Factor to Compare Algorithm

Time ·) Space

Analysis of Algorithm

Learning Factors

① Time Complexity Analysis (TCA)

CA

↳ Algo. Time taken on input size n

② Space CA

↳ Memory taken by Algo.

TCA of Algo.

Time by Dry Running

↳ By Step Count Approch

↳ Assymtotic Notation :

① Big O]- Worse Case

② " Theta (Θ) → Average Case

③ " Omega (Ω) → Best

~~invested~~

∵ We are ~~instead~~ in Worst Case

Big O converts const value to
" Do nothing with variable
" Ignore constant if it
with variable

$2n^2 + n + 8$

$= O(2n^2) + O(n) + O(8)$

$= O(n^2) + O(n) + (1)O \Rightarrow \because O(c) = O(1)$
↳ constant

$= O(n^2)$

Means constant

ignored constant

↳ is liay apay kionkay sab say ziada value ko saktii hay

Lets do an example to find TCA or ki hi
T(n) of algo.

```
Void main()
{int a=0 ---> 1
    while (a<6) ---> 7n+1
        Cout<< DSA ;} ---> 6n
        a++;
}
```

Let n=6
so time w.r.t n. is

Analysis of Loops Depends on

- Starting value
- Condition
- Increment / Decrement

$$T(n) = \begin{array}{r} 1 \\ + \; n+1 \\ + \; n \\ \hline 2n+2 \end{array}$$

$$O(2n) + O(2)$$

$$T(n) = O(n)$$

[01-09-23]

If it asks exact step count then do dry run all calculations in neat.

Time Complexity Analysis

(1)

```
i = 0          — 1
while (i<n)    — n+1   ⟹ O(n)
{
    ___
    i++ }      — n
}
```

(2)

```
i = n          — 1
while(i>0)     — n+1   ⟹ O(n)
{   ___
    i--  }     — n
}
```

Additive / Differentiative for single loop ⟹ O(n)

③

```
for(   ; i<n ; ) ---> n+1
{
    ~~~
```

(5)

```
int i = 1
while (i < n)
{
    i = i*2;
}
```

=> for each alteration i will be x by 2 so max no of ith value will be a^k

where a = factor with which i multiplied

k is no. of alterations

~~Time Complexity is n tan~~

Time Analysis if & else if block

$= 1/$ no. of if els block

eg if 3 if the 1/3, of 4 then 1/4 etc

Best to Worst Complexities

O(1) Constant (1,2)
Log (log n)
Linear (n)
Polynomial (n^2)
Exponential (2^{2n})
Factorial

So loop will terminate when

$2^k >= n$

$=> 2^k = n$

$\log 2^k = \log n$

$k \log 2 = \log n$

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

$k = \log_2 n$

$O(\log n)$ answer

Home Work

Linear search
Binary "
And 3 types of sorting

Linear Search

```
bool
int result = 0
for (int i = 0 ; i < length ; i++)
{
    if (arr[i] == search)
        found = 1; break;
}
if (found
```

```
bool result = 0;
for(int i=0; i < length; i++)        ---> n+1
{
        if (arr(i) == search)
            result = 1; break;       - n
}
if (result == 1)
        cout << "Found";              -> 1/2
else
        cout << "not found";          -> 1/2
```

$\Rightarrow n+1+n+\frac{1}{2}+\frac{1}{2}$

$= 2n+2 \Rightarrow O(n) = n$

$= O(2n)+O(2)$

Binary Search

```
bool result = 0;
while ( low <= high )  -> n+1
  {
    mid = (low+high)/2   3,  (4+6)/2 = 10/2 = 5 = 6
    if (arr[mid] == element)  -> 1/3
    {
        found = 1;
        break;
    }
    else if (arr[mid] > element)  -> 1/3
        high = mid-1;
    else (mid < element)  3 < 6 , 5 < 7
        low = mid+1;  4, 6
  }
if (found == 1)
        cout << "Found";   -> 1/2
else
        cout << "Not found";  -> 1/2
```

low = mid * 2

$i = j * 2 \Rightarrow \log_2 n \Rightarrow \log n$

---

let arr = [1, 2, 3, 4, 5, 6, 7] element = 67

low = 0 , high = 6

$$\text{Time} = n+1 + \log n + \left(\frac{1}{3} \times 3\right) + \left(\frac{1}{2} \times 2\right)$$

$$= n+1 + \log n + 1 + 1$$

$$= n + \log n + 3$$

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

2, 8, 3, 1, 5, 4, 7

Selection Sort

```
int min = 0
for(int i=0; i<len-1; i++) --> n+1
{
    min = i   0,1,2,3
    for(int j = i+1; j<size; j++) -> m-i
        1,2,3,4,5,6   4,5,6
        2,3,4,5,6
        3,4,5,6
        1,2,3,4,5
        2,3,4,5
        3,4,5
        {
            if(arr[j] < arr[min])   m-i-1
                min = j;
        }
        1,2,3,4, 1,3,4,
    swap(arr[min], arr[i]);
}
```

n-1

~~n+1 + n[m-1+m]~~

$= n + (n-1)[(m-i)+(m-i-1)]$

$= n + (n-1)(m-i) + (n-1)(m-i-1)$

$= n + nm - ni - m + i + nm - ni - n - m + i + 1$

As i & 1 are constant so $i = 1 = c$

$= \cancel{n} + 2mn - 2nc - 2m + c + - \cancel{n} + c + c$

$= 2nm - 2nc - 2n + c$

∴ Sum of constant = constant

$O(n) = O(nm) - O(n) - O(n) + O(c)$

$= nm \Longrightarrow n^2$

04-09-23

Lec # 4

(LIFO) DATA STRUCTURES

Stack => The element will be added on top only. And the ~~value~~ element will also be removed only on top. Both of them are done one by one, i.e. Last in, first out.

→ It cannot be accessed randomly. Only the end element will be accessed. Weather you are removing or adding

```
Class Stack {

    int * arr = 0;
    int maxsize, current_size, top ;
public:
-> Stack (int n=5)
  {
      max size = n
      arr = new int [n]
      top = -1
      Curr = 0
  }

-> ~Stack ()
  {
      if (arr != 0)
      { delete [] arr; arr=0
      }
  }
```

```
-> bool isEmpty( )
{
    if (Cur == 0)
        return true;

    return false;
}

-> bool isFull( )
{
    if (Cur == maxsize)
        return true;
    return False;
}

-> bool push (int val)
{
    if (isFull( ))
        return False;

    else
    {
        top++;
        arr[top] = val;

        Cur ++;
        return true;
}
```

```
-> bool pop()
  {
      if(isEmpty())
          return false;

      else {
              cur --;  top --;
              return true;

          }
  }

-> bool getTopValue(int & val)
  {
      if(isEmpty())
              return false;
      else
       {
          val = arr[top];
          return true
       }
  }

};  // end of class

void main()
  {
      Stack s1(50);
      s1.push(39);     st.pop
      s1.push(1);
      s1.pop();
      int a;
```

```
getTopValue(a);

Cout << a;
}
```

HOMEWORK

Buble Sort

```
int temp;

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

n

n-i-1

$n+1 + n(n-i) + n(n-i-1)$

$\cancel{n}+1 + n^2 - ni + n^2 - ni - \cancel{n}$

$2n^2 - 2ni + 1$

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

$\Rightarrow n^2$

~~arr:~~ HOMEWORK

Insertion Sort

```
int i, j, temp;   let n=6
for ( i=1 ; i<=n-1 ; i++ )  ⟶  n
{
    temp = arr[i]
    j = i-1 ; if i=4, j=3 |
    while ( temp < arr[j]  &&  j>=0 )
    {
        arr[j+1] = arr[j] ;
        j--;
    }
    arr[j+1] = temp ;
}
```

1, 2, 3, 4, 5, $\boxed{6}$

Check: $i=5$ $j=4$ | if $i=1$, $j=0$

5, 2, 0, 0, $\boxed{-1}$

(0, $\boxed{-1}$)

4, 3, 2, 1, 0, $\boxed{-1}$

$n-1$

$m-1$

m

arr = {0, 5, 3, 2, 1, 6}

① temp = 5
j = 0
5 < arr[0]

② temp ③ ⟶ arr[2]
j = 1

⟶ {0, 3, 5, 2, 1, 6} ↳ Result of second alteration

3 < arr[1] ⟹ 3<5, ~~1>0~~ 1

arr[2] = 5 ⟹ 3 replaced by 5

j--, j = 2-1 = 1

arr[1] = 3

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

$= \cancel{n} + nm - m + nm - \cancel{n} - m + 1$

$= 2nm - 2m + 1$

$O(n) = O(nm) - O(m) + O(1)$

$= O(nm)$

$O(n) \Longrightarrow n^2$

Recursion

08-09-23 Lec # 5

→ Queue (FIFO) → First In First Out

→ Class Queue

{

int * queueArray = 0; int maxSize;

int currSize; int front; int rear;

Do a similar work as top in Stack

do a work if top variable. which it do in Stack

public :

Queue (int n=5)

{

rear variable work ← Enqueue → push

front variable work ← Dequeue → pop

```
if (rear == (maxSize -1))
        rear = 0
else
        rear++;
rear = (rear + 1) % maxSize
```

maxsize = n;

queueArray = new int [maxSize];

currentSize = 0

front = -1 ; rear = -1;

}

~ Queue ()

{

P.T.O

```
~Queue()
{
        if (queueArray != 0)
        {
                delete [] queueArray;
                que = nullptr;
        }
}


bool isEmpty();
{ bool isfull();

bool Enqueue(int val)
{
        if (isfull())
        { return false; }

        else
        {
                rear = (rear +1) % maxSize;
                queueArray[rear] = val;
                Curr ++;
                return true;
        }
}
```

```
bool dequeue(int& val)
{
    if( isEmpty())  { return false; }

    else
    {
        front = (front + 1) % maxSize;
        val = queueArray[front];
        cur --;
        return true;
    }
}
};
```

ON LINE CLASS STA TASK

Q'

a)

```
int arr
    int range, arraySize; index = 0;
    int* arr = new int[arraySize];
    int num;
for (int i = 0, i < range ; i++)
{
    if (index == arraySize)
        index = 0;
    else
    {   arr[index] = num;
```

```
        index ++;
    }
}
```

b)

```
int range, arrSize, num, index = -1;
int * arr = new int [arrSize];

for (int i=0 ; i < range ; i++)
{
        index = (index +1) % (arrSize);

        arr[index] = num;
}
```

Q2

```
template < class temp>
Class Queue {
    temp* queueArr = 0;
    temp maxSize, CurrSize, front, rear;
public:
```

```
Queue (temp n=5) {

    maxSize = n; temp
    queueArr = new ~~int~~ [maxSize];
    CurrSize = 0; front = -1; rear = -1;
}

~Queue( ) {

    if (queueArr != 0)
    {
        delete [] queueArr;
        queueArr = null ptr;
    }
}

~~temp isempty ( );~~
~~temp isfull ( );~~

temp isempty() {

    if (CurrSize == 0)
        return true;
    return false;
}

temp isfull() {

    if (currSize == maxSize)
        return true;
    return false;
}
```

```
temp Enqueue (temp val)
{
        if (isfull())
                return false;
        else {
                rear = (rear+1) % maxSize;
                queueArr [rear] = val;
                CurrSize ++;
                return true;
        }
}

temp dequeue (temp val)
{
        if (isempty())
                return false;

        else
        {
                front = (front + 1) % maxSize;

                val = queueArr [front];
                CurrSize - -;
                return true;
        }
}
```

15-09-2023 Lec # 8

Data Structures

Link List / List

1) Disadvantage of Arr Data Structure :-

① Fix Size

② Wastage of Memory

③ Contigeous memory allocation

④ Can't add single to existing array
⑤ " delete a " block from " " .

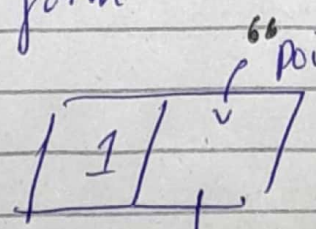
Solution :

Linked ~~or~~ List or List

•) Linked list is all based on Pointers.
↳ Create Memory on run time.

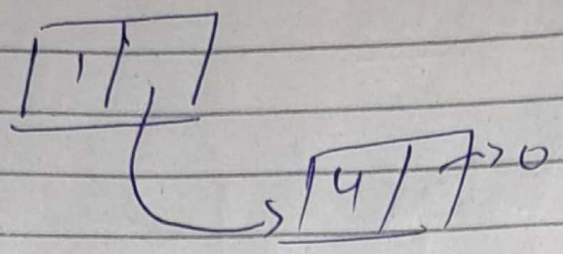
•) Head pointer tells list is empty or not.

•) When one int or variable came to block form



"Pointer pointing to the adres of next no." so that the next block should be attached there (actually it points to the adress of next block).

If no no. came the next block is (No matter where) ptr = 0.



It is not a array. It looks like a array. There is a pointer which links all block spreaded in a memory.

1D Link List IMPLEMENTATI

```
Class Linked list ;

Class Node
{
    int data ;
    Node * next ;
    friend class Linkedlist ;

}; //end of Node Class


Class Linkedlist
{
    Node * head ;
public :

    Linked list ( )
    { head = nullptr }  both means same
      head = 0;         do one of them
    }
```

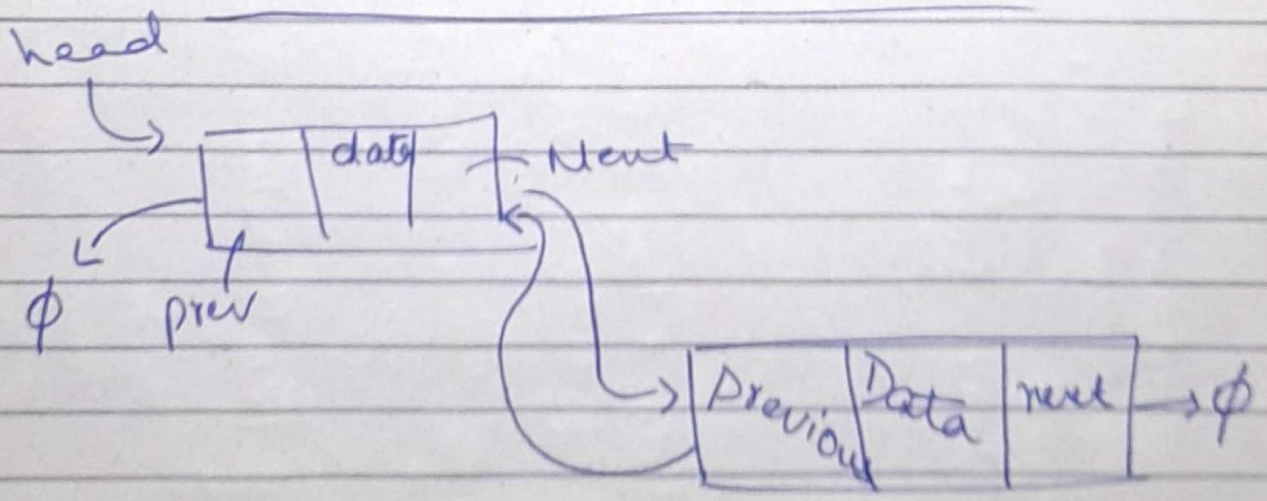
```
void display()
{
    ~~Node * curr = head;~~

    if (head == nullptr)
            cout << List is Empty;
    else
    {   Node * curr = head;
        while (curr != 0)
        {
            cout << curr -> data;
            curr = curr -> next;
        }
    }
}

bool InsertAtStart (int val)
{
    Node * temp = new Node;
    temp -> data = val;
    temp -> next = head;
    head = temp;
    return true;
}

bool deleteFromStart() {

    Node * temp = new Node;
    temp = head;
    ~~delete head;~~
    head = temp -> next;   delete temp;  return true;
}
```

```
                    int
bool insertafterValue(target , int val)
{
    ~~New~~ Node * temp = new Node

    if (head == nullptr)
    {
        temp->data = val ;
        temp->next = 0 ;
        head = temp ;
        return true
    }

    else
    {
        Node * temp = new Node
        temp->data = val ;

        while (curr != nullptr)
        {

            if (curr->data = target)
            {
                ~~node~~
                temp->next = curr->next
                curr->next = temp
                return true ;
            }

            curr = curr->next ;
        }
}
```

```
bool InsertAtEnd(int val)
{
    Node * temp = New node;
    temp->data = val;
    temp->next = nullptr;
    if(head ==0)
        head = temp; return 1;
    else {

        Node * curr = head, * prev = 0;
        while (curr != 0)
        {
            prev = curr; curr = curr -> next;
        }
        prev -> next = temp;
        return 1;
        }
}

bool deleteFromEnd( )

{
        if(head == nullptr)
            return 0;
        else {
                head
        Node * curr = 0, *last = 0, * secondlast = 0;
        while (curr != 0)
        {
            secondlast = last;
            last = curr;
            curr = curr -> next;
        }
```

```
if (secondlast == 0)
        head = 0;
else
        secondlast -> next = 0;

delete last;
last = nullptr;
return 1;
}

}


~ LinkedList()
{
    while (head != 0)
    {
        Node * curr = head;
        head = head -> next
    // ↳ means  head = curr -> next

        delete curr;
        curr = head;
    }
}
```

```
Class DoublyLinkList ;
Class Node
{
    int data ;
    Node * next , * prev ;
    friend class DoublyLinkList ;
};

Class DoublyLinklist
{
    Node * head ;
public:-

    Constructor {
        head = nullptr ;
    }

};
```




```
last -> prev -> next = 0 ;
del last ;
```
} - del from end

HomeWork # 3

Write a function for double link list that will accept address of first node as parameter and merge it with list pointed by this pointer.

Sol:

```
bool mergelist (DLL* ptr)
{
    if (ptr == nullptr)
            return false;

    if (head == 0)
    {
        head = ptr;
        head->prev
        ptr->prev = nullptr; return true;
    }

    DLL* curr = head;

    while (curr != nullptr)
    {
        if (curr->next = nullptr)
        {
            curr->next = ptr;
            ptr->prev = curr
            return true;
        }
```

(Page Turn off)

```
        curr = curr->next;
    }
}
```

Let $n = 4$

eg

```
if(n = 1)
    Cout<<1;

if(n=2)
    Cout << 2;

if(n=3)
    Cout << 3;

if(n=4)
    Cout << 4;
```

→ output

1, 2, 3, 4

because the last no. before termination of call was 4, then 3 then 2 and so on

Lec #13

4-10-28

→ Recurrsion

↳ A recursive / repitative call of a function to itself.

↳ When to Stop Recurrsion :-

There should be a condition called base condition, on which termination of recurrsive call depends.

eg

```
void display (int n)
{
    if (n>0)
    {   cout << "Okay\n";
        display (n-1);
    }
    int s=5;
    cout << s;
    s--
}
```

Compiler store recurrsive call in Stack.

→ Base Condition

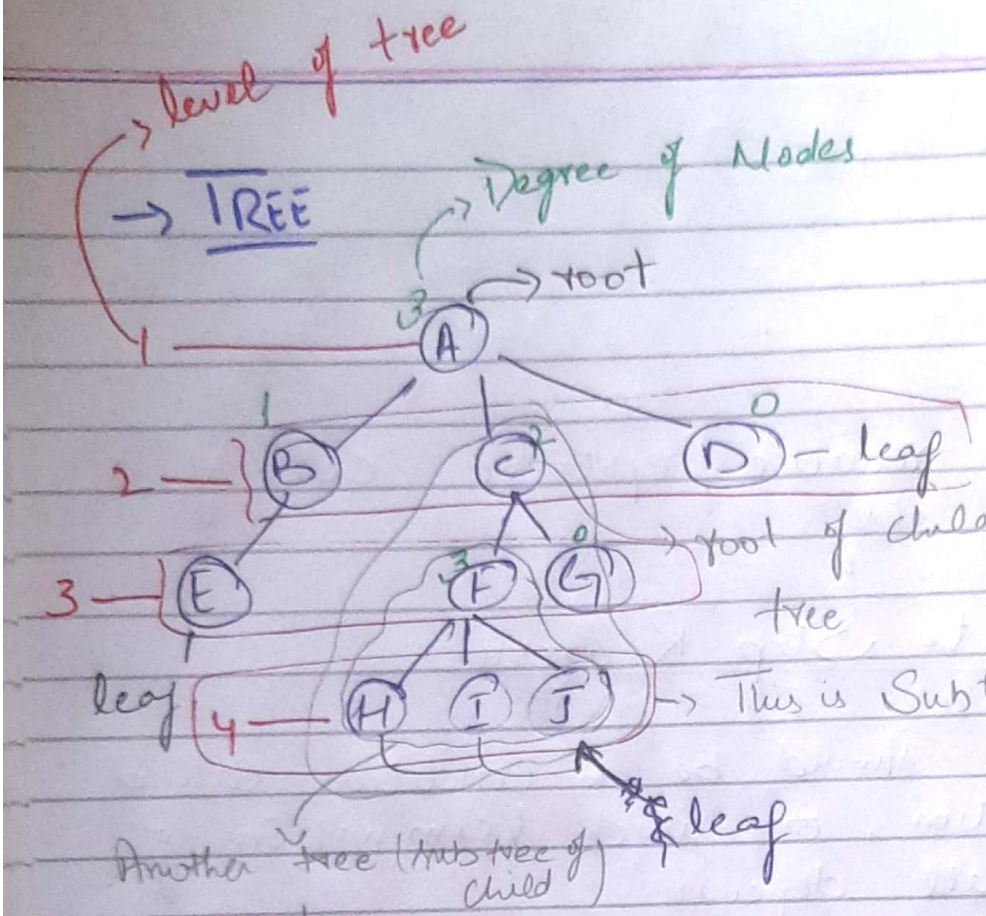
↓

Which will Stops Recurrsive Call

If there is code after recurrsive call (written in block) then Compiler will save it. It will start completing code after recurrsive call in reverse order.

→ Eg on previous pg

~~Disp~~



→ Root of Tree — starting pt. of tree

→ Degree of Node
↳ no of subtrees of node

→ Degree of Tree
↳ Max of all degree of nodes

→ Level of Node

→ Height / Depth of Tree
↳ max level at which node exists

→ Parent → Parent of I is F

→ Parents of parent node. A & C are ancestors of F.

→ Tree of child is subtree

→ Child → E is ~~B's~~ child of B

→ Siblings → Node having same parent eg ~~E & F, H I & J~~ EF, HIJ

→ Ancestors

Degree is also called no. of subtrees.

→ Leaf → end Notes → otherwise terminal node.
→ Nodes with 0 degree.

* Type of Tree

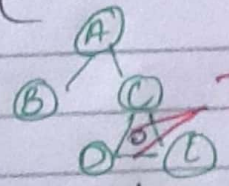
→ Binary Tree

↳ Tree in which node has atleast two childs/sub nodes

Note

Difference b/w tree and graph is:-

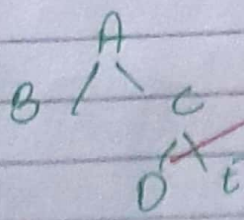
→ Path to reach node from parent is multiple in graph. (i.e circle exists in graph.)



Two path to reach D.

→ Path to reach node from parent is single in tree (i.e circle not exists)

→ One node is both tree and graph i.e.



Single path

• we don't print 3 after 9 because we have already printed that node

16-10-23

Lec #16

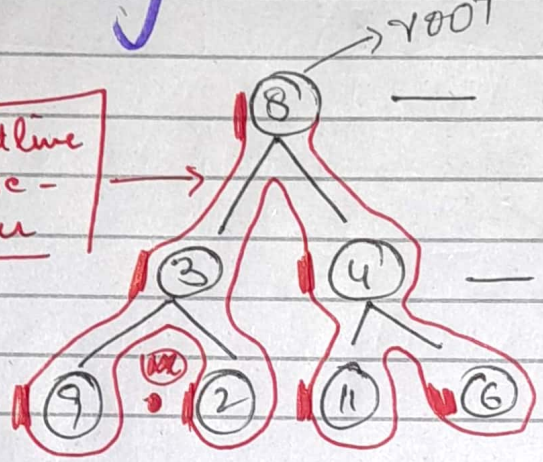
⇒ Binary Tree

→ not exactly same as BST. differ

val < root = left side

val > root = right side

insert in Binary Tree we can insert randomly.



Red outline for pre-order

⇒ Max number of nodes at specific level (k):

2^{k-1} ∵ k is level

Max nodes at level 3 is

$2^{3-1} = 2^2 = 4$

⇒ Max no. of nodes in tree:

$2^k - 1$ ∴ k = max level of tree

↳ For this tree max level = 3

Max no of nodes = $2^3 - 1 = 8 - 1 = 7$

⇒ Traversals in Tree:-

1. Inorder
2. Pre order
3. Post Order

Print when pass bottom

→ Dia on next page of next page

1. Inorder → LNR (Left Node Right)

9, 3, 2, 8, 11, 4, 6

Print when pass left

→ Time Complexity : O(H) → Height / Depth

→ From left we did

② Pre order 8, 3, 9, 2, 4, 11, 6

→ NLR (Node Left Right)

③ Post Order → 9, 2, 3, 11, 6, 4, 8

Print when Pass Right

→ LRN

→ Implement Tree using Array Data Structure:

Root is at index 1 :-

	0	1	2	3	4	5	6	7
i: Node		8	3	4	9	2	11	6

		8	3	4				6

leftChild : 2i

Right " : 2i+1

↳ If 11, 9, 2 absent. Then we leave space of absent elements

Parent = $\lfloor i/2 \rfloor$ → floor of value

if i even

$\frac{i-1}{2}$

↳ Array size : (Max no. of nodes) + 1 ⇒ because we have to skip index 0. (because tree level starts from 1 not from 0)

→ Disadvantage by Array :

if i is odd

→ Wastage of memory.

→ Sekweed Binary Tree :

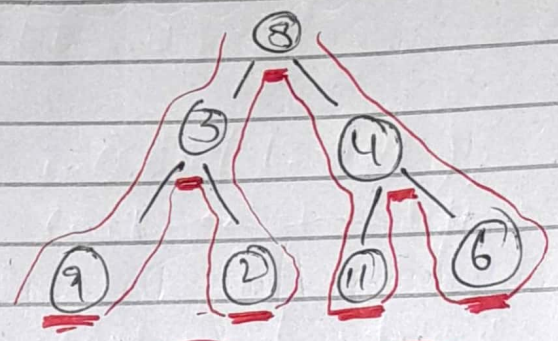
→ Left sekweed ③ - ② - ① (Root)

→ Right sekweed (Root) ① - ② - ③

If we start ~~array~~ root at by Zero index then

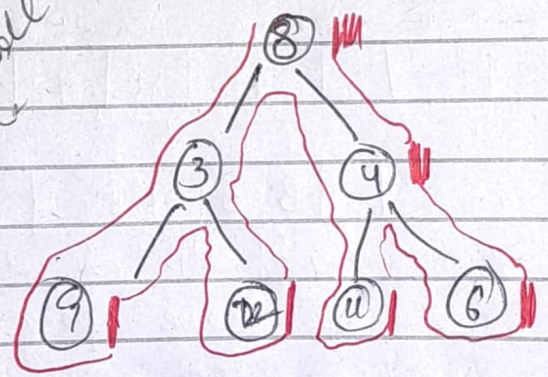
left : 2i+1 right : 2i+2

Inorder



~~For P~~ Post Order

Each node only about prev and next not all at one time



BST → Binary Search Tree

if
data < Node ⇒ Left

data > Node ⇒ Right
data = Node ⇒ Right

growth is expon

One root can hold multiple ch at one time. Chil varies from par to parent

Data Structure

types

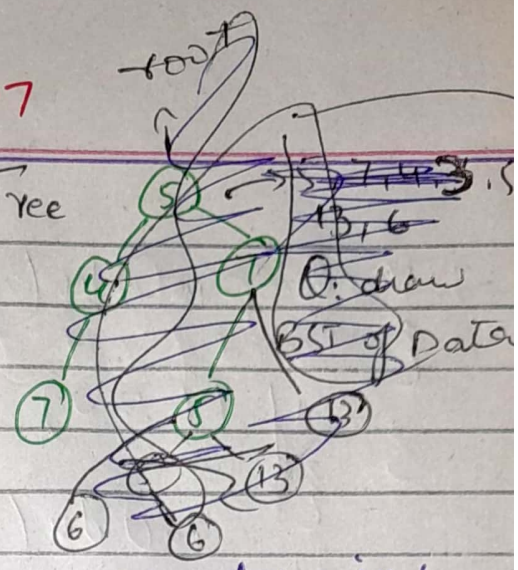
Linear/sequential
→ array
→ linklists
→ stack
→ queue

Nonlinear
→ Tree

18/10/23 Lec #17

Binary Search Tree (BST)

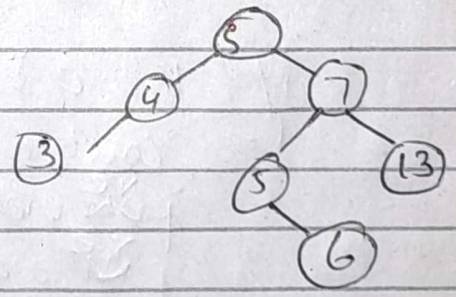
→ Version of Binary Tree



↳ A type of tree in which left child value is less than parent value and and right child value is greater than parent value.

∵ Left and Right Subtrees should also be BST

Draw BST of 5, 7, 4, 3, 5, 13, 6



⇒ Implication in C++

```
Class BST;

Class TreeNode
{
    int data;
    TreeNode *left, *right;
    friend class BST;

};

Class BST
{ private:
    TreeNode * root;
    void ins()
```

⇒ Min node is the end left sub node

⇒ Max node is the right end node

```
=> void inOrder(TreeNode *t)
{
    if (t != 0)
    {
        inOrder(t->left);        L
        ~~cout << t->left;~~

        cout << t->data;         N
        in order(t->right);      R
    }
}

=> void PreOrder(TreeNode *T)
{
    if (t != 0)
    {   cout << t->data          N
        pre order(t->left);      L
        pre order(t->right);     R

        ~~cout << t->data;~~
    }
}

=> void postOrder(TreNode *T)
{
    if (t != 0)
    {
        postorder(t->left);      L
        postorder(t->right);     R
        cout << t->data          N
    }
}
```

```
void insert (TreeNode* &t, Tree Node * newNode)
{
    if(t==0)
    {
        t = NewNode;      -> No data in tree, make it a tree root which value come first
    }
    else if (NewNode->Data < t->data)
    {
        insert(t->left, NewNode);      -> data < Node
    }
    else
        insert (t->right, NewNode)      -> data ≥ Node
}
```

public:

```
-> void inOrder()
   {
       inOrder(root);
   }
```

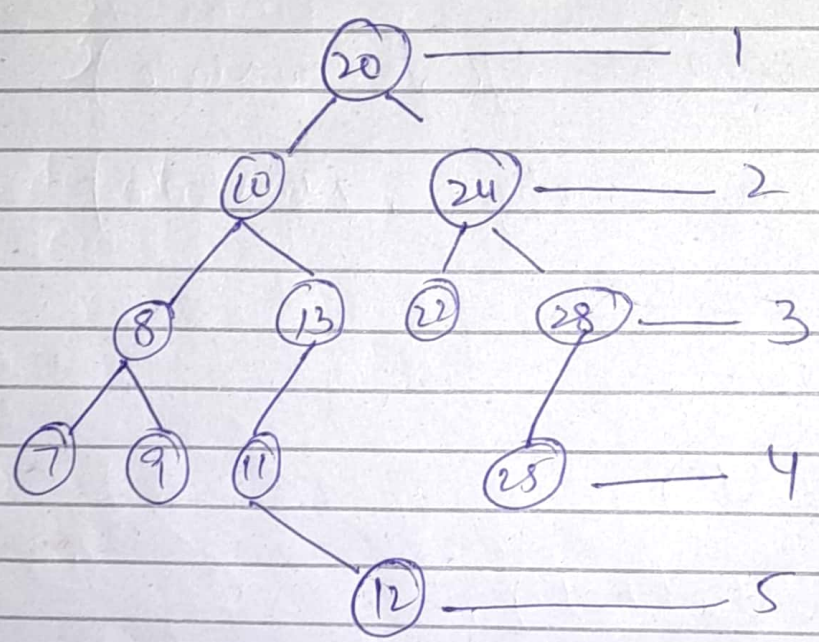
NLR
LNR
LRN

```
-> void preOrder()
   {   preOrder (root);
   }
-> void postOrder()
   {   postOrder (root);
-> void insert (int val)
   {  TreeNode * temp = new TreeNode;
      temp->data = val;
```

```
temp->left = temp->right = ϕ ;
insert ( root, temp);
}
```

→ If the root of the tree is deleted then it become forest. Because there is no starting point so we have to make a new value as a root

23/10/23 Lec #18



→ If we del 10 or 24 then forest will form because the have tree also

```
bool remove (int val)
{
    Node * curr=root , *parent=0;
    while ( curr!=0 && curr->data != val)
    {
        parent = curr;
        if (val < curr->data)
            curr = curr->left
```

```
else
        curr = curr -> right;
}// end of while
if (curr == 0) //if tree is empty or value to
                be founded not exists
    return false;

else // value founded
{
  if (cur -> left != 0 && cur -> right != 0)
  {
        l = curr -> left
    Node * l = cur, * pr = cur;

        while (l -> right != 0)
        {
            pl = l
            l = l -> right;
        }
        curr -> data = l -> data;
        cur = l;
        Parent = pl;
  }
  if (cur -> left != 0)
  {
      chil = curr -> left ;
  else
      Child = cur -> right;
  if (cur == root)
  {
      root => child; }
```

Vise versa: We will make the greatest value of that tree as root.

(We will make the greatest value of left side as root

if we want to del 10 we swap 10 by 9. so

Node * Child;

BST rules can be followed as

≥9 on right of tree

<9 on left of tree

if it has only one child

```
else
{
    if (cur = parent->left)
        parent->left = child;
    else
        parent->right = child;
}

delete cur;
cur = 0;
return true;

}
```

25/10/23 lec#19

Time Complexity BST

n = nodes of tree, h = height of tree It explore every node (i.e it run n times with n++ as iterator)

① Inorder or Pre Order or post order

$O(n)$ $O(n)$ $O(n)$ $\Rightarrow O(h) \neq O(n)$

$O(h)$ $O(h)$ $O(h)$

↳ It also explore nodes at max height

② Insertion in BST

$O(\log n)$ → average Complexity if tree not skewed

$O(n)$ → if skewed → worst Complexity

↳ $O(h)$ → in height

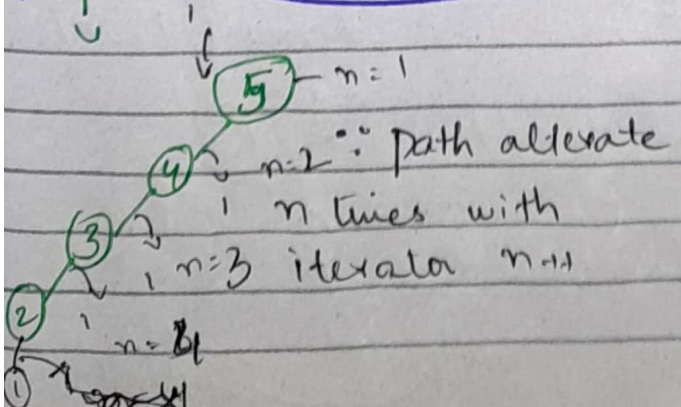
③ Deletion in BST

You will right $O(h)$ if he asked time complexity in term of height

$O(h)$

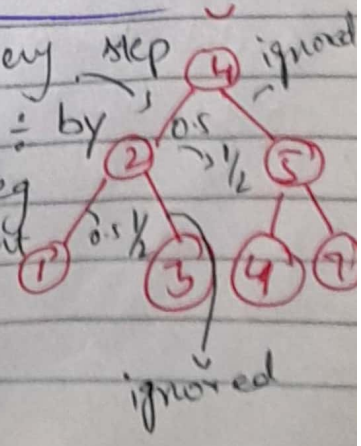
$O(\log n)$ → Average

$O(n)$ → worst



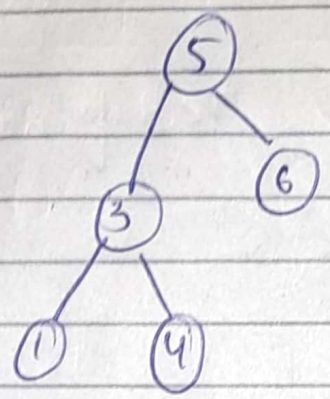
∵ path alterate n times with iterator n++

at every step path ÷ by 2 eg if insert 0.5



Full Binary Tree

- Every node has either 2 or 0 subnode



Complete Binary Tree

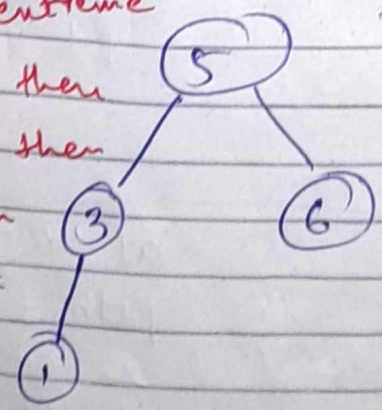
Rule 1 → Every node at k-1 level should have 2 subnode

→ max tree level

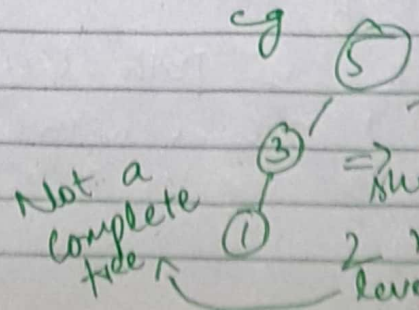
Rule 2 → At k (max level) node should be inserted from left child

in easy word → k-1 level pa jitnay bhi no a saktay tha honay chahi

Start from extreme left node. then insert left then Right then go to next node and so on



eg

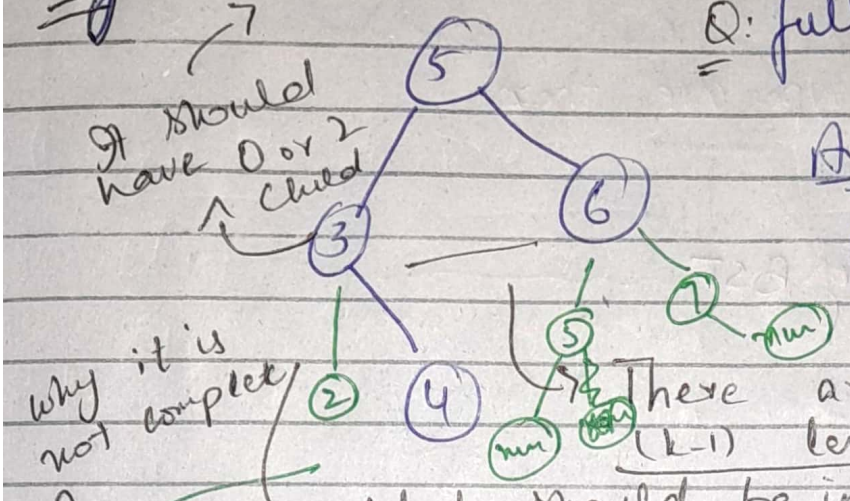


Not a complete tree

⇒ There should be 2 node at level 2

eg

reason of not full



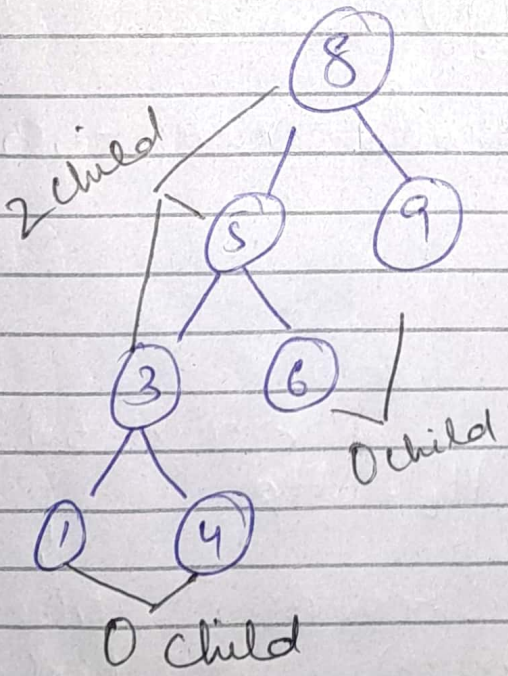
Q: full or Complete?

Ans

No complete
no full

There are max node at (k-1) level

Node should be inserted at right then left. then move to next node and the child should be inserted first left then right and so on



Q: full or Complete?

Ans

Full as every nodes has 2 or 0 child.

→ Consider 2, 5, 7 only now it is complete

→ Consider 2, 5, 7 and all nums Now it is not complete because 'num' should be inserted at right of 5 then at left of 7.

Height Balanced BST

type → many types are there.
→ AVL TREE
↳ version of BST

Only its insert func. will be different

⇒ AVL
↳ BST with height balanced factor

No complete or full in AVL.

⇒ Balancing Factor (BF)

→ Every node should have -1, 0, 1 balancing factor.

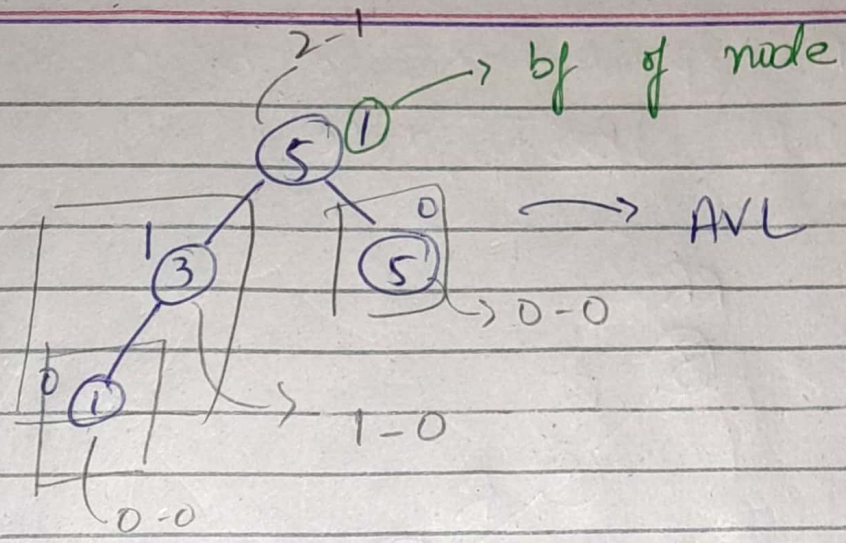
→ BF formula

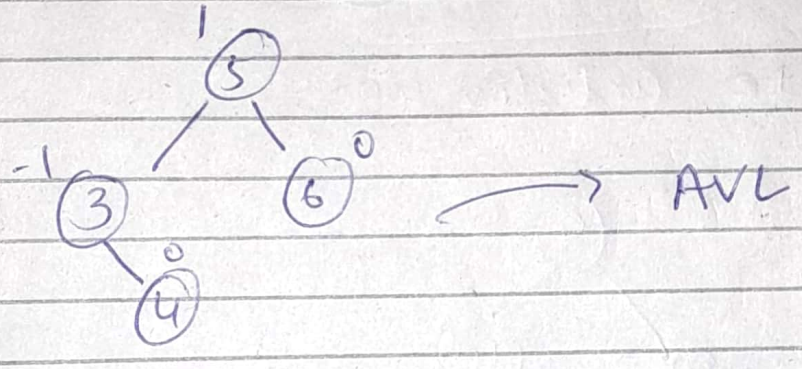
bf of node = $H_L - H_R$

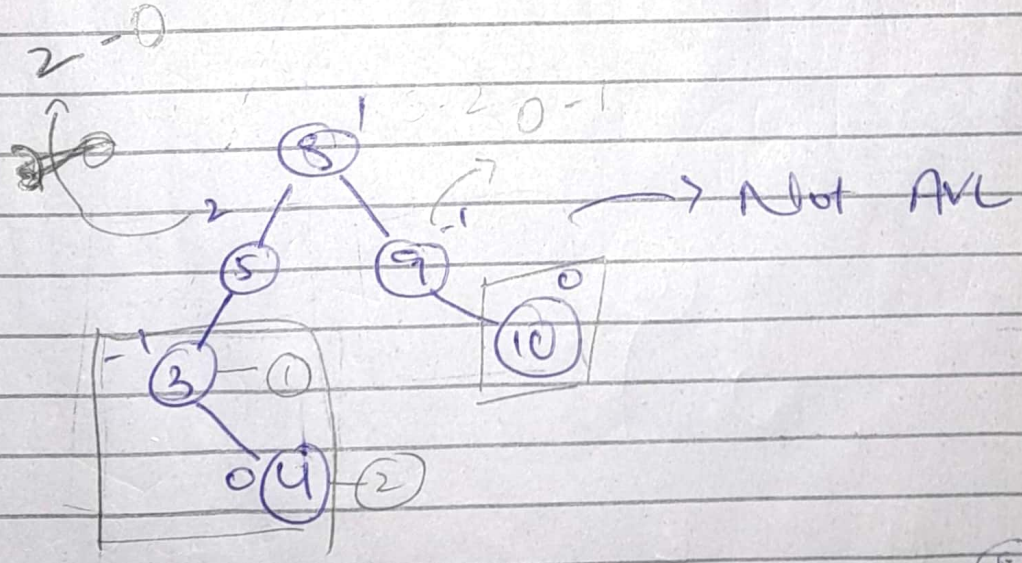
H_L → left subtree(s) height (h) → total levels in left subtree

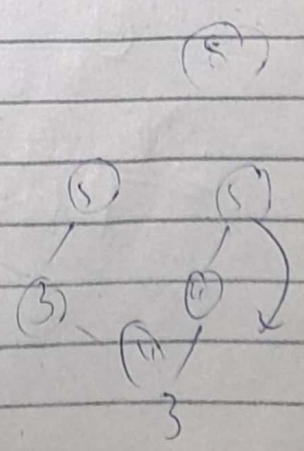
H_R → right s h → total level in right subtree

eg



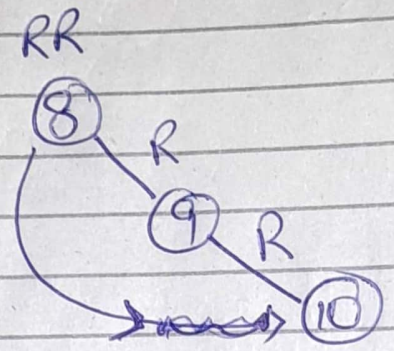






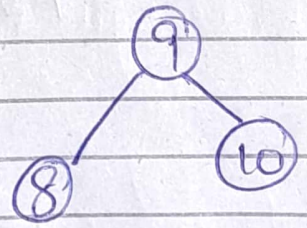
All operation should be according BST rule

Case 1

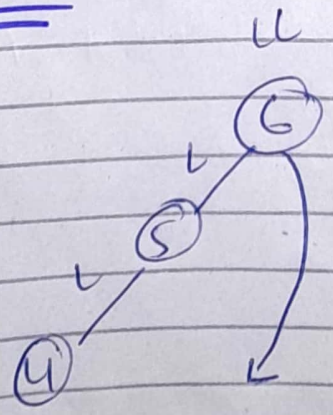


Sol

Rotate anticlock wise

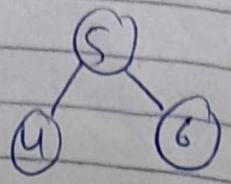


Case 2



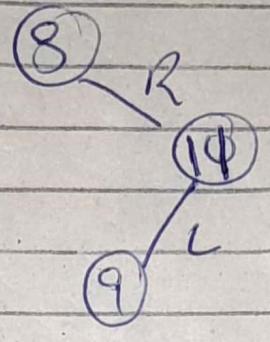
Sol

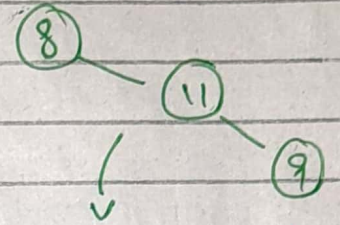
Rotate cloch wise



Case 3

RL

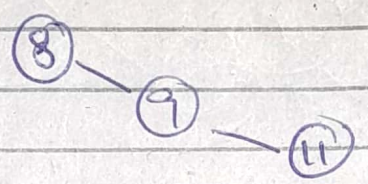




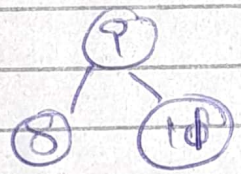
wrong rule violated 9 should be at left

Sol

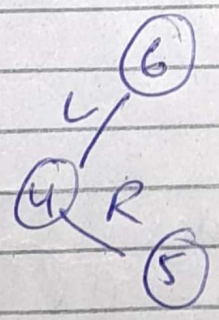
•) Convert in RR



•) Rotate Anticlock wise

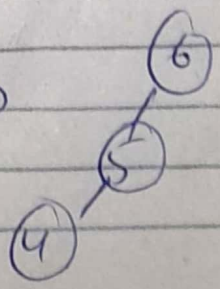


Case 4 LR



Sol

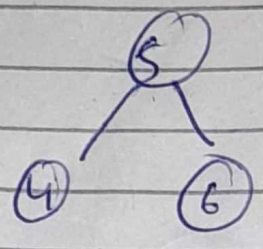
•) Convert in LL →



•) ~~Rotate~~

P.T.O

•) Rotate Clockwise



MaxHeap (int n) Here n passed will be n+1

{ maxsize = n

h = new int [maxsize]

currheapsize = 0

}

13/10/23 Lec # 20

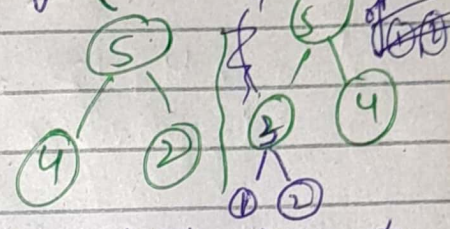
→ Version of Binary Tree

Heap Data Structure

It is not BST It is BT only.

① A complete Binary Tree

② Types → eg



→ Max Heap :

heap in which parent's data / key is greater than child key. Moreover, left and right subtree should also be max heap.

h.. too

→ Min Heap

h.. is less too min heap

Code

Heap for Swap Sorting
BST for Searching

```
Class MaxHeap
{
    int * h,
    int maxsize, curheapsize;
Public:
    MaxHeap(int n)
    { root is at 1 index
        maxSize = n +1 ; 1
        h = new int[maxSize];
        curheapsize = 0
    }
}
```

we do this because we start array in BT by index 1

Max. element of all tree will be found at root for max heap

Similarly, min element at root in min heap.

P.T.O

// Insert
// Delete

};

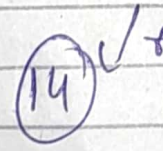
input data

14, 24, 12, 11, 25, 8, 35
↳ maxSize = 7+1 = 8.

→ **Insertion in heap**

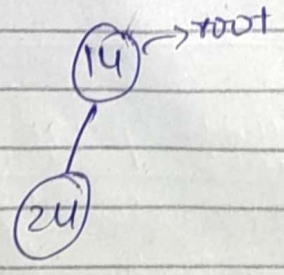
1. Insert value element by element.
2. " " at last available locat
3. Convert tree in max or min heap according to heap type.

i)

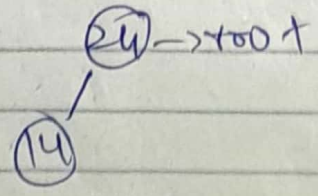


stack – push
queue – enqueue
heap → buildheap

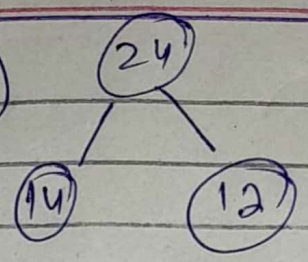
ii)



heapify



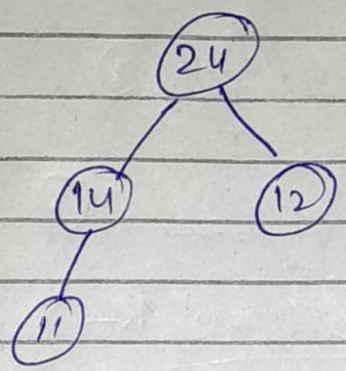
iii)



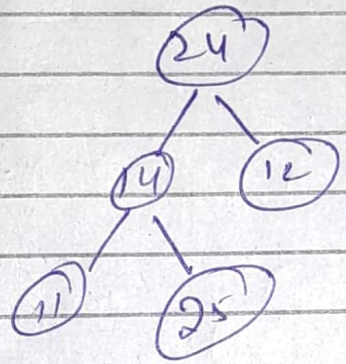
Time Complexity

log n because key or date jump to its Parent not explore whole array.

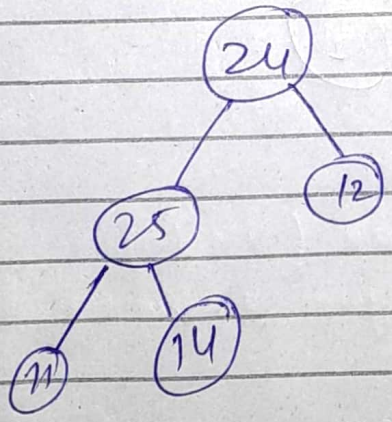
iv)

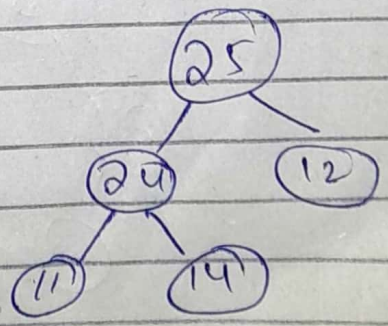


v)

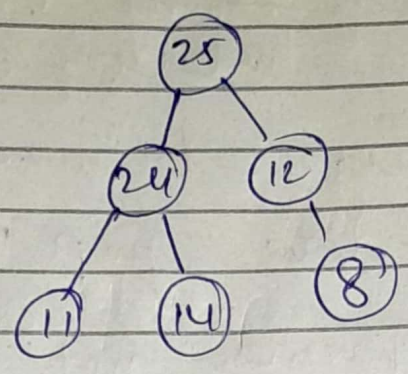


heapify

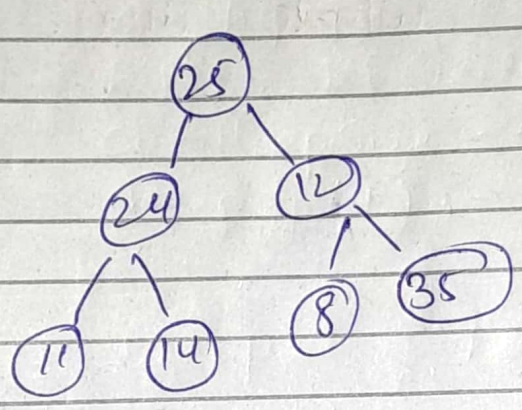




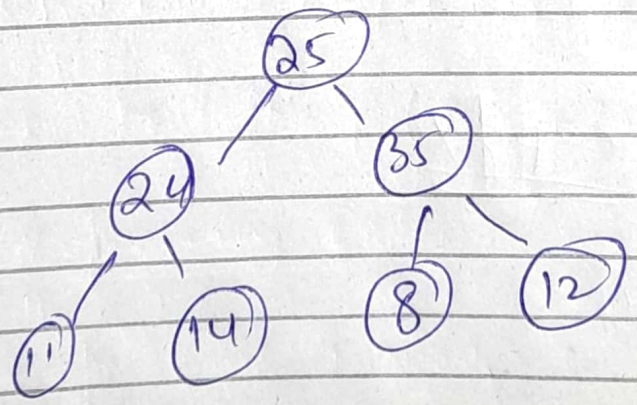
vi)

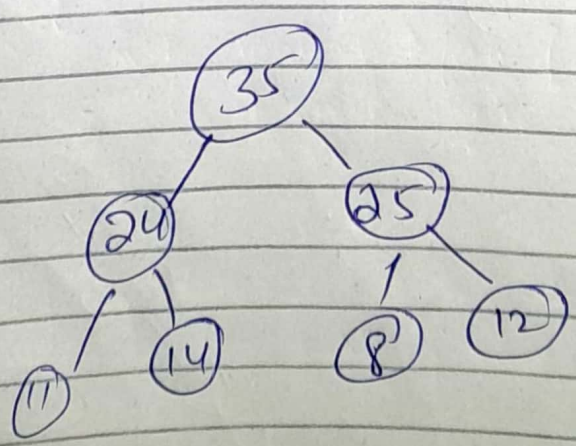


vii)

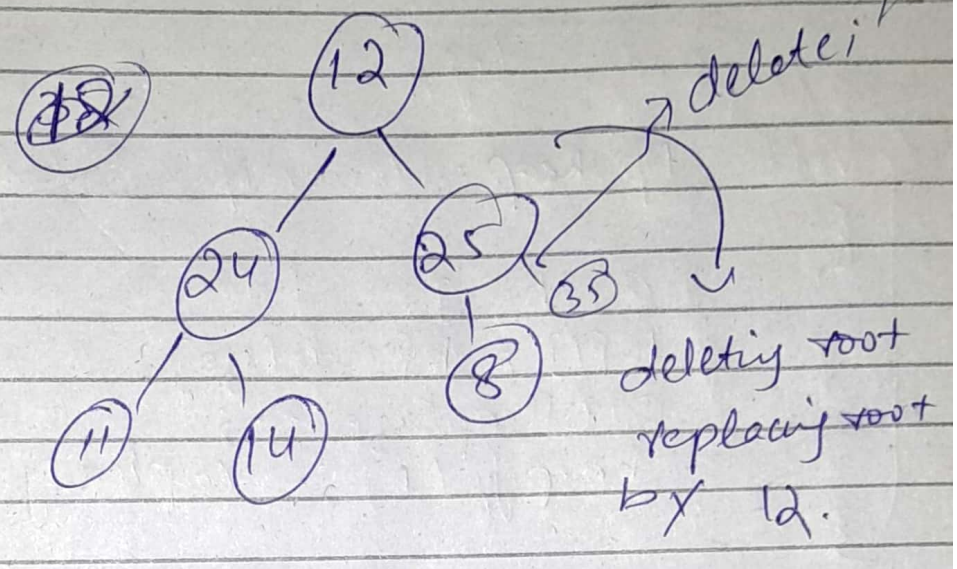


heapify





delete



Now call heapify function so that 12 goes to it actual location

1/11/23 Lec # 21

→ Heap Sort

→ complexity = $\log n$

```
Void MaxHeap: removeMax (int &a)
{
    a = h[1];
    h[1] = h[curr size]; // replacing root by last value
    curHeapSize --;
    heapify (1); --> complexity logn
}
```

→ It is Not the part of heap class

```
Void heapSort (int * arr, int size)
{
    Max Heap mh (size+1);
    ~~mh. build heap (arr, size)~~  because it uses O(n) & heapify O(logn)

    mh. buildheap (arr, size) — O(n log n)

    // Ascending Order
    → This loop is for original array.
    for (int i = size-1; i ≥ 0, i--)
    {
        mh.removeMax (arr[i])
    } ↳ it will give ascending order

}
→ for ~~descendin~~ descending order
    (int ~~i~~ i = 0 ; i < size ; i++)
```

```
void main()
{ ~~int n;~~
    cin >> arraysize;

    // Create dynamic arr
    // input value from user
    // display array values

    heapSort (arr, arraySize)

}
```

→ $O(n \log n)$ → worst / avg.

~~$O(n)$ → best~~

15/11/23 Lec # 23

Conceptual Ques will come not Coding

Data Compression

Lets discuss with example

→ A file has following characters with freq. of each character

Freq: ~~no. of bits required for each character~~

Unique binary code for each character

	Freq	Bit no. required for each character
a	60	3 × 60 = 180
b	10	3 × 10 = 30
c	5	3 × 5 = 15
d	63	3 × 63 = _
e	5	3 × 5 = 15

Eg total bits for ~~whole data~~ file

We used 3 because we can represent 5 numbers in 3 bits. If the no. are 4 then we take 2 because in 2 bits we can represent 4 numbers.

If this is prob eg

a	0.6	3 × 0.6
b	0.1	
c		
d		
e		

Greedy Method

Huffman Coding Scheme / Algorithm

	Prob of freq.	Codes
a	0.3	00
b	0.1	1110
c	0.2	110
d	0.3	010
e	0.1	1111

Remember this and scheme is on next, next page

→ Step 1:

Set data in descending order w.r.t freq / prob. or prob.

agar freq. same hay to pahlay wo entery likha jo orignal table mai first thi

Steps:

a	0.3
d	0.3
c	0.2
b	0.1
e	0.1

a(0.3)
d(0.3)
c(0.2)
be(0.2)

a(0.3)
d(0.3)
cbe (0.4)
a(0.3)
d(0.3)
ad(0.6)
cbe (0.4)
~~abcde~~(1)
adcbe

We maintain descending order of prob.

(Start from end result while giving code)

Also remember

P.T.O

→ We will track the character from back or start through path and which ever ~~character~~ binary comes in its way added in its binary

~~Note : Dont focus on~~ Character ~~except a code separately.~~

⇒ You can start tracking from extreme left or right

⇒ Avg bit of each char = $\frac{\text{Bits of file}}{\Sigma f}$

⇒ Every character has unique code.

⇒ The character having max. character will be represented by min no. of bits.

a	0.3
d	0.3
c	0.2
b	0.1
e	0.1

a (0.3)
d (0.3)
c (0.2)
be (0.2)

a (0.3)
d (0.3)
cbe (0.4)

a (0.3)
dcbe (0.7)

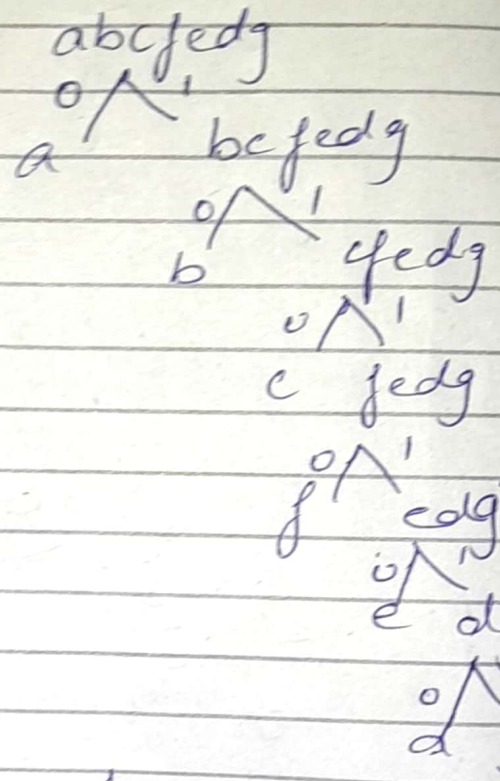
adcbe (1)

One more Example

Freq

a	40
b	30
c	20
d	10
e	14
f	16
g	5

Tree



Arranging

Code

a	40	0	1	1×40
b	30	10	2	2×30
c	20	110	3	3×20
f	16	1110	4	4×16
e	14	11110	5	5×14
d	10	111110	6	6×10
g	5	111111	6	6×5

Bits req. for file

a (40)
b (30)
c (20)
f (16)
e (14)
d (10)
g (5)

a(40)
b(30)
c(20)
f(16)
e(14)
dg(15)

a(40)
b(30)
c(20)
f(16)
edg(29)

a(40)
b(30)
c(20)
fedg(45)

a(40)
b(30)
cfedg(65)

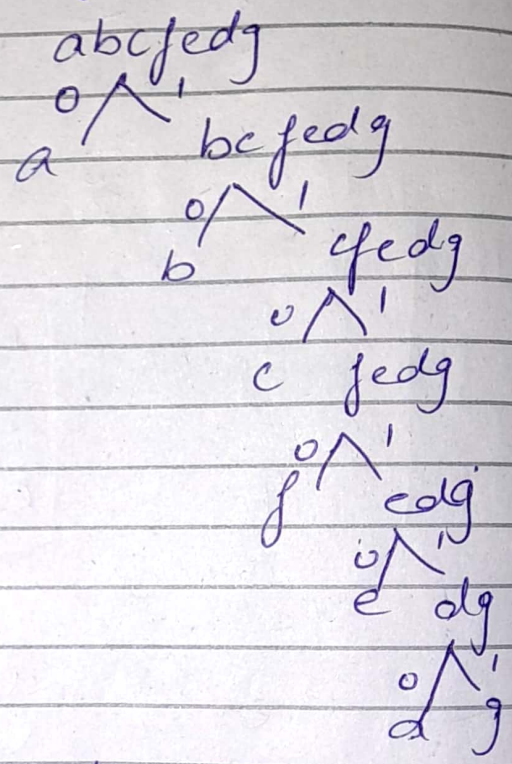
a(40)
bcfedg(95)

abcfed
(135)

One more Example

	Freq
a	40
b	30
c	20
d	10
e	14
f	16
g	5

Tree



Arranging

Code

a	40	0	1	1×40
b	30	10	2	2×30
c	20	110	3	3×20
f	16	1110	4	4×16
e	14	11110	5	5×14
d	10	111110	6	6×10
g	5	111111	6	6×5

Bits req. for file

a (40)
b (30)
c (20)
f (16)
e (14)
d (10)
g (5)

a(40)
b(30)
c(20)
f(16)
e(14)
d(10)
dg(15)

a(40)
b(30)
c(20)
f(16)
edg(29)

a(40)
b(30)
c(20)
fedg(45)

a(40)
b(30)
cfedg(65)

a(40)
bcfedg(95)

abcfed
(135)

27/11/23

Lec # 25

Data Structure

(i) Array based implimentation

Hashing & Hash maps or Hash Table

Terminologies :-

① keys : data or values

② hash key : Index where key has to be stored

③ Hash Table : Data Structure for storage of keys / data

④ Probing factor of key :-

how many time you find index

lets understand the concept using an example

Given Data : -> it will be given in exam

Hash function = h(k) = k mod 10

key / data = 12, 19, 10, 3, 4

Solution :

hash funct will also be given instead of k mod 10 it may be k mod x mod (10)² etc

(Page turn ⇒)

find key index accordy to formula.

How to decide total indexing

if mod 10 then ans = 0 - 9

if mod 6 then ans 0-5

if mod 100 then ans 0 - 99

⇓

Possible remainder which mod can give

Hash Table

0	10
1	
2	12
3	3
4	4
5	
6	
7	
8	
9	19

If our hash key approach to end then we again start from index 0.

⇒ Observed Automatically

Insertion of Keys

⇒ Find hash value of key using hash function

⇒ $k = 12$

$H(k) = 12 \bmod 10 = 2$

0	
1	
2	12
⋮	
9	

⇒ $h(19) = 9$

$h(10) = 0$

$h(13) = 3$

$h(4) = 4$

Problem

What if there is an other value 22 then hash ~~value~~ key of 22 is 2 and index

2 is already occupied. This is "called collision."

if collision then $O(n)$
if not then $O(1)$

Collision Solution

known as ⇒ Collision Resolution in Hashi

① Chaining (open hashing)
- → we create array of pointers while creating hash table
  - eg i.e each index will be a link list seperately

0	10
1	
2	12
3	3
4	4
5	
6	
7	
8	
9	19

⇒ 2 → 12
⇒ 3 → 12
⇒ 4 → 12
⇒ 5 → null ptr
⇒ on next page

⇒ insertion : $O(1)$
Search/delete = $O(n)$

Open Addressing (Closed hashing)
- → Linear Probing
- → Quadratic Probing
- → Double Hashing

linklist

key = 22

H(k) = 2

link list function

insert at start (22)

0	10
1	
2	12
3	3
4	4
5	
6	
7	
8	
9	19

$\Rightarrow$ 22 → 12

0	10
1	
2	22 → 12
3	3
4	4
5	
6	
7	
8	
9	19

Prob factor : 1

because we search index only one time and if collision we don't find index again we go to link list of that index.

② Open Addressing

① Linear Probing

collision function is independent of original hash function

Given Data

$h(k) = k \bmod 6$

Data : 3, 5, 15

if collision exists we use following hash function

$$h'(k) = \left[h(k) + i \right] \bmod 6$$

$h'(k)$ → h compliment

i → No. of attempts / collision

→ Donate next index where we have to update value in Table.

(P.T.O)

0	
1	
2	
3	3
4	15
5	5

$h(3) = 3$

$h(5) = 5$

$h(15) = 3 \Rightarrow$ already filled

$h(15) = 3$

Collision exists

so

$h'(k) = (3+1) \bmod 6 = 4$

Prob factor of 15 is 2

~~Collision eq for putting open addressing~~

② Quadratic: → Problem: Its is possible you find all the time those location which are occupied but there is space tab

key = —

$h(k)$ = — . ↓ because jump of i^2 is taken

if collision exists

$$h'(k) = \left[h(k) + \underline{\underline{i^2}}\right] \bmod 10$$

↓ → Difference

In linear i

In quad. i^2

→ Also linear

③ **Double Hashing :-**

Given

keys : —

h(k) : —

Collision exists

$$h'(k) = \underbrace{\left(h(k) + i(h_1(k))\right)}_{} \bmod \overset{\text{size}}{10}$$

where

$$h_1(k) = \underset{\text{size}}{10} - k \bmod (\underset{\text{size}}{10})$$

All the hashing keys will be given

$$h'(k) = \left(h(k) + i(h_1(k))\right) \bmod n$$

$$h_1(k) = n - k \bmod n$$

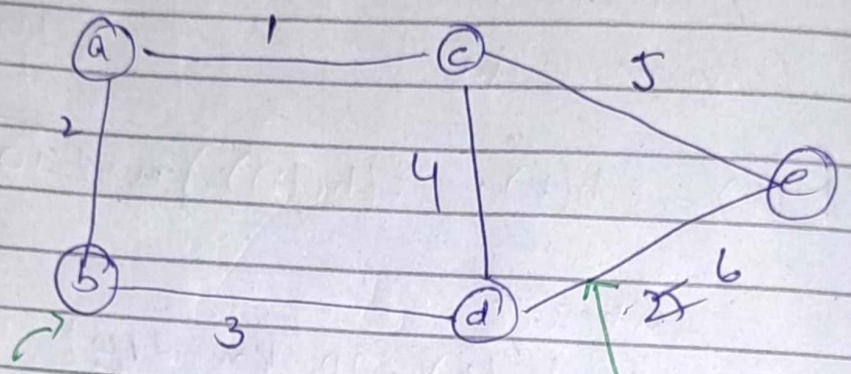
$$\left(h(k) + i(h_1(k))\right) \bmod 10$$

$$h_1 = 10 - k \bmod 10$$

29/11/23

Lec # 26

→ Graph or GRAPH THEORY



→ G(V,E) : Graph having V vertex & E edges.

→ Adjacent Vertex :

Vertex with common edge (E)

(A, C)

→ Adjacent Edges :

(1, 4)

(a, c, d)

common

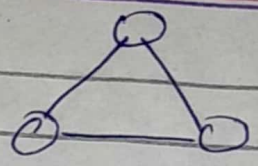
Edge with Comon Vertex

(1, 4) but not (1,3)

Note : Every tree is graph but not viceversa

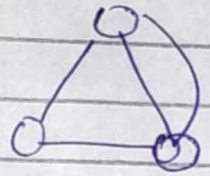
→ Simple graph

Graph with no loop and no ~~multiple path~~ multipath

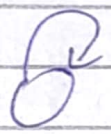


→ Multi graph

having multipath

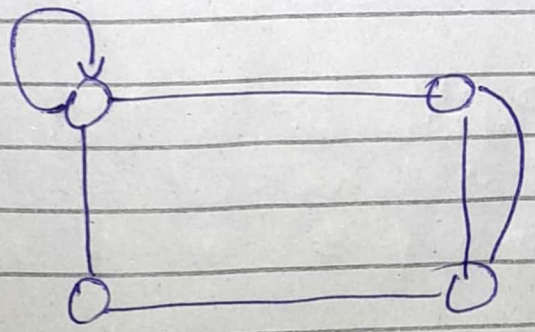


→ Graph with Self loop



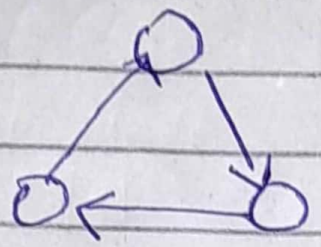
→ Pseudo Graph

Having self loop and multipaths

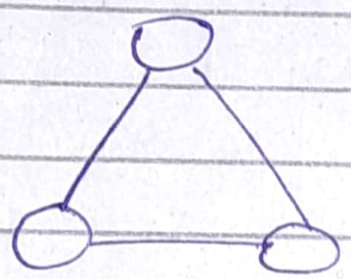


Major Types or Catagories of Graphs.

→ Directed Graph



→ Undirected Graph



→ Order of graph = $|V|$ = vertices no.

→ Size of graph : $|E|$: no. of edges

TRAVERSELS In GRAPH

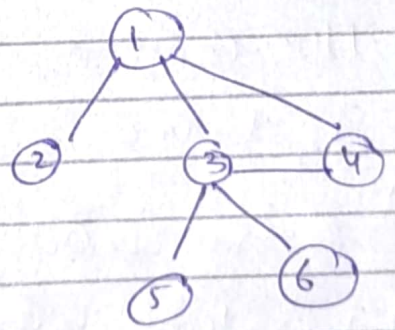
① Breath First Search (BFS)

- → Level Vise traverse
- → No ~~track~~ back tracking (dont look prev. node)
- → Uses Queue Data

② Depth First Search (DFS)

- → Use back tracking
- → Uses stack data Structure

=>



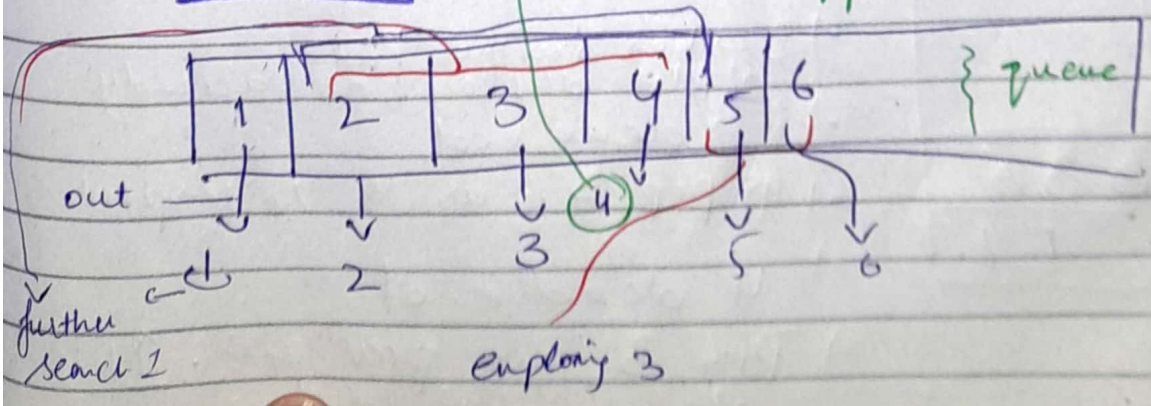
Output

1 2 3 4 5 6

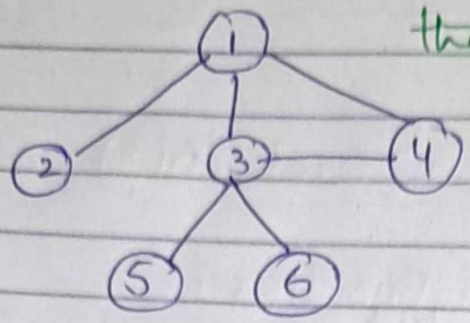
||

BFS

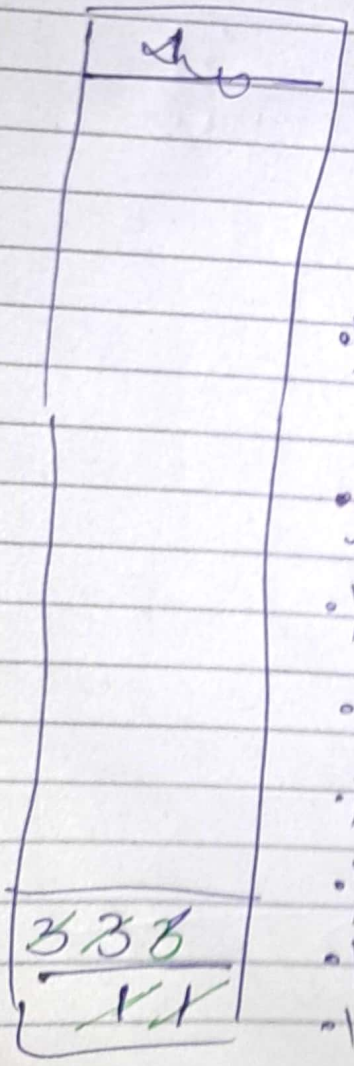
→ When you explore 4. 3 will come but we will not repeat it



→ **DFS** → Capture One node & reach its end. Then the node at the



•) 1 → Capture node and check are there any path of 1. Yes they are, so suspend it (i.e store in stack)



•) 2 → no path have it is dead end so track prev from stack ie 1 it is already displayed

•) Now check another path of 1. Found so suspend

•) 3 → no dead end have path

•) 5 → dead end

•) again track prev from stack.

•) 3 → has another path
  •) Not display 3 as already

•) 6 → dead end

•) pop 3. as 4 so suspend 3

•) 4 dead end

•) Prev 3, no prev path. 1 has already displayed

o) Now 1 has another path of 4 but it is already displayed.

Note: As 3 has 1 also in path but we dont display because it has already displayed

In case of duplicate note ie 2 or more times 4. Then we display it one time

Output

1, 2, 3, 5, 6, 4

When the graph is directed we dont do back because directed shows direction.

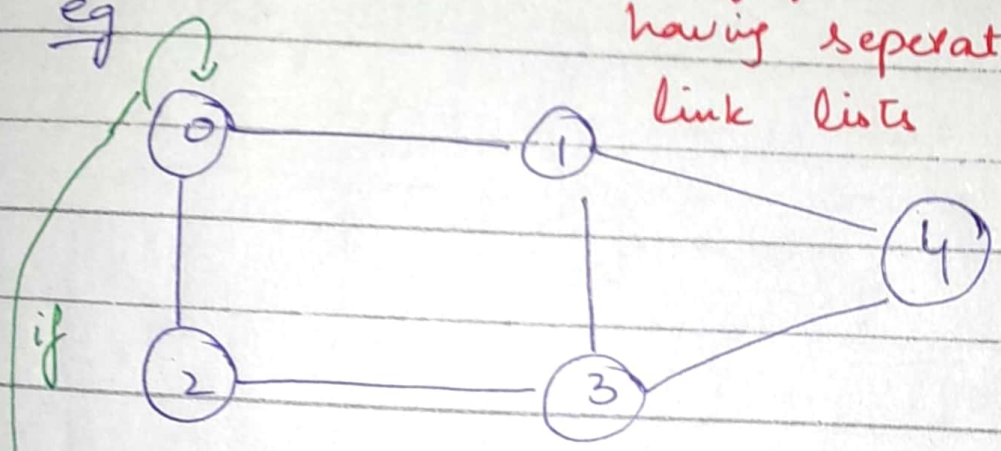
6/12/23 Lec # 28

→ Representation of Graph

① Using Adjency Matrix
② Using Adjency List

Array of pointers having seperately link lists

eg



if

Using Adjency Matrix

	0	1	2	3	4
0	10	1	1	0	0
1	1	0	0	1	1
2	1	0	0	1	0
3	0	1	1	0	1
4	0	1	0	1	0

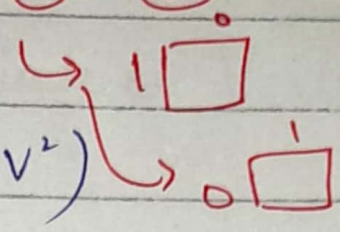
$V \times V = V^2$

✓ $(V+2E)$ because in it you check each nodes 2 times

i.e ① — ⓪

→ **Time Complexity :-**

$O(V^2)$



→ **Space Complexity :-**

$O(V^2)$

② Using Adjency List

① It not shows that 1 & 2 are connected

0	→ [1] → [2] → ϕ
1	→ [0] → [3] → [4] → ϕ
2	→ [0] → [3] → ϕ
3	→ [2] → [1] → [4] → ϕ
4	→ [3] → [1] → ϕ

→ **Space & Time Complexity :-**

For undirected ← $O(V+2E)$ → ~~Edge~~ Edge

② It only shows that 1 is connected to which nodes

$O(V+E)$ → for directed

Important point at end

→ No. of edges connected to the node.

→ In Order Degree of Node or Vertex :-

→ When to use Adjency Matrix or Adjency List

→ Adjency Matrix when graph is "dens".

↳ almost every Node is connected with every node.

→ Adjency list when graph is sparce.

→ eg :- → Indegree of '0' is 2.

→ Indegree of '3' is 3.