

- more important note and Week Lecture / homework / recording
- Data Structure
- Heading
- Subheading or important point

Every week from new page

github.com/  
Ujjwal2327/  
DSA-SUPREME

Ujjwal  
Maheshwari

linkedin.com.in/  
ujjwal-maheshwari-  
164886202

<https://www.linkedin.com/in/ujjwal-maheshwari-164886202/>

<https://github.com/Ujjwal2327/DSA-SUPREME>

Bhaiya Bday  
20 June 1998  
7:15 AM

# IMPORTANT POINTS

- Focus on work on skill and build networks those work in companies and can talk to HR for you
- Internship- do if you want to learn or convert it as PPO
- Rough sols. and dry run are very important
- Do documentation to promote and mail everything you discuss with HR or team
- Think twice, Code once
- To clarify any code you are confused, use cout statements every where to know what is going on in the code
- Code all approaches you can think of and can find & understand from google
- Revise all incorrect & skipped questions in quizes regularly
- Watch sol. only after attempting the question
- Interviewer will ask Time & Space complexity after every sol. you give

→ 2 websites

    └→ cppreference  
    └→ cplusplus.com

→ Think on paper

→ Write readable codes

# IMPORTANT C++ NOTES

- Making global variable is BAD PRACTICE
- To increase range of int / long long , you can use unsigned int / long long
- % → heavy operator
  - ↳ so try to use it less
  - Use bitwise operators instead of this if possible
- arr[n] → BAD PRACTICE
  - arr[100000] is better than arr[n]
- to find min., start ans from INT\_MAX
- to find max, start ans from INT\_MIN
- n & 1 gives rightmost bit of n
- xor → cancels out same elements
  - $0 \wedge 1 = 1$
  - $0 \wedge 0 = 0$
- In ASCII → 'O' → 48  
                  'A' → 65  
                  'a' → 97

## → Search Space

↳ find range of search space (start & end) in ques of. Binary Search Questions

## → In binary search questions

while ( $s \leq e$ ) {

left  
search  
} }  
 $s = mid + 1;$   
 $e = mid - 1;$

right  
search

while ( $s < e$ ) {

left  
search  
} }  
 $s = mid;$  → right search  
 $e = mid - 1;$

OR

left  
search  
} }  
 $s = mid + 1$  → right  
search  
 $e = mid;$

## → Types of ques in binary search

→ 1<sup>st</sup> type → classic questions

→ 2<sup>nd</sup> type → find in search space (range)

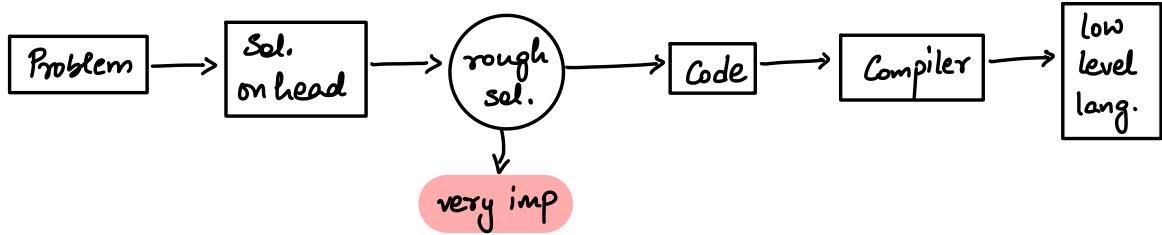
↳ predicate function

→ 3<sup>rd</sup> type → observation in index

Thought process to solve a problem-

W1-L1

- Understand a problem
- input values
- find approach



Algorithm - Sequence of steps

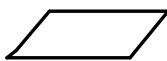
Flowchart - Graphical representation of algo

Components -



terminator

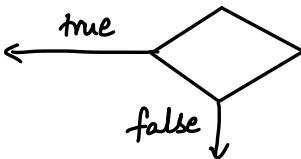
for start / end



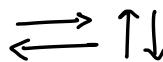
for input /output read /write



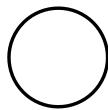
computation / process / declaration



decision making block  
takes condition



flow



Connector  
takes function

Pseudo Code - Generic way of writing algo

Dry Run → Very Important to understand any topic

W1-L2

IDE - Replit, VS-Code

```
# include <iostream>           → preheader file contains implementation of identifiers
using namespace std;
int main () {
    cout << "Namaste Bharat";
}
```

region where scope of identifier is defined

used to point on console/standard display

→ using standard namespace implementation of cout choosing from multiple types of namespace

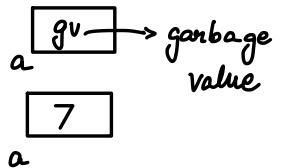
→ to end any statement

→ string

cout << endl; → for next line

cout << '\n';

int a;  $\longrightarrow$  a is an integer  
 cin >> a;  $\longrightarrow$  input a from user  
 ex - 7



## Variables

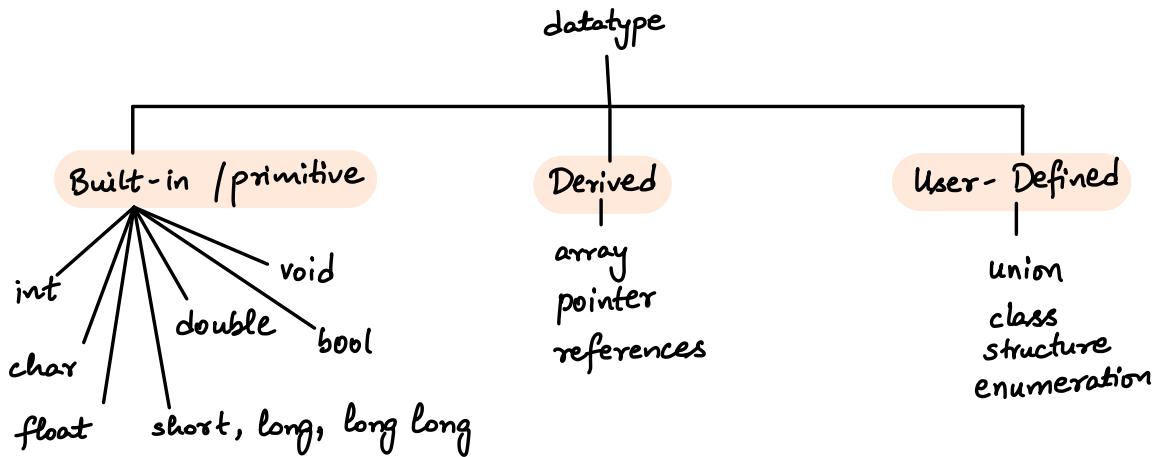
named memory location

int  $\downarrow$   
 datatype  
 a = 5;  
 ↓  
 variable  
 name

value

## Datatypes

type of data



int - 4 byte - 32 bits in memory

$\longrightarrow$   $-2^{31}$  to  $2^{31}-1$  in signed int  
 $\longrightarrow$  0 to  $2^{32}-1$  in unsigned int

char - 1 byte - 8 bits in memory

$\longleftarrow 2^8$  different chars.

## ASCII

↳ char maps with numerical ASCII value

char  $\leftrightarrow$  ASCII value  $\rightarrow$  store in memory

bool  $\rightarrow$  1 byte  $\rightarrow$  8 bits

true - 1

false - 0

↳ because minimum addressable memory is  
1 byte

We cannot address 1 bit in memory

float  $\rightarrow$  4 byte  $\rightarrow$  32 bits

double  $\rightarrow$  8 byte  $\rightarrow$  64 bits

long long  $\rightarrow$  8 byte  $\rightarrow$  64 bits

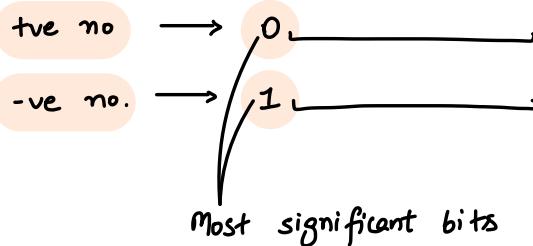
short  $\rightarrow$  2 byte  $\rightarrow$  16 bits

long  $\rightarrow$  4 byte  $\rightarrow$  32 bits

## How data is stored

int a=5;

↳ 32 bits      0...00101  
                  29 bits



### How -ve number is stored in memory

In 2's complement form

→ 1's complement + 1

→ reverse all bits

int a = -7;

7 → 0.....00111 } 32 bits

ignore -ve sign  
find binary equivalent

1's (7) → 1.....11000

find 2's complement

2's (7) → 1.....11001

→ this is how -7 will be stored in memory

### How to read -ve no. present in memory

→ take 2's complement

1....11001

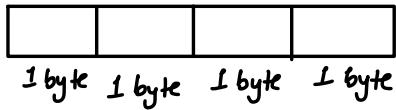
→ 1's complement → 0...00110

2's complement → 0...00111

→ + 7

-7

## Interesting problem



how computer know these are 4 chars or a single integer

↳ Using datatype

↳ tell 2 things

- ↳ type of data used
- ↳ space used in memory

## Signed vs Unsigned

↓  
↳ 0, +ve  
+ve, -ve, 0

↳ by default

int - 4 byte - 32 bits in memory

↳ total no. of combinations -  $2^{32}$

signed int

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

unsigned int

0 to  $2^{32}-1$

} range

(1) 0...0

011...1

0.....0

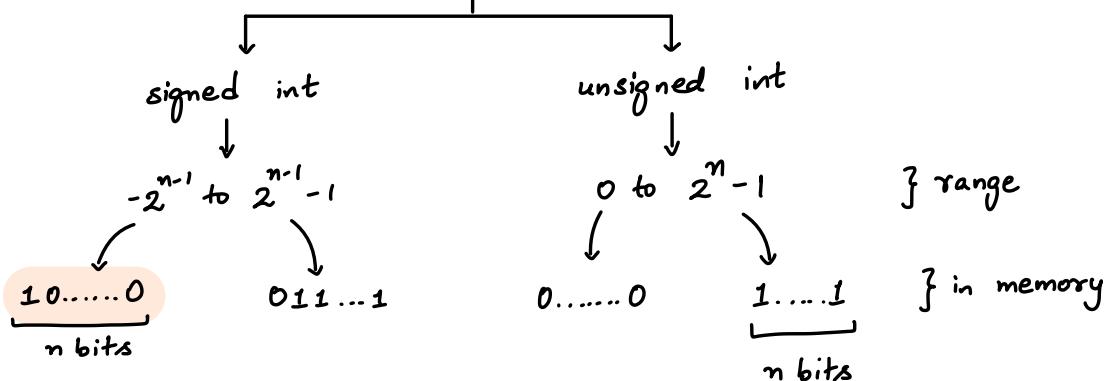
1....1

$2^{15} \rightarrow \underline{10...0} \rightarrow -2^{31}$

} in memory

## General Formula

$n$  bits in memory  
↳ total no. of combinations -  $2^n$



## Typecasting

↳ convert one type of data to another

### implicit typecasting

ex- char ch = 97;  
cout << ch; → (a)

### explicit typecasting

ex- char ch = (char) 97;  
cout << ch; → (a)

overflow ex- char ch = 9999;  
cout << ch;

9999 → 100 111 0000 1111  
binary conversion stores only last 8 bits

so ch stores 00001111 in memory  
 ↓  
 ↓  
 acc. to ASCII table

## Operators -

### Arithmetic Operator

→ +, -, \*, /, %

int op int → int

float op int } float

int op float }

float op float }

bool op bool → int

bool act as 0 or 1

double op int } double

int op double }

double op double }

float op double }

double op float }

3 → int    → by default → cout << sizeof(3.0);  
 3.0 → float / double    ↓  
 3.0 → float / double    ⑧  
 not int

### Relational Operator

a op b

>, <, >=, <=, !=, ==

Output - 0 or 1

false      false  
 true      true

these are different things

## Assignment Operators

=

## Logical Operators

↳ when you have multiple conditions

$a \& \& b$  → and → true if both are true

$a || b$  → or → true if any one is true

$! a$  → not → negate the result

Output - 0 or 1  
false ↕ true

(cond1  $\&\&$  cond2  $\&\&$  cond3)

if cond1 is false

compiler will not check further  
as ans will already false

(cond1  $||$  cond2  $||$  cond3)

if cond1 is true

compiler will not check further  
as ans will already true

## Conditions

if (cond.){  
    execute  
}

if

if (cond){  
    execute 1  
}

else {  
    execute 2  
}

if - else

**W1-L3**

if (cond1)  
    execute 1  
else if (cond2)  
    execute 2

if - else if

```

if (cond 1)
    execute 1
else if (cond 2)
    execute 2
else if (... )
else
    execute n

```

**if - else if - else**

```

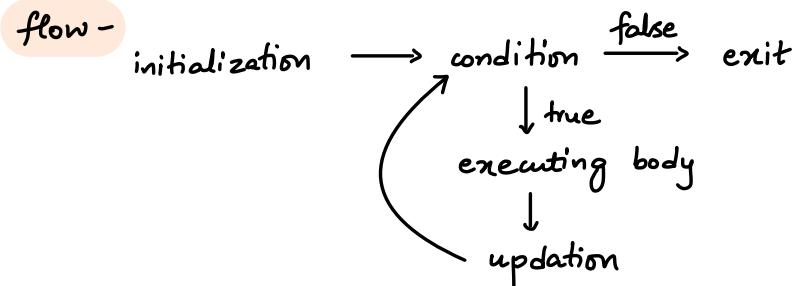
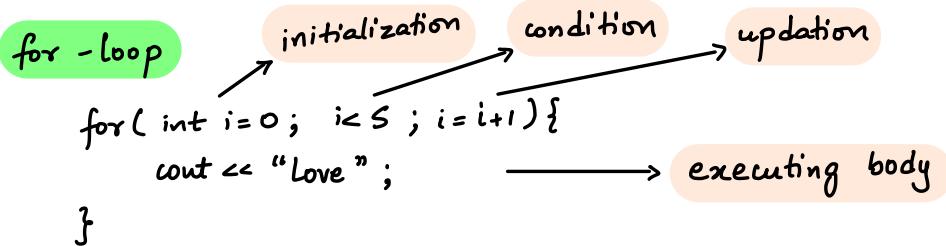
if (cond 1)
    execute 1
else {
    if (...) { }
    else (...) { }
}

```

**nested if - else**

## Loops

↳ to do something repeatedly



initialization  
condition  
updation } none is mandatory  
one or multiple i,c,u can be added  
multiple c → i>5, i<10; → i>5 & i<10

## patterns -

generally 2 loops → outer loop() {  
  → for rows  
  inner loop() {  
  → for cols  
  }  
  }  
  cout << endl ;  
  }

→ a op = b → a = a op b

op → +, -, \*, /, , /

## cin in if()

```
int num;  
if (cin >> num) {  
    cout << "hello";  
}  
else {  
    cout << "hi";  
}
```

it will not give error

output -  
              hello

for all values of num  
  ↓

0, true, -ve

## cout in if()

```
int num = 0;  
if (cout << num << endl) {  
    cout << "hello";  
}  
else {  
    cout << "hi";  
}
```

it will not give error

output -  
              0  
              hello

for all values of num  
  ↓

0, true, -ve

HLL - High level language

↳ human readable and user friendly

W1-L4

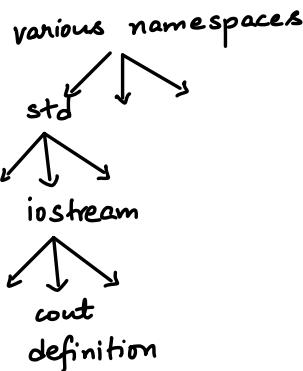
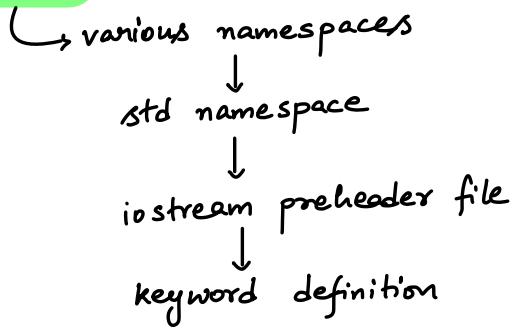
C++, C - Middle Level language

namespace → to avoid collision



multiple definitions of a single keyword

hierarchy



float f = 2.0 + 100;

cout << f ;      → output -

102 or 102.0  
compiler dependent

float f = 2.7;

int n = 157;

int diff = n-f;

cout << diff ;

output -

154

explanation -

$$n-f = 157 - 2.7 = 154.3$$

int diff = n-f

diff = 154

## ternary operator -

W1-HW

↳ syntax

variable = (condition) ? expression2 : expression3

(condition)? variable = expression2 : variable = expression3

by default -

cout << sizeof(2.3); → 8  
                          |  
                          → float

cout << sizeof(a); → 4 → int  
                          |  
                          → -( $2^{31}$ -1) to  $2^{31}-1$

cout << sizeof(- $2^{31}$ ) → 8  
                          |  
                          → long long

↳ how to think

→ finding formula for rows and cols

$n=5$

row	stars
0	0
1	0
2	1
3	2
4	3

→ formula -

0 to  $< n-1$

$n-1$

-1

0

1

2

3

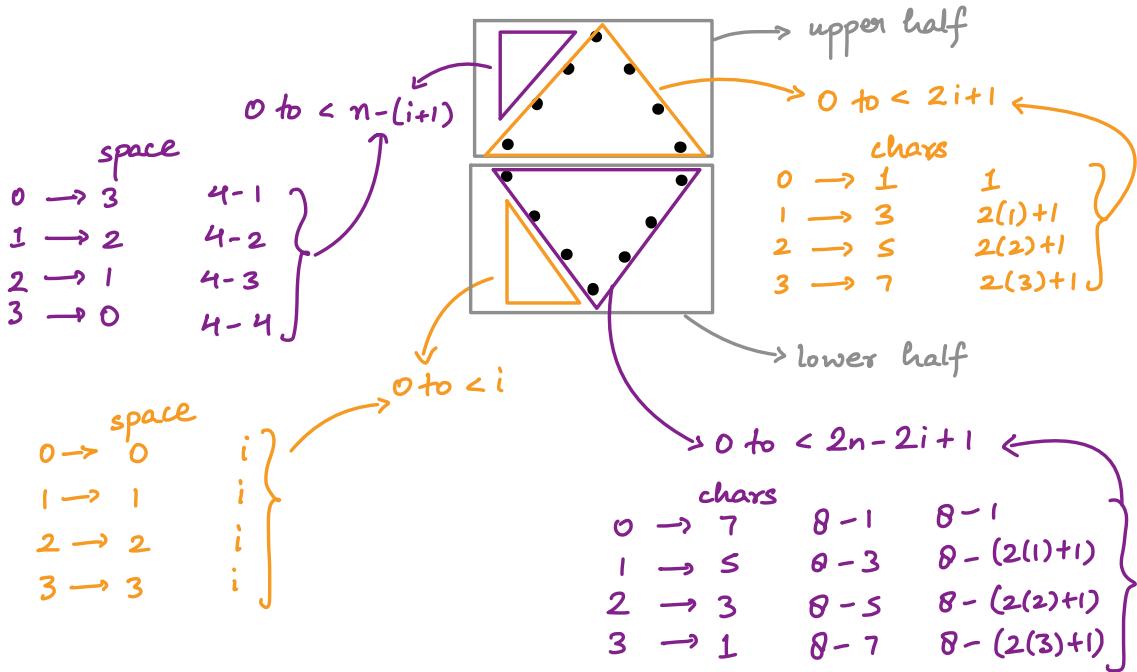
no. of times loop runs

as condition fails ( $0 < -1$ )

→ to do anything  $n$  times

↳ `for(i=0 ; i<n ; i++) {}`

→ break the complex patterns



## Bitwise Operators

W2-L2

→ use on bit level

And  $(a \& b)$  1 if both bits are 1

Or  $(a | b)$  1 if any or both bits are 1

not  $(\sim a)$  negate the result

nor  $(a ^ b)$  same values  $\rightarrow 0$   
diff. values  $\rightarrow 1$

$\sim 5$

$\sim 5 \rightarrow 1 \dots 1010$

$\sim 5 \rightarrow 1 \dots 1010$

↳ how compiler read this

↳ 2's complement

$0 \dots 0101 \rightarrow 1$ 's complement

$0 \dots 0110 \rightarrow 2$ 's complement

-6

So  $\sim 5 = -6$

## Left and right shift operators

<<

shift all bits to left

\* by 2 (not in every case)

↳ if MSB is 1 and

2nd MSB is 0

>>

shift all bits to right

/ by 2 (not in every case)

↳ in -1

$a = a \ll b$  a left shifts, b times  $\rightarrow$  result  $\rightarrow a \times 2^b$

$a = a \gg b$  a right shifts, b times  $\rightarrow$  result  $\rightarrow \frac{a}{2^b}$   
b cant be -ve

↳ in case of -ve

$a = 5;$

↳ gives 8v

$a = a \ll 1;$        $a = 10$

$a = 5;$

$a = a \ll 2;$        $a = 20$

in left shift  $\rightarrow$  filled with 0

in right shift  $\rightarrow$  filled with

0 and 1  
in +ve no.  
in -ve no.

right shift in -ve number

-ve no. in memory  $\rightarrow 1 \dots$

↓ right shift

1 1 ...

signed bit is used to fill  
the vacant bit

ex-

$s \rightarrow 0 \dots 0 101$

$-s \rightarrow 1 \dots 1 011$

$-s \gg 1 \rightarrow 1 \dots 1 01 \rightarrow -3$

$-1 \gg 1 \rightarrow -1$

left shift in number where MSB is 1

and 2nd MSB is 0

no.  $\rightarrow 1 0 \dots \rightarrow$  -ve no.

left shift  $\rightarrow 0 \dots$

$\rightarrow$  +ve no.

## Pre- Post → Increment / Decrement Operator

### pre- increment

↳  $++a$

↳ first increment by 1, then use

### post - increment

↳  $a++$

↳ first use then increment by 1

### pre- decrement

↳  $--a$

↳ first decrement by 1, then use

### post - decrement

↳  $a--$

↳ first use then decrement by 1

```
int a = 5;  
cout << (++a) * (++a);
```

output -

49

↳ due to operator precedence

→ links.txt in repo

## break and continue

### break

↳ exit from that loop

### continue

↳ skip that iteration

## Variable Scoping -

```
int g= 25;           -----> global variable
int main(){
    int a;          -----> declaration
    int b= 5;        -----> initialization
    b = 10;          -----> updation
    //int b= 15;      -----> redefinition is not allowed
    int c= 7;
    g= 30;
    cout << g;       -----> 30
    if (true){
        int b= 15;
        cout << b;     -----> 15
        cout << c;     -----> 7
        g= 50;
        cout << g;     -----> 50
    }
    cout << a;       -----> gv
    cout << b;       -----> 10
    cout << c;       -----> 7
    cout << g;       -----> 50
}
```

Making global variable is very BAD PRACTICE

## Operator Precedence

- order of priority of operator
- no need to remember
- use brackets properly

## Switch Case

```
switch (expression) {
```

```
    case value1 :
```

executing body 1

```
    break ;
```

```
    case value2 :
```

executing body 2

```
    break ;
```

:

```
    case value n :
```

executing body n

```
    break ;
```

```
default :
```

executing body

```
}
```

without break

→ all below executing body will also execute

→ continue cannot be used in switch case

→ can only use in loops

can also have  
nested switch  
case

not  
mandatory

## Function -

- program linked with well defined task
- why
  - reusable
  - readable
- without
  - bulky
  - lengthy
  - buggy if mistake in any place

## syntax -

```
return type function name ( input parameters ) {
    function executing body
}
```

void → empty / no value

```
int main() {
    return 0;
}
```

→ returns 0 to Operating System  
 → 0 is used as means of successful execution

- a cpp file cant have more than 1 main functions
- main cant have return type other than int in offline compiler

## Function Call Stack

function call  $\leftrightarrow$  function invoke

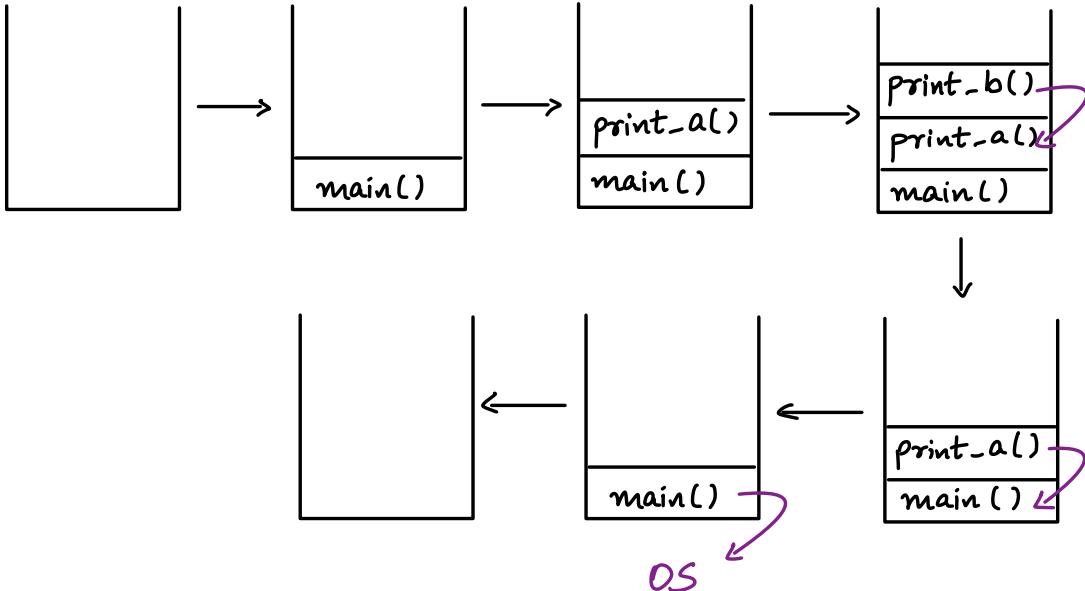
Stack

↳ Last In First Out

- ↳ tells what functions
  - ↳ which function calls which
  - ↳ local variables of function
  - ↳ return type of function

ex -

```
int main() {  
    int a=5;  
    print_a(a)  
    return 0;  
}  
  
void print_a(int a){  
    cout << a;  
    int b=3;  
    print_b(b);  
}  
  
void print_b(int b){  
    cout << b;  
}
```



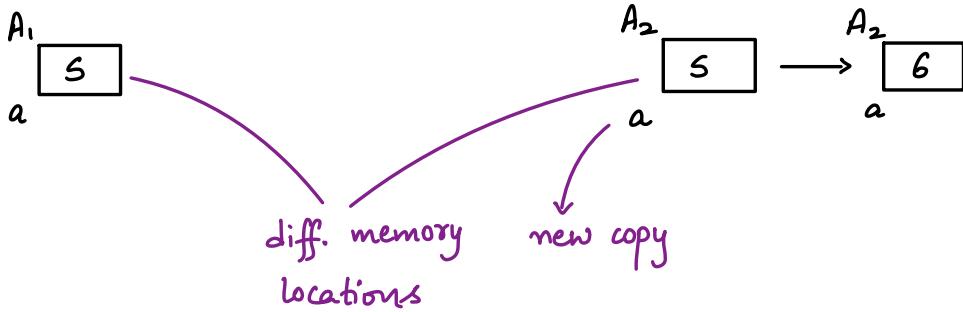
## Pass by value

↳ a copy will be created of variables

```
int main() {  
    int a=5;  
    printNumber(a);  
    cout << a;  
}
```

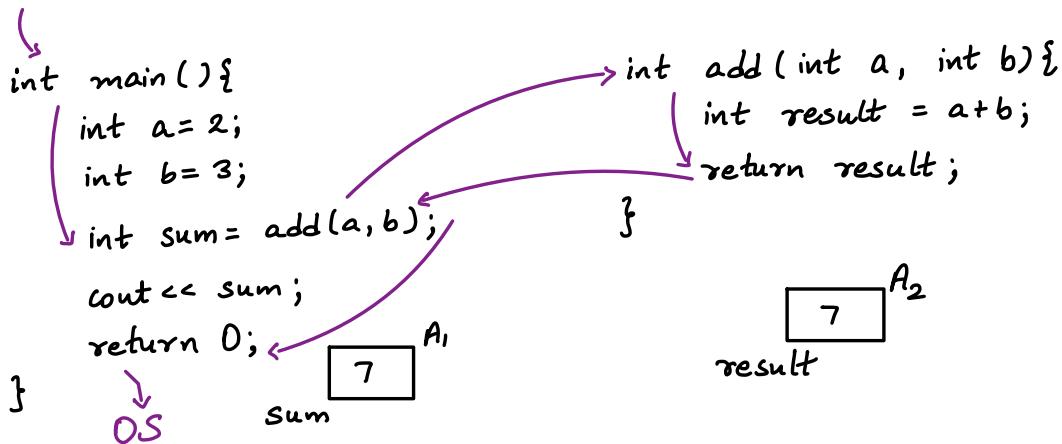
```
parameter  
void printNumber(int a){  
    cout << a;  
    a++;  
    cout << a;  
}
```

argument



## Address Of Operator &

```
int n=5;  
cout << &n;           → output -  
address of n
```



## Function Order

### Order 1

```
int add (int a, int b) {  
    return a+b;  
}  
  
int main () {  
    int a= 3;  
    int b = 5;  
    int sum= add (a,b);  
    cout << sum;  
    return 0;  
}
```

function  
declaration  
and  
definition

### Order 2

```
function declaration  
{  
int add (int a, int b);  
  
int main () {  
    int a= 3;  
    int b = 5;  
    int sum= add (a,b);  
    cout << sum;  
    return 0;  
}  
  
int add (int a, int b) {  
    return a+b;  
}
```

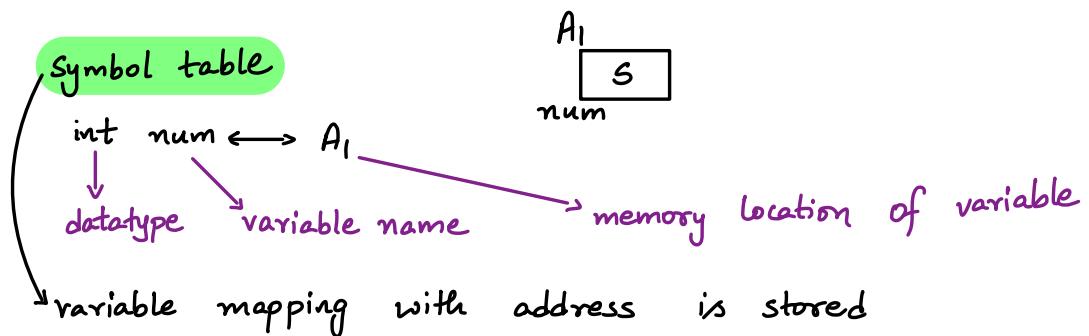
% operator → heavy operator

↳ so try to use it less

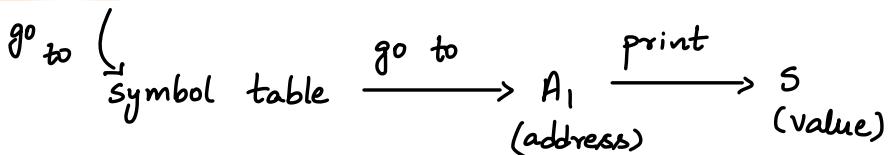
BTS

→ Behind The Scenes

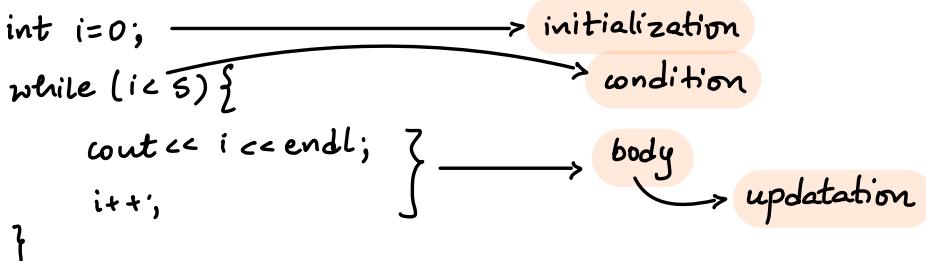
int num=5;



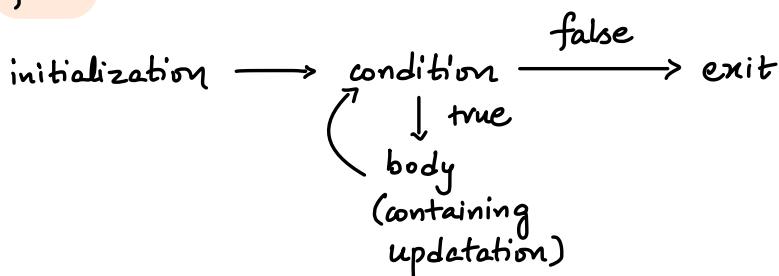
cout &lt;&lt; num;



while loop



flow



## left and right shift operators

int a = 2;

$a \ll 1;$  → no change

$\text{cout} \ll a;$  → 2

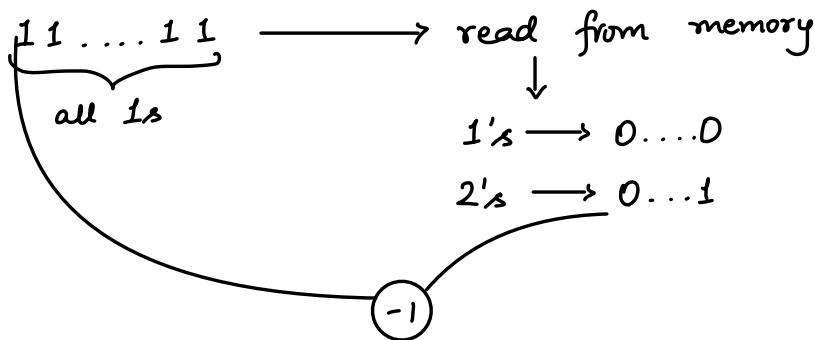
$a = a \ll 1;$  → change → left shift by 1

$\text{cout} \ll a;$  → 4

## right shift in -ve no.

↳ link in links.txt in repo

## How -1 is stored in memory -



$$\sim a = -(a+1) \quad \text{and} \quad \sim(\sim a) = a$$

ex -  $a = 5; \rightarrow 0\dots0101$

$a = \sim a; \rightarrow 1\dots1010 \rightarrow -6 \rightarrow -(5+1)$

$a = -6;$

↳  $1\dots1010$

read

-6

$a = \sim a; \rightarrow 0\dots0101$

↳ 5 →  $-(-6+1)$

## Number System

↳ method to represent numeric values using digits

### Decimal Number System

↳ base 10

↳ digits → 0 to 9

### Binary Number System

→ base 2

↳ digits → 0, 1

→ used in CPU, memory, computer

→ 0 → power off

→ 1 → power on

→ number, images, all files & folder are in binary

### Decimal to Binary

→ divide no. by 2

→ store remainder

→ repeat above steps until no. is 0 or 1

→ reverse the bits so obtained

## Binary to Decimal

- multiply each bit with its place value
  - ↳ base  $i$
- add all products
  - ↳  $2^i$

## Time & Space Complexity

W3-R

### Time Complexity

- amount of time taken by an algo as a function of length of input
- not actual time
- it defines CPU operations
- use case -
  - to make efficient programs
  - ask by interviewer after every sol. you give

### Space Complexity

- ↳ amount of space taken by an algo as a function of length of input

### Units to represent Complexity

Big O → upper bound → worst case

Theta  $\Theta$  → average case

Omega  $\Omega$  → lower bound → best case

## Big O Complexities

$O(1)$  → Constant time

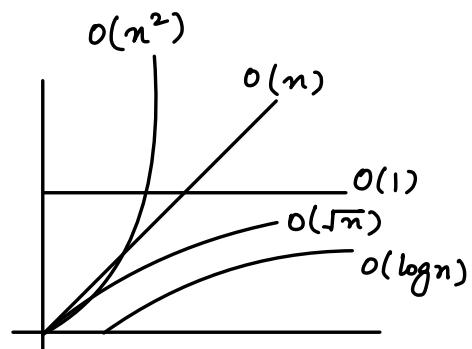
$O(n)$  → Linear time

$O(\log_2 n)$  → Logarithmic time

$O(n^2)$  → Quadratic time

$O(n^3)$  → Cubic time

nesting → multiply → { }  
in sequence → add → { }  
| → { }  
| → { }



$$f(n) = 4n^4 + \frac{n^3}{5} + \log n + n \log n \rightarrow O(n^4)$$

## Complexity Order

$$\begin{aligned} O(1) &< O(\log_2 n) &< O(\sqrt{n}) &< O(n) &< O(n \log_2 n) &< O(n^2) \\ &< O(n^3) &< O(2^n) &< O(n!) &< O(n^n) \end{aligned}$$

# ARRAY

W3-L1

- Data Structure to store similar items
  - ↳ same datatype
- Continuous memory location space
- use case
  - ↳ for multiple huge same kind of data  
`int a[30000];` → 30000 variables are ready

## continuous memory location

↳ memory wastage  
if needable memory is present but not in continuous way

`int a = 5;`

A



a

## symbol table

`int a ↔ A`

`int arr ↔ A1`

## Declaration

`int arr[5];`

$A_1 \rightarrow A_1 + 4 \rightarrow A_1 + 8 \rightarrow A_1 + 12 \rightarrow A_1 + 16$



arr

20 bytes

base address ↳ continuous space

`cout << arr ;`  $\longrightarrow$  A<sub>1</sub>

`cout << &arr ;`  $\longrightarrow$  A<sub>1</sub>

## Initialization

`int arr [7] = { 2, 4, 6, 8, 10 };`

`int arr2 [5] = { 2, 4, 6, 8, 10 };`

`int arr3 [10] = { 2, 4, 6, 8, 10 };`  $\longrightarrow$  remaining 5 will be 0

`//int arr4 [4] = { 2, 4, 6, 8, 10 };`  $\longrightarrow$  ERROR

`int arr5 [10] = { 0 };`  $\longrightarrow$  initializing all values with 0

## Making array at runtime

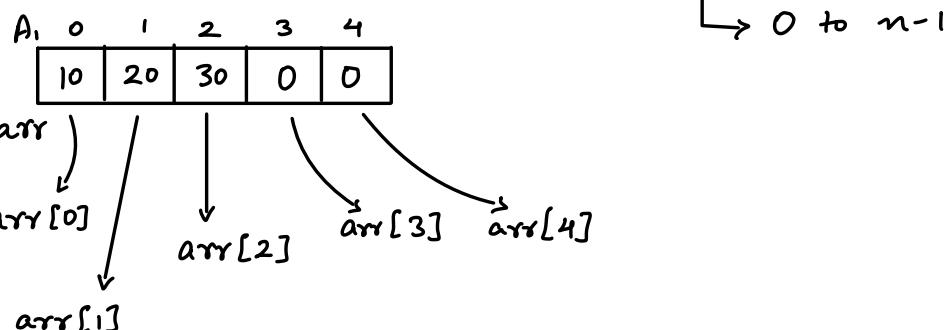
`int n;`

`cin >> n;`

`int arr[n];`  $\longrightarrow$  BAD PRACTICE

## Index and Access in memory

`int arr[5] = { 10, 20, 30 };`  $\longrightarrow$  0<sup>th</sup> based indexing



$\text{arr}[i] \longrightarrow$  value at address  $[\text{arr} + (i * 4)]$   
 thats why 0 based indexing

$A_1$  index  
 due to int (datatype size)

### taking input in array

$\hookrightarrow \text{cin} >> \text{arr}[i];$

due to internal working

### Arrays and Function

$\hookrightarrow \text{func}(\text{int arr[], int size})\{$

}

$\hookrightarrow$  pass by reference  
 $\hookrightarrow$  updation in actual array  
 $\hookrightarrow$  always pass size alongwith arr

```

int main() {
    int arr[] = {5, 6};
    int size = 2;
    func(arr, size);
    return 0;
}
  
```

```

void func(int a[], int size) {
    a[0] = a[0] + 10;
}
  
```

5	6
arr	

`sizeof( int );` → 4 → in bytes

`int arr [5];`

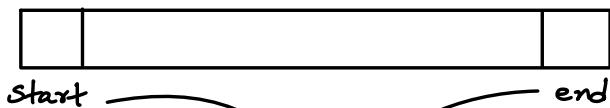
`sizeof( arr );` → 20 → in bytes

linear search in array

INT\_MIN and INT\_MAX

- to find max. , start ans with INT\_MIN
- to find min. , start ans with INT\_MAX

2 pointer approach



use of 2 variables as extreme points

To find size of array

`int arr [ ] = { 1, 2, 3, 4 };`

`int size = sizeof( arr ) / sizeof( int );`

## Vector

## W3-L2

- Data structure
  - Same as array but dynamic
    - ↳ size not fixed
  - default size → 0
  - if gets full and new items are inserted  
size gets doubled
- pass by value in functions

### Initialization

```
vector <int> arr {10, 20, 30}; → [10 | 20 | 30]  
vector <int> arr (5); → [0 | 0 | 0 | 0 | 0]  
vector <int> arr (5, -2); → [-2 | -2 | -2 | -2 | -2]  
int n; size → let n = 5  
vector <int> arr(n); → [0 | 0 | 0 | 0 | 0]  
vector <int> arr (n, 10); → [10 | 10 | 10 | 10 | 10]
```

### Insertion -

```
arr.push_back (5);
```

### Remove

```
arr.pop_back();
```

### Size -

```
arr.size();
```

→ no. of elements it stores

### declaration

```
vector <int> arr;  
→ arr.size() → 0  
→ arr.capacity() → 0
```

## Empty or Not

arr. empty ();  $\longrightarrow$  true if empty

## Capacity -

arr. capacity ();  $\longrightarrow$  \* by 2 if arr gets fully filled  
and a new element is inserted

→ no. of elements it can store

→ in initialization, capacity = size in all methods

sizeof (arr);  $\longrightarrow$  compiler dependent initially

cout << arr ;  $\longrightarrow$  give ERROR

→ Xor  $\longrightarrow$  cancels out same element

$$0 \wedge \text{ans} = \text{ans} \quad \begin{cases} 0 \wedge 1 = 1 \\ 0 \wedge 0 = 0 \end{cases}$$

## for each loop

```
for (auto val: arr){  
    cout << val << ' ';  
}
```

## 2D Arrays

W3-L3

→ use case

→ to work on multiple rows and columns

### Declaration -

`int arr[m][n];` →  $m \times n$  elements

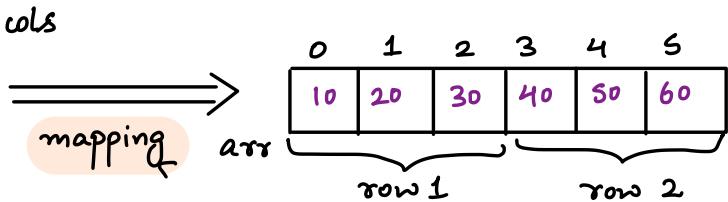
cols → 0 to  $n-1$   
rows → 0 to  $m-1$

`int arr[2][3];`

### visualize -

	cols		
0	10	20	30
1	40	50	60
rows	0	1	2

### in memory -



### Access -

`arr[i][j];`

col index  $0 \leq j < n$   
row index  $0 \leq i < m$

### Mapping -

$$\text{linear\_index} = c * i + j$$

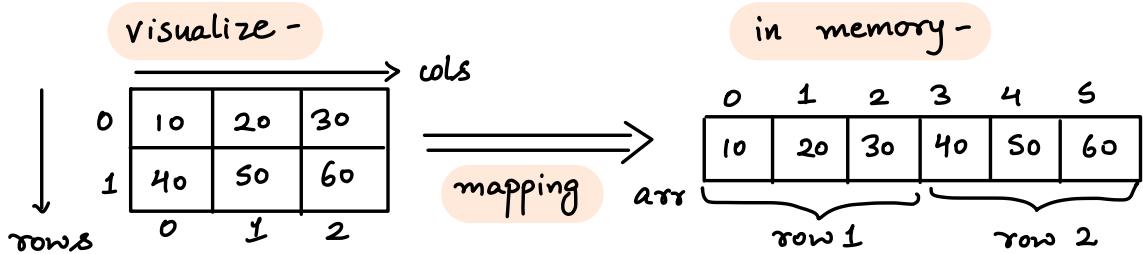
no. of cols  
row index  
col index

$$i = \text{linear\_index} / c$$

j

## Initialization -

```
int arr [2][3] = {{10, 20, 30}, {40, 50, 60}};
```



## 2D Arrays and function -

→ pass by reference

```
func ( arr [ ] [500], int rows, int cols )
```

this value and  
no. of cols  
in array  
passed in func  
should be same

cannot leave blank

why if dont know, put large value

for mapping

→ linear\_index = c \* i + j

→ So that compiler can know

## 2D Array -

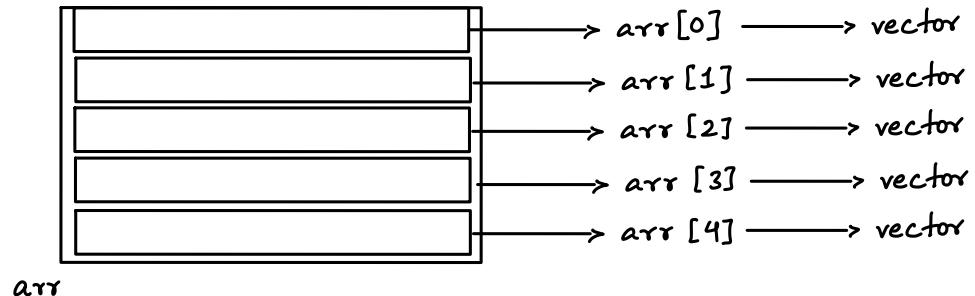
→ to make dynamically

→ vector of vectors

## 2D Vector

### Declaration -

```
vector<vector<int>> arr;
```



### Declaration -

```
vector<vector<int>> arr;  
vector<vector<int>> arr (m);
```

### Number of rows

arr.size()

### Number of cols

arr[i].size()

→ in ith row  
→ size of ith row

### Initialization

```
vector<vector<int>> arr;
```

size  
↑

```
vector<vector<int>> arr (rows, (vector<int>(cols, value)));
```

rows → no. of rows in arr

cols → no. of cols in arr

size of 1D arrays

initialization of 1D  
vectors in arr

value → initial value in all elements

of all 1D vectors

```
vector<vector<int>> arr(2, vector<int>(4, 101));
```



101	101	101	101
101	101	101	101

arr

# Searching and Sorting

W4-L1

## Searching

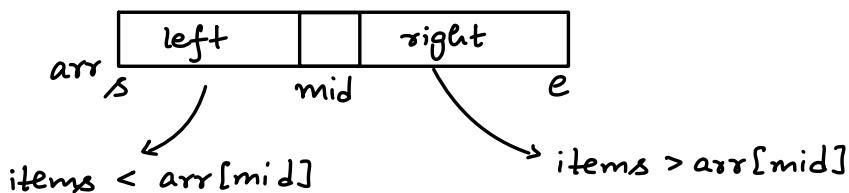
### Linear Search

```
int linearSearch ( vector <int> arr, int target ) {  
    int n = arr.size();  
    for( int i=0; i<n ; i++ ) {  
        if ( target == arr[i] )  
            return i;  
    }  
    return -1;  
}
```

T, C -  $O(n)$

### Binary Search -

- condition → sorted order → monotonic function
- binary → 2 → start and end pointers



```

int binarySearch ( int arr[], int n , int target ) {

    int s= 0, e=n-1;

    int mid = s+(e-s)/2;

    while( s<=e) {

        int element = arr[mid];

        if (target == element)
            return mid;

        else if ( target < element)
            e=mid - 1; —→ search in left subarray

        else
            s= mid + 1; —→ search in right
                           subarray

        mid = s + (e-s)/2;
    }

    return -1;
}

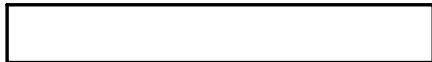
```

issue in  $mid = \frac{s+e}{2}$  ; —→ int overflow  
                  if  $s+e < INT\_MAX$

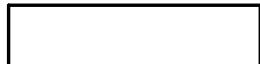
└—————> so use  $mid = s + \frac{e-s}{2}$ ;

T.C -  $O(\log_2 n)$

## T.C. of binary Search



$n$



$n/2$

:



$n/2^k$  after  $k$  times

at last  $\frac{n}{2^k} = 1$

$$k = \log_2 n$$

So loop runs  $\log_2 n$  times

$$\text{T.C.} = O(\log_2 n)$$

**W4-L2**

→ Search Space

- find range of search space (start & end) in ques of. Binary Search Questions
- store mid in ans if needed

→ In binary search questions

while ( $s \leq e$ ) {

$s = mid + 1;$

    left  
    search  
    }

$e = mid - 1;$

    right  
    search

    while ( $s < e$ ) {

$s = mid;$  → right search

        left  
        search ←  $e = mid - 1;$

    OR

$s = mid + 1$  → right

        left  
        search ←  $e = mid;$       search  
        }

cout << (int)(-3.74);

W4-L3

    → - [ int(3.74) ]

    → -(3) → -3

cout << (-22)/7; → -3

cout << 22/(-7); → -3

Types of ques in binary search

→ 1<sup>st</sup> type → classic questions

→ 2<sup>nd</sup> type → find in search space (range)

    → predicate function

→ 3<sup>rd</sup> type → observation in index

