limits needs to  
 int min = INT\_MAX; be imported for

for ( int i=0; i<n; i++ ) using limits.  
 { }

if (max < arr[i])

    max = arr[i];

if (min > arr[i])

    min = arr[i];

y

cout << min << " " << max << endl;

Que

Running sum of Array.

```
for (int i=1; i<n; i++)
```

```
{  
    arr[i] = arr[i] + arr[i-1];  
}
```

Que

Kid with greatest candies

```
int max = INT_MIN;
```

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

```
{  
    cin >> arr[i];
```

```
    if (arr[i] > max)
```

```
{  
    max = arr[i];  
}
```

```
}  
}
```

```
int extra;
```

```
cin >> extra;
```

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

```
{  
    if (arr[i] + extra > max)
```

```
        cout << "true";
```

```
    else
```

```
        cout << "false";  
}
```

```
}  
}
```

Que

Linear Search.

Time Complexity  $\rightarrow O(n)$

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

```
{  
    if (arr[i] == search) {
```

```
        cout << "Element found at index " << i << endl;  
        return 0;
```

```
}  
}
```

```
cout << "Element not found";
```

Que

Binary Search

Time Complexity  $\rightarrow O(\log_2 n)$

```
int binarySearch(int arr[], int n, int key)
{
```

```
    int start = 0;
```

```
    int end = n - 1;
```

```
    while (start < end)
```

```
{
```

```
    if (arr[start] == key)
```

```
        return start;
```

```
    else if (arr[end] == key)
```

```
        return end;
```

```
    int mid = (start + end) / 2;
```

```
    if (arr[mid] == key)
```

```
        return mid;
```

```
    else if (arr[mid] > key)
```

```
        end = mid - 1;
```

```
    else
```

```
        start = mid + 1;
```

```
}
```

```
return -1;
```

```
}
```

## 22.0 SEARCHING IN ARRAY

↳ It is crucial to look for an element in an array as we perform specific operations if element is present.

### 22.1 LINEAR Search $O(n)$

↳ We will iterate through each elements and will check if element is present.

### 22.2 BINARY Search $O(\log_2 n)$

↳ In a sorted array, we look for element by reducing the elements to half.

## 23.0 SORTING AN ARRAY

↳ As linear search is very costly and binary search requires sorted array to work upon, so sorting is an important concept in DSA.

### 23.1) Selection Sort $O(n^2)$

↳ Find the minimum element in unsorted array and swap it with element at the beginning.

#### Ques Selection Sort

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

Inner loop selects the min element in unsorted array and places them in increasing order.

### 23.2) Bubble Sort $O(n^2)$

↳ Repeatedly swap two adjacent elements if they are in wrong order.

#### Ques Bubble Sort

```
int counter = 0;
while( counter < n-1 ) {
    for( int i=0; i<n-counter; i++ ) {
        if( arr[i] > arr[i+1] ) {
```

swap (arr[i], arr[i+1]);

}

}

counter++;

}

If  $\text{arr}[i] > \text{arr}[i+1]$ , swap them. To place the elements in respective position, repeat the operation  $n-1$  times.

### 23.3) Insertion Sort $O(n^2)$

↳ Insert an element from unsorted array to its correct position in sorted array.

#### Ques Insertion Sort

```
for (int i=0; i<n; i++) {
    int current = arr[i];
    int j = i-1;
    while (arr[j]>current && j>=0) {
        arr[j+1] = arr[j];
        j--;
    }
}
```

$\text{arr}[j+1] = \text{current};$

}

Take an element from the unsorted array, place it in its corresponding ~~the~~ position in the sorted array part & shift the elements accordingly.