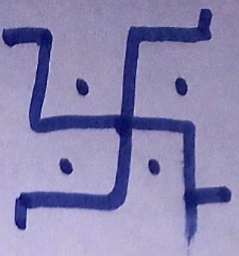
जय श्री राम



PLACEMENT PREP BEGINS 2020 - 2024

Lecture 1 (1/4/23)

1. Understand prob
2. Check given values
3. Move towards approach (Divide & Solve)
4. Make program of that approach

Pseudo de / flowchart

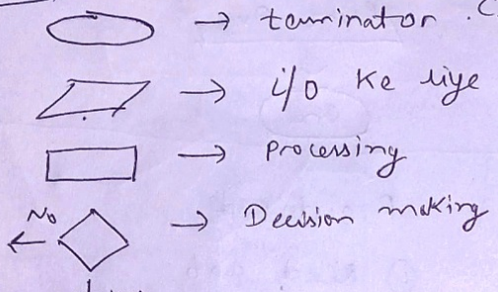
↓

program (HLL) (source code)

↓

Machine understandable code

Flowchart



→ terminator (start / end)

→ i/o ke liye

→ processing

→ Decision making

↓ yes

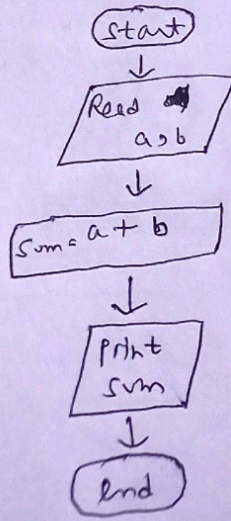
↓↑ → Arrows (flow & connect components)

→ connector

Q Sum of 2 no. flowchart

a = 5, b = 10

sum = a + b = 5 + 10 = 15



Pseudocode

way of representing code which is generic and not any language specific.

just code in english language

— x — x — x — x —

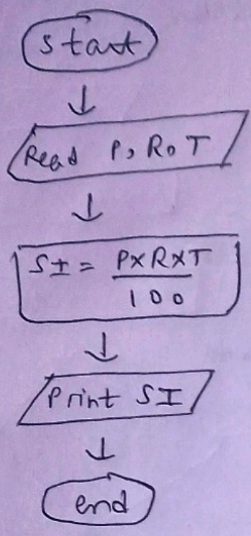
1. Read 2 no's a and b
2. Sum = a + b
3. print sum

or

1. read a
2. read b
3. let Sum = 0
4. Sum = a + b
5. print Sum

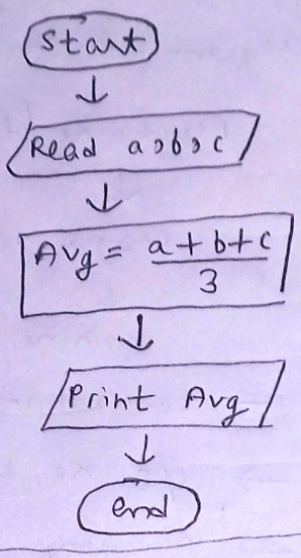
Pseudocode / flowchart for Simple Interest

$SI = \frac{P \times R \times T}{100}$

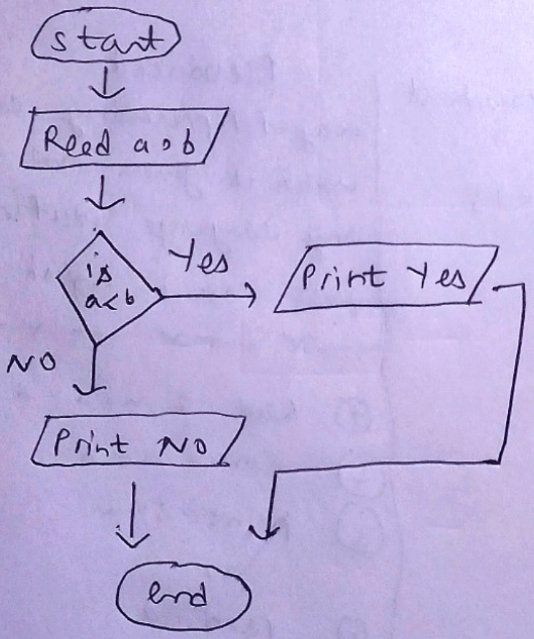


flowchart for avg of 3 no.

$avg = \frac{(a+b+c)}{3}$



check a<b or not
Print yes else no
i/p → a, b
o/p → yes or no



Pseudo code

① Read a, b

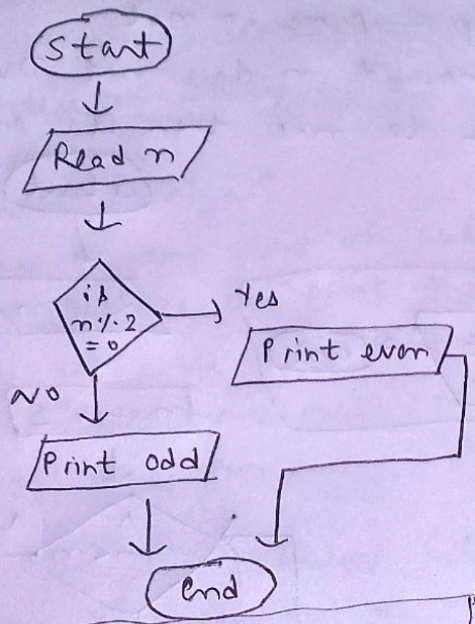
② if a<b
Print Yes
else
Print NO

Check given no. is odd or even

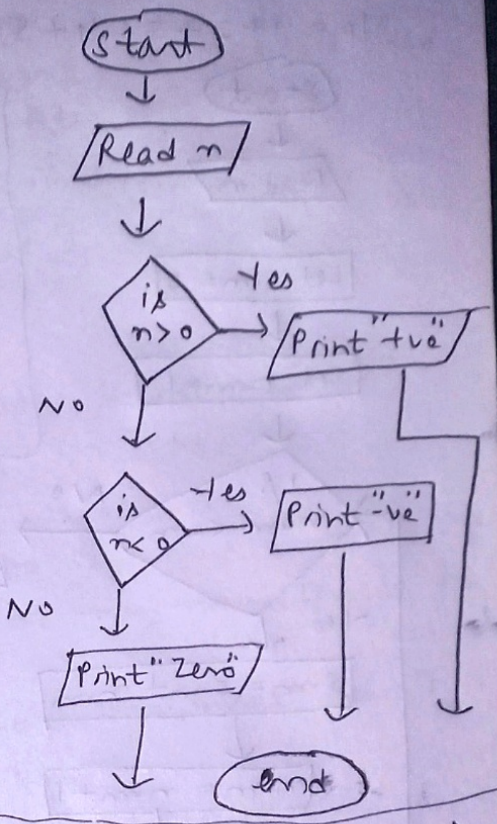
no. % 2 == 0 means even

no. % 2 != 0 means odd

% operator → remainder (modulus operator)



Check given no. is +ve, -ve or Zero



Valid triangle or not

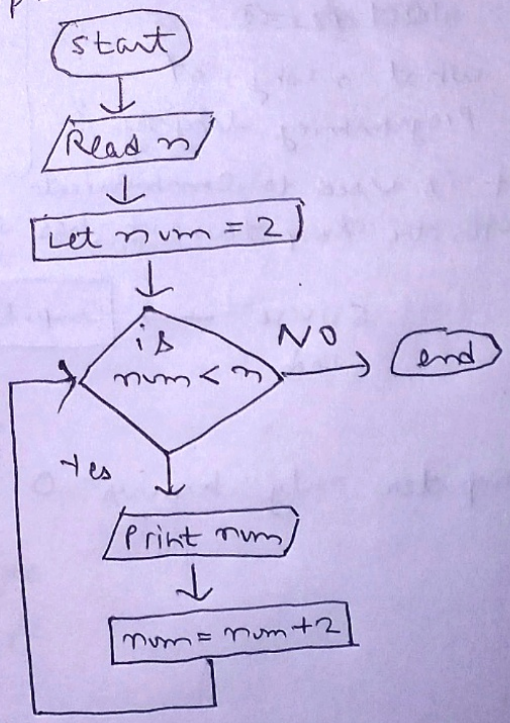
a+b > c, b+c > c, c+a > b

is our condn

Print all even no. from 1 to N

i/p = n

o/p = all even no. b/w 1 to n

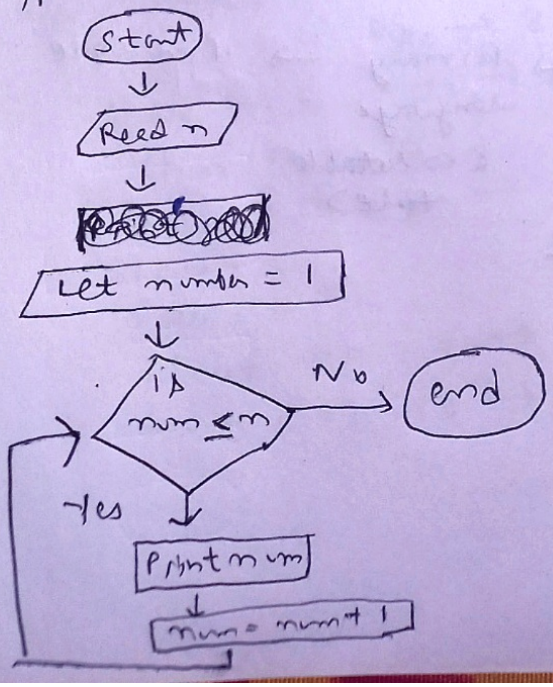


Loops

Print 1 to N where N = given

i/p N = 5

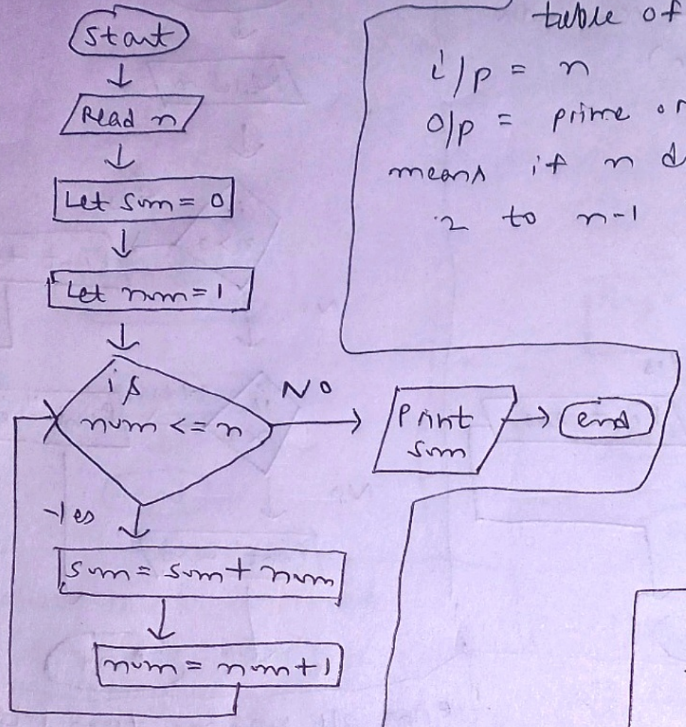
o/p 1, 2, 3, 4, 5



Find sum 1 to N (inclusive)

i/p = 5

o/p = 1+2+3+4+5 = 15



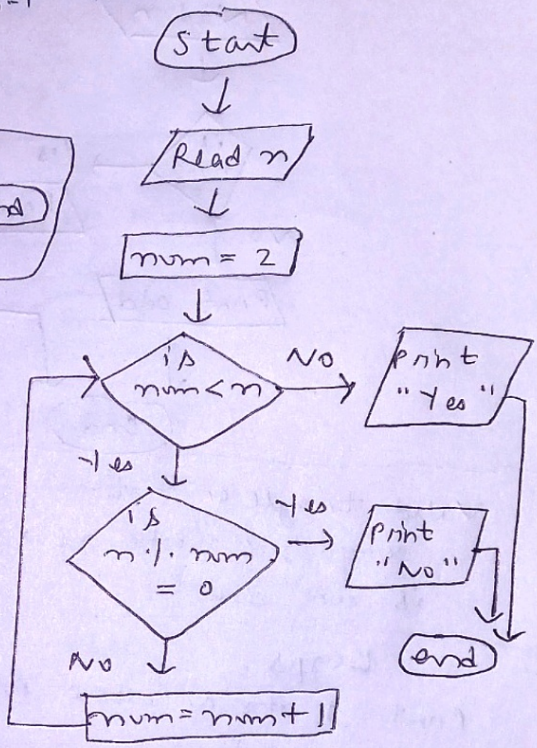
check no given is prime or not

either comes in table of 1 or table of itself.

i/p = n

o/p = prime or not

means if n does not Divisble by 2 to n-1 then its prime (i.e. ≠ 0)



~~Lecture~~

what, why of Programming language?

It is used to Communicate with the computer, it has some rules or semantics

Source code → Compiler → Binary language (executable file) → Run code

Computer only knows 0 or 1

Lecture 2

→ Getting started

made src code using any programming language
comp only understand binary language (0 or 1) for that we need compiler then comp runs or executes that file.

→ Compiler helps in Translation, error finding
IDE (Integrated Development environment)

→ first program
"Namaste Duniya"

→ There should be some start of code (which is int main())

→ using cout, we can print anything in c++.

→ To use cout, we need to include Header file

→ we write "using namespace std" to say, we want to use std vala namespace. as their are many namespaces in c++.

→ endl means newline (put cursour to next line) printing newline. or can use "\n"

→ ";" semicolon is used to end the line just like we use fullstop (.) in english literature.

→ Data types and variable
To store anything in c++ we need to tell its type so that its size can be estimated.

int a
↓ ↓
type variable

int → 4 byte → 32 bit (depend compiler to compiler)
char → 1 byte → 8 bit
char ch = 'a';
bool → 1 byte → 8 bit
True, 1
false, 0
float f = 1.2 → 8 byte
double d = 1.23 → 8 byte

→ There are some rules or convention for variable naming but never put number in starting of variable naming.

```
int a = 123 ;   ✓
int 2a = 123 ;  ✗
char a = 'v' ;  ✓
```

→ sizeof() operator to get size of a variable

→ How data stored in memory ?

```
int a = 8 ;
```

↓ convert to binary (1000)

↓

0000 0000 0000 1000 (32 bit)

for -ve number ?

```
char ch = 'a'
```

ASCII values of a = 97

65-90 is A-Z

97-122 is a-z

→ Type casting (converting one type of another type)

```
int a = 'a' ;
cout << a << endl ;   o/p = 97
char ch = 98 ;
cout << ch ;          o/p = b
```

integer = 4 byte means 32 bit

max value that can be stored = $2^{32}-1$

min value = 0

for char = 1 byte = 8 bit

max value = $2^8 - 1$

min value = 0

How -ve no. are stored

① first bit show no. is +ve or -ve
+ve → 0 , -ve → 1

① ignore -ve sign
② convert left over thing to binary
③ Take 2s complement and store
2s comp means first take 1s comp (0→1, 1→0)
now add 1 to it

int a = -5;

5 = 0 1 1

1 1 1 . . . . . 0 1 1

1s comp = 0 0 0 . . . . . 1 0 0
+ 1

0 0 . . . . . 1 0 1

↓
0 means -ve
so print = -5

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

but 0 and -0 is same so why do we rep it in 2 ways so to save this, we use 2s comp method

→ default we can store both -ve or +ve in int but
unsigned int → only +ve no. (range has increased)
unsigned abc = 111; o/p = 111
unsigned abc = -11; o/p = 4294967285
by default int has signed value i.e both + or - can be stored.

→ Operator
%. → modulo opp
Arithmetic → +, -, *, /
int / int = int
float / int = float
double / int = double

→ Relational operator
=, >, <, >=, <=, !=

→ Logical operators
&&, ||, !

→ Bitwise operator
&, |, ^, <<, >>, ~

Lecture 3

Conditionals & loops

Conditionals

```
if (a>b)
{   cout << "a";
}
if (b>a)
{   cout << "b";
}
```

if-else block

```
if(a>0)
 { print +ve
 }
 else {
      print not +ve
 }
```

→ cin do not read tab '\t', " " space, "\n" enter

→ cin.get(); reads everything whether its space, tab, anything for input.

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

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

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

Loops

→ while loop

```
while (cond^n)
 {
 }
```

Pattern print

① check no. of row
② check no. of col in each row
③ make relation b/w each row & col. based on row.

Lecture 4

Pattern Print

① 1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4

② 3 2 1
3 2 1
3 2 1

③ 1 2 3
4 5 6
7 8 9

row = 3
col = 1
Take count

④ *
* *
* * *
* * * *

⑤ 1
2 2
3 3 3
4 4 4 4

⑥ 1
2 3
4 5 6
7 8 9 10

⑦ 1
2 3
3 4 5
4 5 6 7

⑧ 1
2 1
3 2 1
4 3 2 1

⑨ A A A
B B B
C C C

⑩ A B C
A B C
A B C

⑪ A B C
B C D
C D E

⑫ A
B B
C C C

⑬ A
B C
D E F
G H I F

⑭ A
B C
C D E
D E F G

⑮ D
C D
B C D
A B C D

⑯ *
* *
* * *
* * * *

space + star

⑰ * * * *
* * *
* *
*

⑱ * * * *
* * *
* *
*

Impt

⑲ 1
1 2
1 2 3
1 2 3 4

1
2 1
3 2 1

question has 3 parts

Day 2 | Lecture 5 (2 Apr 2023)

Operator & loop

Bitwise operator

AND → &
OR → |
NOT → ~
XOR → ^

all work in level of bits

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

a = 5 → 101
b = 7 → 111
a & b → 101 = 5

OR operator (|)

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

a = 2 → 10
b = 4 → 100
a | b → 110 = 6

a = 3 → 11
b → 6 → 100
a | b → 111 = 7

(NOT)

X	Z
0	1
1	0

a = 2 → 10 → 0000....0010
int = 4 byte = 32 bit
~a = 1111.....1101

How to print ~a

~a = 1111.....1101

I means -ve

take 2's comp of

-111.....1101

2's comp = -000.....0010
+1
= -00.....0011
= -3

XOR (Dono mai se ek 1)

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

a = 2 → 10
b = 4 → 100
a ^ b = 110

a = 5 → 101
b = 7 → 111
a ^ b = 010

To get 2s comp we get 1s comp first then add +1 in it.

a = 4 → 100
b = 6 → 110
a ^ b = 010 = 2

(Left shift opp)

5 << 1

000.....101 << 1

0000.....1010 = 10

3 << 2

0000.....11 << 2

000.....1100 = 12

a << b

Shift bits of a by b steps

Set bit means bit = 1

any bigger on left shift becomes -ve no.

0 100.....0010 << 1

1000.....0100

↓

show -ve

(Right shift opp)

15 >> 1

5 >> 2

000.....0101 >> 2

000.....00001 = 1

5 >> 1 → 5/2

5 >> 2 → 5/2x2

★ on shifting +ve no. padding is done with 0 i.e extra bit added on shifting either side is 0

★ on shifting -ve no., padding is compiler dependent.

(Inc | Dec operator)

i++ → post inc

++i → pre inc

i-- → post dec

--i → pre dec

i++ → use krne ke baad mai badha dunga

int i=3, a=2

sum = a + (i++);

= 2+3 = 5

but now i = 4

phele purani value use krlo then inc krdo

++i → phele increment phir use karo

int i=11;

int a = ++i = 12

a=2, i=3

sum = a + (++i)

= 2+4

= 6

someway i--, --i are used.

(For loop)

for(int i=0 ; i<=n ;

initialisation condition

i++)

updation

{

}

ini, cond^n, updn nothing is necessary we can also write it like

for(; ;)

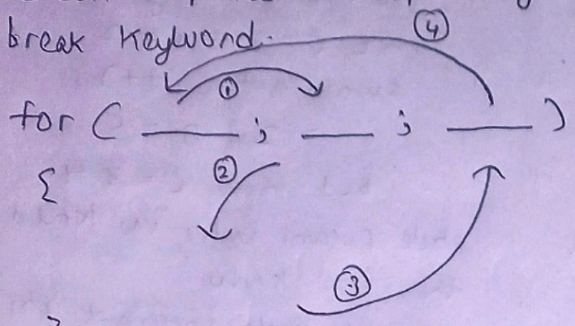
{

}

but it will run into a ∞ loop as this for loop does not know when to stop.

(Break)

We can stop this ∞ loop using break keyword.



Fibbonacci series

0, 1, 1, 2, 3, 5, 8, 13 . . . .

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

```
int a = 0
int b = 1
cout << a << b
for (i = 1; i <= n; i++)
 { int next = a+b;
      a = b;
      b = next;
      cout << next;
 }
```

(Continue)

Skip a particular iteration in the loop.

(Variable & Scope)

- Declare a variable before using it.
- Without assignment any garbage value will be assigned to that variable
- Anything b/w { } will have block scope
- we cannot redefine any variable.
- we cannot declare variable of same name inside same block twice. but its valid for block inside block.

```
if (true) {
  int c = 10;
  if (true)
  { int c = 11;
    cout << c << endl;
      // 11
  }
}
```

but if we remove it cout << c = 10

- if variable is not found inside block then it will be searched outside whether it declared anywhere else outside or not.

(Operator Precedence)

- Use bracket
- Do not cram table

Q Leetcode Problem

To get last digit of no.

= num % 10

To remove that no. from

Digit = num / 10

Go on till num ≠ 0

Q Get no. of '1' bits in any unsigned integer

☆ check last bit and right shift by 1 everytime till no. becomes 0

☆ To check last bit we do n & 1 as

1 & 1 = 1

0 & 1 = 0

So by n & 1 we will get whether last bit is 0 or 1

Lecture 6

(Binary & Decimal Number System)

Decimal = 1, 2, 3, 4 . . . . .

Binary = 0 or 1

Logic for conversion

① n = 10

→ Divide by 2

→ store remainder

→ repeat above step till n = 0

→ reverse the answer left

n = 10

~~10 / 2 = 0~~

Div	Rem
10/2 → 5	0
5/2 → 2	1
2/2 → 1	0
1/2 → 0	1

ans = 0 1 0 1

now reverse the ans = 1 0 1 0

② n = 7

Div	Rem
7/2 = 3	1
3/2 = 1	1
1/2 = 0	1

ans = 1 1 1

reverse = 1 1 1

Binary of 7 = 1 1 1

$2^2 + 2^1 + 2^0 = 7$

(Another logic)

n = 5, 1 0 1

Binary rep of 5 ?

---

num & 1 = 1 means odd

else its even

```
while(n != 0)
{   bit = n & 1;   // get right most bit
    n = n >> 1;    // Right shift
}
int ans = 0
Digit = 1 on n & 1, n >> 1
      = 01 on n & 1, n >> 1
      = 101 on n & 1, n >> 1
```

answer = (10^0 × digit) + answer

answer = (10^1 × digit) + answer

answer = (10^2 × digit) + answer

★ If we want to make a no. in same flow i.e 1, 2, 3 given we want 123

```
ans = 0  // initialise
ans = (ans X 10) + digit
```

★ If we want to make reverse in same flow i.e 1, 2, 3 given we want = 3 2 1

```
ans = 0
run loop for i = 0 to n
ans = (digit X 10^i) + ans
```

★ Convert binary of no. n

```
int ans = 0, i = 0;
while (n != 0)
{ int digit = n % 10;
  ans = (digit * pow(10,i))
        + ans;
  n = n >> 1;
  i++;
}
cout << ans
```

★ Find Binary of -6

① ignore -ve sign

② Find 2's comp of 6 i.e 1's comp + 1

★ Binary to Decimal

1 0 1 0 1

$2^4 * 2^3 * 2^2 * 2^1 * 2^0$

if our bit is 1 then consider that '2 ki power' else if its 0, do not consider '2 ki power anything'

```
n = 1 1 0
while (n != 0)
{ bit = n & 1
  if (bit == 1)
  { ans = ans + (bit * 2^i)
    i++;
  }
}
```

// Code

```
int ans = 0; i = 0;
int n; // in binary i/p
while (n != 0)
{ int dig = n % 10;
if (dig == 1)
{ ans = ans + pow(2,i);
}
n = n / 10;
i++;
}
cout << ans;
```

★ if we are working on bit level

a) we get last bit

b) we Right shift

★ if we work on digit level

a) get last digit by % 10

b) Remove last digit by / 10

Lecture 7

Leetcode Problem solving

① Reverse Integer

Normal → 123
↓
321

if range exceeds return 0

① get last Digit by %10
② ans = ans X10 + last digit
③ Remove last digit by num/10
④ Repeat above steps tills n != 0

now if value > limit then return 0.

Range = [2^{31}, $2^{31}-1$]
↓ ↓
INT_MIN INT_MAX

```
if (ans > INT_MAX/10 || ans < INT_MIN/10)
{   return 0;
}
```

② Complement of base 10 integer

n = 5 → 101
~5 → 010 = 2

return complement of 5

n = 7 → 111
~7 → 000 = 0

① we need to play with bit
② Complement the bits
③ append them in a integer
④ Return that integer.

n = 5 → 0000 0101
~5 → 1111 1010

we need a mask to remove 1111.....13 part

~5 & mask
where mask = 0000...0111

then we can achieve the result

① To create mask we first Remove last bit till num != 0
② now left shift by 1
③ now OR by 1 to attach 1 at the end

```
{ int m = n;
  int mask = 0;
  ans = 0;
  if (n == 0)
      return 1;
  while (m != 0)
  { mask = (mask << 1) | 1;
    m = m >> 1;
  }
  ans = (~n) & mask;
  return ans;
}
```

③ Power of TWO

Given any no. tell its True or false based on if its power of 2 or not.

we check from 2^0 to 2^{30}

if any one == n return true

as constraint is that

$-2^{31} \leq n <= 2^{31}-1$

(Code)

```
{ int ans = 1;
  for(i=0; i<=30; i++)
    { if (ans==n)
      { return true;
      }
      if (ans < INT_MAX/2)
      { ans = ans*2;
      }
    }
    return false;
}
```

(Lecture 8)

Switch case & functions

① In switch case default case is not necessary

② Case ' ' ↓ this take int/char but it does not take float/string in switch case

③ Break is important after every case in switch

④ we can write multiple statement inside a case in switch case.

⑤ we can put another switch inside any particular case of a switch case.

⑥ if-else-if ladder can be converted to switch.

⑦ if we have a switch case inside ∞ loop and we want to get out of ∞ loop by writing one statement inside switch then it will be exit().

⑧ 'continue' inside switch is not valid.

(Functions)

① TO avoid repeatition we use functions, To avoid writing same piece of code again & again we use functions.

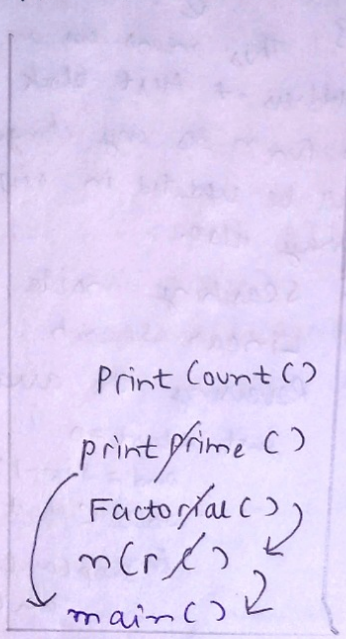
② func signature, func name, parameter name

```
void func(int a, int b)
{ return type
  parameter type
  func body
}
int main()
{
  func(90, 30);
  ↓ Argument name
}
```

Function call stack

stack = shadi ki plates.

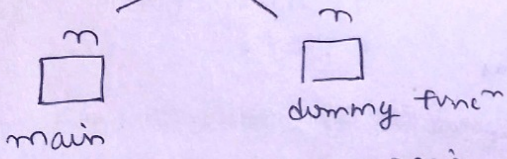
(Last In First Out)



apna kaam hote hi func^n will be out of call stack

(Pass by value)

The way we pass normally is pass by value i.e. value will not be updated in main. (both are different)



if we pass n = 16 from main and make n = 12 in dummy, n will change only in dummy not in main.

Copy of n will be passed to dummy, not the original one

(Lecture 9)

(Day 3) (3 April 2023)

(Arrays)

Array is like a container in which we can store variables of same type. all elemnt should be of same type although it can be of type but all same. (similar type of item)

at contiguous memory location we can access any elemnt through its index.

Indexing starts from zero.

v[0] = first elemnt of array v

int v[10];

v[1] = second elemnt of array v

if size of array = n

index will go from 0 to n-1

(Initialisation)

int a[3] = {5, 7, 11}

or

int a[10000] = {0}

(How to initialise entire array with same value?)

int a[5] = {[0 ... 4] = 1};

or

int a[] = {[0 ... 4] = 1};

(How to get no. of elemnt or size of an array?)

int n = sizeof(arr) / sizeof(int)

☆ Uninitialised Array will have default value = 0

☆ while passing array as parameter of func^n, mentioning size is not important.

```
void print(int arr[], int n)
{
   // body
}
```

☆ size of array = $\frac{\text{sizeof(arr)}}{\text{sizeof(int)}}$

but if size of 15, but it has only 2 elemnt initialised then this formula will give size = 15 not size = 2 so we pass size as a parameter to a func^n also.

☆ int arr[size] ✗✗ (BAD)

int arr[100000] ✓✓

always put no. in size never give variable

☆ max() and min() are 2 predefined func^n to find maximum & minimum elemnt of an array

☆ (Scope in array)

actual/original array is passed as a parameter to a func^n whenever we pass array as a parameter. Any change made inside func^n to Array will be reflected in actual array also.

☆ void func^n(int arr[])

```
{

}
```

```
main()
{ func^n(arr);
```

↓

} This means we are passing address of first block of array to func^n so any change made will be updated in original array also.

☆ Searching inside Array

☆ Linear Search.

☆ Reversing the array

just start = 0

end = size - 1;

while(start < end)

{ swap(arr[start], arr[end]);

i++;

i--;

}

(Lecture 10)

① Swap Alternate

i/p arr[5] = [1, 2, 7, 8, 5]

o/p = [2, 1, 8, 7, 5]

i/p [1, 2, 3, 4, 5, 6]

o/p = [2, 1, 4, 3, 6, 5]

```
for(i=0; i<size; i++)
{   if(i+1 < size)
    { swap(arr[i], arr[i+1]);
      i++;
    }
}
```

② Swap two numbers without using library

we need a dummy variable.

```
temp = arr[1]
arr[1] = arr[0]
arr[0] = temp
```

③ Find unique

1, 3, 4, 1, 3

(1) Count all numbers and form a Hash table

(2) XOR. $a \wedge a = 0$

$0 \wedge a = a$

④ Find Duplicates in an array

```
int ans = 0
for(i=0 → size)
  { ans = ans ^ arr[i];
  }
for(i=1 → size)
  { ans = ans ^ i;
  }
return ans;
```

⑤ Intersection of Arrays

Intersection means resultant array has common element from both array

given: arrays are sorted in non decreasing order

if no int^n present return -1.

i/p arr1 = [1, 2, 3]

arr2 = [3, 4]

o/p = [3]

① pick every element from arr1 and check in arr2. and print common elements. but before break change value of that element in arr2. So that it do not gets mapped again. (TLE)

② elements are already sorted in inc order so

```
if arr1[i] <= arr2[j]
  { break;
  }   (TLE)
```

③ 1, 2, 2, 2, 3, 4 = arr1

2, 2, 3, 3 = arr2

arr[i] < arr[j]

↓

i++

arr[i] == arr[j]

↳ print

i++, j++;

arr[i] > arr[j]

↳ j++;

[2 pointer approach]

⑥ Pair Sum

Find pairs whose Sum = S

given: Sum S, return answer in a vector and each pair should be in sorted order.

① for (0 → n-1)
{ for (i+1 → n-1)
{ if (arr[i] + j == S)
{ // store ans in {min( ), max( )}
}
}
}

// Return all pairs in sorted order of their first values.

sort ( );
return ;
for (i=0 → n-1)
{ for (j = i+1 → n-1)
{ if (arr[i] + arr[j] == S)
{ temp.push(min(arr[i], arr[j]))
temp.push(max( ))
ans.push(temp)
}
}
}
sort(ans.begin, ans.end)
return ans;
}

⑦ Triplet with given Sum

Find triplet whose sum is equal to target.

① Use 3 loops

for (0 → n-1)
{ for (i+1 → n-1)
{ for (j+1 → n-1)
{ (i+j+k == S)
{
}
}
}
}

⑧ Sort 0 1

Put all zeroes at one side and all 1's at other side.

i/p: 0, 1, 1, 0, 0, 1
o/p: 0, 0, 0, 1, 1, 1

① 2 pointer approach

if (arr[i] == arr[j])
{ i++;
}
if (arr[i] != arr[j])
{ swap
}

— x — x — x — x —

int l = 0, r = n-1;
while (l ≤ r)
{ while (arr[l] == 0)
{ l++;
}
while (arr[r] == 1)
{ r--;
}

```
Swap(arr[l], arr[r]);
l++; r--;
}
```

Don't forget to check (.SS left < right) in all 3 conditions.

— x — x — x — x —

Lecture 11

Time & Space Complexity

cnt of time taken by an algo as func^n of length of i/p.

How to say a algo is good or bad?

Based on its Time Complexity.

Let say we run any slow algo in fast machine and fast algo in slow machine then its unfair.

Time Comp is metric for any algo to be called as optimised.

Big O (upper bound) (Hadd se Hadd itna time legi)

↓

Theta Θ (for average case comp)

↓

Omega Ω

(Kam se kam) (lower bound)

Constant time → O(1)

Linear → O(n)

Log time → O(log n)

Quadratic time → $O(n^2)$

Cubic time → $O(n^3)$

O(1) ⌞ O(N!)

O(n) ◺ $O(2^n)$

$O(n^2)$ ◺ $O(n^3)$

O(log n) ◺ $O(N^2)$

O(N log N)

O(N)

O(log N)

O(1)

Complexity ↑

Rules

① lower degree is always ignored

② Constant are always ignored

$4n^4 + 3n^3 = O(n^4)$

$3n^3 + 2n^2 + 5 = O(n^3)$

$12001 = O(1)$

$\frac{n^3}{500} = O(n^3)$

$\frac{n+4}{4} = O(n)$

③ Bahaar ek loop Hai then use +

```
for( ) //O(n)
{ }
for( ) //O(m)
{ }
```

then $O(n+m) = O(n)$

Ek ke andar ek loop Hai then use *

```
for( ) //O(n)
{ for( ) //O(n)
  {
  }
}
```

then $O(n*n) = O(n^2)$

10^8 operation Rule [TLE]

All modern machine can execute any program in 10^8 operation / second

Constraints

$1 <= n < 10^5$

$\leq [10 \cdots 11]$	$O(n!), O(n^6)$
$< [15 \cdots 18]$	$O(2^n * n^2)$
< 100	$O(n^4)$
< 400	$O(n^3)$
< 2000	$O(n^2 * \log n)$
$< 10^4$	$O(n^2)$
$< 10^6$	$O(n \log n)$
$< 10^8$	$O(n), O(\log n)$

Space Complexity

Extra space taken by our algo as func^n of length of i/p.

- Variable do not contribute to SC // $O(1)$
- int arr[5]; // SC: $O(1)$ as array is of fixed size
- int arr[n]; cin>>n // SC: $O(n)$
- for loop running n times do not contribute to SC, it only contribute to TC

Lecture 12

Binary Search

→ we have already studied linear Search

→ we use it for finding a key in an Array

→ Linear search takes $O(n)$ complexity

→ Binary Search takes $O(\log n)$ so it is optimised things used for Searching in an Array. (Condition of BS)

→ Binary Search always work in monotonic func^n (i.e all ↑ing or all ↓ing order)

1) Find mid element of Array
2) Compare mid ele with key
3) if no, then we take either left or right part of mid element and again find its mid.
4) Again compare with key

```
if (arr[mid] == key)
    { return mid;
    }
```

0	1	2	3	4	5
3,	7,	11,	13,	19,	27

Key = 27

mid = $\frac{0+5}{2}$ = 2, ele = 11

(11 ≠ 27) 27 > 11 so go in right part

3	4	5
13,	19,	27

mid = 19 ≠ 27

19 < 27

5
27

mid = $5+5/2$ = 5

mid == key

return 5;

4, 8, 16, 22, 34 key = 4
0 1 2 3 4

mid = 0+4/2 = 2, ele = 16

16 ≠ 4

0 1
4, 8

mid = 0+1/2 = 0

4 == 4

return 0; // index

5) if key is not found we return -1.

5, 11, 13, 17, 19, 27 key = 25
0 1 2 3 4 5

mid = 0+5/2 = 2, ele = 13

13 ≠ 25

17, 19, 27
3 4 5

mid = 3+5/2 = 4, ele = 19

19 ≠ 25

27
5

mid = 5+5/2 = 5

27 ≠ 25 // Array filled up

return -1; // ele not found

// code

```
{ int start = 0, end = n-1;
  ~~while (start <= end)~~
  int mid = (s+e)/2;
  while (s<=e)
  { if (arr[mid] == key)
    { return mid;
    }
    if (arr[mid] < key)
    { start = mid+1;
    }
    else { end = mid-1;
    }
    mid = (s+e)/2;
  }
  return -1;
}
```

Optimisation in BS

mid = (s+e)/2;

we know int has range of $2^{31}-1$

$s = 2^{31}-1$, $e = 2^{31}-1$

then s+e > range so we need to change the formula

$$\left[s + \left(\frac{e-s}{2}\right)\right] = mid$$

Complexity of BS = O(log n)

N or $\frac{N}{2^0}$

$N/2$ or $\frac{N}{2^1}$

$N/4$ or $\frac{N}{2^2}$

$$\frac{N}{2^k} = 1$$

$N = 2^k$

$\log N = k \log 2$

$\frac{\log N}{\log 2} = k$

$$\log_2 N = k$$

Lecture 13

BS Questions

① First and last posn of an element in sorted array

i/p: Left most and Right most occurence of an element

i/p: Size = 8
Key = 2
Arr = {0, 0, 1, 1, 2, 2, 2, 2}

o/p: first occurence = 4
last occurence = 7

if ele do not exist return {-1, -1}

→ Sorted Array search
we can use Binary Search in O(log n)

First most occ
1st part

Last occurence
2nd part

1, 2, 3, 3, 5 Key = 3
0 1 2 3 4

mid = 4+0/2 = 2, ele = 3
3 == 3 first occurence can be at index = 2; Let store it

now search for another 3 in left part of index = 2

so end = mid - 1 and
store index = 2 for first occurence.

```
if (arr[mid] == key)
{   ans = mid;
    end = mid - 1;
}
// rest all same as
BS other cases
```

Last occurence
just find the mid same as first occurence
store mid if == key
and start = mid + 1

```
if (arr[mid] == key)
{   ans = mid;
    start = mid + 1;
}
```

— x — x — x —

DRY RUN

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

Key = 3
mid = 0+6/2 = 3
ele = 3
ans = 3

for last occurence we go to Right part of ind = 3

3, 3, 5
4 5 6
mid = 4+6/2 = 5
ans == key, ans = 3
return 3.

— x — x — x — x — x —

② Find total no. of occurence of an ele in an sorted array

we know how to find first occurence and last occurence

Total no. of occ = (last ind - first ind) + 1

③ Peak Index in a Mountain Array

arr is mountain if

arr.length >= 3

there exist 'i', 0 < i < arr.length - 1

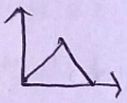
such that

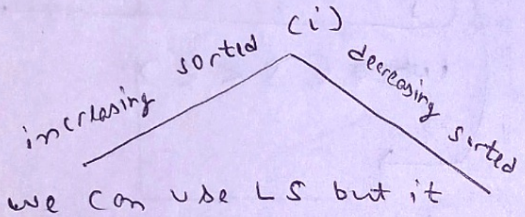
arr[0] < arr[1] . . . . < arr[i]

arr[i] > arr[i+1] . . . . arr[n-1]

Find that index i (Peak element)

0, 1, 0 | 0, 2, 1, 0





1) we can use LS but it take O(n)

2) we will use BS, take O(logn)

i/p [3, 4, 5, 1]
0 1 2 3

mid = 0+3/2 = 1

ele = 4

```
{
if (arr[mid] < arr[mid+1])
 {   start = mid + 1;
 }
else {
        end = mid;
  // not mid-1 because our
     mid can be the peak element also.
 }
mid = s + (e-s)/2;
}
```

3, 4, 5, 1
0 1 2 3

mid = 0+3/2 = 1

ele = 4 < 5

start = 1+1 = 2

ele = 5

mid = 2+3/2 = 2

ele = 5

5 ≠ 1

end = mid = 2 index

mid = 2+2/2 = 2

2 ≠ 1

So while (s<e) // s<=e not

```
 {

 }
 return s;
```

— x — x — x — x — x —

Lecture 14

Binary Search Questions.

① Find Pivot in an array.

pivot means where sum of all no. strictly to left of pivot = sum of all no. strictly to right of pivot.

if index is on left edge, means left sum = 0, same for right edge

return leftmost pivot index.

return -1 if no such thing exist

i/p: nums: {1, 7, 3, 6, 5, 6}

o/p = left sum = 1+7+3 = 11
right sum = 6+5 = 11

o/p = indx 3

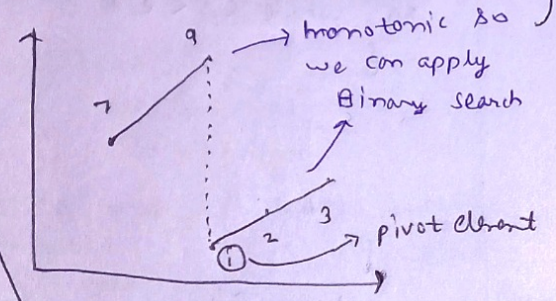
i/p: nums: {2, 1, -1}

o/p = Left sum = 0 as 2 is leftmost
Right = 1 + (-1) = 0

o/p = index = 0.

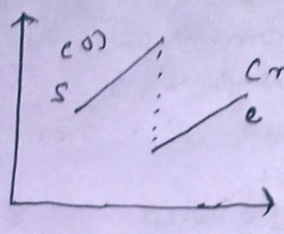
given: sorted array.

→ 9, 1, 2, 3




```
{
    int right =
        accumulate(nums.begin(),
        nums.end(), 0);
    int left = 0
    for(i=0 → n-1)
    {
        right = right - nums[i]
        if(left == right)
        {
            return i;
        }
        left += nums[i];
    }
    return -1;
}
```

Different Question



mid = s + (e-s)/2

```
if (arr[mid] >= arr[0])
{
    start = mid+1;
}
else {
    end = mid;
}
while (s < e)
{
}
}
return s;
```

Implementation

arr = {3, 8, 10, 17, 1}
0 1 2 3 4

mid = 0+4/2 = 2

10 > 3
end = mid;
3 ; 8, 10
0 1 2
s e

mid = 0+2/2 = 1

8 < 10
start = 1+1 = 2
10
2
s e

mid = 2+2/2 = 2

10 < 10
end > start, loop break

return s or
return e

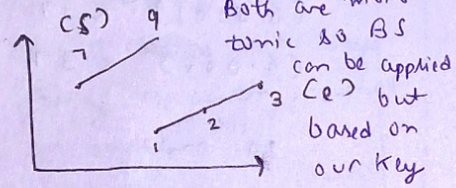
(2) Search in Rotated Sorted Array

given an sorted array and a key array is rotated at some pivot point unknown to us.

eg: Arr = [1, 3, 5, 7, 8]
Rotated at ind = 3 : [7, 8, 1, 3, 5]

o/p: index at which K is present, if not present return -1, no duplicates present, Arr can be rotated only in right direc^{n}.

i/p [7, 9, 1, 2, 3]



Both are monotonic so BS can be applied (e) but based on our key

we can find Pivot and then search for target.

our key can be

arr[pivot] ≤ Target ≤ arr[n-1]

or

arr[0] ≤ Target ≤ arr[pivot]

1) Find pivot
2) then apply Binary search accordingly based on, which side key is present of pivot

```
// get pivot ( )
// now binary search code
if (k >= arr[pivot] && k <=
      arr[n-1])
{  return BinarySearch (
      arr, pivot, n-1, k)
}
else {
  return BS(arr, 0, pivot, k)
}
```

(3) Square root of a number

Given an positive integer N. return its square root if 'N' is not a perfect square, then floor value of sqrt(N).

i/p : N = 27
o/p : $\sqrt{27} = 5.2 \approx 5$

i/p : N = 8
o/p : $2\sqrt{2} = 2.3 \approx 2$

Code

(1) Brute force
check for square of each no. close to N.

(2) if N = 27
our ans will be b/w 0 - 27

0 - 27 is our search space
0, 1, 2, 3 ..... 27 is monotonic, means we can apply BS here

if $18 * 18 > N$
then $19 * 19$ or $20 * 20 > N$
So our search space reduced to 0 - 17.

```
if (mid * mid > num)
  {   end = mid - 1;
  }
```

mid = 8

$8 * 8 < N$

$7 * 7 < N$ So

~~delete mid ted~~

```
if (mid * mid < N)
  {   ans = mid  // store it
      start = mid + 1;
  }
if (mid * mid == N)
  {   return mid;
  }
```

But if mid = $2^{31}-1$ then mid * mid > out of range

So long long int mid;
long long int square = mid * mid;
long long int ans;

How to find floor value of a decimal value

$6 + 0.1 = 6.1 * 6.1$

$6.2 * 6.2 < 37$

⋮

$6.3 \rightarrow (6.3)^2 < 37$

Somewhere

$6.2 + 0.01 \rightarrow 6.21$

$(6.21)^2 < 37$

$(6.22)^2 < 37$

if we want precision till 3 digit we do

$6 + 0.001 = 6.001$

$(6.0001)^2 < 37$

$(6.002)^2 < 37$

⋮

So on

```
double precision(int n, int precision, int IntValue)
{   double factor = 1;
    double ans = IntValue;
  for(i=0; i < precision; i++)
  {  factor = factor/10;
  for(j=IntValue; j*j<n;
          j = j + factor)
  {
      ans = j;
  }
};
return ans;
}
```

Lecture 15

Advanced Binary search problems

① Book Allocation Problem

Lokesh study for test happening in N days. TO score good read M chapters. i^{th} chapter take TIME[i] to study. Ayush has 100*100 sec in day. He always study sequentially i+1 only after Completing i^{th}.

if He starts a chapter then he completes it on same day. He divide his work in N days to ↓ amount of time he studies in a day. Find that minimal time he divide his M chapters in N days.

Given: Array representing no. of pages in books.

no. of student = m

Allocate books to students such that allocation is continuous, each student gets atleast one book Each book should be allocated to a student, allocate books such that max amount [no. of pages allocated to student is minimum.

Solution

ip: 10^0, 20^1, 30^2, 40^3

m = 4 (no. of book)

m = 2 (no. of student)

10 | 20 30 40 — 90 — max = 90

10 20 | 30 40 — 70 — max = 70

10 20 30 | 40 — 60 — max = 60

min (90, 70, 60)

o/p = 60

The array is not sorted here So How to apply BS?

we will ↓ the search space

Let say min = 0

max = Sum of all ele of array

(max, min) is our search space.

Let us check whether our mid i.e mid = max + min/2 is possible answer or not?

m = no. of student = 2

mid = 50

10 20 30 40

I → 10+20 = 30 ✓

10+20+30 > 50 ✗

(10, 20) first partition

II → 30 ✓

30+40 > 50 ✗

(30) is second partition

III → we only have 2 students So 50 is not a possible soln so values < 50 not possible then start = mid+1

Search space is now

51 ——————— 100

mid = 75

I → 10+20+30 ✓

II → 40 ✓

75 can be possible soln as for m=2, we've successfully allotted the books. So values > 75 are also possible so end = mid-1 as we need minimum ans

so 1) store ans

2) end = mid-1

51 —— 74

mid = 62

I → 10 + 20 + 30 ✓

II → 40 ✓

ans = 62

end = mid - 1

51 —— 61

mid = 56

I → 10 + 20 + ✓

II → 30 + 40 ✗

start = mid + 1

57 —— 61

mid = 59

I → 10 + 20 ✓

II → 30 + 40 ✗

start = mid + 1

60 — 61

mid = 60

I → 10 + 20 + 30 ≤ 60

II → 40 ≤ 60

(60 is our answer)

end = mid - 1

= 59

start > end so loop breaks

return ans ;

② Painter's Partition Problem.

N = no. of boards

each element rep length of each board, k = no. of painter available to paint these boards, each unit of board take 1 unit time. Return area of min time to get this job done of painting N boards such that any painter will paint continuous section of boards only.

5, 5, 5, 5

k = 2

no. of painter = no. of student

no. of pages = no. of boards

minimum time when each board takes one unit of time to paint.

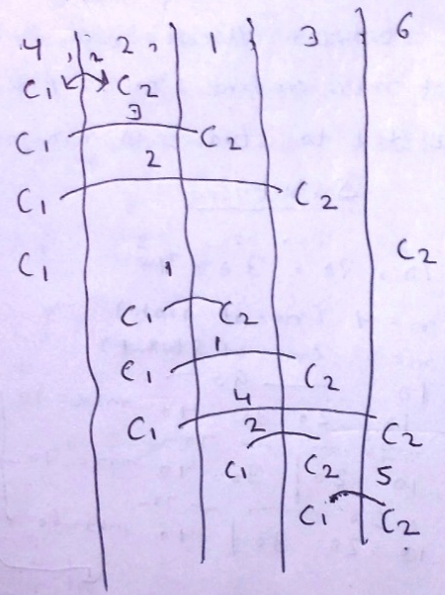
// same as Book allocation Problem

③ Aggressive Cows

Given array of N integers each ele denote posn of stall, k = no. of cows assign stalls to k cows such that min dist b/w any 2 cows is max possible

i/p = arr: [4, 2, 1, 3, 6]

k = 2 (C1, C2)



O/p = max(2, 3, 7, 2, 1, 1, 4, 2, 5, 3) = 5

maximum of minimum distance b/w 2 cows.

Search space

min dist = 0

max dist = max val of arr − min val of arr = 6 − 1 = 5

mid = 0+5/2 = 2

1) Store possible ans
2) Do start = mid+1 as we need max value in our answer

place first cow at a posn

now place second cow such that (Dist 2nd − Dist 1st <= mid)

Dist > mid
{ ans = mid − 1
}

till start < end

but at the beginning Sort the vector first

— x — x — x — x —

lecture 16 (sorting)

Selection Sort

1) Easiest Sorting algorithm

arr[] = {1, 7, 9, 2, 3, 0}

Sorted Ting arr[] = {0, 1, 2, 3, 7, 9}

what ?

There are different rounds in each round, we pick smallest element and put it in its right place.

arr[] = {64, 25, 12, 22, 11}
0 1 2 3 4

Round 1

64, 25, 12, 22, 11

i = 0

put smallest value of arr in i = 0

11, 25, 12, 22, 64
i=1

Round 2

put min value of 25, 12, 22, 64 at i = 1 posn

11, 12, 25, 22, 64
i=2

Round 3

put smallest val at i = 2

11, 12, 22, 25, 64
i=3

Round 4

11, 12, 22, 25, 64
i=4

Round 5 (not needed)

11, 12, 22, 25, 64

Total element = n

no. of rounds = n − 1

Code

```
// do not consider last element
for(i=0 ; i<n-1 ; i++)
{   minInd = i // let first ind be min
    for(j= i+1 ; j<n ; j++)
    {   if(arr[j] < arr[minInd])
        {   minInd = j;
        }
        swap(arr[minInd], arr[i])
    }
}
```

Examples

1) 6 2 8 4 10

2 6 8 4 10

2 4 8 6 10

2 4 6 8 10

2) 4 3 2 1

1 3 2 4

1 2 3 4

Space Complexity

$= O(1)$

Time Complexity

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

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

$= O(n^2)$

Best Case TC = array already Sorted = $O(n^2)$

Worst Case TC = $O(n^2)$

When to use?

When size of array is small and no memory constraints we can consider Selection Sort

— x — x — x — x —

Lecture 17

Bubble Sort

We have an array

10 1 7 6 14 9

There are multiple rounds

Round 1 (This is 1st pass)

Compare 10 and 1

if 10 > 1, swap (10, 1)

1, 10, 7, 6, 14, 9

Round 2

Compare 10, 7

1, 7, 10, 6, 14, 9

Round 3

1, 7, 6, 10, 14, 9

Round 4

1, 7, 6, 10, 14, 9

compare 14, 9

1, 7, 6, 10, 9, 14

In 1st Pass, 1st largest element will come at its right place.

2nd Pass

last element is already sorted so ignore it

1 7 6 10 9 14

1 6 7 10 9 14

1 6 7 10 9 14

1 6 7 9 10 14

sorted

3rd Pass

3rd largest element will come at its right posn now.

1	6	7	9	10	14
1	6	7	9	10	14
1	6	7	9	10	14
1	6	7	9	10	14

Sorted

Sameway we do 4th & 5th Pass. when no. of element = 6 = n

no. of Passes = 5 = n-1

Code

In every i^{th} round & pass of bubble sort, we fix i^{th} largest element in its right position.

In every round we are leaving one element from last of array.

j → 0 → < (n-1)
0 → < (n-2)
0 → < (n-3)
⋮
0 → < 1

So (j = 0 → < n-i)

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

if (i = 0 → i<n-1)
then (j = 0 → j<n-i-1)

Time Complexity:

In outer loop we do
(n-1) Comparison
(n-2)
⋮
1 = $\frac{n(n-1)}{2}$ = $O(n^2)$

Space Complexity = O(1)

Optimisation? if In any round during Comparison is done means there are no swaps, means array already sorted.

```
for(i=1 → i<n)
{  bool swapped = false;
   for(j=0 → j<n-i)
   {  if ( )
      {  swap();
         swapped = true;
      }
   }
   if (swapped == false)
   { break; }
}
```

Best Case TC = O(n)

Lecture 18

Insertion Sort

Card example

we have 4 cards, each card has some number, we get card one by one. we want to keep these cards in our hand in a sorted order.

{4 12 11 20}

we have 4

{4}

we have 12, 4 < 12 {4, 12}

we have 11, 4 < 11, 11 < 12

{4, 11, 12}

we have 20,

4 < 20, 11 < 20, 12 < 20

so {4, 11, 12, 20} is our sorted array

Another example

[10 1 7 4 8 2 11]
0 1 2 3 4 5 6

Round 1 {10}

Round 2, 1 < 10, {1, 10}

Round 3, i < 7, 10 > 7, {1, 7, 10}

R 4, 4 < 7, 4 > 1, 4 < 10,

{1, 4, 7, 10}

R 5, 8 > 7, 8 < 10, 8 > 1, 8 > 4

, {1, 4, 7, 8, 10}

R 6, {1, 2, 4, 7, 8, 10}

R 7, {1, 2, 4, 7, 8, 10, 11}

o/p = {1, 2, 4, 7, 8, 10, 11}

(R7 can be ignored)

no. of element = 7 = n

no. of round = n - 1 = 6

we need not to swap

we need to shift elements here.

Code

```
for(i=1 -> i<n)
{ int temp = arr[i];
  // store it so that we
     can shift
  int j = i-1;
// Compare with prev elements
for( ; j>=0; j--)
  { if(arr[j] > temp)
     { // shift
       arr[j+1] = arr[j];
     }
    else { // Hold on
       break;
     }
  }
  arr[j+1] = temp;
}
```

Why Insertion Sort ?

It is most adaptable means we do less repetitive work for sorting.

It is stable.

9, 3, 3', 5, 1, 5', 2, 1, 3

// unsorted

1, 2, 3, 3', 3'', 5, 5', 6, 9

// stable

1, 2, 3'', 3', 3', 5', 5, 6, 9 // unstable

Stable means preserves the order of records with equal keys.

Insertion, Bubble Sort are Stable algo.

Quick Sort, Selection Sort are unstable sorting algo

Adaptive algo

algo which changes its behaviour at time it is run, based on available info.

Space Comp = O(1)

Time Comp = $O(n^2)$

Best Case Comp = O(n)

Worst Case Comp = $O(n^2)$

— x — x — x — x —

Lecture 19

C++ STL

(Complete code in github no theory)

— x — x — x — x —

Lecture 20

Question on Arrays

① Reverse an array

arr[] = {1, 2, 3, 4}

↓

{4, 3, 2, 1}

Given = array and pos^n m, Reverse array after mth pos^n.

take start = 0

end = n-1

Swap(arr[start], arr[end])

till start < end

{4 3 2 1} = arr[]

i j

{1 3 2 4}

i j

{1 2 3 4}

```
int s = m+1, e = size-1;
while (s < e)
{
    swap(arr[s], arr[e]);
    s++;
    e--;
}
```

② Merge Sorted Array

arr[] = {1, 3, 5, 7, 9}

arr2[] = {2, 4, 6}

merge both sorted array into a third array which is also sorted

arr3[] = {1, 2, 3, 4, 5, 6, 7, 9}

arr1[] = {1, 3, 5, 7, 9}
i

arr2[] = {2, 4, 6}
j

Compare i and j and jo chota hain usko daal denge. arr3[] mai

1 < 2 ; arr3 = {1}, i++

3 > 2 , arr3 = {1, 2}, j++

3 > 4, arr3 = {1, 2, 3}, i++

5 < 4, arr3 = {1, 2, 3, 4}, j++

5 > 6 , arr3 = {1, 2, 3, 4, 5}, i++

7 < 6 , arr3 = {1, 2, 3, 4, 5, 6}

j++.

now j is out of range so the array in which elements are left just make while loop and copy elements in it.

```
Size of arr1 = m
Size of arr2 = n
int i=0, j=0, k=0
vector<int> nums3 (m+n);
while (i<m && j<n)
{   if (nums1[i] < nums2[j])
    { nums3[k++] = nums1[i++]
    }
    else {
      nums3[k++] = nums2[j++]
    }
}
while (i<m)
{ nums3[k++] = nums1[i++]
}
while (j<n)
{ nums3[k++] = nums2[j++]
}
for (i=0; i<nums3.size(); i++)
{ nums1[i] = nums3[i];
}
```

③ Move Zeroes

Given an array, move all 0's to end of it while maintaining order of non zero elements.

i/p = [0, 1, 0, 3, 12]

o/p = [1, 3, 12, 0, 0]

means keep all non-zero element sorted and shift all 0's to end of array.

2 pointer approach

0, 1, 0, 3, 12, 0

move 1st non-0 value at i = 0

" 2nd " " i = 1

" 3rd " " i = 2

if we encounter 0, ignore

if we encounter non-0, swap.

```
int i = 0;
for (j=0 → j<nums.size)
{ if(nums[j] != 0)
  { swap(nums[j],
         nums[i]);
    i++;
  }
}
```

— × — × — × —

Lecture 21

① Rotate Array

Given an array, rotate array by k steps

i/p = [1, 7, 9, 11], k=2

o/p = [9, 11, 1, 7]

i/p = [-1, -100, 3, 99], k=2

o/p = [3, 99, -1, -100]

★ .'. n any no gives o/p b/w [0 — (n-1)] always

★ Array index is always b/w [0 — (n-1)]

[11, 12, 13, 14, 15], k=3

0 1 2 3 4

Let say we want to shift 15 by k=3 steps

current index of 15 = n-1

3 step jump = n-1 → n → n+1 → n+2

but array size = n

so (n+2) .'. n = 2

we get correct posn of 15 after rotn like this.

arr[(i+k)%n] = arr[i])
will shift i^{th} term by k steps in cyclic way in an array.

```
vector<int> temp;
for (i=0 → i<nums.size())
 { temp[(i+k)%nums.size()] =
        nums[i];
 }
nums = temp;
```

We take temp so that values do not get overwritten in nums array

② check if array is sorted and Rotated.

There may be duplicates in original array.

i/p = {3, 4, 5, 1, 2}

o/p = true

There can be 3 cases

I Sorted, but not Rotated
[1, 2, 3, 4, 5]

II Rotated, but not sorted
[3, 4, 5, 1, 2]

III neither sorted nor rotated
[3, 5, 7, 1, 6]

~~wrong case~~

IV all elements are equal
[1, 1, 1]

```
{ int count = 0;
 int n = nums.size();
 for (i=1 → i<n)
 { if (nums[i-1] > nums[i])
    { count++;
 } }
 if (nums[n-1] > nums[0])
  { count++;
  }
  return count <= 1;
}
```

③ Sum of 2 Arrays

2 arrays given arr1, arr2. add them

arr1 = [1, 2, 3, 4]
arr2 = [6]

o/p =
```
  1  2  3  4
        +  6
  ----------
  1  2  4  0
```

o/p = [1, 2, 4, 0]

i/p arr1 = [1, 2, 3]
arr2 = [9, 9]

o/p =
```
  1  2  3
     9  9
  ----------
  2  2  2
```

o/p = [2, 2, 2]

1) we start adding from last element of array.

3 cases may arise

1) arr1 is large, arr2 is small

arr1 = 1 2 3 4

arr2 = [there will be missed] 6

2) arr2 is large, arr1 is small

arr2 = 6

arr1 = 1 2 3 4

[these digits will be missed]

3) both arr1 and arr2 has same length

arr1 = 9 9 9

arr2 = 9 9 9

[Carry will be missed]

carry = sum / 10

sum = sum % 10

arr[i] = sum

for example: 2 + 13 = 16

carry = 16 / 10 = 1

sum = 16 % 10 = 6

arr[i] = 6

At the end our ans will be stored in reverse order so just reverse the o/p array.

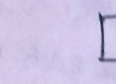
[code]

```
vector<int> add(a, int n, b, int m)
{   i = n-1, j = m-1
    carry = 0, vector<int> ans;
    while(i>=0 && j>=0)
    {  sum = a[i] + b[j] +
              carry;
       carry = sum/10;
       sum = sum%10;
       ans.push_back(sum);
       i--;
       j--;
    }
    // 1st case when arr1 large
    while(i>=0)
    {  sum = a[i] + carry;
       carry = sum/10;
       sum = sum%10;
       ans.push(sum);
       i--;
    }
    // 2nd Case
    while(j>=0)
    {  sum = b[j] + carry;
       { rest same
       j--;
    }
    // 3rd case
    while(carry)
    {  int sum = carry;
       carry = sum/10;
       sum = sum%10;
       ans.push(sum);
    }
    return reverse(ans);
}
vector<int> reverse(ans)
{   s=0, e = ans.size-1;
    while(s<e)
    {  swap(ans[s++], ans[e--])
    }
    return ans;
}
```

Lecture 22

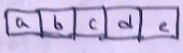
Strings : char arrays

1) `char a = 'z'` // variable of character type



but we can store only single character, to store multiple character we use string.

2) 1D char type array is called string



3)
```
int a[10];          // int array
char ch[10];        // char array
cin >> n;           // int input
cin >> arr[i];
char name[20];      // char array declaration
cin >> name;        // char array i/p
```

4) null character '\0' is used as a terminator in char array or string which signifies where string is ending.

5) It will print string till '\0' null character.

6) O/p char array?
```
cout << name;       // will print character by character
```

7) As soon as compiler sees '\0' it will stop printing.

8) cin stops execution when it sees space, tab '\t' or newline '\n'

9) length of string
start counting from index = 0
stop at '\0' (null char)
```
int count = 0;
for(i=0; name[i] != '\0'; i++)
{
    count++;
}
return count;
```

10) Reverse a string
```
int s = 0, e = n-1;
n = length of string
while(s<e)
{ swap(s[s++], name[e--]);
}
```

11) check Palindrome
a string whose reverse is same as original string.
NITIN ⇌ NITIN
NOON ⇌ NOON
Question statement : skip all special characters in the string 0) make string to lowercase
1) Reverse the string

```
for(i=0; i<s.length(); i++)
{
    char ch = s[i];
    if(ch<='z' && ch>='A')
    { ch = ch - ('A' - 'a');
      s[i] = ch;
    }
}
```

2) Compare original string & Reversed string by putting pointer at 0th index or both

```
if (isalnum(s[i])==0)
  {  ++i;
  }
else if (isalnum(reversestring[s[j]])==0)
 {  ++j;
  }
else if (s[i]== reversestring[j])
  {  ++i;
     ++j;
   }
   else {
        return false;
     }
}
return true;
}
```

12) String is a class which has some methods & properties.
String stores data internally in form of null terminated C-string, but in normal usage does not allow us to access the null terminator
We can use push_back, pop_back, find, compare etc methods in a string also.

13) We have some in-built func^n in string.

19) Diff b/w string and char array
String is used as single entity & single data type.
In char array each element is a separate entity.
String can be stored in random order in memory, char array stored in contigous location.
String is a class so can use in-built func^n but char array cannot
String can be concatenated using + or Concate() but char array can't
String is immutable (immutable means strings with same content share storage in a single pool to minimize creating copy of same value) hence one string is created its content can't be changed, changing its content will lead to generation of new string. but char array are mutable.

14) Reverse words in a string - II
In a string words are given in space seperated manner, Reverse every word of string such that spaces are intact.

1) Reverse string func^n bnaya
2) Start & start kiya and stored that words in a temp string till space is encountered.
3) Reverse temp string
4) pushback that temp string in answer and store that many spaces in resultant string.
5) As soon as i get '\0' reverse string again

15) Return maximum occuring character in a i/p string
print character occ max times if more than one character do so then print lexicographically smaller character first.

String = test

t → 2
e → 1
s → 1
} t = o/p

We have total 26 characters we make array of size 26 and store count in it
we consider 'a' & 'A' with same block in that array.

```
int arr[26] = {0}
for (i=0 → i < s.length()) {
  if (s[i] is lowercase)
  { num = ch - 'a' ; // ch = s[i]
  }
  arr[num]++;
  if (s[i] is uppercase)
  { num = ch - 'A';
  }
  arr[num]++;
}
int max = -1
int ans = 0;
for (i = 0 → < 26)
{ if (maxi < arr[i])
  { maxi = arr[i];
    ans = i
  }
}
return ans + 'a';
```

16) cin.getline
To get spaces also stored in char array to take i/p use
cin.getline (string, string length);

17) In-built string func^n
// length strlen(name);
// Compare strcmp(s1, s2);
// Copy strcpy (distination str, sourc string);

18) Replace spaces.
Replace all spaces in a string & put @40.
i/p: My Name is Khan
o/p: My@40Name@40is@40Khan.

```
if (str[i] == ' ')
{ temp.pushback('@');
  ' '  '4')
  ' '  '0')
}
else { temp.push(str[i]);
}
```

19) Remove all occ of a string.

Given String s and part
Remove all occ of part from s by starting from leftmost occ of part and remove it from s.
part is a substring.

i/p: s = "daabcbaabcbc"
part = "abc"

after removing abc
s = dababcbc
again removing abc
s = dabc
again same work
s = dab

o/p = dab

Code

s.find(part) is used to find whether 'part' string is present in string 's' or not if present, it gives first occurance of part in s.

s.erase(part.begin(), 'end()) is used to erase some part of string s from part.begin to end

```
while (s.length != 0 && s.find(part)
        < s.length())
{   s.erase(s.find(part), part.
        length());
}
return s;
```

20) Permutation in string

Given two strings s1 and s2, return true if s2 contains a permutation of s1 or false otherwise.
Return true, if s1's permutation is substring of s2.

s2 = eidbaooo
s1 = ab
o/p = true

1) Store count of s1 in an char arr[26];
2) make a window of size of s1.length.
3) Search within that window in s2 and compare with arr[26] whether the characters are same or not.

21) Remove all adjacent Duplicates in a string

given string in lowercase, remove elemt which are duplicates & adjacent to each other.

i/p: s = abbaca
o/p: s = ~~abbaca~~ aaca
↓
o/p: ca

i/p: azxxzy
↓
azzy
↓
o/p: ay

```
string ans;
ans.push(s[0]);
for(i=0 → i<s.len())
{  if(s[i]==ans.back())
   {  ans.popback();
   }
   else {
      ans.pushback(s[i]);
   }
}
return ans;
```

22) String Compression

i/p: a, a, b, b, b, c, c, c

o/p: a → 2
b → 3
c → 3

o/p: {a, 2, b, 3, c, 3}

if any character appear only 1 time, then count is not needed; just o/p the character only.

i/p: a, a, b, c, c, d, d

o/p = {a 2 b c 2 d 2}

Approach

1) Take count of character
2) if count == 1, sirf char daaldo
3) if (count > 1)
   { if (c < 10)
      { // print char with count
      }
   else // count is 2 digit no.
   { // convert to single digit and do the same
      // print char with count
   }
}

// ans should be stored in same i/p array, no extra space to be taken.

Code

```
i=0;  ansInd = 0;
n = chars.size();
while(i<n)
{  j=i+1;
   while(j<n && chars[i]==chars[j])
   {  j++;
   }
   // store all char in array
   chars[ansInd++] = chars[i];
   int count = j-i;
   if(count>1)
   {  string cnt = to_string(count)
      for(char ch: cnt)
      {  chars[ansInd++] = ch;
      }
   }
   i=j;
}
return ansInd;
}
```