why 2 started??

DSA have Babbar

→ video no. 34 ; lecture no. 31.

Recursion

↳ fxn calls itself

Eg → sum {(n+n)}

fun (n);

}

⇒ Big / complex problem की solⁿ
depend करती है some type की choti problem पर.
then use recursion.

Eg 2^n

$\Rightarrow 2^n = 2 \times 2^{n-1}$

$f(n) = 2 \times f(n-1)$ (recursive relation)

Eg → $n! = n \times (n-1)!$

↳ Stop at base case (that we already know).

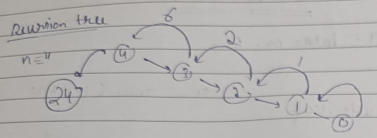
* base case में return लिखना mandatory है.

factorial using recursion.

↳ in PC

* if (base case)
  data नहीं रखना;
else {
  segmentation error
}

Recursion tree



Recursive fxn

- Base case → return
- Recursive relation knowledge
- Processing

[must]

[head & tail recursion]

2^n →

```
int power (int n){
    if ( n==0){
        return 1;
    }
    int small prob = power(n-1);
    int big prob = 2 * smaller match

    return bigprob;
}
```

[function]

Counting using recursion.

Read about recursion.
(documentation).

Fibonacci series using recursion

↳ 0, 1, 1, 2, 3, 5, 8, 13, 21 ....

↳ give 8th no.

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

$n=2$ { $F(0) = 0$, $F(1) = 1$

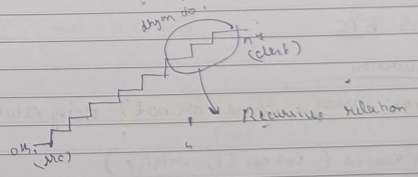
⇒ code in PC.

climb stairs

↳ initially at 0th → reach to n^{th}.

→ each time u can either climb one step or two steps.

return the no. of distinct ways from 0 to N.



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

```
if (n<0){
    return 0;
}
```

```
if (n==0){
    return 1
}
```

Code :→

```
int countway (long long n){
    if (n < 0){
        return 0;
    }
    if (n == 0){
        return 1;
    }
    int ans = countway (n-1) + countway (n-2);

    return ans;
}
```

Say digits.

input → 4 1 2

output → four one two.

string arr[10] = {"zero", ...}

→ code in PC

Sorted recursion

check whether array is sorted or not) using recursion.

```
bool isSorted ( int arr [], int size)
    // base case.
    if (size == 0 || size == 1)
        return true;
    }
    if (arr[0] > arr[1])
        return false
    else
        bool ans = isSorted (arr+1, size-1)
        return ans;
}
```

(HW) i/p → | 3 | 2 | 5 | 16 |

O/p → sum of all elements in array.

in laptop code.

binary search recursion

base case → size == 0?
false

Code in PC

binary search recursion

| 2 | 4 | 6 | 10 | 14 | 16 |

Algo → mid = $\frac{0+5}{2}$ = 2

key = 14 (eg).

14 > 6

=> | 10 | 14 | 16 |

mid = $\frac{2+5}{2}$ = 14

ans = found.

if (arr[mid] < key)
    search in right half
}
else {
    search in left half
}

Dikkat Bhi iterative use mat karo while using recursion.

search in right → l = mid+1
search in left = r = mid − 1

binarysearch (arr, l, r, k)

binarysearch (arr, mid+1, r, k)

binarysearch (arr, l, mid−1, k).

Code in PC.

H.W → Solve all of previous Binarysearch questions using recursion.

Que. i/p → "abcde"
o/p → "edcba"
→ reverse a string using recursion.

reverse (i, j, s) {
// base case
if (i>j)
return;
swap (s[i], s[j])
i++;
j--;
reverse (i, j, s);

H.W → do with only 1 pointer.

Stack int / float - - etc.

Q. check palindrome using recursion.

// Base case
if (i>j)
return true;
}

code in PC

H.W → do with one pointer only
hint → (i, n−i) type.

Q. i/p → a & b
o/p → a^b
using recursion.

↳ a^b → b is even → $a^{b/2} \times a^{b/2}$
→ b is odd → $a \times (a^{b/2} \times a^{b/2})$

Eg. 3, 11 → 3 × 243 × 243
3, 5 → 3 × 9 × 9 = 243
3, 2 → 3 × 3
3, 1 → 3

Recursion tree.

* binary = char (Ch Not 2) + '0' + binary;
N /= 2;
→ to convert number to binary.

bubble sort using recursion.

code on laptop (✓✓)

Hw → Selection sort, Insertion sort] using recursion.

Imp # Merge Sort (Using Recursion)

Approaches

→ merge 2 sorted arrays [concept as well as a question].

create new array ← copy well

indexes (s, e, mid)

→ Code in PC.

Hw → Read application of Merge sort.

Hw → Inversion count Question.

Imp Time complex → O(nlogn) → How?

match again (difficult

Quick sort

0						5
3	5	1	8	2	4	

↳ element → rightness place

4

--- [e] ---

<e >e

] partition partition karna

→ | 1 | 5 | 3 | 8 | 2 | 4 |

↳ | 1 | 2 | 3 | 8 | 5 | 4 |

↳ then recursion.

```
void quicksort (int arr[], int s, int e){
    //base case
        if (s>=e)
            return;

    // partition
        p = partition (arr, s, e)

    //recursion
        quicksort (arr, s, p-1);
        quicksort (arr, p+1, e);
}
```

• partition fxn. pivot (3) | 4 | 1 | 5 | 2

↳ take a pivot

↳ count all element < pivot

↳ pivot → s + count

[4 | 1 | 3 | 5 | 2]

[a | a | a] → condn.

[4 | 1 | 3 | 5 | 2]

i d

2 1 3 5 4

i d

2 1 3 5 4 → 1 pass complete

→ Use recursion now!

Code in pc

H.W → [why quicksort preferred over mergesort in Arrays
why ~~quicksort~~ mergesort preferred over quicksort in linked list]

S.C → O(n)

T.C → O(nlogn)

worst case → O(n²)

Day 7 ⊗ powerset. [include exclude recursion]

i/p = arr[3] = {1, 2, 3}

o/p → [[ ], {1} {2} {3} {1, 2} {2, 3} {1, 3} {1, 2, 3}]

2^n = total elements.

Approach

({1, 2, 3}, {})

exclude / include

{1, 2, 3}, {} / {1, 2, 3} {1}

{1, 2, 3}, {} / {1, 2, 3} {2} / {1, 2, 3} {1} / {1, 2, 3} {1, 2}

exclude / include

Code:-
in pic

#⊗ Subsequences.

str = "a, b, c"

Output → (bc, ac, ab, a, b, c, abc, "") 2^n

n = length of str.

(Same approach).

H.W → Solve the above question through bits.

Day 6

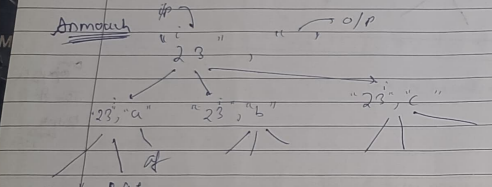
Phone keypad problem.

i/p → string : "1234" [2 - 9]

Output ⇒ 2 4 → dg cs bs
1 ghi dh ch ph
def di ci bi

2 - abc
3 - def

Approach



Code → In PC

Day 9

Permutation of a string.

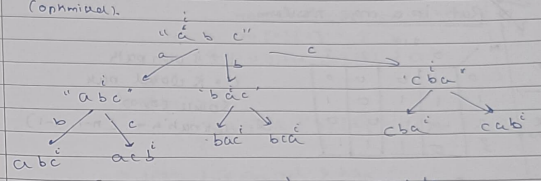
i/p → "abcd"

o/p → abc bac cab
acb bca cba

#. how place ⇒ have ~~was~~ character at fix kardo.

↳ n! for length n.

Approach ②
(optimal).



(Sabka apne aap ~~at~~ swap karda
→ i bahar chala gya].

Code :→

```
void solve (vector<int> nums, ~~vector<int>&~~ vector<vector<int>>&ans, int index){
    //base case
    if (index >= nums.size()){
        ans.push_back (nums);
        return;
    }
    for (int j = index; j<nums.size(); j++){
        swap (nums[index], nums[j]);
        solve(nums, ans, index+1);
        //backtrack
        swap (nums[index], nums[j]);
    }
}

vector<vector<int>> permute (vector<int>& nums){
    vector<vector<int>>ans;
    int index=0;
    solve (nums, ans, index);
    return ans;
}
```

Day 12

Rat in a mage problem

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

1 → is open path

0 → is closed path

source (0, 0)

Co. destination → (n-1, n-1)

← rat → (all based momentum.

↓

all existing cell →

Approach →

up → (x-1, y)

down - (x+1, y)

left - (x, y-1)

right - (x, y+1)

// movement combination

source (0, 0) → 2 | L | R | U

path = RRRRRR

(∞ loop ho sakta hai)

(To rasta already liya vo lena na pade).

visited

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

m

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

thus called.

(i, j) → (k, l)

↳ 1. matrix k andar hai chahiye

↳ 2. m[k][l] = 1

↳ 3. visited[k][l] = 0

safe to move.

↳ when move {

visited[k][l] = True

}

↳ when function call return

↳ visited[k][l] == false

back track.

}

Code:

```
vector <string> findPath (vector <vector <int>> &m, int n){
    vector <string> ans;
    int srcx = 0;
    int srcy = 0;

    vector <vector <int>> visited = m;
    //initialize with zero
    for (int i = 0; i<n; i++){
        for (int j=0; j<n; j++){
            visited[i][j] = 0;
        }
    }
    string path = "";
    solve (m, n, ans, srcx, srcy, visited, path);
    sort (ans.begin(), ans.end());
    return ans;
    if (m[0][0] == 0){
        return ans;
    }
}
};
```

// now solve for n.

(1)

```
bool isSafe (int x, int y, int n, vector<vector<int>> visited,
        vector<vector<int>> &m) {
    if ( (x>=0 && x<n) && (y>=0 && y<n) &&
        visited[x][y] == 0 && m[x][y] == 1){
        return true;
    }
    else {
        return false;
    }
}
```

(2)

```
void solve (vector<vector<int> &m, int n, vector<string> & ans,
        int x, int y, vector<vector<int>> visited, string path){
    // you have reached x, y here
    // base case
    if (x==n-1 && y==n-1){
        ans.push_back(path);
        return;
    }
    visited[x][y] = 1;

    // 4 choices = D, L, R, U
    // down
    int newx = x+1;
    int newy = y;
    if (isSafe(newx, newy, n, visited, m)){
        path.push_back('D');
        solve (m, n, ans, newx, newy, visited, path);
        path.pop_back();
    }
    visited[x][y] = 0;
}
// left   // right   // up.   [write]
```

Time & Space complexity.

↳ linear search :-

T.C → O(n)
↳ array size

S.C → O(1) → for loop used it (1)

↳ Recursion :- Fact | Pow | counting

(1) (a) Factorial → T.C → time required as function of input f(n).

(1) Eqn + recurrence

$F(n) = n * F(n-1)$

$T(n) = k_1 + k_2 + T(n-1)$

$T(n) = k + T(n-1)$

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

$\vdots$

$T(1) = k + T(0)$ → base case $= k_1$

$= k_1$

$\Rightarrow T(n) = n*k + k_1$

$= kn + \cancel{k_1}$

$T(n) = n$ $\quad O(n) = TC$

(2) Binary search.

$F(n) = \underset{k}{\log 2} + f(n/2)$

$T(n) = \underbrace{k_1 + k_2}_{k} + T(n/2)$

$T(n) = k + T(n/2)$

$T(n/2) = k + T(n/4)$

$\vdots$

$T(2) = k + T(1)$

$T(1) = k$

a lim

$T(n) = a * k$

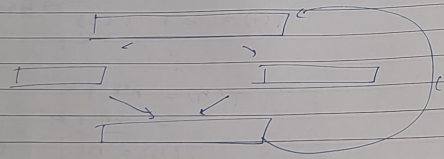
$n \to n/2 \to n/4 \ldots$ ①

$\Rightarrow \frac{n}{2^a} = 1$

$\Rightarrow a = \log(n)$

$\Rightarrow T(1) = O(\log(n))$

② Merge Sort.



Steps → ↳ break 2 array left & right

↳ Recur → sort (left / right)

↳ new array → merge

↳ copy new array content → original array

$T(n) = k_1 + k_2 + \overbrace{T(n/2)}^{\text{left}} + \overbrace{T(n/2)}^{\text{right}} + k_3(n) + k_4(n)$

$= \underbrace{k}_{\text{const}} + 2T(\tfrac{n}{2}) + n(k_5)$

$T(n) = 2T(n/2) + n(k_5)$

$T(n/2) = 2T(n/4) + n/2\, k$

$\vdots$

$T(1) = k$

x2 and odd

$\Rightarrow T(n) = an$

$= n \log n$ $T(c)$

④ Fibonacci

$T(n) = \underbrace{k_1 + k_2}_{k} + T(n-1) + T(n-2)$

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

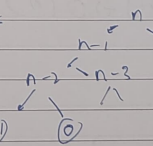
$\vdots$

$T(1) = k_1$

$T(0) = k_2$

✗ not solved.

Rec tree



Each node → takes k time

⇒ total time = k * total node.

Total node = $2^{n+1} - 1$

2^n

$\Rightarrow \boxed{T(k) = 2^n}$ $O(2^n)$

H.W → Find T(c) of monday recursion question.

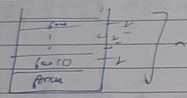
(a) lecture (41)
↳ code studio link MCQ value.

Space complexity

↳ space req^n as fn of input.

(* max space at any instant of time) = S(c)

(1) Factorial.



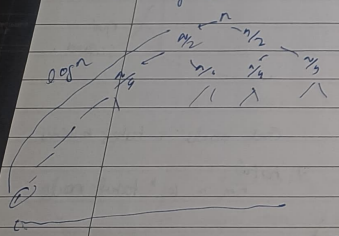
→ $S(c) = kn$

$O(n)$

(2) Binary search

$a \cdot k \Rightarrow O(\log(n))$

(3) merge sort



$S(c) = \cancel{k \log n} + n$ (merge)

$S(c) \Rightarrow O(n)$

fibonacci → $O(n)$

H.W → same as TC.

Lecture 42

Object Orient Programming. (OOPs)

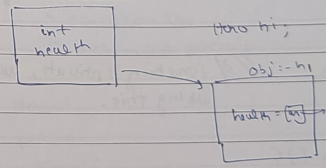
Object → entity → memory / state
→ behaviour

⇒ class :- (User defined data type).
int a
string str
char

code:

```
class Hero {
    char name [100];        // memory
    int health;
}
int main() {
    Hero h1;        // creation of object
    cout << "size: " << sizeof(h1) << endl;
    return 0;
}
```

class :- Hero



→ empty class
then ans = 1.

[object ko 1 byte allocate hojata hai
to identification & tracking

→ create class Hero.cpp and save it.

now in different code, to include above; use -

#include "Hero.cpp". (header code)

↳ to access members, use dot operation.

code:-
```
int main(){
    Hero Ramesh;
    cout << "health is:" << Ramesh.health << endl;
}
```

- Access modifier
  - ↳ Public
  - ↳ Private
  - ↳ protected.

By default: Private.

code:-
```
class Hero{
    public:
    int health;
    int level;
}
```
// if we use private then can access the member only in class hero and not in main fn.

- Getter / setter // If something is private, we can use it using this.

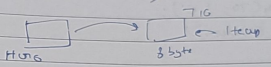
code=
```
class hero{
    int gethealth (){
        return health;
    }
    void setHealth (int h){
        health = h;
    }
```

```
int main(){
    cout << Ramesh.gethealth() << endl;
}
```

(H.W) → Read about padding & greedy alignment.

- dynamic allocation

```
{ Hero *b = new Hero;
```




```
cout << "level is" << (*b).level << endl;
cout << "level is" << b->level << endl;
}
```
2 ways to access.

- Behind the scene
  - → constructor
    - ↳ invokes at the time of object creation.
    - ↳ no return type
    - → no input parameter

2 types constructor.

code:
```
class Hero {
    public:
    Hero(){
        cout << "constructor called" << endl;
    }
}
int main(){
    Hero Ramesh;
}
```

- Parameterized constructor.

```
Hero (int health){
    this -> health = health;
}
cout << "address of Ramesh" << &Ramesh << endl;
```

- Copy constructor

```
{ Hero Suresh(70, 'c');
Hero [redacted](suresh);
    [redacted] b
    a.print();
}

Hero (Hero temp){
}
```

pass by value

but do pass by reference

- Shallow & Deep Copy

→ default copy constructor do shallow copy.

↓

accessing same memory using diffrt names.

- copy assignment operator.

```
Hero a(10, 'c');
Hero b(20, 'n');
a = b;
```

- Destructor

↳ to de-allocate memory.

```
~Hero(){
    cout<<
}
```

Use tilde sign

→ obj

static allocation ⇒ destructor automatically calls.

dynamic " ⇒ manually call destructor

H.W → what is const ~~key~~ keyword

what is initialization list

- Static keyword

↳ Agar data member create ~~karna~~ ki

↳ which belongs to class.

↓

to access hamein object ~~ki~~ ~~need~~ ~~ka~~ need nahi hoti.

Initialize:-

datatype classname :: field name = value;

→ Static function :-
↳ object create ~~and~~ at need nahi hai
└ ~~this keyword~~ ✗

Video 2

① Encapsulation → Information Hiding
↳ wrapping up data member & function

class is capsule → (class) → data member → member function
└ behaviour / method

→ Fully encapsulated class
↳ all data members are private marked.

Adv :- Data hide karsakte ho. [Security]
→ Read only class
→ code reusable
→ helps in unit testing

• Implementation

Code :
```
class student {
    private :
        string name;
        int age;
        int height;
    public :
    int get Age () {
        return this -> age;
    }
}
```

② Inheritance

class A
class B
Inherit karaliya.

Eg. Human → parent class / base class
Male
↳ sub-class / child class

Code → In PC.

⇒ Super class → Access modifier property data members
Sub-class → Mode of Inheritance

1)	public	public →	public
2)	public	private →	private
3)	public	protected →	protected
	protected	public / protected →	protected
	~~public~~ protected	private →	private
	private	→	NA

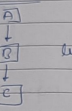
↳ private data memory of any class cannot be inherited.

• protected → similar to private
↳ child class can access it.

Types of Inheritance

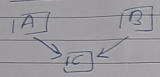
- Single

- Multilevel

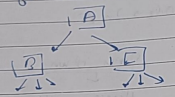


levels wise

- Multiple

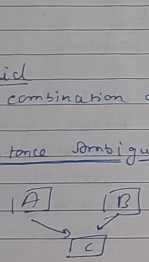


- Hot Hierarchical
  → one class serves as parent for more than one class.



- Hybrid
  → combination of above cases. (more than one).

Inheritance Ambiguity



→ If name of fxn inside A & B are same, and then you call in C; then which will be called ??

→ {

```
obj . A::Func();
obj . B::Func();
```

→ rash of Ambiguity.

Polymorphism :-

↳ existing in multiple forms

1. Compile time polymorphism
2. Run time polymorphism. (dynamic polym :-)

① CT
- Func overloading
  ↳ Operator overloading (Operator ki Apna custom kaam karna)

syntax

```
return type operator + ()
{
}
```

② RT
↳ Method overriding

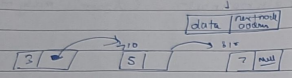
Abstraction
↳ Implementation Hiding.

H.W
- http/:- have Babbar OOPS cheatsheet.

Linked list

Day 1

↳ Data structure
↳ collection of nodes.



why needed ?

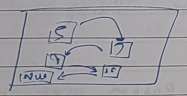
arr[10] → run time - ?
change to 100 ? can we ?

Sol^n

" L.L ↳ dynamic D.S. (can grow | shrink run time)
↓
No memory wastage.

↳ insertion / Deletion → easy.
(No shift needed)

" data heap ke andar pada hogo.

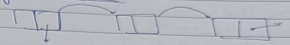


Types of LL

→ Singly
→ doubly
→ circular
→ circular doubly.

* vector is actually a dynamic array.

Singly linked list



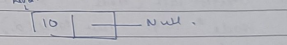
pointer of node type.

```
class linked list node {
    public:
        int data;
        linked list Node * next;
};
```

Code: → In PC

Operation at head.

```
Node *Head = NULL;
Node *Node1 = new Node(10);
```




```
temp -> next = head | node1;
head = temp;
```

Code:-

```
void InsertatHead(Node* &Head, int d){
    Node* temp = new Node(d);
    temp -> next = head;
    head = temp;
}
```

- **traversal**

```
node * temp = head
cout << node -> mint
temp ki Baat Bachhad.
```

Jab tak temp Null nahi hota.

Code:

```
void mint (node * &Head){
    Node * temp = head;
    while (temp != Null){
        cout << temp -> data << " ";
        temp = temp -> next;
    }
    cout << endl;
}
```

- **insert at tail** (will add new node ending node ke baaje).

```
code - void insertAtTail (Node * &tail, int d){
    Node * temp = new Node(d);
    tail -> next = temp;
    tail = ~~tail -> next~~; temp;
}
```

- **Insert in Middle.**

```
Void insertat position (int position, int d, Node * &tail, Node * &head){
    Node * temp = head;
    int count = 1;
    while (count < position - 1){
        temp = temp -> next;
        count++;
    }
    //creating a node for d
    Node * nodetoinsert = new Node(d);
    nodetoinsert -> next = temp -> next
    temp -> next = nodetoinsert;
}
if (position == 1){
    insert at head (head)
    return;
}
```

//starting a node.

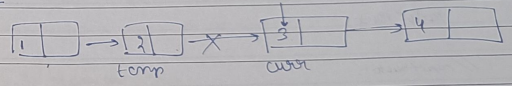
```
//inserting at last position
if (temp -> next == NULL){
    insert at tail (tail, d);
    return;
}
```

Insertion completed.

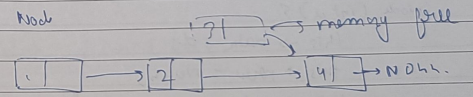
Deletion

```
void deleteNode (int position, Node * &Head){
```

Approach:-



① Middle Node



temp -> next = curr -> next

② last node.

prev -> next = curr -> next;

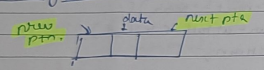
(2) First node → delete.

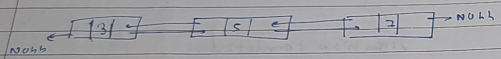
prev → next

* destructor likhna padega.

Code → in PC

Doubly linked list

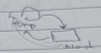





```
class node {
    public:
    int data;
    Node *prev;
    Node *next;

//constructor
Node (int d) {
        this -> data = d;
        this -> prev = NULL;
        this -> next = NULL;
    }
}
```

* traverse

- insert at head



1. create temp
2. temp → next = head
3. head → prev = temp
4. head = temp

- insert at tail

1. create node.
2. tail → next = NULL
3. temp → prev = tail
4. tail = temp;

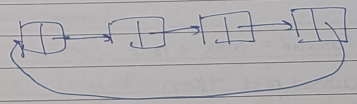
- insert at position.

I. create Node
II. node to insert → next = temp → next
III. temp → next → prev = node to insert
IV. node to insert → prev = temp

- deletion

1. temp → next → prev = NULL.
2. temp → next = NULL.
4. Memory free.

Circular linked list.



```
class node {
    public:
    int data;
    Node * next;

    // constructor
    Node (int d) {
        this -> data = d;
        this -> next = NULL;
    }
    ~Node () {
        int value = this -> data;
        if (this -> next != NULL) {
            delete next;
            next = NULL;
        }
```

// destructor

↳ i/p - data

→ found → new node add

- Insert

① empty list

tail = NULL

① → new node

② tail = temp

③ temp → next = temp;

② 1 Node

I. Forward = curr → next

II curr → next → temp

III temp → next = forward.

→ traverse -

→ deletion -

(value & another node)

① prev → next = curr → next

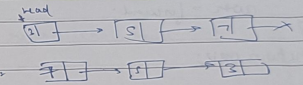
② curr → next = NULL

③ Memory free

linked list Questions

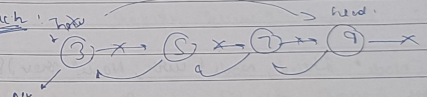
Q/1 Reverse a linked list

i/p :



o/p :

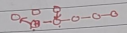
Approach :




```
Node *
    prev = NULL;
    Node * curr = head
while (curr != null) {
    forward = current -> next
    current -> next = prev
    ~~current = forward~~;
    prev = current;
    curr = forward;
}
```

Code :-

```
if ( head == NULL || Head -> next == Null){
    return head;
}
Node * prev = NULL;
Node * curr = head;
Node * forward = NULL;

while (curr != NULL){
    forward = curr -> next;
    curr -> next = prev;
    prev = current;
    curr = forward;
}
return prev;
}
```



$T(c) = O(n)$

$S(c) = O(1)$

Approach II recursive.

```
void reverse (Node * &head, Node * curr, Node * prev){
    //Base case
    if ( curr == NULL){
        head = prev;
        return;
    }
    Node * forward = curr -> prev;
    reverse (head, forward, curr);
    curr -> next = prev;

Node * reverse linked list (Node * head){
    Node * curr = head;
    Node * prev = NULL;
    rev (head, curr, prev);
    return head;
}
```

Approach III

```
Algo Node * reverse I (Node* head){
    //Base case
    if (head == NULL || head -> next == NULL){
        return head;
    }
}
Node* ~~int~~ chota head = reverse I ( head -> next);

head -> next -> next = head;
head -> next = NULL;

return chota head;
}

Node* reverse linked list (Node * head){
    return reverse I (head);
}
```

$T(c) = O(n)$

Q. Reverse Doubly linked list.

Q. Middle of LL.

1 - 2 - 3 - 4 (even)

Ans

1 - 2 - 3 (odd).

Ans

Approach ①

↳ traverse L
↳ length of L.L
len = even | odd.

↳ mid = $\frac{len}{2}$ = $\frac{5}{2}$ = ② + 1
= $\frac{6}{2}$ = ③ + 1

=> mid = $\frac{length}{2}$ + 1 → Answer if node return of (2n) + 1

Code:-

```
Node * findMiddle (Node * head){
    int len
int getLength (Node * head){
    int len = 0;
    while (head != NULL){
        len ++;
        head = head -> next;
    }
    return len;
}
Node * findMiddle (Node * head){
    int len = getLength (head);
    int ans = (len/2) + 1;
    ~~return ans;~~
    Node * temp = head;
    int count = 0;
    while (count < ans){
        temp = temp -> next;
        count ++;
    }
    return temp;
}
```

T(C) = O(n)

S(C) :

→ Tortoise & hare method.

Approach 2

Pro (1m) ——— 5sec ——— 10m (n)

noob (1m) ——— 5m (n/2)

① if Complete list → return NULL
② node → head
③ 2 node → head → next
④ Algo

```
Node * fast = head -> next
Node * slow = head
while (fast != NULL).
```

Code:-

```
Node * getMiddle (Node * head){
    if (head == NULL || head -> next == NULL)
        return head;
    if (head -> next -> next == NULL){
        return head -> next;
    }
    Node * slow = head;
    Node * fast = head -> next;
    while (fast != NULL){
        fast = fast -> next;
        if (fast != NULL){
            fast = fast -> next;
        }
        slow = slow -> next;
    }
    return slow;
}
```

T(C) :

Q. linked list reverse [k - group]

i/p - 3 → 2 → 7 → 8 → 9 → 11 → x

k → 2

o/p 2 → 3 → 8 → 7 → 11 → 9

Approach:- recursion + solve 2.
Yaahi mai dekhlunga.

Also

iterative algo (count < k)

first k node reverse

↳ head → next = recursion call

↳ return head of reversed ll
+ return.

(I) Iteration

(II) head → next → recursion call

(III) return prev

Code:

```
Node* kReverse (Node* head, int k){
    //base cal.
    if (head == NULL){
        return NULL;
    }

    Node* cursor = head;
    for (int i = 0; i<k; i++){
        if (cursor == NULL) return head;
        cursor = cursor -> next;
    }

    // reverse first k nodes
    Node* next = NULL;
    Node* curr = head;
    Node* prev = NULL;
    int count = 0;

    while (curr != NULL && count < k) {
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;   count++;
    }
    // recursion
    if (next != NULL){
        head -> next = kReverse (next, k);
    }
    // return head of reversed list
    return prev;
```

T (c) → O(n)

Q. check whether circular or not?

Approach 1 =

(1) empty list → head == NULL
return true;

(2) 1 node →
if (head → ~~next~~ next) == NULL){
return false;

② >1 node.

↳ Node * temp = head → next;

while (temp != NULL && temp != head) {
temp = temp → next
}

if (temp == NULL) (non circ)
return false.

if (temp == head)
return true;
}

Code:-

```
bool is circular list (Node* head) {
    if ( head == NULL) {
        return ~~false~~ true;
    }
    Node * temp = head -> next;
    while ( temp != NULL && temp != head) {
        temp = temp -> next;
    }
    if ( temp == head) {
        return true;
    }
    return false;
}
```

T(c) → O(n)

S(c) → O(1)

Approach 2 → Using Map.

HW Floyds cycle detection

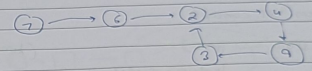
Read

③ Detect & Remove loop.

(A) Three ways to ask →

- detect cycle in LL
- Remove cycle from LL
- beginning/start node of loop in LL

Approach :-



① detect cycle.

map

3	"str"
4	"hello"
7	"foo"

map < Node*, bool > visited;

Code:-

```
#include <map>
bool detectLoop (Node* head) {
    if (head == NULL) {
        return false;
    }
    map < Node*, bool > visited;
    Node * temp = head;
    while (temp != NULL) {
        ~~visited[temp]~~
        // cycle is present
        if ( visited [temp] == true) {
            return 1;
        }
```

Using Hash Maps

```
visited [temp] = true;
temp = temp -> next;
}
return 0;
}
```

$S(c) = O(n)$
$T(c) = O(n)$

Approach 2 Floyd cycle detection algorithm.

↳ slow & fast mein ott concept.

```
if ( slow == fast )
    ↳ loop is detected
```

(some meet kar gye).

```
if ( fast == null )
    ↳ no loop.
```

Code:

```
Node * floydDetectLoop (Node * head){
    if ( head == NULL)
        return ~~false~~ NULL;
```

not needed.

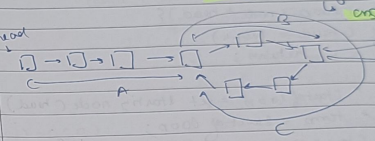
```
    Node * slow = head;
    Node * fast = head;
    while ( slow != NULL && fast != NULL ){
        fast = fast -> next;
        if ( fast != NULL){
            fast = fast -> next;
        }
        slow = slow -> next;
        if ( slow == fast){
            return ~~fast~~ slow;
        }
    }
    return ~~false~~ NULL;
}
```

$T(c) = O(n)$
$S(c) = O(1)$

Starting node of loop.

Approach - I

1. FCQ Algo → point of intersection.
2. slow = head
   ↳ slow, fast ↓ run with same pace
   ↳ when (slow == fast)
      ↳ out.



Distance by fast pt = 2* distance by slow pointer

$$(A + x*C + B) = 2*(A + y*C + B)$$

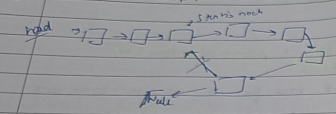
$$\Rightarrow C(x - 2y) = A + B$$

$$\Rightarrow \underbrace{A + B}_{\text{cycle complete number}} = kC$$

Code:

```
Node * getStartingNode (Node * head){
    if (head == NULL)
        return NULL;
    Node* intersection = floydDetectLoop (head);
    Node * slow = head;
    while (slow != intersection){
        slow = slow -> next;
        intersection = intersection -> next;
    }
    return slow;
}
```

Q. Remove a loop.



Code :-

```
void Remove loop (node * head){
        if ( head == NULL)
                return ;

        Node * startof loop = set starting node (head);
        Node * temp = startof loop;
        while (temp -> next != startof loop){
                temp = temp -> next;
        }
        temp -> next = NULL;
        return head;
}
```

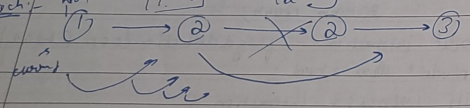
```
if (startof loop == NULL)
        return head;
```

Q. Remove Duplicates from sorted LL.

i/p - 1 → 2 → ~~2~~ → 3 → ~~3~~ → ~~3~~ → ~~3~~ → 4 → X

o/p - 1 → 2 → 3 → 4 → X

Approach :- head, 1!=2 ✓, 2==2 → delt



Code :- Node * unique sorted list (Node * head){

```
//empty list
if (head == NULL)
        return NULL;

//non empty list
Node * curr = head;

while ( curr != NULL ){
        if ( curr -> next != NULL) && curr->data == curr -> next->data){
                Node * next = curr -> next -> next;
                Node * node to delete = curr -> next;
                delete ( node to delete);
                curr -> next = next, curr -> next -> next;
        }
        else{
                curr = curr -> next
        }
}
return head;
}
```

T(n) = O(n)

S(c) = O(1)

Q. Remove duplicates from unsorted LL.

Algo For 1 element

↳ Age ek sarii LL traverse & delete duplicate.

```
curr = head
while ( curr != NULL){
        Node * temp = curr -> next;
        while ( temp != NULL){
                if ( curr = temp){
                        delete
                }
```

TC O(n²)

SC O(1)

Approach 2
↳ ① sort
② do previous question.

$T(C) = O(n \log n)$
$S(C) = O(1)$.

Approach 3 Using map.
↳ $O(n)$.

H.W
① Code above approaches.
② Q. Split circular LL in 2 halves.

Q. Sort 0, 1, 2s in LL.

Approach ⇒
Algo
↳ 1. count 0, 1, 2
2. replace data in order.

Code:-
```
Node * SortList (Node * head){
    int zerocount=0;
    int one count=0;
    int two count=0;

    Node * temp = head;
    while ( temp != NULL ){
        if ( temp -> data ==0)
            zero count ++;
        else if ( temp -> data ==1)
            one count ++;
        else if ( temp -> data ==2)
            two count ++;
        temp = temp -> next;
    }
    temp = head;
    while (temp != NULL){
        if ( zero count != 0){
            temp -> data =0;
            zerocount--;
        }
        else if ( one count != 0){
            temp -> data =1;
            one count --;
        }
        else if ( two count != 0){
            temp -> data =2;
            two count --;
        }
        temp = temp -> next;
    }
    return head;
}
```

$T(C) = O(n)$
$S(C) = O(1)$

Approach 2 Data replacement is not allowed.
↳ change links
↳ 3 linked list (0, 1, 2)
↳ merge them.

//separating into 3 LL using dummy node.

Code:-

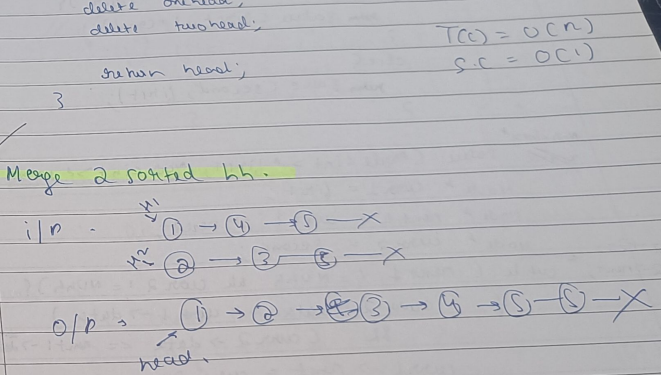
```
Node * ~~separate list (Node * head){~~
void insertatTail (Node * & tail, Node * curr){
    tail -> next = curr;
    tail = curr;
}
Node * sortlist (Node * head){
    Node * zerohead = new Node(-1);
    Node * zerotail = zero head;
    Node * onehead = new Node(-1);
    Node * onetail = one head;
    Node * twohead = new node (-1);
    Node * two tail = two head;

    Node * curr = head;
    while (curr != NULL){
        int value = curr -> data;
        if (value == 0){
            insert at tail (zero tail, curr);
        }
        else if (value == 1){
            insertattail (one tail, curr);
        }
        else if (value == 2){
            insertattail (two tail, curr);
        }
        curr = curr -> next;
    }

// merge 3 LL code.

    if (one head -> next) = NULL){
        zero tail -> next = one head -> next;
    }
    else{
        zero tail -> next = two head -> next;
    }
    one tail -> next = two head -> next;
    two tail -> next = NULL;

    head = zero head -> next

// delete dummy node.
    delete zero head;
    delete one head;
    delete two head;

    return head;
}
```

$T(C) = O(n)$

$S.C = O(1)$

Merge 2 sorted LL.

i/p :



Approach 1

```
if (head1 == NULL)
    return head2
if (head2 == NULL)
    return head1
```

jit fit cases.

max = 5 temp = data = curr → temp ka data do
Search mili.

Falsy
↳ curr ki Agle ki
prev ki Agle kardo

Code :-

```
Node <int>* sortTwoList (Node <int>* first,
                         Node <int>* second) {

    if (first == NULL)
        return second;
    if (second == first)
        return first;

    if (first -> data <= second -> data) {
        return solve (first, second);
    }
    else {
        return solve (second, first);
    }

Node <int>* solve (Node <int>* first, Node <int>* second) {
    Node* curr1 = first;
    Node* next1 = curr1 -> next;
    Node* curr2 = second;
    Node* next2 = curr2 -> next;
    while (next1 != NULL && curr2 != NULL) {
        if (curr2 -> data >= curr1 -> data)
            && (curr2 -> data <= next1 -> data)) {
            curr1 -> next = curr2;
            next2 = curr2 -> next;
            curr2 -> next = next1;
            curr1 = curr2;
            curr2 = next2;
        }
        else {
            // curr1 & next1 ko aage badhao.
            curr1 = next1;
            next1 = next1 -> next;

            if (next1 == NULL) {
                curr1 -> next = curr2;
                return first;
            }
        }
    }
    return first;
}

// if only 1 element in first list.
    if (first -> next == NULL) {
        first -> next = second;
        return first;
    }
```

T(C) = O(n)
S(C) = O(1)

• odd at the starting of solve fn to handle edge case

Palindrome check in linked list.

↳ list content = reversed list content

Approach →

1. Create Array.
2. Copy content of LL in Array
3. check palindrom for Array

S(C) = O
T(C) =

code:-

```
#include <vector>
bool isPalindrom (Node * head){
    vector <int> arr;
    Node * temp = head;
    while (temp != NULL) {
        arr.push_back (temp -> data);
        temp = temp -> next;
    }
    return checkPalindrom (arr);

bool checkPalindrom (vector <int> arr){
    int n = arr.size();
    int s = 0;
    int e = n-1;

    while (s<=e){
        if (arr[s] != arr[e]){
            return 0;
        }
        s++;
        e--;
    }
    return 1;
}
```

Approach 2

1. Middle (find)
2. reverse LL after mid
3. compare both halves → yes (✓)
   no (not palindrom).
4. repeat step 2

$S(c) = O(1)$

$T(c) = O(n)$

Code:-

```
Node * getMid (Node * head){
    Node * slow = head;
    Node * fast = head -> next;
    while (fast != NULL && fast -> next != NULL){
        fast = fast -> next -> next;
        slow = slow -> next;
    }
    return slow;
}
bool isPalindrom (Node * head){
    if (head == NULL || head -> next == NULL){
        return true;
    }
    Node * middle = getMid (head);
    //reverse list after middle.
    Node * temp = middle -> next;
    middle -> next = reverse (temp);
    //compare both halves
    Node * head1 = head;
    Node * head2 = middle -> next;
    while (head2 != NULL){
        if (head1 -> data != head2 -> data){
            return false;
        }
        head1 = head1 -> next;
        head2 = head2 -> next;
    }
    //repeat step 2
    temp = middle -> next;
    middle -> next = reverse (temp);
    return true;
}
};
```

```
Node * reverse (Node * head){
    Node * curr = head;
    Node * prev = NULL;
    Node * next = NULL;
    while (curr != NULL){
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

Add 2 numbers represented by ll.

i/p →

a — (3) → (4) → X

b — (2) → (3) → (0) → X

o/p

head — (4) → (6) → (2) → X

Approach → 1. Reverse each one.

2. Add from left to right

3. Reverse ans.

Code = Node * reverse {

/* (written in previous question). */

}

PUBLIC:

```
struct Node * add_two_lists (struct Node * first, struct Node * second){
    first = reverse (first);
    second = reverse (second);

    // add 2 LL.
    Node * ans = add (first, second);

    ans = reverse (ans);
    return ans;
}
}
```

// digit = sum%10;

// carry = sum/10;

```
struct Node * add (struct Node * first, struct Node * second){
    int carry = 0;
    ~~int sum~~
    while (first != NULL && second != NULL){
        int sum = carry + first -> data + second -> data;
        int digit = sum%10;
        Node * temp = new Node(digit);
```

— — —

↳ Code in PC.

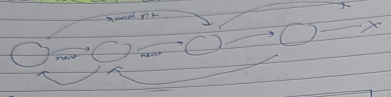
$T(c) = O(M+N)$

$S(c) = O(\max(M,N))$

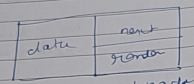
```
void insertatTail {

    */ also needed for this code /*
}
```

Q/ Clone a linked list with random pointer.

i/n -




```
node {
    int data;
    node* next;
    node* random;
}
```



(node)

Approach #1

① Create a clone list (using original list next ptr)

② 2 for loop

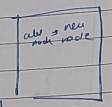
check random ptr of each node and copy.

$T(c) = O(n^2)$.

Approach #2

↳ ① step ① of previous.

② map.



clone Node → random

= mapping [original Node → random] Mapping

$T(c) = O(n)$

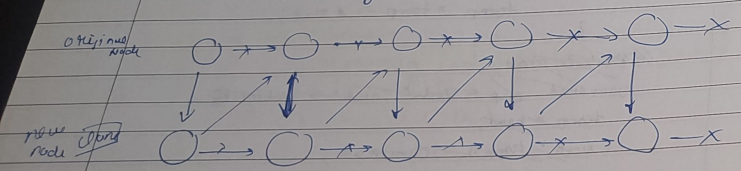
Code :-

```
private {
    void insertatTail (Node* &head, Node* &tail, int d){
        Node* newNode = new Node (d);
        if (head == NULL) {
            head = new Node;
            tail = new Node;
            return;
        }
        else {
            tail -> next = newNode;
            tail = new Node;
        }
    }
public {
    Node* copyList (Node* head) {
    // create clone list
        Node* cloneHead = NULL;
        Node* cloneTail = NULL;
        Node* temp = head;
        while (temp != NULL) {
            insertatTail (cloneHead, cloneTail, temp->data);
            temp = temp -> next;
        }
        // create a map
        unordered_map <Node*, Node*> oldToNew;
        temp = head;
        Node* temp2 = cloneHead;
        while (temp2 != NULL && temp2 != NULL) {
            oldToNew
```

```
//Create Map//
unordered_map <Node*, Node*> oldToNew Node;
Node* originalNode = head;
Node* cloneNode = cloneHead;
while (originalNode != NULL && cloneHead != NULL)
    oldToNew[originalNode] = cloneNode;
    originalNode = originalNode -> next;
    cloneNode = cloneNode -> next;
}
originalNode = head;
cloneNode = cloneHead;

while (originalNode != NULL){
    cloneNode -> arb = oldToNew Node[originalNode -> arb];
    originalNode = originalNode -> next;
    cloneNode = cloneNode -> next;
}
return cloneHead;
}
```

Approach 3# Map hatana hai tabhi $S(C) = O(1)$.

⇒ changing links.



① Create clone.
② cloneNode add btw original list.
③ Random ptr copy.

temp → next ⇒ random = temp → random → next

④ Reset change done in step ②.
⑤ return ans [cloneHead]

$T(C) = O(n)$
$S(C) = O(1)$

Code → In PC

Lect 16 LL

Merge Sort linked list

↓ data structure not allowed
→ change links to solve question.

Approach → Break in 2 halves (find mid then this).
→ sort the 2 halves (recursively).
→ merge the 2 halves. (sort

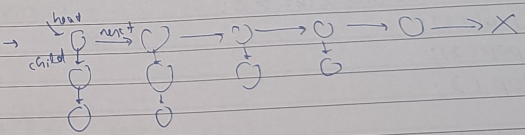
Code → in PC

$TC = O(n \log n)$.
$SC = O(\log n)$

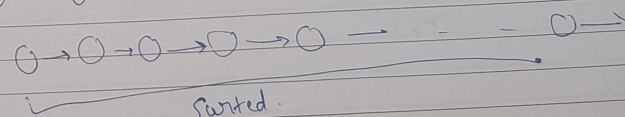
H/W → why merge sort preferred in linked list?

Flatten a linked list.

i/p →



o/p →



H.W flatten (head)
{
Node * down = head;
down → next = NULL;
Node * right = flatten (head → right);

Node * ans = merge (down, right)
return ans;
}

H.W → Solve more questions on linked list.

1 June

Stack

→ Last in first out.

push → Insert
pop → remove

peek → top element
empty

Syntax → stack <int> s;
s.push(2);
s.pop(2);
s.top();

s.empty()

Code:

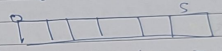
```
#include <stack>

int main()
    stack <int> s;
    s.push(2);
    s.top();
    return 0;
}
```

Sth.

0 Implementation of stack :- using array using LL.

class stack {

top = -1



• → arr int top → index
int * arr
int size

• push() :→
↳ check if space available.
top++
arr [top] = value;

• pop() →
top--;

• empty() →
top == -1;

• top() → return ( ) top( );

Code → In PC.

Q. Implement stack using linked list.

Q. 2 stacks in an array.

→ approach

top = -1

array size | | | | | | | | | top2 = size

top1 — stack 1 L to R

top2 — stack 2 R to L

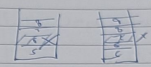
- code in PC (copy).

Q. Reverse string using stack.

Code:

```
string str = "babbar";
stack <char> s;
for (int i=0; i< str.length(); i++) {
        char ch = str[i];
        s.push(ch);
}
string ans = "";
while (! s.empty()) {
        char ch = s.top();
        ans.push_back(ch);
        s.pop();
}
cout << ans << endl;
return 0;
}
```

Q. delete middle element from stack.



Code:

```
void solve (stack<int> &inputstack, int count, int size){
    //base case
    if (count == size/2 ){
        inputstack.pop();
        return;
    }
    int num = inputstack.top();
    inputstack.pop();
    //recursive call
    solve (inputstack, count+1, size);      //incrementable.
    inputstack.push(num);
}
void deletemiddle (stack<int> &inputstack, int N) {
    int count = 0;
    solve (inputstack, count, N);
}
```

Q. Valid parenthesis (imp)

{} ✓

{()} ✓

[(] ✗

⇒ Jab closing bracket aayega, tab stack ka top par waala corresponding opening bracket milna chahiye.

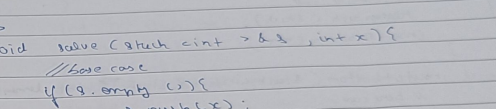
Code :-

```
bool isvalidparenthesis (string expression ){
    stack <char> s;
    for (int i=0; i<expression.length(); i++){

        char ch = expression[i];

        // opening brackets, stack push
        // if close brackets, stack top check & pop.

        if (ch == '(' || ch == '{' || ch == '[']){
            s.push(ch);
        }
        else{
            // for closing brackets
            if (!s.empty()){
                char top = s.top();
                if (matches(top, ch)){
                    s.pop();
                }
                else{
                    return false;
                }
            }
            else{
                return false;
            }
        }
    }
    if (s.empty())
        return true;
    else
        return false;
```

if (ch == ')' && top == '(') ||
(ch == '}' && top == '{') ||
(ch == ']' && top == '[')){

Insert an element at the bottom in a given stack

Approach



Code :->

```
void solve (stack <int> &s, int x){
    //base case
    if (s.empty()){
        s.push(x);
        return;
    }
    int num = s.top();
    s.pop();

    solve (s, x);
    s.push(num);
}
stack <int> pushAtbottom (stack<int> &mystack, int x){
    solve (mystack, x);
    return mystack;
}
```

Reverse Stack -> using Recursion.

Approach -> top element nikaalo,
-> reverse rest of stack using recursion,
-> insert the top element at bottom.

Code

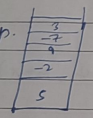
```
void reverse stack (stack <int> &stack) {
    // base case
    if (stack.empty()) {
        return;
    }
    int num = stack.top();
    stack.pop();
    // recursive call
    reverse stack (stack);
    insert at bottom (num);
}
void insert at bottom (stack <int> & stack, int element) {
    // base case.
    ~~if (s.empty()) {~~
        ~~stack.push()~~
```

-> // write previous ~~solve~~ fxn //

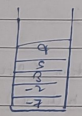
Q) Sort a stack.

- connot use loop.

i/p.



o/p ->



Approach
- top element nikal ~~k~~ side ~~k~~ Rakhlo.
- empty stack
- voapas fate hue sorted way ~~k~~ insert

```
    return ans;
}
```

- Sorted insert?

Code :->

```
void sorted insert (stack <int> &stack, int num) {
    // base case
    if (s^ek empty() || (s^ek empty() && s^ek top() < num)) {
        stack.push (num);
        return;
    }
    int n = stack.top();
    stack.pop();
    // recursive call
    ~~sorted stack (stack, num)~~
    sorted insert (stack, num);
    stack.push (n);
}
void ~~sorted~~ sort stack (stack <int> &stack) {
    // base case
    if (stack.empty()) {
        return;
    }
    int num = stack.top();
    stack.pop();
    // recursive call
    sort stack (stack);
    sorted insert (stack, num);
}
```

Q) Redundant brackets

- useless bracket.

Eg. ((a+b)) -> return.

Approach

```
For ( )
{
    ch = str[i];
    if (ch -> '{' || open bracket)
        stack.push(ch)
    else
        '}' or lower case
        if (ch == '}')
        {
            stack -> open bracket
                yes -> reducement (pop, top)
                NO -> reducement
        }
}
```

$T(c) = O(n)$

Code -> In PC

Q.. -> Q. Minimum cost to Make string valid.
OR bracket Reversal.

i/p -> { { { }

o/p -> 1 reversal.

Approach

① if str length :-> odd
    return -1

② -> Invalid string cond"
- {{{{
- }}}}
- }}} {{{

valid part hatake
remaining part check.

③ -> a -> count of closed brac
b -> count of open brac.

case I. { { { { - 4
↓
b/2

case II } } } }
a/2

case 3 } } } } { { { {
a=4 b=4
a -> b = 2
ans = 4

cm = $\left(\frac{a+1}{2}\right) + \left(\frac{b+1}{2}\right)$

Code -> in PC.

Q. Next smaller element. -> (first smaller at RHS)

i/p -> [2, 1, 4, 3]
o/p -> 1 x 3 x

Approach
2 -> {1, 4, 3} -> (n-1)
1 -> {4, 3} -> (n-2)
4 -> -> 1
3 -> 0

$T(c) = O(n^2)$
X
Bekar.

Approach 2

[2, 1, 4, 3]

2 -> next smaller element
↳ choto element kon
(stack)
choto
Bada
only store
choto kaun.

Code:

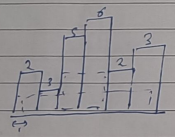
```
#include <stack>
vector <int> nextsmaller (vector <int> &arr, int n){
    stack <int> s;
    s.push(-1);
    vector <int> ans(n);

    for (int i=n-1; i>=0; i--){
        int curr = arr[i];
        while (s.top() >= curr){
            s.pop();
        }
        ans[i] = s.top();
        s.push(curr);
    }
    return ans;
}
```

$T(n) = O(n)$

imp

Largest Rectangular Area in Histogram.

⇒



Brute force =

area = length × breadth

max

↳ For (int i=0; i<n; i++){

while (left)

while (right)

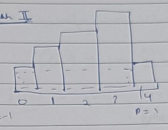
area = max (area, breadth).

}

$T(n) = O(n^2)$

Behar.

Optimum II



(max)

width = n - p - 1

= (4)

Code:- int largest Rectangle Area (vector <int> &heights){

int n = heights.size();

vector <int> next(n);

next = nextSmaller Element (heights, n);

vector <int> prev(n);

prev = prevSmaller element (heights, n);

int area = INT_MIN;

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

int l = heights[i];

~~int b = next[i] - prev[i] - 1;~~

if (next[i] == -1){

next[i] = n;

}

int b = next[i] - prev[i] - 1;

int newArea = l * b;

area = max(area, newArea);

}

return area

}

};

Private:

vector <int> nextsmaller Element (vector <int> arr, int n){

}

vector <int> prevsmaller element (vector <int> arr, int n){

}

→ 2n (PC)

Q/ Celebrity problem.

→ everyone knows celebrity.

→ celebrity knows no-one.

check whether celebrity exists in the party.

Eg ⇒

	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

① → Ignore diagonal

↳ knows no one

everyone knows ②

⇒ 1 is celebrity.

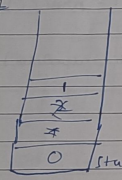
Approach Brute force

→ celebrity → row zeros.

→ each cell one's except diagonal element.

$T(c) = O(n^2)$ ⇒ Bekar

Approach #



Stack → celebrity element

Algo →

→ put all elements in stack

→ Jab stack size != 1

A → s.top → s.pop

B → s.top → s.pop

if (A knows B)

↳ B wapas daaldo.

→ if (B knows A)

↳ A wapas daaldo.

→ now A = 1 B = 0 (Repeat).

Jo single element bacha hai → wo potential celebrity ho sakta hai.

→ verify → check its row & col^n condi^n.

Code → in PC.

T.C → O(N)

H.W → check more approaches of this question.

Q/ Max rectangle in binary matrix with all 1's.

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

Ans → ③

Approach →

① → 1st row

1	1
0	0

max area = ~~2~~

② select first 2nd row

1	2	2	1

max area = ~~4~~

③ first + second + third row.

2	3	3	2	3

max area = 8

④ consider whole matrix

3	4	0	0

max area = ~~6~~

Algo ⇒

① compute max-area for 1st row

→ for remain row

→ upto valid row & element add to current row pair

② Compute area.

Code:-

```
vector<int> nextsmallerelement (vector<int> arr, int n){

}

vector<int> previousmallerelement (vector<int> arr, int n){

}

int largestrectangleArea (vector<int> height){

}
```

hint from question code

Public:

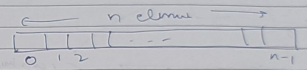
```
int maxArea (int M[max][Max], int n, int m){
    //compute area for first row
    int area = largestRectangleArea (M[0], m);
    for (int i=1; i<n; i++){
        for (int j = 0; j<m; j++){

            if (M[i][j] != 0)
                M[i][j] = M[i][j]+M[i-1][j];
            else
                M[i][j] = 0;
        }
        area = max (area, largestRectangleArea (M[i], m))];
    }
    return area;
}
};
```

$T(c) = O(n \times m)$

$S(c) = O(m)$

(2) N stuck in an array. (single array to answer).

Approach 1



k stack

→ "k" stack

entire array - N length

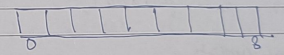
divide in k parts →

1 part = n/k

drawback

→ not space optimised.

Approach 2



3 Stacks →

2 additional things → top[] → index of top element. → k size

→ array → next[] → point to next element of the stack top

→ pt to next free place in a block

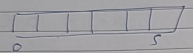
(if arr[i] → stored a element

else → arr[i] → nothing stored.

Eg// N → no of stack → 3 S_1 S_2 S_3

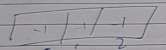
S → size of array → 6

Q → no. of queries → 5



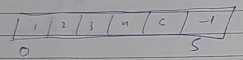
Q = { push (10, 1)
push (20, 1)
push (70, 2)
pop (1)
pop (2) }

top →



/ constructor.

next →



freespot = [0]

→ push (10, 1)

(I) //find index
int index = freespot;

(II) // freespot update
freespot = next [index];

T.C = O(1)
S.C → O(3+n)

(III) // insert in array
arr [index] = x;

(IV) // update next
next [index] = top of [n-1]

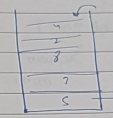
(V) // update top
top [m-1] = index;

Code → 2n PC.

IMP **

Q. Design stack that supports getMin() in O(1) & O(1)
TC SC

Approach →



5, 3, 1, 2, 4

int mini = INT_MAX;
mini = min (mini, arr)
min (5, 3) = 3
min (3, 1) = 1
⋮
check.

→ O(N) space

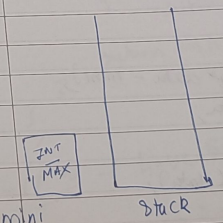
push()
- (I) - i/p → normal push
(II) - mini value → 1st i/p.

T.C = O(1)

pop() →
(I) - store stack of pop

getMin() - return second stack top().

Approach II Solve using variable to have S(1).



push operation
↳ overflow condn.
↳ for 1st element → normal push
+ mini push
↳ further element
+ if (value < mini)
val = 2*value - mini
→ push (value)
→ mini = value update

else {
↳ normal push
}.

pop()
↳ overflow condⁿ check
↳ if (curr > mini)
normal pop().
else
value = 2*mini - curr.
mini = value
pop();

getMin()
return
mini.

formula used → push → 2* curr - mini
pop() → 2* mini - curr
} → without minin
we lose prev mini
find out hard.

Code →
```
#include <stack>
class SpecialStack {
    Stack <int> s;
    int mini; ~~= INT_MAX~~
void push (int data){
    if (s.empty()) {          //for first element//
        s.push (data);
        mini = data;
    }
    else {
        if (data < mini){
            s.push (2* data - mini);
            mini = data;
        }
        else {s.push (data);
        }
}
}
```

```
int pop () {
    if (s.empty()) {
        return -1;
    }
    int curr = s.top();
    s.pop();
    if (curr > mini) {
        return curr;
    }
    else {
        int prevMin = mini;
        int value = 2* mini - curr;
        mini value;
        return prevMin;
    }
}
int top() {
    if (s.empty())
        return -1;

    int curr = s.top();
    if (curr < mini) {
        return mini;
    }
    else {
        return curr;
    }
}
bool isempty() {
    return s.empty();
}
int getMin() {
    if (s.empty())
        return -1;

    return mini;
}
};
```

"correct code in PC"
overflow error in this code.

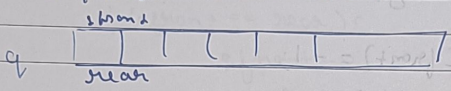
-H.W → have Babbar Sheet questions.

X Stacks Done X

lecture # 60

Queue

→ First in First out.

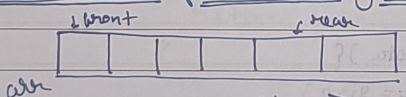


push → rear ⇒ hoga
pop → front ⇒ hoga

STL :- #include <queue>
queue <int> q;

q.push()
q.pop()
q.size()
q.empty()
q.front()
q.rear()

Implementation of queue using Array.



size →

class queue
~ 4 data members.

→ at Starting front = 0
rear = 0

if (rear = n)
⇒ full.

if (front = rear)
⇒ empty

~ Push
→ if (full) → Queue full
else {
arr[rear] = element
rear++;

→ pop

→ if (empty) → x

```
arr[front] = -1;
front++;
if (front == rear) {
    front = 0;
    rear = 0;
}
```

Code :-

```
class queue {
    int *arr;
    int qfront, rear, size;
PUBLIC:
    Queue() {
        size = 10001;
        arr = new int[size];
        qfront = 0;
        rear = 0;
    }
```

Push:

```
void enqueue (int data) {
    if ( rear == size)
        cout << "full";
    else {
        arr [rear] = data;
        rear ++;
    }
}
```

Pop:

```
int dequeue () {
    if (qfront == rear) {
        return -1;
    else {
        int ans = arr[qfront];
        arr[qfront] = -1;
        qfront ++;
        if (qfront == rear) {
            qfront = 0;
            rear = 0;
        }
        return ans;
    }

int front() {
    if (qfront == rear) {
        return -1}
    else {
        return arr[qfront];
    }
bool isempty () {
    if (qfront == rear) {
        return true;
    else {
        return false;
    }
}
};
```

T(c) = O(1)

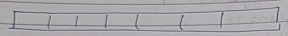
S(c) =

Circular Queue :-

rear n ki nahi aayega → ghum k arr ki starting ki aajayega.

push =

queue



front = -1
rear = -1

→ if (full) → "queue full"

· (front == 0 && rear = size - 1)

or

· rear == (front - 1) %% (size - 1)

↳ if (front == -1) // first element
front = rear = 0;
arr [queue] = data;
}

↳ if (rear == size - 1 && front != 0) {
rear = 0
arr [rear] = data;
}

else {
rear ++
arr [rear] = data;

Pop

→ if (empty) → " empty "
↳ front = -1

↳ // single element in queue
if (front == rear) {
front = rear = -1;

↳ if (front == size - 1) {
front = 0;

↳ front ++ ;

⇒ Code in PC :- //

6 Input restricted queue → i/p (from rear only)
→ o/p → both side possible.

· push - back (rear)
pop - front
pop - back

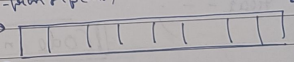
° Output restricted queue - i/p → both side possible
- o/p → only front side

→ push - front → pop - front.
→ push - back

Doubly ended queue

push - front (front)

push - back (rear)



pop - front (front)

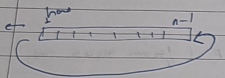
pop - back (rear)

STL Implementation

→ [| | | | | | | |]

front = -1;
rear = -1

push front
↳ if full → Queue is full
↳ // first element
front == -1 → front = rear = 0
↳ if (front = 0) → front (n-1)
↳ normal front → front--;



push rear → same logic as above
pop front → same logic as above.

→ pop-back
↳ if (empty) → "queue is empty"
↳ // single element → front = rear = -1;
↳ // cyclic nature
if (rear == 0) → rear = n-1
↳ // normal
rear--;

Code in PC

STL :-

deque <int> d;
d.push-front (12)
d.push-back (14)
d.pop-front ()
d.pop-back ()

front → 12, rear → 14

d.empty().

Queue Reversal.

Approach → Use stack.

Algo → 1. queue ki nikalo stack ki dalo
2. stack ki nikalo queue ki dalo

O(n)
T(c) S(c)

Approach-2 → Use Recursion.

Sol^n - approach (1).

Code.
```
queue <int> rev (queue <int> q) {
    stack <int> s;
    while (!q.empty()) {
        int element = q.front();
        q.pop();
        s.push(element);
    }
    while (!s.empty()) {
        int element = s.top();
        s.pop();
        q.push(element);
    }
    return q;
}
```

Q2 i/p → {-8, 2, 3, -6, 10}
k = 2
→ return negative element in a window.

→ ans → {-8, 0, -6, -6}
o/p

approach.

make → deque < int > dq;

① first k element (first window) mean.

answer =>

② loop

(age vaali window mean).

```
// removal.
(if (dq.front() - i >= k)
        dq.pop - front();

//addition
    if (arr[i] < 0)
        dq.push - back(i);
```

ans ⟶ ans → dq.pop

⟶ 0

Code:- vector < long long int > printFirstNegativeInteger(long long int A[], long long int N, long long int k){

```
deque <long long int> dq;
vector < long long> ans;
//process first window of k.
for (int i=0; i<k; i++)
    if (A[i] < 0){
        dq.push_back(i);
    }
}
//store answer of above work done.
if (dq.size() > 0){
    ans.push_back (A[dq.front()]);
}
else{
    ans.push_back(0);
}
//mean for remaining window.
for (int i=k, i<N; i++){
    //removal
    if (!dq.empty() && i - dq.front() >= k){
        dq.pop_front();
    }
    if (A[i] < 0){
        dq.push_back(i);
    }
    //ans store.
    if (dq.size() > 0){
    ans.push_back (A[dq.front()]);
    }
    else{
        ans.push_back(0);
    }
}
return ans;
}
```

H.W → Solve above question using optimal approach.

Q. Reverse first k element of queue.

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

k = 3

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

Approach → 1. fetch first element from Q & put in stack.

2. fetch element from stack and put in Q.

3. fetch first n-k element from Q and push back.

Code:

```
queue<int> modifyQueue(queue<int> q, int k){
    stack<int> s;
    for (int i=0; i<k; i++){
        int val = q.front();
        q.pop();
        s.push(val);
    }
    while (!s.empty()){
        int val = s.top();
        s.pop();
        q.push(val);
    }
    int t = q.size() - k;
    while (t--){
        int val = q.front();
        q.pop();
        q.push(val);
    }
    return q;
}
```

$S(c) = O(k)$

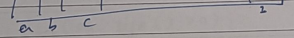
Q) First non-repeating character in a string.

i/p → a a b c

o/p → a # b b

* # for repeating.

approach - QS

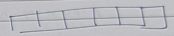


map<char, int> count; or arr[26]

0 → 'a'

1 → 'b'

→ QS (a)



a a b c

① count [a] ++;

② q.push (a);

q.front → repeating → pop();

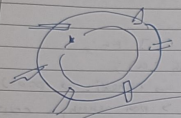
→ non repeating → cout;

q.empty() → #

Code:- string FirstNonRepeating (string A){

```
    Unordered_map<char, int> count;
    queue<int> q;
    string ans = "";
    for (int i=0; i<A.length(); i++){
        char ch = A[i];
        count[ch]++;
        q.push(ch);

        while (!q.empty()){
            if (count[q.front()] > 1){
                q.pop();
            }
            else{
                ans.push_back(q.front());
                break;
            }
        }
        if (q.empty())
            ans.push_back('#');
    }
    return ans;
}
```

Circular tour



p.pump → petrol data (& dist)
→ next p.p distance (by km).

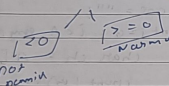
- find a starting pt so that touch completes whole circle.

i/p -

0	1	2	3
(4,6)	(6,7)	(7,3)	(4,5)

(Petrol, distance).

balance → Q → petrol ?? left



⇒ Shift s++;

→ Again calculate by adding balance to previous petrol.

if successful circle
→ return index of start.

approach - Brute force → $O(n^2)$

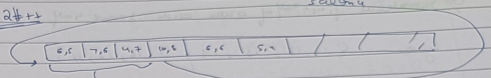
approach 2nd front & rear.

if travel possible n-0 >= 0
↳ rear++;
if (front == rear)
⇒ circle complete
return front.

else
front & rear+1; (not front++) ??
rear = front)

Optimum 2nd ++

balance



temi = -1
collision

if (balance + temi >= 0)
⇒ ans possible.

code:-

```
int tour (petrol pump p[], int n){
    int deficit=0;
    int balance=0;
    int start=0;
    for (int i=0; i<n; i++){
        balance = balance + p[i].petrol
                  - p[i].distance;

        if (balance<0){
            deficit = deficit +balance;
            start=i+1;
            balance=0;
        }
    }
    if (deficit + balance >=0){
        return start;
    }
    else{
        return -1;
    }
}
```

//start index & Arge aaya//

Q. Interleave first half of queue with second half.

i/p → { 11, 12, 13, 14 ; 15, 16, 17, 18 }

o/p → { 12, 15, 12, 16, 13, 17, 14, 18 }

Approach (1) 1. fetch first half from queue & push into new queue

q = 15, 16, 17, 18

newq = 11, 12, 13, 14

2. while (! newq.empty()) {

int val = newq.front

newq.pop()

q.push(val);

val = q.front()

q.pop();

q.push(val);

}

TC = O(n)

SC = O(n)

Approach 2 Using stack.

// should know implementation of stack using queue and vice versa //

Algo → 1. first of q → push in stack

2. stack → vacant queue.

| 15 | 16 | 17 | 18 | 14 | 13 | 12 | 11 |

3. first of q. pop & push.

| 14 | 13 | 12 | 11 | 15 | 16 | 17 | 18 |

4. first half of q + vacant stack s

while (! s.empty()) {

int val = s.top()

s.pop()

q.push(val)

val = q.front()

q.pop()

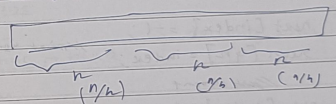
q.push(val)

}

H.W ↑ solve.

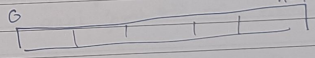
Q. "k" queue in an array.

Approach 1 → Brute force.



array break → k parts.] → cannot utilize space optimally.

Approach 2 →



arr

front[k] → [-1 | -1 | -1] if k = 3

Rear[k] → [-1 | -1 | -1]

next[n] → [1 | 2 | | | | 7 | -1] (0, 1, 2, 3, 4, 5, 6, 7)

free spot → [0]

push

↳ overflow check → free = -1

↳ // find index, where we want to insert

int index = freespot

↳ // update freespot

free spot = next [index]

↳ // first elem

if (front [qn] == -1)

front[qn] = index;

else { next [rear [qn]] = index;

}

↳ next [index] = -1; // point rear to index

↳ rear [qn] = index;

// push element

↳ arr [index] = k;

Code → 2n PC...

pop

↳ is empty → underflow.

↳ // find index

int index = front [qn]

↳ front [qn] = next [index]

// freespot manage.

↳ next [index] = free;

free = index

Q Sum of min Max element of k subarray :: size k.

i/p → { 2, 5, -1, 7, -3, -1, -2 }

k = 4

sum = 7+(1) = 6

sum = 4

sum = 4

→ O/P

Approach 1

For (0 → <n) {

for (k times)

{ max

min = sum store.

}

}

T(c) = O(n*k)

Approach 2

k size

→ deque → [max] → track → decreasing order of element range.

→ deque → [min] → track → increasing order of element hai

maxi. front → max^m element in k size window

mini. front ⇒ min element in k size window

{ 2, 5, -1, 7, -3, -1, -2 }

For (i = k <n) {

max = maxi. front(), mini

sum += max + min;

// next window

// removing

```
// removal
while ( maxi.front - i >= k)
        pop();
    while (mini-
    }
// addition
        (✓) same as previous.
}
sum += (   )  + (   )
}
return sum;
}
```

Code:-

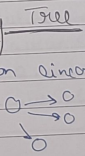
```
int solve (int *arr, int n, int k) {
    deque <int > maxi(k);
    deque <int > mini(k);
    //addition of first k size window
    for (int i=0; i<k; i++) {
        while (!maxi.empty() && arr[maxi.back()]<= arr[i]){
            maxi.pop_back();
        while (!mini.empty() && arr[mini.back()]>=arr[i]
            mini.pop_back();
        maxi.push_back(i);
        mini.push_back(i);
    }
    int ans = 0;
    ans += arr[maxi.front()] + arr[mini.front()];

    // remaining window by process here.
```

```
for (int i=k; i<n; i++){
    //next window
    //removal
    while (!maxi.empty() && i - maxi.front() >= k){
        maxi.pop_front();
    while (!mini.empty() && i - mini.front() >= k){
        mini.pop_front();
    }
    // addition.
    while (!maxi.empty() && arr[maxi.back()]<= arr[i]
        maxi.pop_back();
    while (!mini.empty() && arr[mini.back()] >= arr[i])
        mini.pop_back();

    maxi.push_back(i);
    mini.push_back(i);
    ans += arr[maxi.front()] + arr[mini.front()];
}
return ans;
}
```

$T(c) = O(n)$;

Binary Tree

↳ non linear DS

↳  multiple nodes.

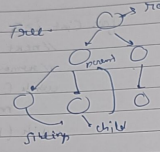
- Basic terms

1. Node
2. Root
3. children
4. parent
5. Siblings
6. Ancestors
7. Descendent
8. Leaf

~~Node~~
~~{~~
~~int data;~~
~~}~~

→ In Binary tree,
each node has
(<=2) child

Tree



Leaf → Iska koi child nahi hai.

```
Node {
    int data;
    node* left;
    node* right;
}
```

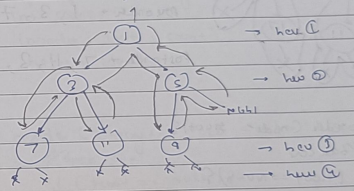
▲ n-ary tree

```
node {
    int data
    list < node* > child;
}
```

Creation of Binary tree

```
classnode {
    public:
        int data;
        node* left;
        node* right;

    node (int d){
        this -> data = d;
        this -> left = NULL;
        this -> right = NULL;
    }
};
```

```
~~node* buildtree () {~~
node* buildtree (node* root) {
    int data;
    cin >> data;
    root = new node (data);
    if (data == -1) {
        return NULL; }

    root -> left = buildtree (root -> left);
    root -> right = buildtree (root -> right);
    return root;
}
```



Level order traversal.

```
void level order traversal (node* root){
    queue < node* > q;
    q.push (root);
    while (! q.empty ()){
        node* temp = q.front();
        q.pop();
        if (temp -> left ) {
            q.push (temp -> left);
        }
        if (temp -> right) {
            q.push (temp -> right);
        }
    }
}
```

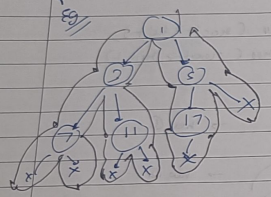
→ Full code in PC.

(H.W) → recursive level order traversal Code.

- Inorder traversal

Inorder	Pre order	Post order
LNR	NLR	LRN

eg



Inorder → 7, 3, 11, 3, 1, 17, 5

preorder → 1, 3, 7, 11, 5, 17

Postorder → 7, 11, 3, 17, 5, 1

Code

```
void inorder (node* root){
    //base case
    if (root == NULL){
        return;
    }
    inorder (root -> left);
    cout << root -> data << " ";
    inorder (root -> right);
}
```

∴ → Write Post order & Pre order ~~yourself~~ yourself //easy.

H.W → In, Pre, Post without using recursion.

- build from level order
  ↳ Code in PC.

Count leaf Nodes
↳ Is a node child no no.

Code

```
int no of LeafNode (BinaryTreeNode <int> * root){
    int count = 0;
    inorder (root, count);
    return count;
}

void inorder (BinaryTreeNode <int> * root, int &count){
    //base case
    if (root == NULL){
        return;
    }
    inorder (root -> left, count);
    // leaf node
    if (root -> left == NULL && root -> right == NULL){
        count++;
    }
    inorder (root -> right, count);
}
```

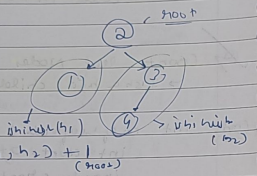
T(n) = O(n)

Height of Binary tree.

↳ longest path btw root node & leaf node.

total nodes in btw.

Approach 1 Recursion



$$ans = \max(h_1, h_2) + 1 \quad (\text{root})$$

Code

```
int height (struct Node* node) {
    // base case
    if (node == NULL) {
        return 0;
    }

    int left = height (node -> left);
    int right = height (node -> right);
    int ans = max (left, right) + 1;
    return ans;
}
```

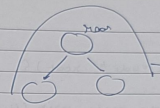
$T(c) = O(N)$

$S(c) = O(\text{height})$.

Diameter of a tree.

↳ longest path btw 2 end nodes.

Eg



→ ans = 3.

- longest path btw 2 end nodes.

can be leaf node, can be root node.

Approach → Diameter

- ans in left subtree
- ans in right subtree
- ans IS passing through root

op1 → diameter (root → left)

op2 → diameter (root → right)

op3 = height (root → left) + 1 + height (root → right)

Code :

```
int diameter (Node* root) {
    if (root == NULL) {
        return 0;
    }
    int op1 = diameter (root -> left);
    int op2 = diameter (root -> right);
    int op3 = height (root -> left) + height (root -> right) + 1;
    int ans = max (op1, max(op2, op3));
    return ans;
}
};
int height (      ) {
    // previous code //
}
```

~~$T(c) = O(n^2)$~~

$T(c) = O(n^2)$

optimization

```
pair<int, int> diameterfast (Node* root) {
    if (root == NULL) {
        pair<int, int> p = make_pair(0,0);
        return p;
    }
    pair<int, int> left = diameterfast (root -> left);
    pair<int, int> right = diameterfast (root -> right);
    int op1 = left.first;
    int op2 = right.first;
    int op3 = left.second + right.second + 1;

    pair<int, int> ans;
    ans.first = max(op1, max(op2, op3));
    ans.second = max(left.second, right.second) + 1;
    return ans;
}
int diameter (Node* root) {
    return diameterfast (root).first;
}
};
```

$T(c) = O(N)$

$S(c) = O(H)$

Q. check for balanced tree.

$\left[\text{height}(\text{left}) - \text{height}(\text{right})\right] <= 1$ → for (all nodes).

absolute

code:-

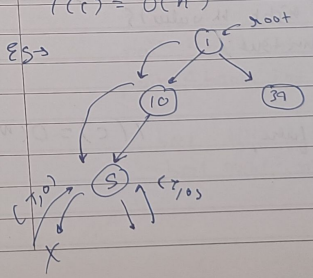
```
bool isbalanced (Node* root) {
    if (root == NULL) {
        return true;
    }
    bool left = isbalanced (root -> left);
    bool right = isbalanced (root -> right);
    bool diff = abs (height (root -> left)
                - height (root -> right)) <= 1;

    if (left && right && diff) {
        return 1;
    }
    else {
        return false
    }
}
int height (      ) {

    // previous code //

}
```

$T(c) = O(n^2)$

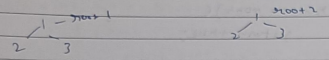
optimised code → in PC.

$T(c) = O(n)$

Eg →



Check if trees are identical or not.



Recursion

→ node value $\xrightarrow{T}$ left, right

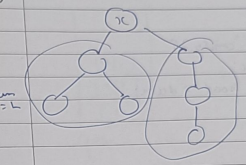
false → return false.

Code :- bool isIdentical (Node * r1, Node * r2) {
    if (r1 == NULL && r2 == NULL) {
        return true;
    }
    if (r1 == NULL && r2 != NULL) {
        return false;
    }
    if (r1 != NULL && r2 == NULL) {
        return false;
    }
    bool left = isIdentical (r1 → left, r2 → left);
    bool right = isIdentical (r1 → right, r2 → right);

    bool value = r1 → data == r2 → data;
    if (left && right && value) {
        return true;
    }
    else {
        return false;
    }
}

$T(c) = O(N)$

Sum Tree



x = L + R

→ check for each node.

→ isSum (root → left); → T
→ isSum (root → right); → T

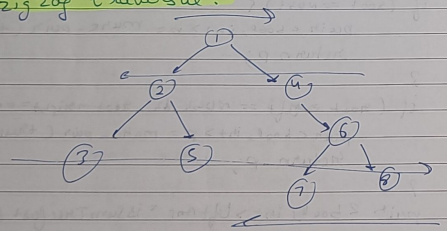
(w/ Node
    root → data
    L + R)

Code pair <bool, int> isSumTreefast (Node * root) {
    if (root == NULL) {
        pair <bool, int> p = make_pair (true, 0);
        return p;
    }
    if (root → left == NULL && root → right == NULL) {
        pair <bool, int> p = make_pair (true, root → data);
        return p;
    }
    pair <bool, int> leftAns = isSumTreefast (root → left);
    pair <bool, int> RightAns = isSumTreefast (root → right);

    bool left = leftAns.first;
    bool right = rightAns.first;

```
bool condn = root -> data == left Ans.second + right Ans.second;

pair<bool, int> ans;
if (left && right && condn){
    ans.first = true;
    ans.second = 2* root -> data;
}
else {
    ans.first = false;
}
return ans;
}

bool isSumTree (Node* root){
    return isSumtreepair (root).first;
}
```

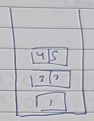
Q. zigzag traversal.



ans: 1 4 2 3 5 6 8 7

L to R then R to L
↳ repeat.

Dsmunch :- through level order traversal.



queue

flag
↳ left to right
- True → ~~child~~ right to left
- False → left to right.

```
(code) vector<int> zigzag traversal (Node* root){
    vector<int> result;
    if (root == NULL){
        return result;
    }
    queue<Node*> q;
    q.push(root);          bool leftToRight = true;
    while (!q.empty()){
        int size = q.size();
        vector<int> ans(size);
        // level order
        for (int i = 0; i < size; i++){
            Node* frontNode = q.front();
            q.pop();
            int index = leftToRight ? i : size - i - 1;
            ans[index] = frontNode -> data;
            if (frontNode -> left)
                q.push(frontNode -> left);
            if (frontNode -> right)
                q.push(frontNode -> right);
        }
        // direction change.
        leftToRight = !leftToRight;
```

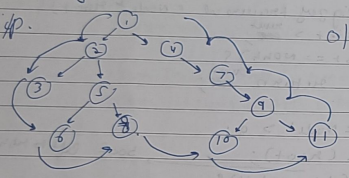
```
for (auto i : ans) {
        result.push_back(i);
    }
    return result;
}
};
```

$T(C) = O(N)$

$S(C) = O(N)$.

Boundary traversal

Ip.



O/p → 1, 2, 3, 6, 8, 10, 11, 9, 7, 4,

algo → 1. left part (without leaf node).
2. leaf node print
3. Right node print (reverse order). (without leaf).

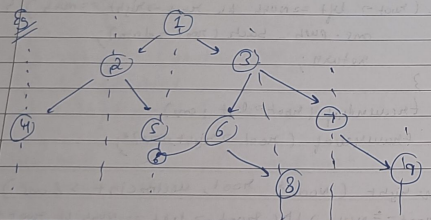
left part	leaf part
→ root → print	
- if (left)	
↳ left call	
→ else	
↳ right call	

```
void traversleft (Node* root, vector<int> &ans) {
    if (root == NULL || (root -> left == NULL && root -> right == NULL)) {
        return;
    }
    ans.push_back (root -> data);
    if (root -> left)
        traversleft (root -> left, ans);
    else {
        traversleft (root -> right, ans);
    }

void traversleaf (Node* root, vector<int> &ans) {
    if (root == NULL)
        return;

    if (root -> left == NULL && root -> right == NULL) {
        ans.push_back (root -> data);
        return;
    }
    traversleaf (root -> left, ans);
    traversleaf (root -> right, ans);
}

void traverse right (Node* root, vector<int> &ans) {
    if (root == NULL || (root -> left == NULL && root -> right == NULL))
        return;
    if (root -> right)
        traverse right (root -> right, ans);
    else
        traverse right (root -> left, ans);

    ans.push_back (root -> data);
}
```

```
vector <int> boundary (node* root) {
    vector <int> ans;
    if (root == NULL)
        return ans;
    ans.push_back (root -> data);

    traversleft (root -> left, ans);
    traversleaf (root -> right, ans);
    traversright (root -> right, ans);

    return ans;
}
};
```

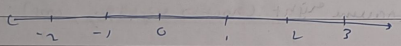
Q vertical order traversal.

Eg



ans → 4, 2, 1, 5, 6, 3, 8, 7, 9

approach



(number line)

→ horizontal distance ग्रहण kay gya.

<0> → 1, 5, 6   <-2> = 4

<-1> - 2   <2> = 7

<1> - 3, 8   <3> = 9 → print sorted.

⇒ mapping

↳ of hori, on tree structure + boundaries.

→ map <int, map <int, vector <int>>> nodes;

H.D   level   list of nodes.

⇒ required: H.D, level

→ queue <Node*, pair <int, int>>;

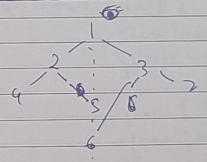
<pair

Code :→ 2n PC

(Tough Question).

Q Top view of BT

Eg



O/P → 4 2 1 3 7

Approach - using map. (Just like previous one).

(remove hidden nodes)

(one to one mapping only).

Code :- vector <int> topview (node* root) {

```
    vector <int> ans;
    if (root == NULL)
        return ans;

    map <int, int> topNode;
    queue <pair <Node*, int>> q;
```

```
q.push (make_pair (root, 0));
while (!q.empty ()) {
    pair<Node*, int> temp = q.front();
    Node* frontNode = temp.first;
    int hd = temp.second;

    if (topNode.find (hd) == topNode.end());
        topNode[hd] = frontNode->data;
    if (frontNode -> left)
        q.push (make_pair (frontNode->left, hd-1));
    if (frontNode -> right)
        q.push (make_pair (frontNode->right, hd+1));
}
for (auto i: topNode) {
    ans.push_back (i.second);
}return ans;
}
};
```

q.pop();

Bottom view in BT.

Algo → horizontal distance ki daalte jao, pichle vaale ko kaat te jao.

Aathiri ki print karado.

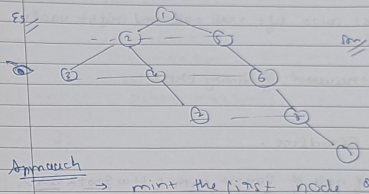
Code → Remove above * line.

Rest code is same.

→ Also code in PC.

left view in BT.

Eg



op → 1 2 3 7 9

Approach

→ print the first node of each level.

level ++;

↳ track each level

↳ func (root, level) {

```
    //base case
    if (root == NULL)
        return;
    func (left), level+);
    func (right, level+);
```

* if (level == vector.size())
⇒ entered a new level.
↳ store data.

Code

```
void solve (Node* root, vector<int> &ans, int level) {
    if (root == NULL)
        return;
    if (level == ans.size())
        ans.push_back (root -> data);
    solve (root->left, ans, level+1);
    solve (root->right, ans, level+1);
}

vector<int> leftview (Node* root) {
    vector<int> ans;
    solve (root, ans, 0);
    return ans;
}
```

Q. Right view in BT

Code → Just swap left vaali call and right vaali call.

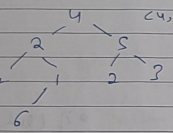
H.W → Diagonal traversal of Binary tree.

Q. Sum of longest Bloodline.

↓

sum of nodes of longest path (b/w root & leaf).

Approach



<4, 1>

< sum, length of path >

↳ has path ki Traverse karlo. to find < sum, len >

↳ store in ans.

Code:

```
void solve (Node* root, int sum, int & maxsum, int len,
                                    int &maxlen){
    if (root == NULL){
        if (len > maxlen){
            max len = len;
            max sum = sum;
        }
        else if (len == maxlen){
            maxsum = max(sum, maxsum);
        }
        return;
    }
    sum = sum + root->data;
    solve(root->left, sum, maxsum, len+1, maxlen);
    solve(root->right, sum, maxsum, len+1, maxlen);
}
```

```
int sumofLongRootToLeafpath (Node* root){
    int len = 0;
    int maxlen = 0;
    int sum = 0;
    int maxsum = INT_MIN;
    solve (root, sum, maxsum, len, maxlen);
    return maxsum;
}
```

$T(c) = O(N)$

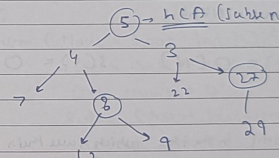
$S(c) = O(h)$

$h <= n$

Q. lowest common ancestor in Binary tree.

i/p → node1, node2

Es →



→ LCA (Sabse nehla mila).

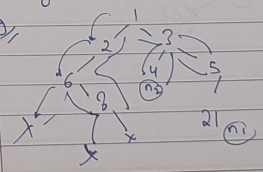
$8 = n_1$

$27 = n_2$

Approach → Root to n1 → path
Root to n2 → path } → find common LCA.

→ Using Recursion.

Eg//



leftAns ← rec → NULL

rightAns ← rec → NULL

return NULL

if left == NULL && right == NULL
right ++

when both (!NULL) → Ans.

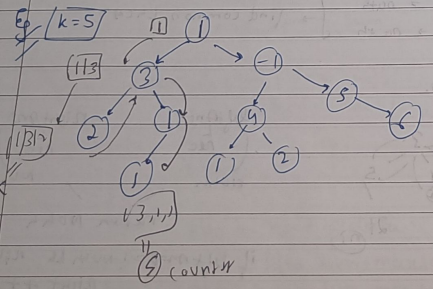
Code:-

```
Node* lca (Node* root, int n1, int n2){
    if (root==NULL)
        return NULL;
    if (root->data == n1 || root->data == n2)
        return root;
    Node* leftans = lca (root->left, n1, n2);
    Node* rightans = lca (root->right, n1, n2);

    if (leftans != NULL && rightans != NULL)
        return root;
    else if (leftans != NULL && rightans == NULL)
        return leftans;
    else if (leftans == NULL && rightans != NULL)
        return rightans;
    else
        return NULL;
};
```

$T(c) = O(N)$ $S(c) = O(h)$

Q K Sum path.

- Find no. of paths in the tree which have their sum equal to k. (from any node to any node).

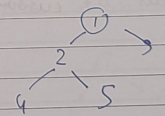


Approach :-

Code -

```
void solve (Node* root, int k, int kcount, vector<int> path){
    if (root == NULL)
        return;
    path.push_back (root->data);
    solve (root->left, k, count, path);
    solve (root->right, k, count, path);
    //check for k sum
    int size = path.size(), sum = 0;
    for (int i = size-1; i >= 0; i--){
        sum = sum + path[i];
        count++;
    }
    path.pop_back();
}
int sum (Node* root, int k) {
    vector<int> path;
    int count = 0;
    solve (root, k, count, path);
    return count;
}
};
```

Q kth Ancestor in BTree.

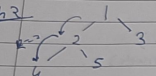


k=2 for node 4

⇒ Ans = 1.

Approach → Path nikal do root node se kisaat node tak.
and store in vector.
→ Fhir ans nikal do (I.W)

Approach 2



$\boxed{k=2}$ $\boxed{n=4}$

1. find node.

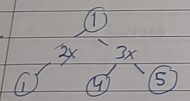
left ans right ans
1 NULL NULL
→ ==0 (true) root
→ !=0 (1 NULL value ki upper bhej do).

Code: Code in PC
↳ k^{th} Ancestor.

$T(c) = O(N)$
$S(c) = O(h)$.

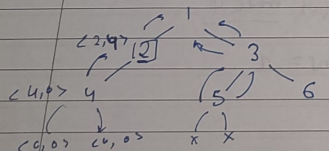
Max sum of Non adjacent nodes. → adjacent (connected node nahi lo sakte).

Ex



ans = 1 2 5
$\boxed{max = 11}$

approach → < a, b >
a → max sum by including nodes at current level.
b → max sum excluding nodes at current level.



⇒ 14th June

Code: → pair <int, int> solve (Node* root) {
if (root == NULL) {
pair <int, int> p = make_pair (0, 0);
return p;
}
pair <int, int> left = solve (root → left);
pair <int, int> right = solve (root → right);
pair <int, int> res;

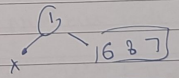
res.first = root → data + left.second + right.second;
res.second = max (left.first, left.second) +
max (right.first, right.second);

return res;
}
int getMaxSum (Node* root) {
pair <int, int> ans = solve (root);
return max (ans.first, ans.second);
}
};

Hard questions.

Q i/p → inorder [1 6 8 7] LNR
preorder [0 6 7 8] NLR

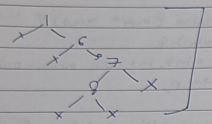
O/p → make tree using above.



$\boxed{6|7|8}$ now inorder $\boxed{6|7|8}$
preorder $\boxed{6|7|8}$

Sub problem Banegi

⇒ in =



post order = 8 7 6 1

Algo ⇒ take index of preorder as root element.
↳ find root element ki position in inorder
↳ root → left
(inorder start, inorder (pos − 1))
↳ root → right
(

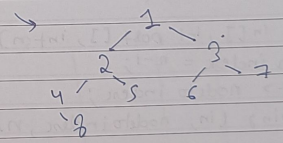
Code

```
Node* solve (int in[], int pre[], int &index,
            int inorderstart, int inorder end, int n){
    //base case
    if (index >= n || inorderstart > inorder end){
        return NULL;
    }
    int element = pre[index++];
    Node* root = new Node(element);
    int position = find position (in, element, n);
    //recursion call
    root -> left = solve (in, pre, index, inorderstart,
                          position -1, n);
    root -> right = solve (in, pre, index, position+1,
                           inorder end, n);
    return root;
}
```

```
int findposition (int in[], int element, int n){
    for (int i=0; i<n; i++){
        if (in[i] == element)
            return i;
    }
    return -1;
}
Node* buildtree (int in[], int pre[], int n){
    int preorder index = 0;
    Node* ans = solve (in, pre, preorder index, 0, n-1, n);
    return ans;
}
};
```

Build tree using inorder & postorder traversals.

Eg. - in [] = [4, 8, 2, 5, 1, 6, 3, 7] → LNR
post [] = [8, 4, 5, 2, 6, 7, 3, 1] → LRN
↑ logic
end ⇒ start.



Code - previous code with few changes.

~~H.W. - Write yourself.~~

→ next page

Code:

```
void createmapping (int in[], map<int,int> &nodetoindex,
                    int n){
        for (int i=0; i<n; i++){
            nodetoindex [in[i]] = i;
        }
}

Node* solve (int in[], int post[], int &index,
             int inorderstart, int inorderend, int n,
             map<int, int> & nodetoindex){
    if (index < 0 || inorderstart > inorderend){
        return NULL;
    }
    int element = post[index--];
    Node* root = new Node(element);
    int position = nodetoindex (element);
//recursive call// //Right subtree पहले करेंगे//:
    root->right = solve (in, post, index, position+1,
                         inorderend, n, nodetoindex);
    root->left = solve (in, post, index, inorderstart,
                        position-1, n, nodetoindex);
    return root;
}

Node* buildtree (int in[], int post[], int n){
    int postorderindex = n-1;
    map<int, int> nodetoindex;
    createmapping (in, nodetoindex, n);

    Node* ans = solve (in, post, postorderindex,
                       0, n-1, nodetoindex);
    return ans;
}
};
```

$T(c) = O(n \log n)$.

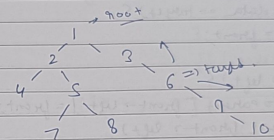
Q) Burning Tree.

↳ target given.

Find min time required to burn the BT if target is set on fire.

(in 1 second, all nodes connected to a given node get burned).

Eg//



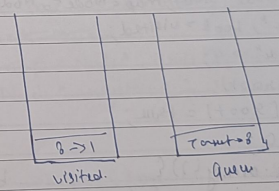
Ans = 5 sec

Approach

1. → node to parent mapping create.
   map < Node*, Node* > nodetoParent
   ↳ using level order traversal.] O(N)

2. Find target Node.] O(n)

3.



4. If any add^n in queue, True → time ++;

Code :-

```
Node* create Parent mapping (Node* root, int target,
                             map<Node*, Node*> & nodeToParent){
    Node* res = NULL;
    queue<Node*> q;
    q.push(root);
    nodeToParent[root] = NULL;
    while (!q.empty()){
        Node* front = q.front();
        q.pop();
        if(front -> data == target){
            res = front;
        }
        if(front -> left){
            nodeToParent[front -> left] = front;
            q.push(front -> left);
        }
        if(front -> right){
            nodeToParent[front -> right] = front;
            q.push(front -> right);
        }
    }
    return res;
}

int ~~void~~ burnTree(Node* root, map<Node*, Node*> & nodeToParent){
    map<Node*, bool> visited;
    queue<Node*> q;
    q.push(root);
    visited[root] = true;
    int ans = 0;
    while (!q.empty()){                    bool flag = 0;
        int size = q.size();
        for (int i = 0; i < size; i++){
            Node* front = q.front();
            q.pop();
            if(front -> left && !visited[front -> left]){
                q.push(front -> left);
                visited[front -> left] = 1;
                flag = 1;
            }
            if(front -> right && !visited[front -> right]){
                q.push(front -> right);
                visited[front -> right] = 1;
                flag = 1;
            }
            if(nodeToParent[front] && !visited[nodeToParent[front]]){
                q.push(nodeToParent[front]);
                visited[nodeToParent[front]] = 1;
                flag = 1;
            }
        }
        if(flag == 1){
            ans++;
        }
    }
    return ans;
}

int minTime(Node* root, int target){
    ~~int ans = 0~~;
    map<Node*, Node*> nodeToParent;
    Node* targetNode = create Parent Mapping(root,
                                             target,
                                             nodeToParent);
    int ans = burnTree(targetNode, nodeToParent);
    return ans;
}
```

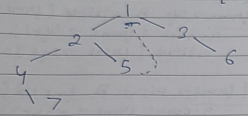
$T(c) = O(N \log N)$

$S(c) = O(N)$.

Concept

Morris Traversal

$T(c) = O(N)$
$S(c) = O(1)$



(temporary link)

Algo ⇒ curr = root
while (~~root~~ curr != NULL)
  if left not exists visit (current)
    current = current → right
  else
    predecessor = find (curr)
    if (pred → right == NULL)
      pred → right = current
      current = current → left
    else
      pred → right = NULL
      visit (current)
      current = current → right.

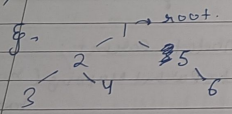
temp; link creation

temp link hatane hai

//

Q/ Flatten a BT to LL. (in place) → O(1) space.

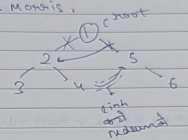
Eg.



Approach 1 → traverse → each each node create hote / memory

Approach 2 → Recursion → O(N) ✗

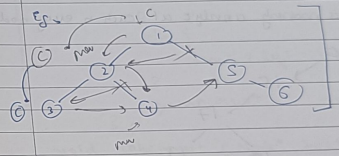
Approach = Morris.



Pre-order → NLR (Ar ⇒ remain)

inorder mein.
↳ right = curr → right

Eg.



⇒ explanation of Morris.

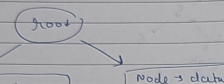
1 2 3 4 5 6

Code ⇒ void flatten (Node * root) {
  Node * curr = root;
  while (curr != NULL) {
    if (curr → left) {
      Node * pred = curr → left;
      while (pred → right)
        pred = pred → right;
      pred → right = curr → right;
      curr → right = curr → left;
    }
    curr = curr → right;
  }
  // left part NULL
  curr = root;
  while (curr != NULL) {
    curr → left = NULL;
    curr = curr → right;
  }
}

15th June

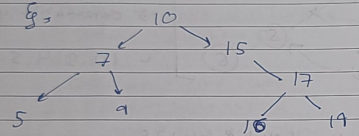
lecture 69

Binary Search Tree



↳ This property applies at each n every node

Eg =



Algo → no. → "n"

→ common with root → data.

> (right part)

< left part.

Code Implementation.

```
class Node {
public:
    int data;
    Node* left; Node* right;
    Node (int d) {
        this -> data = d;
        this -> left = NULL;
        this -> right = NULL;
    }
};
```

$T(n) = O(\log n)$.

```
Node* insertinBST (Node* & root, int d) {
    //base case
    if (root == NULL) {
        root = new Node (d);
        return root;
    }
    if (d > root -> data) {
        root -> right = insertinBST (root -> right, d);
    }
    else {
        root -> left = insertinBST (root -> left, d);
    }
    return root;
}

void takeinput (Node* root) {
    int data;
    cin >> data;
    while (data != -1) {
        root = insertinBST (root, data);
        cin >> data;
    }
}

int main() {
    Node * root = NULL;
    cout << "Enter data to create" << endl;
    takeinput (root);
    return 0;
}
```

// for printing →

copy code of level order traversal → pre, post, in.

```
int main() {
    levelorder traversal (root);
```

Search in BST.

Algo → if reach NULL → return false
→ if root → data > x → left root.
else right root.

Code - bool searchinBST (BinaryTreeNode <int> * root, int x){
if (root == NULL){
return false;
}
if (root → data == x){
return true;
}
if (root → data > x){
return searchinBST (root → left, x);
}
else{
return searchinBST (root → right, x);
}
};

$T(c) = O(H)$
$S(c) = O(H)$

→ optimization.
Iterative sol^n ⇒ $S(c) = O(1)$.

Code bool searchinBST (BinaryTreeNode <int> * root, int x){
BinaryTreeNode <int> * temp = root;
while (temp != NULL){
if (temp → data == x){
return true;
}
if (temp → data > x){
temp = temp → left;
}
else{
temp = temp → right;
}
}
return false;
}
};

Inorder (LNR) of BST is sorted always.

minvalue

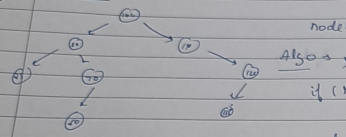
temp = root;
while (temp → left != NULL)
temp = temp → left;
}
return temp;

max value

temp = root
while (temp → right != NULL){
temp = temp → right;
}
return temp;
}

H.W. Inorder predecessor & inorder successor.
Code →

Deletion



node to delete = 120.

Algo → reach that node.

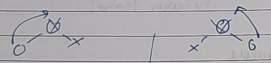
```
if (root -> data == x) {
}
else if (root -> data > x)
    { left + root
else
    right root
}
```

→ Node to delete

↳ 0 child

→ delete node;
return NULL

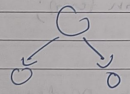
↳ 1 child



temp = root -> left
delete root;
return temp;

same

→ 2 children.



1. left ka max ka replace karo
   → us node ko delete karde
2. right ka min value utharlo
   ↳ phir usko delete karda.

→ Need to write minval fun in below code also.

```
(or) Node* deleteFromBST (Node* root, int val){
    //base case
    if (root == NULL)
        return root;
    if (root -> data == val){
        //0 child
        if (root -> left == NULL && root -> right == NULL){
            delete root;
            return NULL;
        }
        // 1 child
        //left child
        if (root -> left != NULL && root -> right == NULL){
            Node* temp = root -> left;
            delete root;
            return temp;
        }
        // right child
        if (root -> left == NULL && root -> right != NULL){
            Node* temp = root -> right;
            delete root;
            return temp;
        }
        // 2 child
        if (root -> left != NULL && root -> right != NULL){
            int mini = minval (root -> right) -> data;
            root -> data = mini;
            root -> right = deleteFromBST (root -> right, mini
            return root;
        }
    else if (root -> data > val){
        root -> left = deleteFromBST (root -> left, val)
        return root;
```

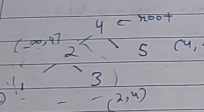
```
    else {
        root->right = deleteFromBST(root->right, val);
        return root;
    }
}
```

Lecture 70 [Questions];

Q. validate a BST?

Approach (1) → find inorder traversal
↳ if sorted → true
else → false

Approach (2)



→ satisfy ranges
→ true
else
false;

Code :→

```
bool isBST (BinaryTreeNode <int> *root, int min, int max){
    //base case
    if (root == NULL)
        return true;
    if (root->data >= min && root->data <= max){
        bool left = isBST(root->left, min, root->data);
        bool right = isBST(root->right, root->data, max);
        return left && right;
    }
    else {
        return false;
    }
}
```

```
bool validBST (BinaryTreeNode <int> *root){
    return isBST(root, INT_MIN, INT_MAX);
}
```

$T(c) = O(N)$

$S(c) = O(H)$

Q. Find kth smallest element.

k = 3

Approach 1
⇒ Inorder is sorted
⇒ 2 3 4 5 6 7 8 10
Ans.

Code :→

```
int solve (BinaryTreeNode <int> *root, int& i, int k){
    //base case
    if (root == NULL){
        return -1;
    }
    int left = solve(root->left, i, k);
    if (left != -1){
        return left;
    }
    i++;
    if (i == k)
        return root->data;
    return solve(root->right, i, k);
}
int kthSmallest (BinaryTreeNode <int> *root, int i, int k){
    int ans = solve(root, i, k);
    return ans;
}
```

$T(c) = O(N)$

$S(c) = O(H)$

H-W → solve above question using Morris Traversal.

Predecessor & Successor in BST

Algo → solve inorder of tree in a Q.S. then traverse the key to find.

$T(n) = O(N)$
$S(n) = O(N)$.

Approach 2

↳ search key (node find out).

↳ predecessor → left subtree → max element

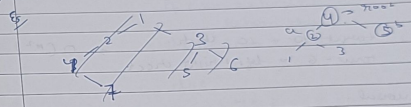
↳ successor → right subtree → min element

Code :→

```
pair<int,int> predecessorSuccessor (Binary Tree Node <int> * root, int key) {

    Binary Tree Node <int> * temp = root;
    int pred = -1;
    int succ = -1;

    while (temp -> data != key) {
        if (temp -> data > key) {
            ~~temp = temp -> left;~~
            succ = temp -> data;
            temp = temp -> left;
        }
        else {
            ~~temp = temp -> right;~~
            pred = temp -> data;
            temp = temp -> right;
        }
    }

    Binary Tree Node <int> * leftTree = temp -> left;
    while (leftTree != NULL) {
        pred = leftTree -> data;
        leftTree = leftTree -> right;
    }
    Binary tree Node <int> * rightTree = temp -> right;
    while (rightTree != NULL) {
        succ = rightTree -> data;
        rightTree = rightTree -> left;
    }

    pair<int,int> ans = make_pair(pred, succ);
    return ans;
```

$T(n) = O(N)$
$S(n) = O(1)$

LCA in a BST (lowest common ancestor).



LCA of {a, b} = { }

Approach → 1. root < a && root < b
↳ right part of Tree

2. root > a && root > b
↳ left part of Tree

3. root > a && root < b
or
root < a && root > b
→ root → return.

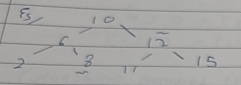
Code :→

```
TreeNode <int> * LCAinBST (TreeNode <int> * root, TreeNode <int> * p,
                           TreeNode <int> * q) {

    if (root == NULL)
        return NULL;
    if (root -> data < p -> data && root -> data < q -> data)
        return LCAinBST (root -> right, p, q);
    if (root -> data > p -> data && root -> data > q -> data)
        return LCAinBST (root -> left, p, q);

    return root;
}
```

use while (root != NULL) { } [for iterative sol^n

Two Sum in a BST.

Ex



target = 20.

↳ 8 + 12

→ true.

approach → Brute

10, target - 10 = 10 check. → $O(N^2)$

6, target - 6 = left, right check

approach 2 → single traversal.

↳ find inorder traversal ⇒ {2, 6, 8, 10, 12, 11, 15}

$T(c) = O(N)$

$S(c) = O(N)$

↳ arr[i] + arr[j] → sum

sum == target

→ return arr.

↳ if sum > target.

j--

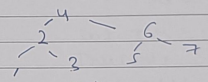
else

i++;

```
void inorder (BinaryTreeNode<int>* root, vector<int> &in){
    if (root==NULL)
        return;
    inorder (root->left, in);
    in.push_back (root->data);
    inorder (root->right, in);
}
```

```
bool twosumInBST (BinaryTreeNode<int>* root, int target){
    vector<int> inorder_val;
    inorder (root, inorder_val);
    int i = 0, j = inorder_val.size() - 1;

    while (i<j) {
        int sum = inorder_val[i] + inorder_val[j];
        if (sum == target)
            return true;
        else if (sum > target)
            j--;
        else
            i++;
    }
    return false;
}
```

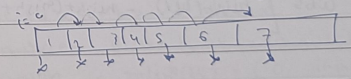
Flatten BST to sorted LL



↳ root 1 → 2 — 3 — 4 — 5 — 6 — 7 → X

approach → save inorder in an array. ✓ vector<Node*>

1.



2.) last element (both pointer to NULL).

$T(c) = O(N)$

$S(c) = O(N)$

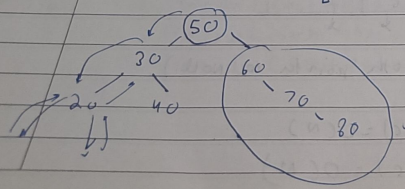
Code :-

```
void inorder {
        (same as question nothing).
    }

TreeNode <int> * flatten (TreeNode <int> *root){
    vector <int> inorderval;
    inorder (root, inorderval);        int n = inorderval.size();
    TreeNode <int> * new root = ~~new TreeNode<int>~~
                                new TreeNode <int> (inorderval[0]);
    TreeNode <int> * curr = newroot;
    for (int i = 1; i<n; i++){
        TreeNode <int> * temp = new TreeNode <int> (inorderval[i]);
        curr -> left = NULL;
        curr -> right = temp;
        curr = temp;
    }
    curr -> left = NULL;
    curr -> right = NULL;
    return newroot;
}
};
```

Convert to Balanced BST.

Eg. ↳ For any node

abs [height(left) − height(right)] <= 1;



→ Unbalanced

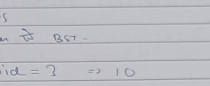
Approach:- 1. save inorder

2, 4, 6, 10, 12, 16, 20

now question is

inorder to BST.

↳ mid = 3 ⇒ 10



→ Recursion recursion & extra space.

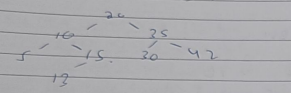
Code :-

```
void inorder {
        // same as previous question.
    }

TreeNode <int> * inorderToBST (int s, int e, vector<int> &in){
    if (s>e)
        return NULL;

    int mid = (s+e)/2;
    TreeNode <int>* root = new TreeNode<int> (in[mid]);
    root -> left = inorderToBST (s, mid-1, in);
    root -> right = inorderToBST (mid+1, e, in);
    return root;
}
TreeNode <int> * balancedBST (TreeNode <int>* root){
    vector <int> inorderval;
    inorder (root, inorderval);
    return inorderToBST (0, inorderval.size()-1, inorderval)
}
```

$T(n) = O(N)$.

BST from Preorder

Approach 1 [20, 10, 5, 15, 13, 35, 30, 42]



→ $O(N^2)$.

Approach 2 1. Find inorder using sort(Preorder).

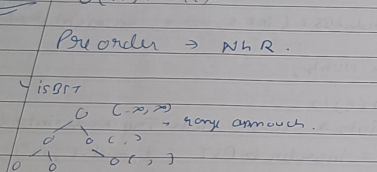
2. Create tree using preorder & inorder.

$T(c) = O(N \log N)$.

Approach 3 $O(N)$. ← $T(c)$

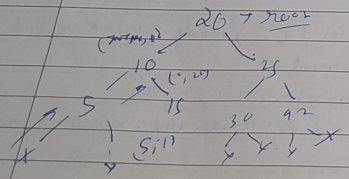
Preorder → N L R.

isBST



→ range approach.

→ INT_MIN, INT_MAX]

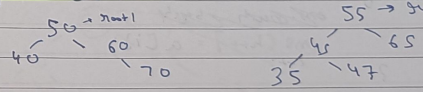


$O(N) + T(c)$

```
(solve - binaryTreeNode <int> * solve (vector<int> & preorder, int mini,
                                       int maxi, int &i)
{
    if (i >= preorder.size())
        return NULL;
    if (preorder[i] < mini || preorder[i] > maxi)
        return NULL;

    BinaryTreeNode <int> * root = new BinaryTreeNode
                                  <int> (preorder[i]);
    root -> left = solve (preorder, mini, root -> data, i);
    root -> right = solve (preorder, root -> data, maxi, i);
    return root;
}
BinaryTreeNode <int> * preorderToBST (vector<int> & preorder)
{
    int mini = INT_MIN;
    int maxi = INT_MAX;
    int i = 0;
    return solve ( preorder, mini, maxi, i);
}
```

IMP

Merge 2 BST.



Approach 1 - Inorder to BST.

1. → Store inorder of both
2. → merge 2 sorted arrays.
3. → inorder to BST convert.

$T(c) = O(M+N)$

$S(c) = O(M+N)$

Code:-

```
void inorder (TreeNode<int> * root, vector<int> &in){
    if (root == NULL)
        return;
    inorder (root -> left, in);
    in.push_back (root -> data);
    inorder (root -> right, in);
}
TreeNode<int> * mergeBST (TreeNode<int>* root1,
                          TreeNode<int> * root2){
    vector<int> bst1, bst2;
    inorder (root1, bst1);
    inorder (root2, bst2);

    ~~mergeArray~~
    vector<int> mergeArray = mergeArrays (bst1, bst2);
    ~~return mergeArray;~~ int s=0, e= mergeArray.size()-1;
    return inorderToBST (s, e, mergeArray);
}
vector<int> mergeArrays (vector<int>&a, vector<int>&b){
    vector<int> ans (a.size() + b.size());
    int i=0, j=0, k=0;
    while (i<a.size() && j<b.size()){
        if (a[i] < b[j]){
            ~~ans.push_back~~
            ans [k++] = a[i];
            i++;
        }
        else {
            ans [k++] = b[j++];
        }
    }
    while (i<a.size()){
        ans [k++] = a[i];
        i++;
    }
    while (j<b.size()){
        ans [k++] = b[j];
        j++;
    }
    return ans;
}

TreeNode<int>* inorderToBST () {
    // from inorder traversal.
}
```

$T(c) = O(m+n)$ & $S(c)$.

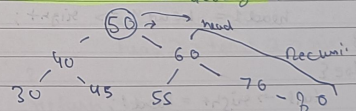
Approach-2 $S(c) = O(h_1 + h_2)$.

↳ i/p → root1, root2

1. flatten BST to sorted LL.
2. merge 2 sorted linked list. SC
3. convert sorted LL to BST.
   (Traversal).

Subquestion:

(1) convert a BST to doubly LL.



algo → convert right subtree into LL → (head of LL).

→ root → right = head;

head → left = root;

→ head → root;

→ convert left subtree into LL → (head), return.

Code

```
void convertintosorteddll ( TreeNode<int> *root,
                            TreeNode<int> *&head) {
    if (root == NULL)
        return NULL;

    convertintosorteddll (root->right, head);
    root->right = head;
    if (head != NULL)
        head->left = root;
    head = root;

    convertintosorteddll (root->left, head);
}
```

Subquestion 2 // Merging the sorted lists.

```
TreeNode<int> mergelinkedlist (TreeNode<int> *head1,
                               TreeNode<int> *head2) {
    TreeNode<int> *head = NULL;
    TreeNode<int> *tail = NULL;
    while (head1 != NULL && head2 != NULL) {
        if (head1->data < head2->data) {
            if (head == NULL) {
                head = head1;
                tail = head1;
                head1 = head1->right;
            }
            else {
                tail->right = head1;
                head1->left = tail;
                tail = head1;
                head1 = head1->right;
            }
        }
        else {
            if (head == NULL) {
                head = head2;
                tail = head2;
                head1 = head2->right;
            }
            else {
                tail->right = head2;
                head2->left = tail;
                tail = head2;
                head2 = head2->right;
            }
        }
    }
    while (head1 != NULL) {
        tail->right = head1;
        head1->left = tail;
        tail = head1;
        head1 = head1->right;
    }
    while (head2 != NULL) {
        tail->right = head2;
        head2->left = tail;
        tail = head2;
        head2 = head2->right;
    }
    return head;
}
```

Subquestion 3

// Sorted linked list to BST.

Algo →
1. Starting at n/2 node → * left + tree
2. Create root
   root → left = left tree
3. head → next
4. Right subtree.

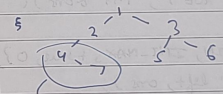
Code :→ ~~Tree~~

```
int countNodes (TreeNode <int> * head){
    int count = 0;
    TreeNode <int> * temp = head;
    while (temp != NULL){
        count++;
        temp = temp -> right;
    }
    return count;
}
TreeNode <int> * sortedLLtoBST (TreeNode <int> * & head,
                                int n){
    if (n <= 0 || head == NULL)
        return NULL;
    TreeNode <int> * left = sortedLLtoBST (head, n/2);
    TreeNode <int> * root = head;
    root -> left = left;
    head = head -> next;
    root -> right = sortedLLtoBST (head, n-n/2-1);
    return root;
}

//final function.

*/ next step /*
```

```
TreeNode <int> * mergeBST (TreeNode <int> * root1,
                           TreeNode <int> * root2){
    TreeNode <int> * head1 = NULL;
    convertIntoSortedDLL (root1, head1);
    head1 -> left = NULL;

    TreeNode <int> * head2 = NULL;
    convertIntoSortedDLL (root2, head2);
    head2 -> left = NULL;

    TreeNode <int> * head = mergeLinkedList (head1, head2);

    return sortedLLtoBST (head, countNodes (head));
}
```

lecture 17

Largest BST in a Binary tree.
↳ return size of largest BST.



Ans = 2

Approach ① → every node, check valid BST,
→ store size.
→ Return max size.

$T(c) = O(N^2)$.

Approach 2 - optimised.

↳ a node can be a valid BST if.

1. left subtree is valid BST → O(1)
2. right — — — → O(1)
3. max (left) < root → data < right (min).

info we need
int ↳ max, min
bool ↳ is BST
↳ size.

Code :→ class info {
Public:
int maxi;
int mini;
bool isBST;
int size;
};

info solve (TreeNode <int> * root, int &ans) {
if (root == NULL) {
return { INT_MIN, INT_MAX, true, 0};
~~int~~ info left = solve (root → left, ans);
~~int~~ info right = solve (root → right, ans);

info currNode;
currNode.size = left.size + right.size + 1;
~~currNode.size = max~~
currNode.maxi = max (root → data, right.maxi);
currNode.mini = min (root → data, left.mini);

if (left.isBST && right.isBST &&
(root → data > left.maxi && root → data < right.mini))
{
curr.Node.isBST = true;
}
else
currNode.isBST = false;

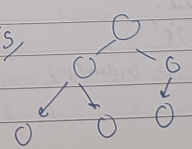
// ans update.
if (currNode.isBST) {
ans = max (ans, currNode.size);
}return currnode;
}
int largestBST (TreeNode <int> * root) {
int max size = 0;
infotemp = solve (root, maxsize);
return maxsize;
}

Lecture 74

Heaps
(what)
→ is a complete Binary tree that comes with a heap order property.

⊙ CBT ⇒ · every level is fully filled except the last level
· · nodes always added from the left,

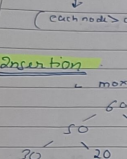
Eg,



o heap order property:
↳ maxi or mini
(each node > child) (each node < child)

Insertion

↳ max heap:



imp

Node → i^{th} index
then left child → $2*i^{th}$ index
right child = $2*i^{th}+1$ index
parent = $(i/2)^{th}$ index

→ using array.

x	60	50	40	30	20	55
0	1	2	3	4	5	6

→ whatever value to be inserted first it in back side

1. insert at end.
2. take it to correct position.
↳ compare with parent $(i/2)$.

Code

```
class heap {
    public:
        int arr[100];
        int size;
        heap() {
            arr[0] = -1;
            size = 0;
        }
        void insert(int val) {
            size = size + 1;
            int index = size;
            arr[index] = val;
            while (index > 1) {
                int parent = index/2;
                if (arr[parent] < arr[index]) {
                    swap(arr[parent], arr[index]);
                    index = parent;
                }
                else {
                    return;
                }
            }
        }
        void print() {
            for (int i=1; i<=size; i++) {
                cout << arr[i] << " ";
            } cout<<endl;
        }
};
int main() {
    h.insert();
    ↳ put value.
```

T.C → $O(\log n)$

Deletion

1. Swap the root node with the last node
2. Remove last node.
3. Propagate the new root node to its correct position.

Code

```
void deleteFromHeap() {
    if (size == 0) {
        cout << "nothing to delete" << endl;
        return;
    }
    arr[1] = arr[size];
    size--;
```

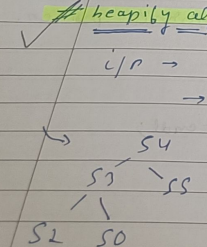
```
int i = 1;
while (i < size) {
    int leftindex = 2*i;
    int rightindex = 2*i + 1;

    if (leftindex < size && arr[i] < arr[leftindex])
    {
        swap(arr[i], arr[leftindex]);
        ~~swap(arr~~
        i = leftindex;
    }
    else if (rightindex < size && arr[i] < arr[rightindex])
    {
        swap(arr[i], arr[rightindex]);
        i = rightindex;
    }
    else {
        return;
    }
}
}
}
```

$T(n) = O(\log(n))$

heapify algorithm.

i/p → | 54 | 53 | 55 | 52 | 50 |

→ convert into heap.



In a CBT → leaf node

↳ $\left(\frac{n}{2}+1\right)$ → nth node

⇒ we need to macs 0 to $n/2$ only.

In case of 0 based indexing

left = 2*i + 1;

right = 2*i + 2;

→ for (i = n/2 → >=0) {

heapify(arr, size, i)

}

→ ~~इस~~ node ~~से~~ वाले सभी leaves को हटाकर वाले फर्क ~~से~~ Num से heapify करें.

(code) Max Heap

```
void heapify (int arr[], int n, int i){
    int largest = i;
    int left = 2*i;
    int right = 2*i + 1;
    if ( left < n && arr[largest] < arr[left]){
        largest = left;
    }
    if ( right < n && arr[largest] < arr[right]){
        largest = right;
    }
    if ( largest != i) {
        swap (arr[largest], arr[i]);
        heapify (arr, n, largest);
    }
}
}
```

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

Build Min heap.

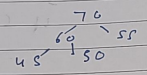
(code) ~~void heapify (int arr[]~~

```
void heapify (vector<int> &arr, int n, int i){
    int smallest = i;
    int left = 2*i + 1;   int right = 2*i + 2;

    if ( left < n && arr[smallest] > arr[left]){
        smallest = left;
    }
```

```
    if (right < n && arr[smallest] > arr[right])
        smallest = right;
    }
    if (smallest != i) {
        swap(arr[smallest], arr[i]);
        heapify(arr, n, smallest);
    }
}
vector<int> builtMinHeap(vector<int> &arr) {
    int n = arr.size();
    for (int i = n/2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
    return arr;
}
```

$T(c) = O(N)$

Heap Sort



↳ we have max heap. !!!

Algo :-

1. swap a[i] → a[n] (root → last)
   size--;
2. root node → correct position.

(while (size > 1))

Repeat

Code :-

```
void heapSort(int arr[], int n) {
    int t = n;
    while (t > 1) {
        swap(arr[t], arr[1]);
        size--;
        heapify(arr, size, 1);
    }
```

Take care of 0 & 1 based indexing (<=)

STL for heap.

Priority Queue.

Max heap / Min heap

→ #include <queue>.

priority_queue <int> pq; // max heap

↳ If we have to make min heap,

⇒ priority_queue <int, vector<int>, greater<int>> minHeap;

H.W → read documentation.

Q. k^{th} smallest element.

i/p = {7, 10, 4, 3, 20, 15}

k = 3

o/p → 7

Approach 1 → Sort the array → O(n log n).

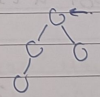
→ return arr[k-1]

Approach 2 → through heap.

1. First k element, make a max heap.
2. For rest element, if element < heap.top()
   ↳ heap.pop() [top at root].
   → heap.push(element).

↳ heap → k element

n element → k small elements



→ ans = heap.top();

Code →

```
int kth smallest (int arr[], int l, int r, int k){
    priority_queue <int> pq;
    for (int i = 0; i<k; i++){
        pq.push (arr[i]);
    }
    for (int i = k; i<=r; i++){
        if (arr[i] < pq.top()){
            pq.pop();
            pq.push (arr[i]);
        }
    }
    int ans = pq.top();
    return ans;
}
```

→ Use min heap to get largest kth element.

Q. is BT a heap?

↳ satisfy max heap here.

Algo:→ solve (){

```
    if (isCBT() && isMaxOrder()){
        true →
    }
    false;
```

```
is CBT (root, i, total count){
    | left/right > total count |
        => Not a CBT
    }
}
```

```
is MaxOrder (root ){
    if leaf → return true;
    if (ri,ri == NULL)
        ↳ check parent data > child.
```

1. Both child
2. leaf node
3. only left node

Code → ~~bool~~

```
int countNodes (struct Node* root){
    if (root == NULL){ return 0; }
    int ans = 1 + countNodes (root->left) + countNodes (root->right);
    return ans;
}
bool isCBT (struct Node* root, int index, int count){
    if (root == NULL){ return true; }
    if (index >= count){ return false; }
    else{
        bool left = isCBT (root->left, 2*index+1, count);
        bool right = isCBT (root->right, 2*index+2, count);
        return (left && right);
    }
}
bool isMaxOrder (struct Node* root){
    if (root->left == NULL && root->right == NULL){
        return true;
    if (root->right == NULL){
        return (root->data > root->left->data);
    }
    else{
        bool left = isMaxOrder (root->left);
        bool right = isMaxOrder (root->right);
        return (left && right &&
            (root->data > root->left->data && root->data >
                root->right->data));
    }
}
bool isHeap (struct Node* true){
    int index = 0; int totalCount = countNodes (true);
    if (isCBT (true, index, totalCount) && isMaxOrder(true))
        return true; }else{ return false; } };
```

Q/ Merge 2 Binary Max heaps.

i/p → MH1, MH2
↓ vector a, ↓ vector b

vector c.
↳ heapify → ans.

Code :→

```
void heapify (int arr[], int n, int i){
    // already written
}
vector <int> mergeHeap (vector <int> &a, vector <int> &b,
                        int n, int m){
    vector <int> ans;
    for (auto i : a)
        ans.push_back (i);
    for (auto i : b)
        ans.push_back (i);
    int size = ans.size();
    for (int i = size/2 ; i>=0 ; i--){
        heapify (ans, n, i);
    }
    return ans;
}
};
```

→ above code is not working as it is 1 based indexing.
convert it to zero based.

↳ heapify fn also in 0 based indexing.

0 - based indexing → $i = n/2 - 1$, $>= 0$

1 - based indexing → $n/2$ → >0

Q/ Min cost of ropes.

↳ cost to connect 2 ropes = length of their ropes sum
↳ task is to have min cost for n-ropes

i/p → {4, 3, 2, 6}

o/p → 2, 3 → {4, 5, 6}
↓
5

4, 5 → {2, 6}
↓
9

→ {15}

ans = 5 + 9 + 15 = 29

Algo → while (size>1)
{
int a = pq.top();
pq.pop();
int b = pq.top();
pq.pop();
int sum = a+b;
pq.push (sum);
}
return cost.

Code :→

```
long long mincost (long long arr[], long long n){
    priority_queue <long long, vector <long long>,
                    greater <long long>> pq;
    for (int i=0; i<n; i++){
        pq.push (arr[i]);
    }
    long long cost = 0;
    while (pq.size() >1){
        long long a = pq.top();
        pq.pop();
        long long b = pq.top();
        pq.pop();
```

```
long long sum = a+b;
cost = cost + sum;
pq.push(sum);
}
return cost;
}
```

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

$S(c) = O(N)$

H.W Q convert BST to min heap.

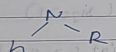
↳ already a CBT (structure ✗ no change).

↳ satisfy min order property.

given condition ↳ left subtree element < right + root + subtree } for each node.

approach

↳ find inorder



↳ N<L & N<R

↳ L<R

⇒ N<L<R ⇒ preorder.

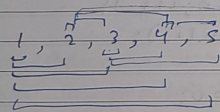
↳ output.

Q kth largest sum subarray.

i/p → { 1, 2, 3, 4, 3 } "subarray is contiguous"

o/p → {1} {1,2} {1,2,3} . . .

approach →




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

subarrays aaye sare.

→ Store in vector.

find kth sum.

$T(c) = O(N^2)$ $T(c) = (n^2 \log n)$

```
code:- int getkthlargest (vector<int> &arr, int k){
    vector<int> sumstore;
    int n = arr.size();
    for (int i=0 ; i<n ; i++){
        int sum = 0;
        for (int j=i ; j<n ; j++){
            sum = sum + arr[j];
            sumstore.push_back(sum);
        }
    }
    sort(sumstore.begin(), sumstore.end());
    return sumstore[sumstore.size() - k];
}
```

approach 2 :→ Use Min heap.

→ For (i=0 i<n){

For (j=i ; j<n){

sum =

if (min_heap.size() < k){

push sum

}

else {

if (sum > min_heap {

min heap.pop

push sum }

}

}

return min_heap.top();

code :→ int get kth largest (vector<int> &arr, int k){

priority_queue<int, vector<int>, greater<int>> min

int n = arr.size();

```
for (int i=0 ; i<n ; i++){
    int sum = 0;
    for (int j = i ; j<n ; j++){
        sum = sum + arr[i];
        if (mini.size() < k){
            mini.push (sum);
        }
        else{
            if (sum > mini.top()){
                mini.pop();
                mini.push (sum);
            }
        }
    }
};
```

$S(c) = O(k)$

Hard Q Merge k sorted arrays.

i/p → arr1 , arr2 , - - , arrk

o/p → [ ] [ ] [ ] [ ] ans

approach 1 → 1. Create ans array
2. insert all element (n*k) in ans;
3. Sort the ans array . $T.C. = n*k \lg(n*k)$.

Approach 2# → Use new min - heap
→ Iske ग्रुप starting में हर array के first element रखो.
→ To iski top से → Put in ans array
→ insert next element of same array.

Code :-
```
class node {
    public:
        int data, i, j;
    node (int data, int row, int col){
        this -> data = data;
        i = row;
        j = col;
    }
};

class compare {
    public:
        bool operator () (node* a, node* b){
            return a->data > b->data;
        }
};

vector<int> mergeKsortedArrays (vector<vector<int>>& kArrays,
                                int k){

    priority_queue <node*, vector<node*>, compare> minheap;
    for (int i=0 ; i<k ; i++){
        node* temp = new node(kArrays[i][0], i, 0);
        minheap.push (temp);
    }
    while (minheap.size() >0){
        node* temp = minheap.top();
        ans.push_back (temp->data);
        minheap.pop();
        int i = temp->i;
        int j = temp->j;
```

```
if (j+1 < kArrays[i].size()) {
    node* next = new node(kArrays[i][j+1],
                          i, j+1);
    minheap.push(next);
}
}
return ans;
}
```

$T.C = O(n*k \log(k))$

$S.C = O(k) + O(n*k)$

Q. Merge k sorted linked list.

↳ return head of final linked list.

Approach 1 → sort ~~vect~~ linked list

→ Join all linked list.

→ replace LL node ⇒ data with vector ⇒ data.

Approach 2 → min heap.

↳ take first element.

↳ while (not empty) {

```
if (head == NULL) {
    head = tail = minheap.top();
    minheap.pop();
    if (head->next) != NULL) {
        minheap.insert
    }
else {
    tail->next = minheap.top();
    minheap.pop();
    tail = tail->next
    if (tail->next != NULL) {
        minheap.insert }
return head;
```

Code

```
#include <queue>
class compare {
public:
    bool operator()(node<int>* a, node<int>* b) {
        return a->data > b->data;
    };
};
Node<int>* mergeKlists(vector<Node<int>*> &listArray) {
    priority_queue<Node<int>*, vector<Node<int>*>,
                   compare> minHeap;
    int k = listarray.size();
    if (k==0) { return NULL; }
    for (int i=0; i<k; i++) {
        if (listarray[i] != NULL) {
            minheap.push(listarray[i]);
        }
    }
    Node<int>* head = NULL;
    Node<int>* tail = NULL;
    while (minHeap.size() > 0) {
        Node<int>* top = minHeap.top();
        minHeap.pop();
        if (top->next != NULL) {
            minheap.push(top->next);
        }
        if (head == NULL) {
            head = top;   tail = top;
        }
        else {
            tail->next = top;
            tail = top;
        }
    }
    return head;
}
```

$T(C) = O(n*k \log k)$

$S(C) = O(k)$

Q Smallest Range in k-list.

i/p: k (sorted) lists
D → D → D → ×
D → D → D →
⋮

o/p: give smallest range [x, y] that has at least one elements.

eg. k = 3, n = 3
{1, 10, 11}
{2, 3, 20}
{5, 6, 12}
→ ans = [1, 5]

Approach 1 Brute force.
↳ sabhi range nikal lo. ($n^2 k^2$)
↳ Has diff range ki difference check karte rho aur smallest ko store karo.

Approach 2
k size
pt1 → {1, 10, 11}
pt2 → {2, 3, 20}
pt3 → {5, 6, 12}

pt [ptr | ptr | ptr | ... |]

↳ min = 1 max = 5

↳ smallest range & atleast 1 elements.

[1, 5]

• max decrease not possible
→ so we can ↑ min value

⇒ increase min; check range;
if minimum store in ans.

$T(c) = O(n^k k^2)$
$S(c) = O(k)$

Approach 3 Using heaps

Algo → make min-heap (k).
↳ has list of starting element details
→ create que
↳ keep a track of max element.

```
int max=start, ans end;
while (!pq.empty()) {
    min = pq.top()
    pq.pop();
    if (max-min < ans.start - ans.end) {
        update ans;
    }
max update → if (min.size() < n) {
        max = max (max, arr[i][j])
        call i;
        pq.push (make node (
        ));
    }
    else {break;
    }
    return ans;
```

Code :→ 2n PC.

Q Median in a stream.
↳ in a random order.
↳ It is a running stream.

Approach 1 → Insertion sort is a potential option.
($O(n^2)$)

Approach 2 optimised sol?

{ _ _ _ M _ _ _ }

< M > M

median = arr[n/2]

= (arr(n-1/2) + arr(n/2)) / 2

Odd size → | left n-1 , right n | or | left n , right n-1 | (cons 0, cons 1)

even size → | left n = , right n | cons 3

* signum (a, b)

a == b , → 0

a > b , → 1

a < b , → -1

→ we will need this fxn

Algo →

→ Watch video. ←

Hashmaps

what → ? → used in Dev algo.

Data structure.

→ Time complexity wise really good. (O(1))

Q/ String str = "abcbdeaf"

O/P → maximum occurring character

sol → a : 3

approach = ? → a → count find → store it.

then b then c → find max.

approach 2 = arr [| | | | |] → 2

'a' 0 ... 25

for (int i=0 → str.size()) {

char ch = str[i];

count[char - 'a']++;

}

for (int i=0 ; 25) {

display max.

Implementation

o In built

→ Ordered - map → O(log n)

→ Unordered _ map → O(1).

Code → STL

```
#include <iostream> <map> <unordered_map>
using namespace std;
int main(){
    unordered_map <string, int> m;
//1  pair <string, int> p = make_pair("bubbar", 3);
   m.insert(p);
//2  pair <string, int> pair2 = make_pair("love", 2);
   m.insert(pair2);
   m["mera"] = 1;
   m["mera"] = 2;   } updation.

//searching
cout << m["mera"] << endl;
cout << m.at("bubba") << endl;

cout << m.at["unknown"] << endl;   } interchange
cout << m["nonmount"] << endl;     } then and check
cout << m.size() << endl;

cout << m.count( )  →  0 & 1 outcome
m.erase("low");

for (auto i : m) {
    cout << i.first << " " << i.second << endl;
//or iterator//.
unordered_map <string, int> :: iterator it =
                                m.begin();

while (it != m.end()) {
    cout << it -> first << " " << it -> second << endl;
    it++;
}
return 0;
}
```

Max freq no.

Code

```
int max freq (vector <int> & arr, int n){
    unordered_map <int, int> count;
    for (int i = 0 ; i < arr.size() ; i++){
        count[arr[i]]++;
    }
    int maxi = INT_MIN;   int ans = -1;
    for (auto i : count) {
        if (i.second > maxi) {
            maxi = i.second;
            ans = i.first;
        }
    }
    return ans;
}
```

→ full code in PC.

Bucket Array & Hash fxn

hash code → common fxn.

contents (Strings, char, obj) to int

common fxn → compresses into range.

Hash codes

1. identity fxn
2. sum of ascii
3. etc etc etc.

collision.

↳ what ??

- Collision Handling

→ Open hashing ⇒ same place ⇒ ⇒ Jaa.

→ Block ⇒ linked list ⇒ head store hota hai and linked list ⇒ individual Jaate rehte ⇒

- remainder changing.

→ Open addressing

↳ gali khali Jagah dekh ke vaha daaldo.

$H_i(a) = h(a) + f_i(a)$.

- linear probing
  $f(i) = i$

- quadratic probing
  $f(i) = i^2$

- complexity analysis

$O(k)$ → (word length).

n = total no. of search

if $n >> k$ ⇒ $O(1)$.

constant time.

* Load factor ?

(H.W) → Read Documentation.

* Rehashing → increasing bucket size.

(H.W) → have Babbar DSA sheet coding ninjas.

Lecture 79

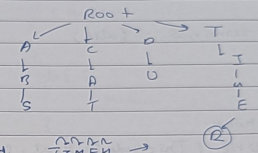
Trie

Eg → Implement dictionary

add word → search → remove word

* Hashmap can be used but in worst case, the T.C will differ & won't be $O(1)$.

Trie →



Add word TIMEN →

→ Aise hi ho gaki hai vo banado trie ⇒.

Algo

↳ character

↳ search → present → Agle bucket pao → Jab string khatam hogyi

→ ashak check node

```
TrieNode {
    char data;
    Trie* children[26];
    bool isTerminal
}
```

Implementation

Code :→

Code :->

```
class TrieNode {
    public:
        char data;
        TrieNode * children [26];
        bool isterminal;
        TrieNode (char c) {
            data = ch;
            for (int i=0; i<26; i++) {
                children [i] = NULL;
            }
            isterminal = false;
        }
};
class Trie {
    public:
        TrieNode * root;
        void insertUtil (TrieNode * root, string word) {
            if (word.length() == 0) {
                root -> isTerminal = true; return;
            }
            // assumption, word will be in CAPSLOCK
            int index = word [0] - 'A';
            TrieNode * child;
            if (root -> children [index] != NULL) {
                child = root -> children [index];
            }
            else {
                child = new TrieNode (word [0]);
                root -> children [index] = child;
            }
            //Recursion
            insertUtil (child, word.substr (1));
        }
        void insertword (string word) {
            insertUtil (root, word);
        }
        bool searchUtil (TrieNode * root, string word) {
            if (word.length() == 0) {
                return root -> isterminal;
            }
            int index = word [0] - 'A';
            TrieNode * child;
            if (root -> children [index] != NULL) {
                child = root -> children [index];
            }
            else {
                return false;
            }
            //recursion.
            return searchUtil (child, word.substr(1));
        }
        bool searchword (string word) {
            return searchUtil (root, word);
        }
};
int main() {
    Trie *t = new Trie();
    t -> insertword ("TIME");
    cout << "Present or not" << t -> searchword ("Time") << endl;
    return 0;
}
```

Trie() {
root = new TrieNode('0');
}

(Constructor)

T(c) → Insertion → O(L)
searching → O(L)

Remove word :-> O(L)

Algo → 1. Search the word
2. Terminal node ko false karlo.

Code :-> (H.W) write code.

Q/ Implement trie
→ a)

Q/ Longest common prefix.

arr → { "coding", "codeser", "codingning", "codon" }

Ans → cod

Approach 1 1. take first string
2. compare characters with remaining string
↳ ans ko store and ++.

Code
```
string longestcommonprefix (vector<string> &arr, int n)
{
    string ans = "";
    for (int i=0; i<arr[0].length(); i++){
        char ch = arr[0][i];
        bool match = true;
        for (int j=1; j<n; j++){
            if (arr[j].size() < i || ch != arr[j][i])
            {
                match = false;
                break;
            }
        }
        if (match == false)
            break;
        else
            ans.push_back(ch);
    }
    return ans;
}
```

T.C → O(M*N)
S.C → O(M)

Approach 2 :-> 1. insert all string in trie.
2. → if child count = 1
↳ add in ans.

T(C) = O(m*n)
S(C)

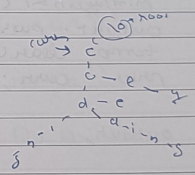
Code :-> Watch video
(too long).

Q/ Implement a phone directory (Hard++)

i/p → prefix
o/p → all suggestions of search bar.

Approach → Trie.

Eg → cod
coding
codding
codu
coday



prefix = coding

Algo →
prefix str = c
last character = c
↳ last character found
mint = dissection.
↳ current move ++

↳ keep track of last character
↳ mint
↳ then mov to curr move.

Code :→ Copy class TrieNode.
Copy class Trie
(without the searching part)

→ Inside class Trie ↓

```
vector <vector <string>> getSuggestions (string str){
    TrieNode* prev = root;
    vector <vector <string> > output;
    string prefix = "";
    for (int i = 0; i < str.length(); i++){
        char lastch = str[i];
        prefix.push_back (lastch);

        TrieNode* curr = prev -> children[lastch-'a'];
        if (curr == NULL){ break; }
        ~~next suggestion (curr)~~
        vector <string> temp;
        printSuggestions (curr, temp, prefix);
        output.push_back (temp);
        temp.clear();
        prev = curr;
    }
    return output;
}

void printSuggestions (TrieNode* curr, vector <string> &temp,
                       string prefix){
    if (curr -> isTerminal){
        temp.push_back (prefix);
        ~~return~~;
    }
    for (char ch = 'a'; ch <= 'z'; ch++){
        TrieNode* next = curr -> children[ch-'a'];
        if (next != NULL){
            prefix.push_back (ch);
            printSuggestions (next, temp, prefix);
            prefix.pop_back();
        }
    }
}

o vector <vector <string>> phoneDirectory (vector <string> &contactList,
                                           string &queryStr){
    Trie *t = new Trie();
    for (int i = 0; i < contactList.size(); i++){
        string str = contactList[i];
        t -> insertWord (str);
    }
    return t -> getSuggestions (queryStr);
}
```

$T(c) = O(n * m^2)$

$S(c) = O(m * n)$

H.W → Do Dry Run of the above.

Backtracking (3 lecture).

→ what ??

↳ purana path dobara check nahi.

Q. Rat in a Maze Problem.

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

→ 0 path 0 → dead

src → (0,0)

dest = (n-1, n-1) = (3,3)

→ print all the paths that exist.

Ans → DDRDRR
DRDDRR

Approach

f(0,0 ; "") → (2, 4, R, D)

Step 0 →
→ mat[newx][newy] = 1
→ visited [newx][newy] ≠ 1
→ newx >= 0 && < n
newy >= 0 && < n
→ then move.

↳ watch video for dry run example.

Code → In PC

T.C → 4^{n^2}

S.C → O(n×m).

Q. N-Queen Problem

↳ placing N chess queens on an N×N chessboard such that no two queens attack each other.

→ queen will attack if
· in same row || coln || diagonal.

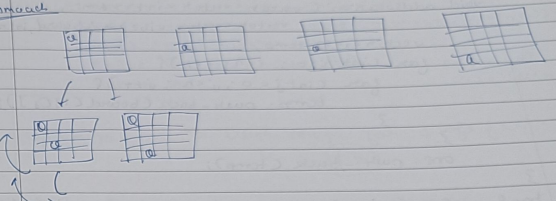
n=4

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

Condn →
1. each row has 1 queen
2. each coln has 1 queen
3. no two queen attack each other

↳ possible soln.

Approach



(Backtracking Tree of setup of all the states)

→ Has 1st case solve next → Backtrack recursion.

Code →

```
void solve (int col, vector<vector<int>> &ans, vector<vector<int>> &board, int n){
    if (col == n) {
        add solution (ans, board, n);
        return;
    }
    for (int row = 0; row < n; row++) {
        if (isSafe (row, col, board, n)) {
            board [row][col] = 1;
            solve (col+1, ans, board, n);
            board [row][col] = 0;
        }
    }
}
```

```
void addSolution (vector <vector <int>> &ans,
                  vector <vector <int>> &board, int n){
    vector <int> temp;
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++){
            temp.push_back (board [i][j]);
        }
    }
    ans.push_back (temp);
}
bool isSafe ( int row, int col, vector <vector <int>> &board,
              int n){
    int x = row; int y = col;
    while (y>=0){
        if (board [x][y] == 1){
            return false
        }
        y--;
    }
    x = row;  y = col;
    while ( x>=0 && y>=0 ){
        if (board [x][y] == 1)
            return false;
        x--; y--;
    }
    x = row;  y = col;
    while ( x<n && y>=0){
        if (board [x][y] == 1)
            return false;
        y--;  x++;
    }
    return true;
}
```

```
vector <vector <int>> nQueens (int n){
    vector <vector <int> > board (n, vector <int> (n,0));
    vector <vector <int> > ans;
    solve (0, ans, board, n);
    return ans;
}
```

$T(n) = n!$

$S(n) = n \times n$

Optimization → Using map for isSafe $f \times n$.

(H.W).

Q. HARD Sudoku solver

→ for 9×9

Cond^n → 1. each digit occurs exactly once in row & col?

2. in 3×3 small grids, each no. exactly once.

Approach →

```
For (int i = 0 to 9) {
    For (int j = 0 to 9) {
        cell = empty
        ↳ For (int value 1 to 9)
            if (safe)
                ↳ board [i][j] = val
                    ↳ recursion
                    backtrack
}
```

Code :→

~~Back~~ page →

```
void solve Sudoku ( vector <vector <int>> & sudoku){
    solve (sudoku);
}
```

```
bool isSafe (int row, int col, vector <vector <int>>& board,
                                                    int val) {
    for (int i=0; i<board.size(); i++){
        if (board [row][i] == val)
            return false;
        if (board [i][col] == val)
            return false;
        if (board [3*(row/3)+i/3][3*(col/3)+i%3]
                                                    == val)
            return false;
    }
    return true;
}
bool solve (vector <vector <int>>& board) {
    int n = board[0].size();
    for (int row = 0; row < n; row ++){
        for (int col=0; col < n; col++ ) {
            if (board [row][col] == 0 ){
                for (int val = 1; val <= 9 ; val++) {
                    if ( isSafe (row, col, board, val)) {
                        board [row][col] = val;
                        bool solpossible = solve (board);
                        if ( solpossible )
                            return true;
                        else {
                            board [row][col] = 0;
                        }
                    }
                }
                return false;
            }
        }
    }
    return true;
}
```