

## Stacks

Example:- Bilags plates arranged in a drum which is hot & sterilized so that plates should be drawn from Top. There is a spring which push a plate up when a plate is drawn.

### Applications

- (1) Recursion
- (2) Conversion of infix to postfix expression
- (3) Parsing Eg:- Matching parenthesis, tags, items
- (4) Browsing, Text editors (Back, Redo, Undo)
- (5) Evaluation of postfix expression
- (6) Tree traversals and Graph traversals

### Implementation of Stack using arrays

Disadvantage:- Predict the size of stack (array)

Advantage:- ① Faster ② 2nd disadvantage

All operations take constant time (in case of Arrays & LL)

```
int sstack[MAX];
```

```
int top=-1; //empty
```

```
void push (int item)
```

```
{
```

```
if (top == (MAX-1)) {
```

```
    printf ("Overflow");
```

```
else {
```

```
    top = top+1;
```

```
    sstack [++top] = item;
```

```
    if (top == -1) {
```

```
        printf ("Underflow");
```

```
    best use of stack is best to
```

```
if = best
```

```
int pop ()
```

```
{ int temp;
```

```
if (top == -1)
```

```
    printf ("Underflow");
```

```
return -1;
```

```
else {
```

```
    tempitem = stack[top]; // int temp = stack[top-1];
    top = top - 1; // return temp;
}
```

Note:-

Here, if we delete an element, top is decremented.

Later we push some element, we override the previous element

Takes  $O(1)$  time.

Linked list implementation of stack

Elements are created from heap memory dynamically

Struct node

```
{
```

```
    int i;
    struct node *link;
```

push(item int item)

```
{
```

```
// p->size
    struct node *p = (struct node*) malloc(sizeof(struct node));
```

```
if (p == NULL)
```

```
{
```

```
    printf(" Malloc error");
```

```
    return;
```

```
}
```

```
    p->data = item
```

```
(E+90) = qd
```

```
p->link = NULL; qd = (qd) + node
```

```
p->link = head;
```

```
// new created node linked to old head
```

```
head = p
```

```
// old head is updated to new head
```

```
}
```

int pop()

```
{
```

```
int item; struct node *p;
```

```
if (head == NULL)
```

STOP JAP

return; // Condition for singly linked list is empty

cout << "Pointed("Underflow")"; // Information about

error occurs when list is empty - it data base without all data

while (item != head) { // until item is "front" than "front" is head

item = item->next; // item pointer to next item than "front" is head

if (item == head) { // if item is head then "front" is head

temp = head; // If item is head then "front" is head

head = head->next; // front = head->next

free(temp); // free memory allocated to item

}

overflow error

Null pointer

### Implementation of a Queue using circular array

A Queue is an empty hollow cylinder. Both its ends are open

We shall put watermelons into it.

It will drop due to gravitational force from front to rear

Front → Return first element in the queue (first element)

Rear → Return last element in the queue (last element)

Deletion is done from front so that the insertion order of

Insertion is done from rear because it is maintained

de));

enqueue(item)

{

rear = (rear + 1) mod n;

if (front == rear)

{

front

3



// Rear &amp; front points to 0

APPEND IT IS overflow

if (rear == n - 1) pointf ("Q is full");

else if (rear == 0) return 1; // if (front == rear)

else front = rear - 1; // if (front == rear)

else if (q[rear] == item) return 1; // if (q[rear] == item)

else { q[rear] = item; return 1; }

else

front = (front + 1) mod n;

item = q[front];

return item;

102 A5E

GATE 2012

Suppose a circular queue of capacity  $(n-1)$  elements is implemented with an array of  $n$  elements. Assume that the insertion and deletion operations are carried out as using 'REAR' and 'FRONT' as array index variables respectively. Initially,  $\text{REAR} = \text{FRONT} = 0$ . The conditions  $\text{REAR} = \text{FRONT} = 0$  are the conditions to detect queue full and queue empty are:

Sol:-

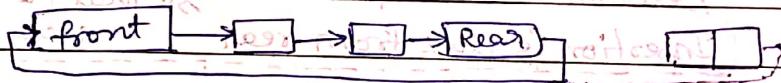
Queue Full

$$(\text{rear}+1) \bmod n = \text{front}$$

Queue Empty

$$\text{rear} = \text{full}$$

Gate 2004 A circularly linked list is used to represent a Queue. A single variable 'p' is used to access the queue. To which node should p point such that both the operations enqueue and dequeue can be performed in constant time?



Ans:-

 $p \rightarrow ?$ 

Sol:-

$$p \rightarrow \text{rear} \quad (\text{rear} + 1) = \text{rear}$$

Question

Gate 1994

$$(\text{rear} + 1) = \text{rear}$$

(option)

Which of the following permutations can be obtained in the output (in the same order) using a stack assuming the input is, in that order 1, 2, 3, 4, 5 in that order?

- a) 3, 4, 5, 1, 2    b) 3, 4, 5, 2, 1    c) 1, 5, 2, 3, 4    d) 1, 5, 4, 3, 2

Check  
for a)

5	2	3
4		
3		
2		

b) is  
3 4 5 2 1

8	4	2	1
7	3		
6			
5			

5, 4, 3, 1, 2

Gate 1997

(T.B) ~~four~~ three days.

A priority queue 'Q' is used to implement a stack 'S' that stores characters.  $\text{PUSH}(c)$  is implemented as  $\text{Insert}(Q, c, k)$  where  $k$  is appropriate integer key chosen by implementation as  $\text{DElemin}(Q)$ . For a sequence of push operations, the keys chosen are in

- a) non-increasing order // decreasing order

b) non-decreasing order

c) strictly increasing order

d) strictly decreasing order

Gate 2003

Let 'S' be stack of size  $N \geq 1$ , starting with the empty stack. Suppose we push first ' $n$ ' natural numbers in sequence, and then  $n$  pop operations. Assume that push & pop operations take  $X$  seconds each, and  $Y$  seconds elapse between the end of one such stack operation and the start of the next operation. For  $m \geq 1$ , define the stack life of  $(m)$  as the time elapsed from the end of  $\text{push}(m)$  to the start of the  $\text{pop}$  operation that removes  $m$  from  $S$ . The average stack-life of an element of this stack is

5

komplexes 3er-System push or pop: x je vorhanden

Carsten Bleibrecht ist der Blumenzopf aus dem Plattenbau

1. introduction

## Push + Stack life - Pop

One element in  $X$

- X

$$\text{Two elements: } 2x + 2y - 2x = 2(x+y) - x$$

With best regards to your family

$$-nx = n(x+y) - x$$

$$+ 10) + 10S_{100} = 2000 \text{ thousand}$$

Printed 2006

An implementation of a queue, using two stacks S1 & S2, is given below

void insert(Q, x)

if stack empty(S1) then  
push(S1, x)  
else push(S1, x) and mark top of S1 node.

void delete(Q, x)

{

if stack empty(S2) then  
pop(S1) and mark top of S2 node  
if stack empty(S1) then  
return  
else  
point("Q is empty");  
return;

Sol:

else while (!stack empty(S1))

do {  
x = pop(S1);  
push(S2, x);

if stack empty(S1) then  
push(S2, x);  
return

else if stack empty(S2) then  
x = pop(S1);  
push(S2, x);  
return

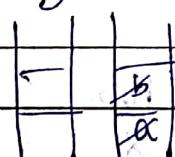
else if stack empty(S1) then  
x = pop(S2);  
push(S1, x);  
return

else if stack empty(S2) then  
x = pop(S1);  
push(S2, x);  
return

Let  $m$  insert &  $n$  ( $\leq n$ ) delete operations be performed  
in arbitrary order on an empty queue  $Q$ . Let  $x \neq y$  be  
the number of push and pop operations performed

respectively in the process. Which one is true for all  $m$  &  $n$ ?

Sol:



Elements :

2 push  $\Rightarrow m$   $\leq x \leq 2n$  (Ans)

2 pop  $\Rightarrow 2m \leq y \leq m+n$  (Ans)

1 push  $\Rightarrow$  Best case:  $m+n - m = m$  // Removal of first

1 pop  $\Rightarrow$  Worst case:  $2n$  // Insertion of last

Pop: Best case =  $2m$  // Removal of first ele

Worst case =  $2n + (m-n)$  // Removal of last ele

$= m+n$

Push: Best case =  $m+n$  // Insertion of first ele

Worst case =  $2m$  // Insertion of last ele

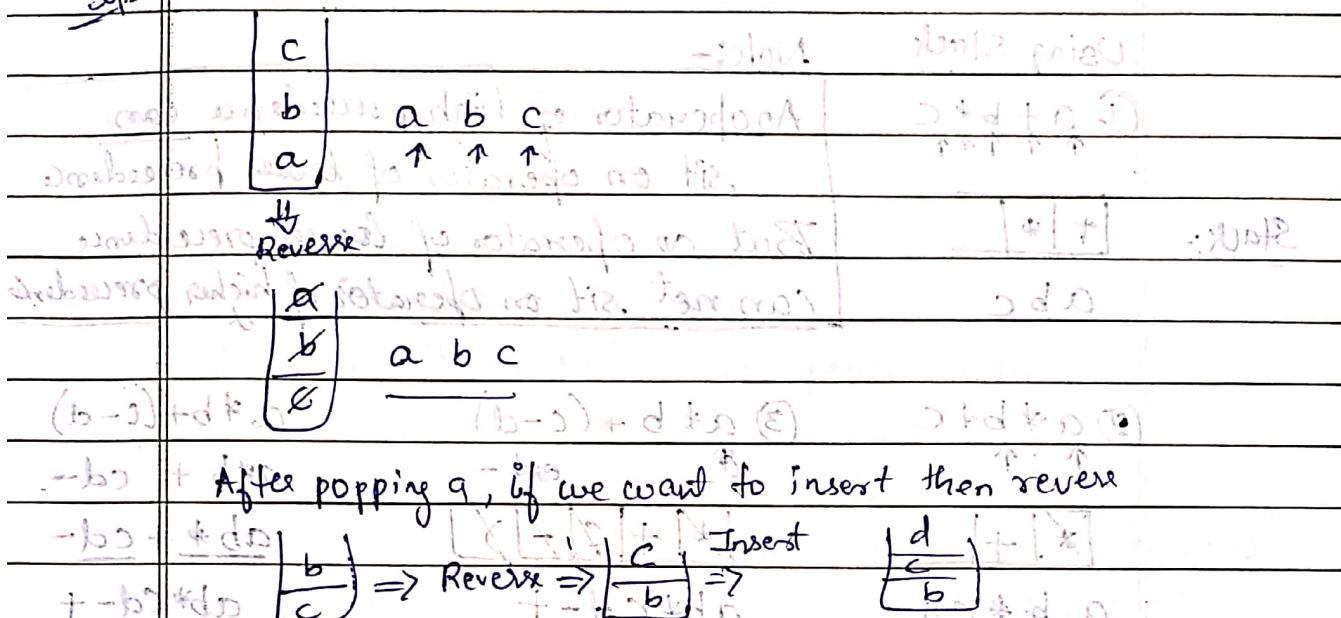
$m+n \leq x \leq 2n$  (Ans)

$2m \leq y \leq 2n + (m-n)$  (Ans)

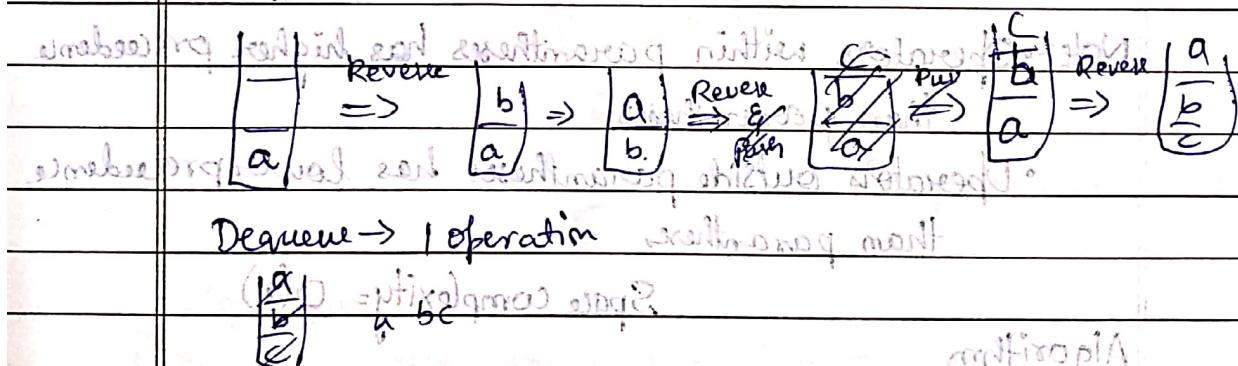
## GATE-2000 ↔ GATE 2014

- Q1 Suppose a stack implementation supports, in addition to PUSH and POP, an operation "reverses" the order of elements on the stack. Implement a queue using the above stack implementation. Show how to implement "ENQUEUE" using a single operation and "DEQUEUE" using a sequence of 10 operations.

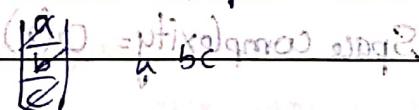
Sol:-



Engineer → Reverse Stack → 3 operation



Degener → 1 operation ~~excluding mont~~



~~all else  
other &  
interest~~

h of  
le to  
es  
ick &  
now

# Applications of Stacks

SURYA Gold

Date \_\_\_\_\_

Page \_\_\_\_\_

## 1) Conversion of infix expression to postfix expression

Eg:-  $a + b * c$   $\rightarrow$   $a b * + c$

$$\textcircled{1} \quad a + b * c \quad \textcircled{2} \quad a + b - c \quad \textcircled{3} \quad a + (b - c)$$

$$ab * + c \quad a + bc * \quad ab + - c \quad a + bc -$$

$$ab * c + \quad abc * + \quad ab + c - \quad abc - +$$

Using Stack

Note:-

$$\textcircled{1} \quad a + b * c$$

An operator of high precedence can sit on operator of lower precedence

Stack:-

$| + | * |$

abc.

But an operator of lower precedence can not sit on operator of higher precedence

$$\textcircled{2} \quad a * b + c$$

$$\textcircled{3} \quad a * b + (c - d)$$

$$a * b + (c - d)$$

$| * | + | b |$

$| * | + | c | | - | d |$

ab \* c +

ab \* cd - +

$a * b + c d -$

+ < (

Note:- Operators within parentheses has higher precedence than parentheses

• Operators outside parentheses has lower precedence than parentheses

Space complexity =  $O(n)$

Algorithm

①

Create a stack

②

for each character 't' in the input

    if ('t' is an operand)

        output 't';

    else if ('t' is a right parenthesis)

        pop and output tokens until a left parenthesis is popped (but don't output)

else // t is an operator or left parenthesis  
 { ~~while (t != '+' & t != '-' & t != '\*' & t != '/' & t != '(')~~

pop and output tokens until one of lower priority than 't' is encountered or a left parenthesis is encountered or stack is empty.

Push t

}

~~above function~~

→ Time complexity :-  $O(n)$

~~(stack push)~~

Uses operator stack ~~(stack push)~~

b) Postfix evaluation algorithm: ~~(stack push)~~

Time complexity:-  $O(n)$ , Uses operand stack

Ex: ①  $3 * 2 + 1$     ②  $3 + 2 * 1$     ~~(stack push)~~ ~~(stack push)~~ ~~(stack push)~~ ~~(stack push)~~ ~~(stack push)~~

$6 + 1$      $5 + 1$     ~~(stack push)~~ ~~(stack push)~~

Postfix:-

$3 * 2 + 1$

First operand becomes

$3 2 * +$

Second operand

and vice versa

$3 2 * +$

using (a) O cannot be ~~(stack push)~~ ~~(stack push)~~ ~~(stack push)~~ ~~(stack push)~~ ~~(stack push)~~

we can see that if we encounter an operator, then we have to apply it to previous elements.

$3 + 2$

Space complexity:-  $O(n)$

$5$

~~(stack push)~~

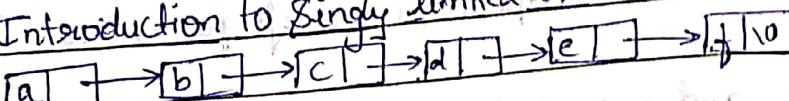
~~(stack push)~~

Algorithm

- ① Scan the postfix string from left to right
- ② Initialize an empty stack
- ③ Repeat steps 4 & 5 until all the characters are scanned
- ④ If the scanned character is an operand, push it onto stack
- ⑤ If the scanned character is an operator, & if operator is unary then pop an element from the stack. If the operator is binary, then pop two elements from the stack. After popping the elements, apply the operator to those popped elements. Push result onto the stack
- ⑥ After all the elements are scanned, the result will be in the stack

## Linked List

## Introduction to Singly linked list



**head** → **center** of mass → **center** of gravity → **center** of buoyancy

Struct node

char data;

char data;  
struct node \*link; struct node \*head;

3.

Q. ① struct node \*p; ~~struct node \*head;~~

$p = \text{head} \rightarrow \text{link} \rightarrow \text{link};$  (a) Optimal length unit?

$p \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} = p;$

~~head->link = b->link;~~

i. `printf("%c", head->link->link->link->link->data);`

Advantage :-

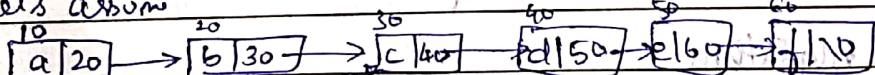
1) Sequential access of element. Takes  $O(n)$  time

In arrays, it takes  $O(1)$  time

2) Searching takes  $O(n)$  time if elements are unordered

Ans für Q

Let's assume



$p = \text{head} \rightarrow \text{link} \rightarrow \text{link item}$ , so it's right, forget it.

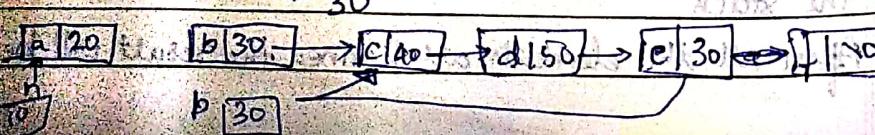
$20 \rightarrow \text{link}$

© 2018 Pearson Education, Inc.

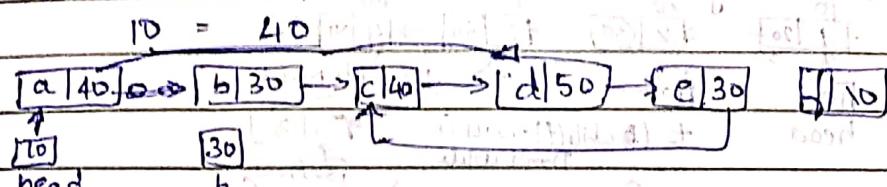
$\Rightarrow$  link  $\rightarrow$  link  $\rightarrow$  link  $\vdash b$  and elements in first row

40  $\rightarrow$  link  $\rightarrow$  link = 50  $\rightarrow$  link.

$$50 - \frac{b}{2} = b$$



$\text{head} \rightarrow \text{link} = p \rightarrow \text{link}$ ;  $p$  must be initialized.



( $\text{head} \rightarrow \text{link}$ ) initial ( $\&$  three pointers) - work done below.

print  $\text{head} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{data}$

$40 \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{data}$

$50 \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{data}$

$30 \rightarrow \text{link} \rightarrow \text{data}$

total of 4 pointers  $\rightarrow 40 \rightarrow \text{data}$  since  $\text{link}$ .

$d_{\text{node}} = \text{data} + \text{more}$  : initialized  $\text{data}$   
 $\text{more} = \text{length}$

Traversing a singly linked list :  $\&$  three pointers.

On a list, we can perform (from  $i=1$ )  $\text{list[i]}$  along EA

i) Traversing  $\text{list[i]} = p$

II) Inserting ( $\text{list[i]} = \{\text{list[i-1]}, p\}$ )  $\text{list[i]}$

iii) Deleting  $\text{list[i]} = p$

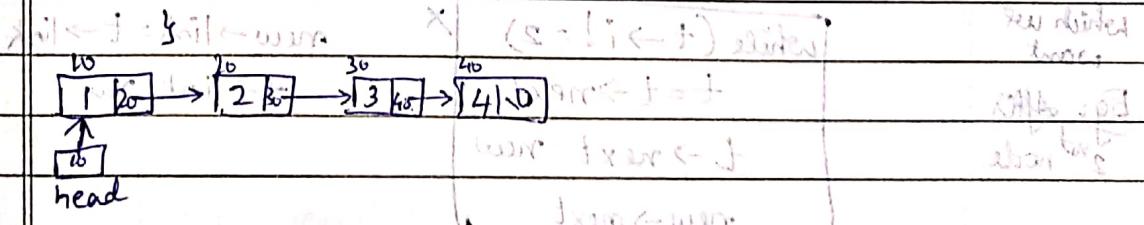
struct node

{

int i;

struct node \*link;

head: it shows length of list



Moving head may cause loss of nodes created dynamically

(since it does not have any name) are lost?

struct node \*t; // t is temporary variable

$t = \text{head}$

while ( $t \neq \text{NULL}$ ) {

printf("%d", t->data); }  $\oplus$  printf("%d", t->i);  $\&$  show more

printf("%d", t->link);  $t = t \rightarrow \text{link};$

nodes (10, 20, 30, 40)  $\&$  10, 20, 30, 40

while ( $t \neq \text{NULL}$ )  $\oplus$  while ( $t \rightarrow \text{link} \neq \text{NULL}$ ) {

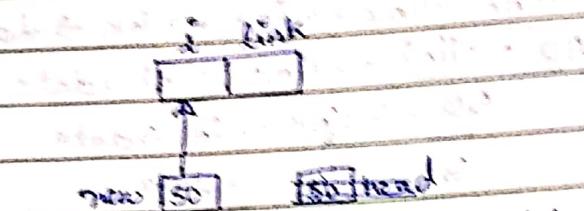
(10, 20, 30, 40)

Inserting an element in SLL



new node =  $t$  ( $t = \text{new}(\dots)$ )  
link =  $t \rightarrow \text{link}$

Struct node \*new; (Struct node \*) malloc(sizeof(Struct node))



head = new // Don't do this info is lost

At beginning :- new  $\rightarrow$  link = head

head = new

At end :-

Struct node \*t; t = head; what is problem?

while ( $t \neq \text{NULL}$ ) // Don't write in a loop

$t = t \rightarrow \text{next}$

while ( $t \rightarrow \text{next} \neq \text{NULL}$ ) //  $t \rightarrow \text{next}$  is temporary variable

$t = t \rightarrow \text{next}$

$t \rightarrow \text{next} = \text{new}$

$\text{new} \rightarrow \text{next} = \text{NULL}$

After node :-

which we want

Eg: after  
2nd node

Struct node \*t = head; (main char \*line)

while ( $t \rightarrow i \neq 2$ )

$t = t \rightarrow \text{next}$

$t \rightarrow \text{next} = \text{new}$

$\text{new} \rightarrow \text{next}$

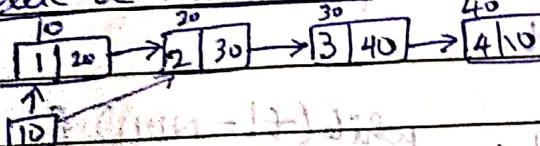
$\text{new} \rightarrow \text{link} = t \rightarrow \text{link}$

$t \rightarrow \text{link} = \text{new}$

head

After inserting 10 into 2nd position (first new) now head given M

Delete a node in SLL (most time work is to find node)



At beginning :-

Struct node \*t = head; head = head  $\rightarrow$  next

free(t); // free the deleted node

if (head == NULL) return;

if (head  $\rightarrow$  next == NULL) (work = if) linked

free(head)

head = NULL;

From:  
tail

struct node \*t = head  
while ( $t \rightarrow \text{next} \rightarrow \text{next} \neq \text{NULL}$ )  
 $t = t \rightarrow \text{next}$   
free ( $t \rightarrow \text{next}$ )  
 $t \rightarrow \text{next} = \text{NULL}$

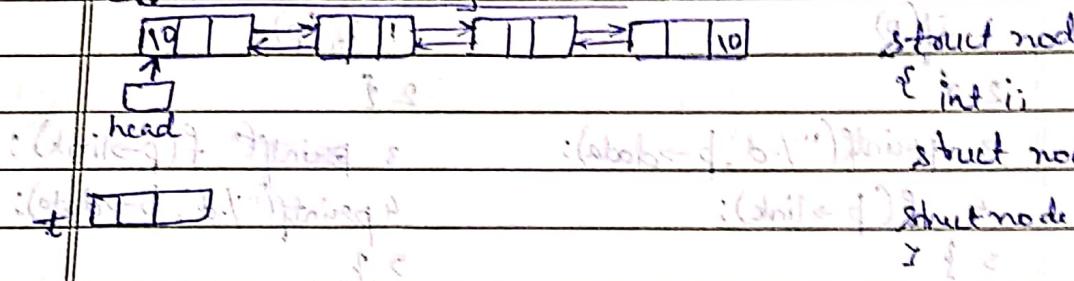
A practical  
note:  
Eg: 3

struct node \*t = head  
while ( $t \rightarrow \text{next} \rightarrow i \neq 3$ )  
 $t = t \rightarrow \text{next}$ ,  $i = i + 1$   
struct node \*l =  $t \rightarrow \text{next}$   
 $t \rightarrow \text{next} = t \rightarrow \text{next} \rightarrow \text{next}$   
free ( $t \rightarrow \text{next}$ )

(deletion from left)

(deletion from right)

### Insertion into doubly linked list



At  
beginning:  
 $t \rightarrow \text{next} = \text{head}$

$\text{head} = t$

$t \rightarrow \text{previous} = \text{NULL}$

~~$t \rightarrow \text{previous} = \text{head}$~~

At  
end:  
while ( $p \rightarrow \text{next} \neq \text{NULL}$ )

$p = p \rightarrow \text{next}$

$p \rightarrow \text{next} = t$

$t \rightarrow \text{previous} = p$

$t \rightarrow \text{next} = \text{NULL}$

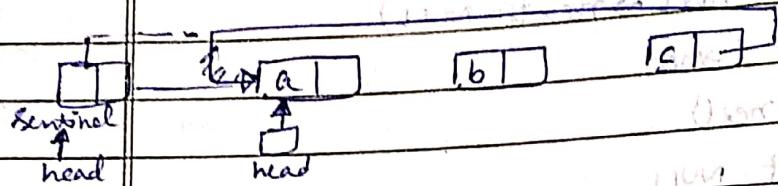
After  
Node:  
Eg:

$t \rightarrow \text{prev} = p$

$t \rightarrow \text{next} = b \rightarrow \text{next}$  (here  $b \rightarrow \text{next}$  is NULL)

$b \rightarrow \text{next} = t$

$t \rightarrow \text{next} \rightarrow \text{prev} = t$

Circular linked list $b = \text{head}$ while ( $b \rightarrow \text{next} \neq \text{head}$ )

Sentinel is a special node that contains count of nodes pointed by head

Printing the elements of SLL using recursion

Program 1:-

void f(struct node \*p)

{

1 if(p)

{

3 printf("%d", p-&gt;data);

4 f(p-&gt;link);

5 }

{

head

p  $\rightarrow$  100, popped offp f()  $\rightarrow$  400p f()  $\rightarrow$  300p f()  $\rightarrow$  200p f()  $\rightarrow$  100

main()

(1) Program 2:-

void f(struct node \*p)

{

1 if(p)

{

3 printf(f(p-&gt;link));

4 printf("%d", p-&gt;data);

5 }

{

head = front = p

front

p  $\rightarrow$  100, f()p  $\rightarrow$  400, f()p  $\rightarrow$  300, f()p  $\rightarrow$  200, f()p  $\rightarrow$  100, f()

main()

O/p: 1 2 3 4

O/p: 4 3 2 1

Reversing an SLL using iterative program + sentinel node

int t3() {  
    return sum of all even numbers

struct node \*next;

head & done function

(f) reverse linked list (using previous node)

struct node \*reverse (struct node \*cur)

{

    struct node \*prev = NULL, \*nextNode = NULL;

    while (cur){

        nextNode = cur->next;

        cur->next = prev;

        prev = cur;

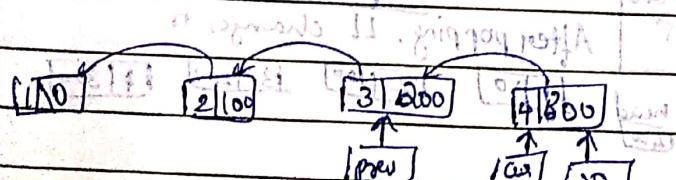
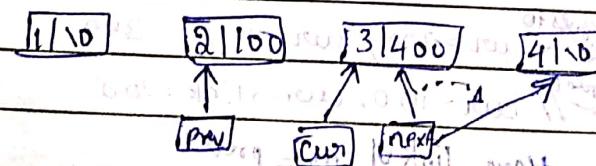
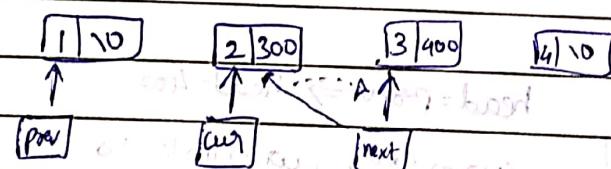
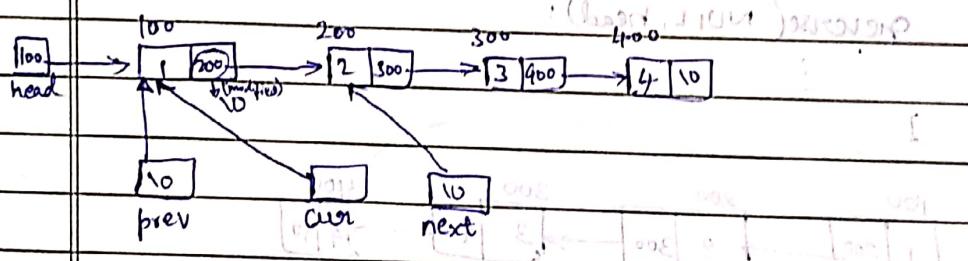
        cur = nextNode;

}

    return prev;

}

beginning



head

400

100 → 200 → 300 → 400 a list of pointers is initialized with 400 as head.

Recursive program for reversing a Singly Linked List

Struct node \*head;

Void reverse(Struct node \*prev, Struct node \*cur)

1 if(cur)

{ If cur is not NULL then go to step 2.

2 reverse(prev, cur->link);

3 cur->link = prev;

4 }

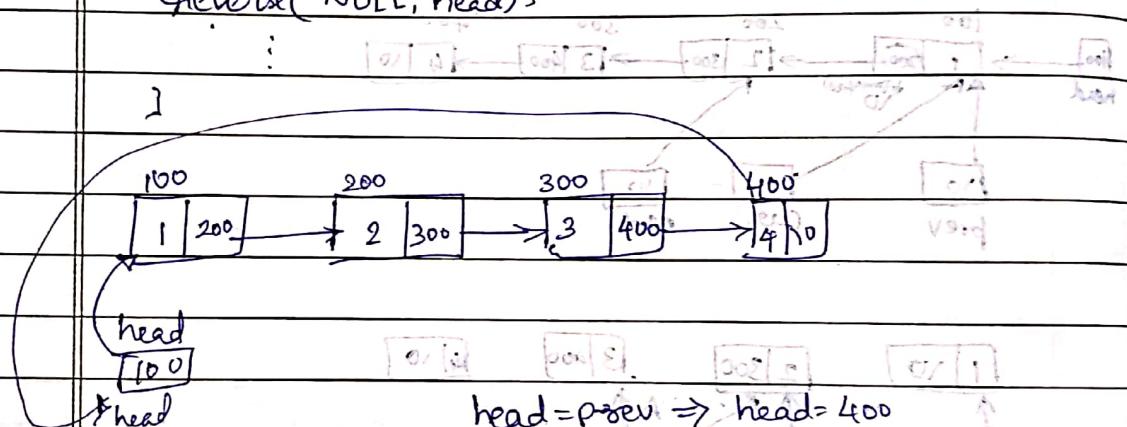
else

head = prev;

void main()

:

reverse(NULL, head);



head = prev  $\Rightarrow$  head = 400

400	rev	400 prev cur	cur $\rightarrow$ 400, cur $\rightarrow$ link = 100
-----	-----	--------------------	---

400	rev	300 prev cur	cur = 300 cur $\rightarrow$ link = 400
-----	-----	--------------------	--

400	rev	300 prev cur	go to (cur = 200, cur $\rightarrow$ link = 300)
-----	-----	--------------------	---

400	rev	200 prev cur	go to (cur = 100, cur $\rightarrow$ link = 200)
-----	-----	--------------------	---

400	rev	100 prev cur	Here link of cur = prev;
-----	-----	--------------------	--------------------------

main() After popping, LL changes to

head	100	12100	13200	14300
------	-----	-------	-------	-------

Gate 2008

The following C function takes a singly linked list of integers as a parameter and rearranges the elements of the list.

The list is represented as a pointer to the structure..

The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in given order. What will be the contents of the list after the function completes execution.

`struct node {`

`int value;`

`struct node *next;`

`};`

`void rearrange(struct node *list)`

`{`

`struct node *p, *q;`

`int temp;`

`if (!list || !list->next) return;`

`p = list;`

`q = list->next;`

`while (q)`

`{`

`temp = p->val;`

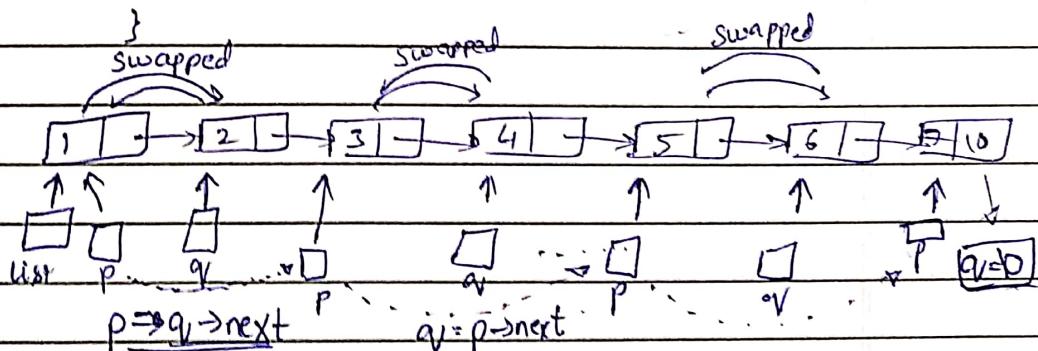
`p->val = q->val;`

`q->val = temp;`

`p = q->next;`

`q = p ? p->next : 0;`

`}`



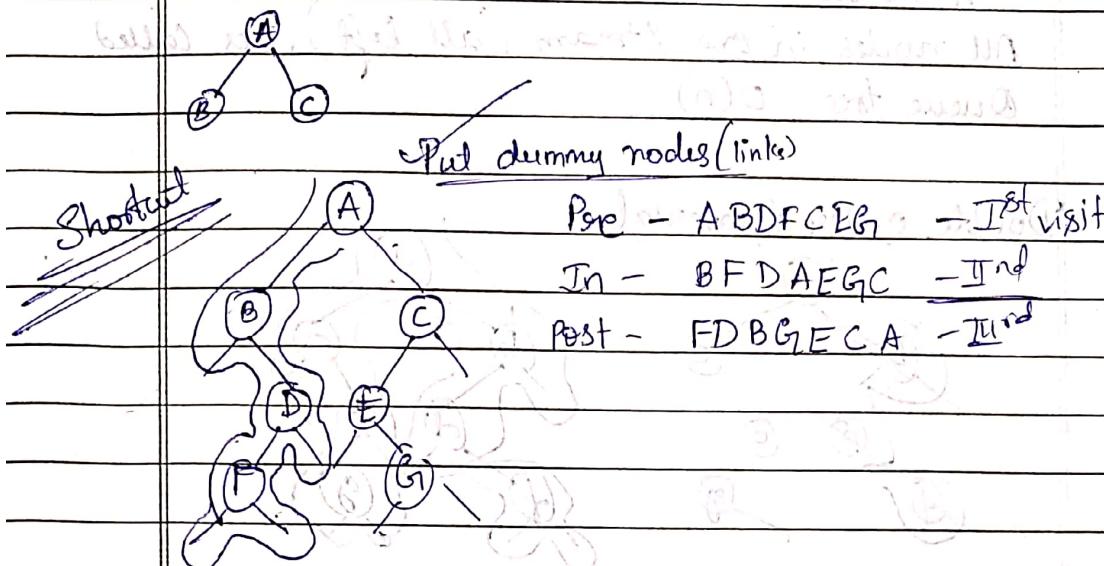
O/P: 2 1 4 3 6 5 7

Gate 2010

The follow

Introduction to tree traversals

- 1) Inorder traversal - Left Root Right - A B C
- 2) Preorder traversal - Root Left Right - ABC
- 3) Postorder traversal - Left Right Root - BCA



Pre - A B D F C E G - I<sup>st</sup> visit  
 In - B F D A E G C - II<sup>nd</sup>  
 Post - F D B G E C A - III<sup>rd</sup>

Implementation of traversals and time and space analysis

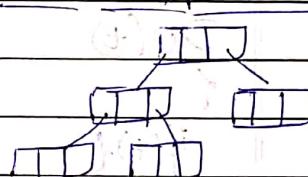
struct node

{

int data;

struct node \*left, \*right;

}



void Inorder(struct node \*t)

{

if(t!=NULL) {

if(t)

1 Inorder(t-&gt;left);

2 printf("%d", t-&gt;data);

3 Inorder(t-&gt;right);

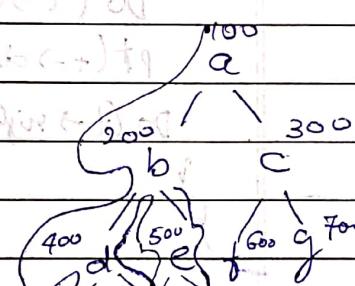
}

points

node during second visit

Op: 400 200 500

d b e a b.



t=100	t=100	t=100

# visiting node takes constant time

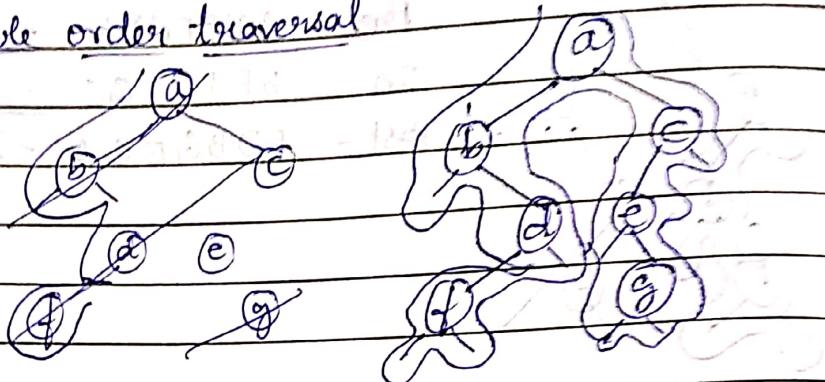
Every node visited 3 times

Time complexity:  $O(n)$

Space complexity: Depends on levels of tree  
 $n$  nodes  $\rightarrow n$  levels

All nodes in one stream (all left), It's called  
Queue tree.  $O(n)$

### Double order traversal



b f d a e c g o

explains along two unit how traversal is maintained

DO(t)

if ( $t \neq \text{null}$ )

pt( $t \rightarrow \text{data}$ ); (Info in node  $t$  is printed)

DO ( $t \rightarrow \text{left}$ )

pt( $t \rightarrow \text{data}$ ) (Info from node  $t$  is printed)

DO ( $t \rightarrow \text{right}$ )

(Info in node  $t$  is printed)

### Triple Order Traversal

TO ( $t$ )

if ( $t \neq \text{null}$ )

if ( $t \neq \text{null}$ )

if ( $t \neq \text{null}$ )

traversing prints like this

and so on

and so on

and so on

```

    pf("%d", t->data) print provided
    TO(t->LC); print left child b
    Pf("%d", t->data)
    TO(-t->RC)
    Pf("%d", t->data).
}
  
```

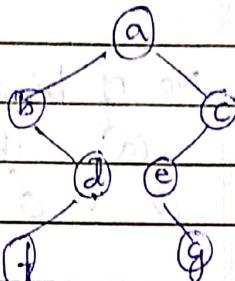
a b d d d b b a c c e e c a result in left skewed tree

### Indirect recursion on trees

void A(struct node \*t)

{ if(t)

1. printf("%d", t->data);
2. B(t->left)
3. B(t->right)



void B(struct node \*t)

{ if(t)

1. printf("%d", A(t->left));
2. printf("%d", A(t->right));
3. A(t->right);

A(B) <= Recursion, sekar yg

}

if(t) print

Skew binary tree is tree which has only one type of subtrees. If a tree has only left subtrees then it is left skewed tree. vice versa

Skew binary tree is tree which has only one type of subtrees. If a tree has only left subtrees then it is left skewed tree. vice versa

Number of binary trees possible

① Node  $\Rightarrow$  No. of binary trees possible

1	1	
2	2	
3	3	
4	4	

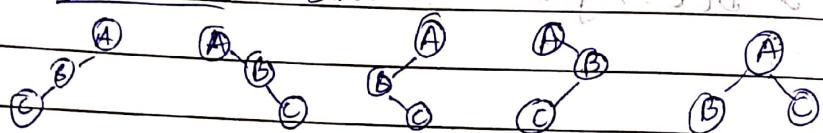
No. of binary trees for  $n$  nodes (for unlabeled nodes)

$$3 \text{ nodes} = \frac{2n}{n+1} C_n = \frac{2 \times 3}{4} C_3 = \frac{6}{4} C_3 = 5$$

No. of binary trees for  $n$  nodes (for labeled nodes)

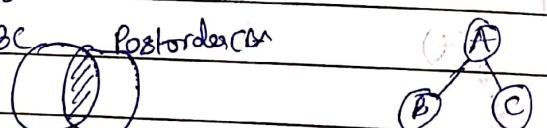
$$\begin{array}{c} \text{Diagram of a binary tree with 5 nodes labeled A, B, C, D, E.} \\ \text{Left side: } = 2n C_n = \frac{6!}{3!3!} = \frac{6!}{3!3!} \\ \text{Right side: } (n+1)n! = 4 \times 3 \times 2 \times 1 = 4! \\ \text{Bottom part: } (n-1)n(n-2) \dots 3 \times 2 \times 1 = 4 \times 3 \times 2 \times 1 \end{array}$$

Preorder ABC - 5 trees



Preorder ABC & Postorder CBA

Preorder ABC      Postorder CBA



Inorder - Back Preorder

Inorder BCA & Postorder CBA

(Inorder CBA)  $\rightarrow$  Postorder CBA

$n$  nodes, Preorder  $\rightarrow$   $\frac{2n}{n+1} C_n$   
Binary trees

Preorder & Postorder - NO unique BT or more than

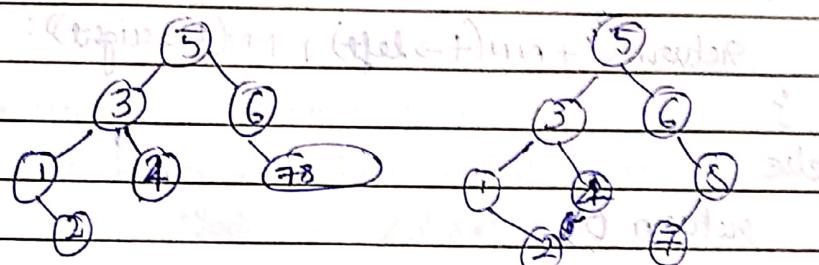
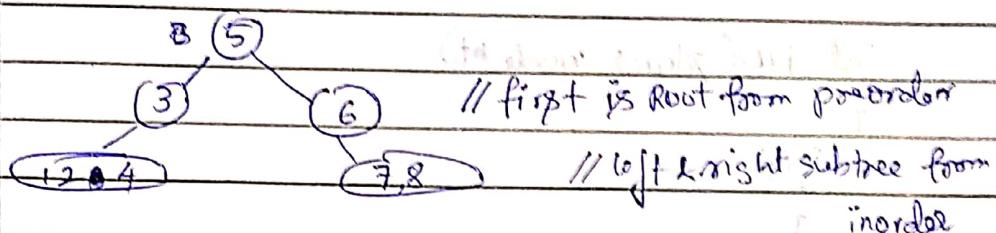
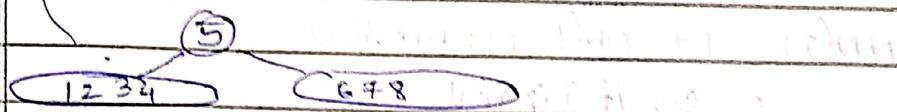
Inorder, Postorder, Preorder  $\rightarrow$  Unique BT & exactly one

Construction of unique binary tree

Inorder: 1, 2, 3, 4, 5, 6, 7, 8

Preorder: 5, 3, 1, 2, 4, 6, 8, 7  
Postorder = ?

Inorder: 1, 2, 3, 4, 5, 6, 7, 8

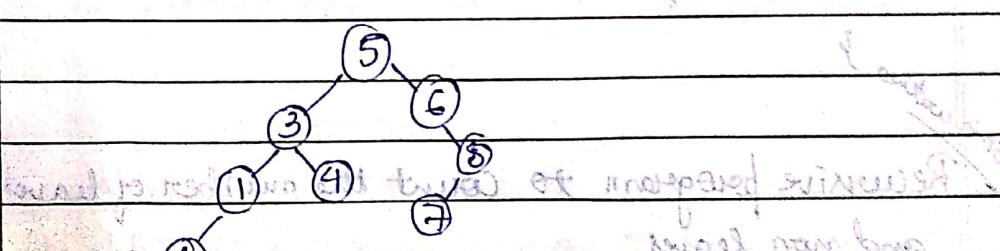
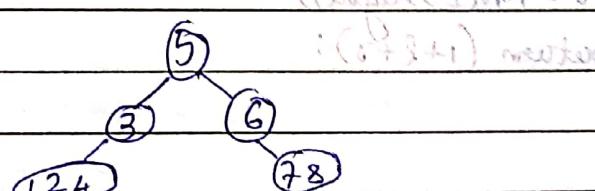
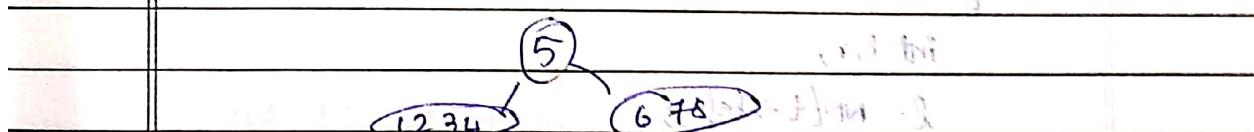


Postorder: 1, 2, 4, 3, 7, 8, 6, 5 // 3<sup>rd</sup> time

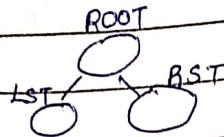
(It does not work this)

② Inorder: 1, 2, 3, 4, 5, 6, 7, 8

Postorder: 2, 1, 4, 3, 7, 8, 6, 5



*Time watched*  
Recursive program to count the number of nodes



$$NN(T) = 1 + NN(LST) + NN(BST)$$

: 0; if T is null

int NN(struct node \*t)

{ if (t) { } else { } }

return {

    return (1 + NN(t->left) + NN(t->right));

}

else { }

    return 0;

}

OR

int NN(struct node \*t)

{

if (t)

{ }

int l, r;

l = NN(t->left);

r = NN(t->right);

return (1 + l + r);

}

else

    return 0;

}

*Time watched*  
Recursive program to count the number of leaves  
and non leaves

$$NL(T) = 1 ; \text{ Tree is a leaf}$$

$$= \frac{NL}{t} (LST) + NL(RST) + 1$$

(if t is leaf, then NL(t) = 0)

$NL(\text{struct node } *t)$  : a function to find NL(t)

{

```

if (t == NULL)
    return 0;
if (t->left == NULL & t->right == NULL)
    return 1;
else
    return (NL(t->left) + NL(t->right));
}

```

(if t is a full node, then NL(t) = 1)

Recursive program to find the full nodes.

Node having 2 children - Full node.

Full Node is a subset of Non leaf

Not every non leaf is a full node

if  $FN(T) = 0$ ,  $T = \text{NULL}$  or it has 1 child

if  $T$  has 2 children, then  $FN(T) = 1$  if all its children are full nodes

$= FN(T \rightarrow LST) + FN(T \rightarrow RST)$  i.e.,  $T$  has 1 child

i.e., it is full if  $FN(T \rightarrow LST) + FN(T \rightarrow RST) + 1 = 1$

$$0 \leq 1 - 1 \leq 0$$

int FN(struct node \*t), more to be added (A)

(i.e., 0-1) condition is added to reduce the above

if (!t) return 0;

if ( $t \rightarrow \text{left} \neq t \rightarrow \text{right}$ ) i.e., non full (B)

return 0;

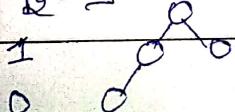
else if  $\text{return } FN(t \rightarrow \text{left}) + FN(t \rightarrow \text{right}) + (t \rightarrow \text{left} \neq t \rightarrow \text{right})$

1:0;

more realistic test program is to type minimum edit (F)

Recursive program to find the height of a tree.

Height of any node, to leaf its distance



$$H(T) = \begin{cases} 0 & \text{if } T \text{ is null} \\ 0 & \text{if } T \text{ is a leaf} \\ 1 + \max(H(LST), H(RST)) \end{cases}$$

int H(struct node \*t)

{

    if (!t) return 0;  
    if (!t->left & !t->right) return 0;  
    return (1 + H(t->left)) > H(t->right))

int l, r;

l = H(t->left)

r = H(t->right)

//return (1 + max(l, r)).

return (1 + H(l > r) ? l : r);

GATEFORM

Binary Search Trees = binary tree

Properties:-

- ① A tree with  $n$  nodes has exactly  $(n-1)$  edges.
- ② In a tree, every node except root has exactly one parent.
- ③ Maximum no. of nodes in a binary tree of height  $k$ :  

$$2^{k+1} - 1, k \geq 0$$
- ④ Consider a nonempty binary tree with  $n$  nodes in number of nodes with  $i$  children ( $i=0,1,2$ ) then,  

$$n_2 = n_0 - 1$$
- ⑤ Maximum no. of nodes in a binary tree at level  $i$ :  

$$2^i, i \geq 0$$
- ⑥ Maximum depth of a binary tree with  $n$  nodes is  

$$\log n$$
- ⑦ The minimum height of a binary tree with  $n$  nodes is  

$$\log_2 n$$
 (at half of maximum possible).
- ⑧ Maximum no. of leaves in a binary tree of height  $h$  (level):  

$$2^h$$
 or  $2^l$

GATE 2017

- ① How many distinct binary search trees can be created out of 4 distinct keys?

$$n = 4$$

$$\text{B}_n = \frac{1}{n+1} \binom{2n}{n} = \frac{2^n C_n}{n+1} = \frac{3}{1}, \frac{12}{2}, \frac{45}{3}, \frac{120}{4}, \frac{210}{5}, \dots, \frac{1}{1}, \frac{6!}{4}, \frac{120}{3}, \frac{360}{2}, \frac{720}{1}$$

$$= \frac{1}{4+1} \times \frac{8!}{4!4!} = \frac{1}{5} \times \frac{8 \times 7 \times 6 \times 5}{4 \times 3 \times 2 \times 1} = \frac{1}{5} \times \frac{6 \times 5 \times 4}{3 \times 2 \times 1} = 5$$

$$= 14.$$

GATE FORM

- ⑩ If a binary tree T has n leaf nodes, the number of nodes of degree 2 in T is  $n-1$

- ⑪ No. of nodes in a complete binary of depth d has  $2^{d+1}-1$

- ⑫ A complete binary tree with n non-leaf nodes contains  $2n+1$  nodes

- ⑬ The depth of a complete binary tree with n nodes is  $\log_2(n+1) - 1$

- ⑭ The depth of a complete ternary tree with n nodes is  $\log_3(2n+1)-1$

- ⑮ The average total path length of a randomly built binary search tree with n nodes is  $O(\log_2 n)$

- ⑯ Let i be the internal path length. Average number of comparisons needed for successful search in a binary tree of n nodes is  $(i+n)/n$

- ⑰ If m pointer fields are set aside in each node of a general tree to point to a max no. of m children and if the number of internal nodes in the tree is n, then number of leaves is  $n(m-1)+1$

- ⑱ No. of different binary trees with n nodes =  $\frac{1}{n+1} \binom{2n}{n}$

- ⑲ There are  $2^n - 1$  nodes in a full binary tree

- ⑳ For a binary tree, maximum no. of nodes at height  $h$  are  $2^{h+1}-1$  nodes

To Root  
Root Left Right  
Root Left Right Post

- (21) Total number of external nodes in a binary tree are  
internal nodes + 1  
 $e = i + 1$

(22) The average total path length of a randomly built binary search trees with  $n$  nodes is  $O(\log_2 n)$

(23) Time complexity of building a max-heap of  $n$  elements is  $O(n)$

(24) Heap sort  $\rightarrow O(n \log n)$

(25) Height of heap  $\rightarrow O(\log n)$

(26) In heap property,

function PARENT( $i$ )  $\rightarrow \lfloor i/2 \rfloor$

LEFT( $i$ )  $\rightarrow 2i$

RIGHT( $i$ )  $\rightarrow 2i+1$

(27) Number of comparisons in searching an element in BST is  $O(n)$

(28) In a complete binary tree of  $n$  nodes, the maximum distance between 2 nodes is  $2\log_2 n$

Max heap, insertion & deletion  $\sim O(\log n)$

(1) In a run of quicksort of 100 items, the main loop of quicksort has already completed 32 iterations. How many elements are guaranteed to be in final spot?

Sol:-

(2) In a selection sort of  $n$  elements,  $n-1$  times the swap function called in the complete execution of an algorithm.

(3) Given ' $n$ ',  $k$ -digit numbers where each digit can take up  $d$  values. Running time of radix sort is  $O(k(n+d))$

(4) Using universal hashing and collision resolution by chaining in an initially empty hash table with  $m$  slots, the expected time to handle  $n$  insertion operations is  $O(n)$

(5) Given an open-address hash table with a table of  $m$  slots and  $n$  elements present, the expected number of probes for a successful search of a key is  $\frac{m}{n} [\log_2 m - \log_2(m-n)]$

(6)  $O(1)$ ,  $O(m^2)$  probe sequences are used in quadratic probing & double hashing

(7) ~~1 to  $n-1$  iterations of relaxing the entire set of edges are performed in Bellman-Ford algorithm~~

(8) Height of decision tree on  $n$  elements  $\log_2(n!)$

(9)  ~~$2^{n-1}$  Comparisons are required for merging 2 sorted lists of  $n$  element items~~

(10) Running time of an efficient algorithm to find an Euler tour in a graph is  $O(|E|)$

(11) Running time of the fastest algorithm to calculate the shortest path between any 2 vertices of a graph where all edges have equal weight  $O(E + V)$

(12)  $\frac{n(n-1)}{2}$  no. of swaps required in an algorithm to calculate the transpose of a directed graph represented as adjacency matrix

(13) Running time of an efficient algorithm to compute the square of a directed graph represented as adjacency matrix  $\Theta(V^3)$

(14) Running time of an efficient algorithm to find all cut vertices of a graph  $\Theta(|E|)$

(15) Let  $H$  be a finite collection of hash functions that map a universe  $U$  of keys into  $\{0, 1, \dots, m-1\}$ .  $H$  is said to be universal if for each pair of distinct keys  $k_1, k_2 \in U$ , the no. of hash function  $h \in H$  for which  $h(k_1) = h(k_2)$  is at most  $\frac{1}{m}$ .

Note: In Preorder, Root is first element

In Postorder, Root is last element

Steps:

- ① Find the root.
- ② Find left & right subtrees. If root found
- ③ Find the root in corresponding LST & RST
- ④ Repeat step 2 and 3 until empty

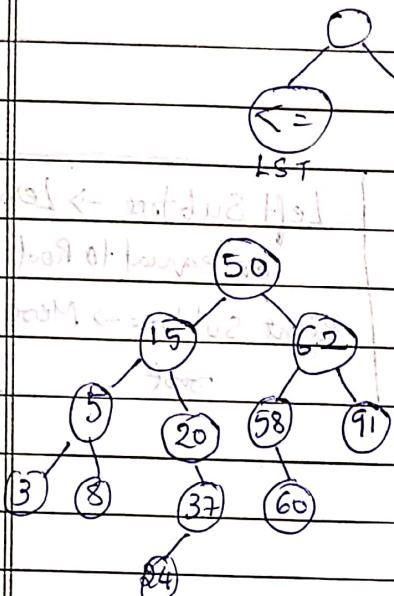
Binary Search Tree

It is a special kind of binary which are helpful in searching of elements

GATE 96

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24

Nodes are not ~~heat~~ repeated. Used to store heap. Points to records.



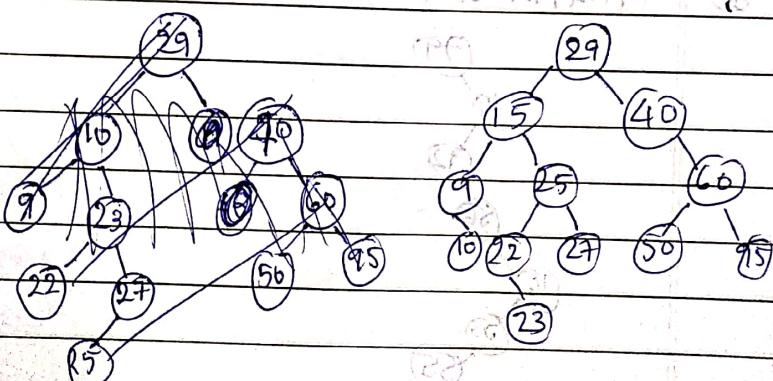
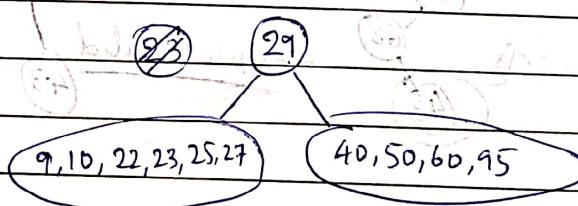
GATE 2005

Postorder: 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

Inorder: Sorted order  $\rightarrow$  9, 10, 15, 22, 23, 25, 27, 40, 50, 60, 95

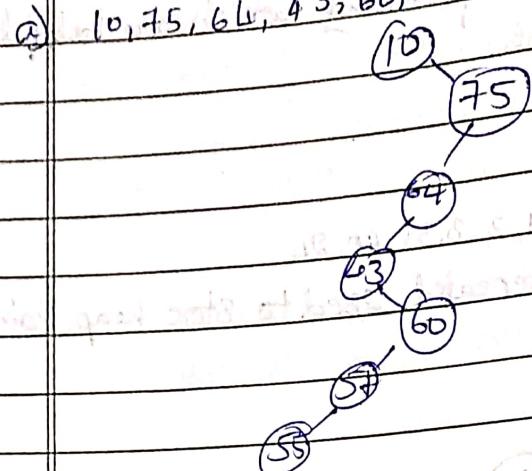
Preorder:-

Preorder: 29, 15, 9,  
10, 25, 22, 23, 27,  
40, 60, 50, 95

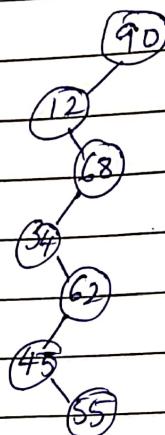


GATE 2006  $\rightarrow$  55 search

10, 75, 64, 43, 60, 57, 55



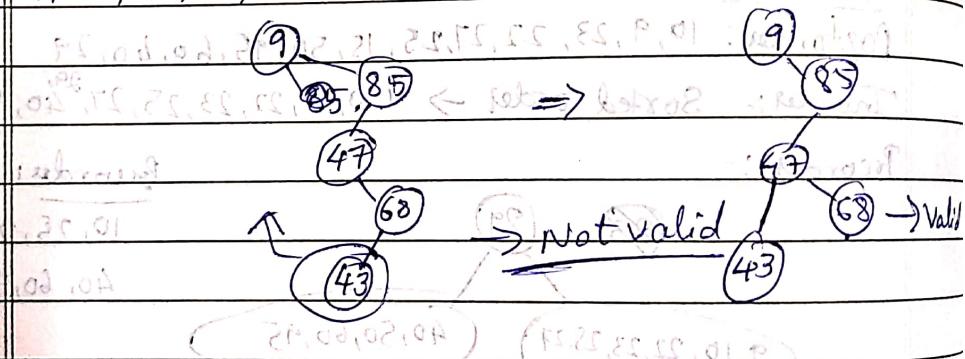
b) 90, 12, 68, 34, 62, 45, 55



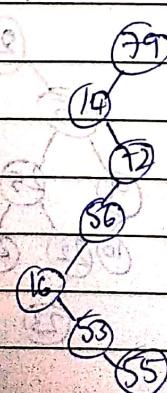
Left Subtree  $\rightarrow$  Less than  
or equal to Root

Right Subtree  $\rightarrow$  More than  
Root

c) 9, 85, 47, 68, 43, 57, 55



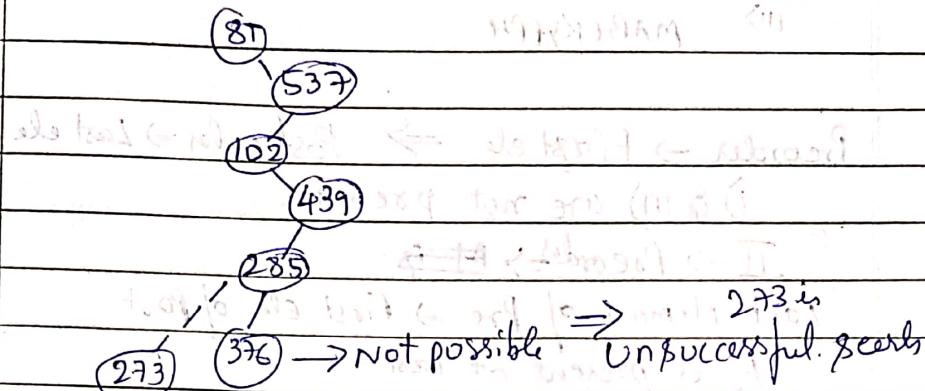
d) 79, 14, 77, 56, 16, 53, 55



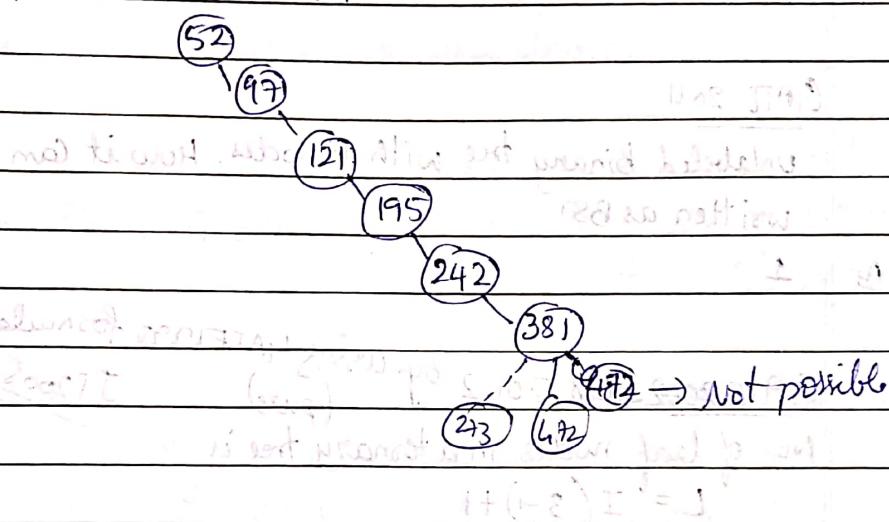
GATE 2008

273 is unsuccessful search

- a) 81, 537, 102, 439, 285, 376, 305

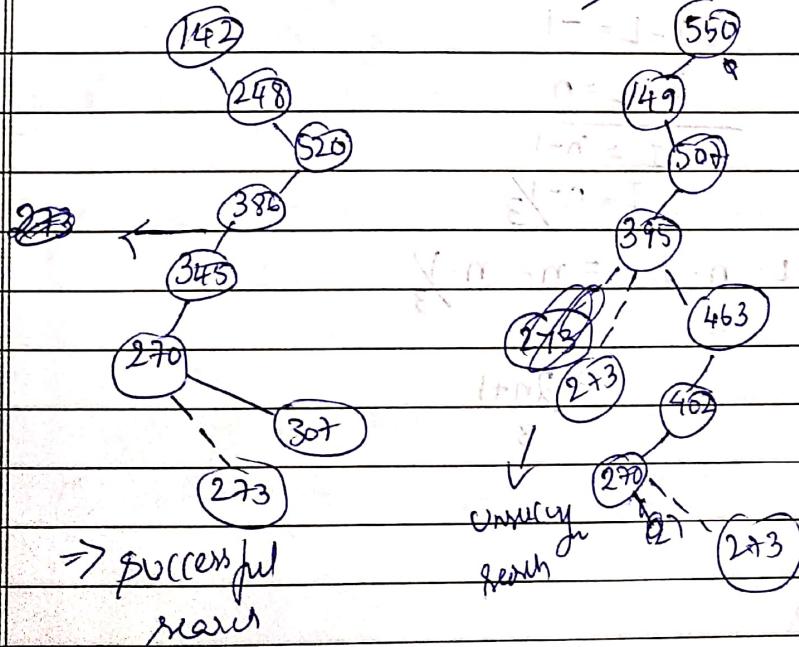


- b) 52, 97, 121, 195, 242, 381, 472



- c) 142, 248, 520, 386, 345, 270, 307

- d) 550, 149, 507, 395, 463, 402, 270



$\Rightarrow$  successful search

GATE - 2008

①

- I) MBCAFHFYK  
II) KAMBYPFH  
III) MABCKYfPH

Preorder  $\rightarrow$  First ele  $\Rightarrow$  Post order  $\Rightarrow$  Last ele

I) & III) are not preorder

II  $\rightarrow$  Preorder  $\Rightarrow$  ~~MBCAFHFYK~~

Last element of Pre  $\rightarrow$  First ele of Post

K is present at last

I  $\rightarrow$  Postorder

III  $\rightarrow$  Inorder

GATE 2011

Unlabelled binary tree with  $n$  nodes. How it can be written as BST.

n

1

By using GATE 1998 formula

GATE 2002 1998 2002 By using GATE 1998 formula (pure) JT 2005

No. of leaf nodes in a ternary tree is

$$L = I(3-1)+1$$

$$\text{Assume } L+I = n$$

$$2I - L = -1$$

$$I + L = n$$

$$3I = n-1$$

$$I = \frac{n-1}{3}$$

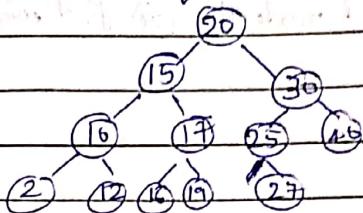
$$L = n - I = n - \frac{n-1}{3}$$

$$\geq \frac{2n+1}{3}$$

(Ans)

(Erg)

What is  $\leq$   
Answer

Delete a Node from BST

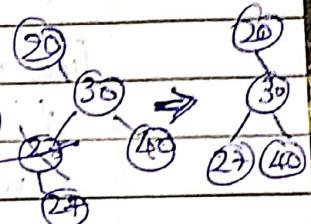
3 types of deletion

① Leaf

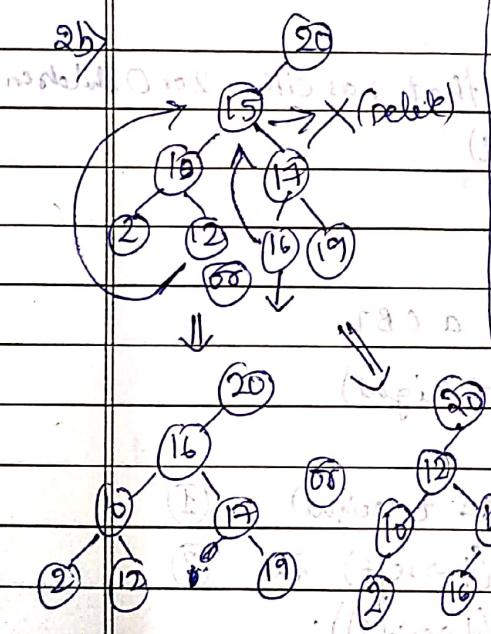
② Non Leaf [ a) one child

b) two children

2 a)



- If a node has one child and we are deleting that node, make that child pointing to grandparent.

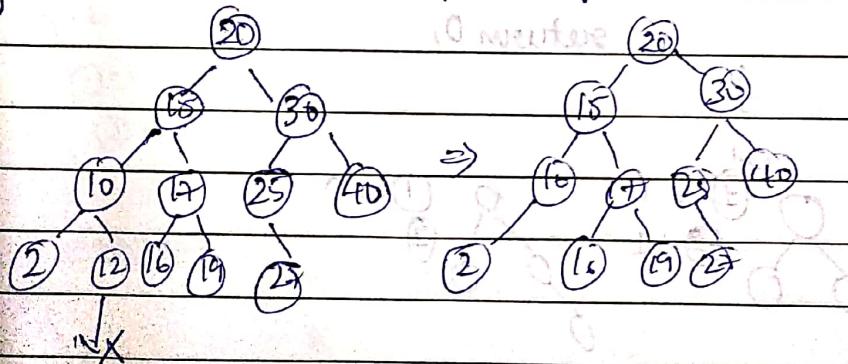


If a node has 2 children, go to right subtree, take least element & replace it in inorder successor

If a node has 2 children, go to left subtree, take greatest element & replace it

inorder predecessor

- If a node to be deleted is a leaf, then delete it.



Finding maximum & minimum in a BST  
find inorder pre( struct node \*t) (①) find min(struct node \*t)  
{  
 while ( $t \rightarrow \text{left}$ )  
  $t = t \rightarrow \text{left}$   
 }

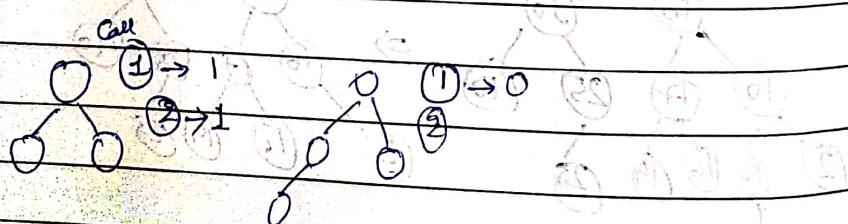
find max ( struct node \*t)  
{  
 while ( $t \rightarrow \text{right}$ )  
  $t = t \rightarrow \text{right}$   
 }

Recursive program on testing whether a tree is  
complete binary tree

Complete binary tree is a tree that has either 2 or 0 children

isComplete (struct node \*t)

```
if ( $t == \text{NULL}$ ).  
else if ( $t \rightarrow \text{left} \& t \rightarrow \text{right}$ )  
    return 1;  
else if ( $t \rightarrow \text{left} \& t \rightarrow \text{right}$ )  
    return 1;  
else if ( $t \rightarrow \text{left} \& t \rightarrow \text{right}$ )  
    return (isComplete ( $t \rightarrow \text{left}$ ) && (②)  
           isComplete ( $t \rightarrow \text{right}$ ));  
else  
    return 0;
```



AVL treesLinked List Search time  $O(n)$ Tree reduces time  $B$  but some cases like

1 2 3 4 5

(1)

(2)

(3)

(4)

(5)

→ Here time -  $O(n)$ Height -  $O(n)$ 

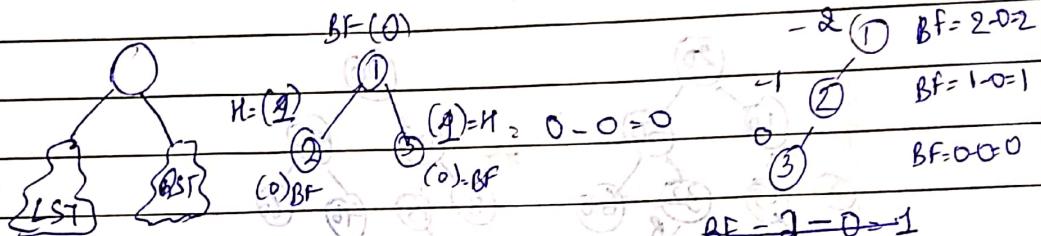
So comes concept of Balancing

AVL - Balanced tree

It is a balanced BST

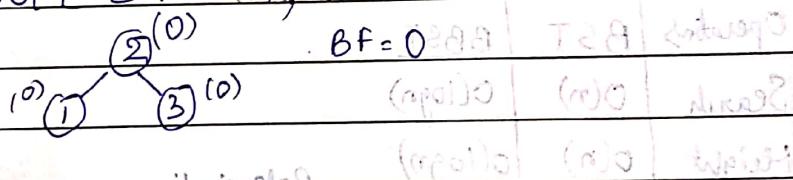
Height Nodes -  $n$ , Height -  $O(\log n)$ Search time -  $O(\log n)$ 

Balance Factor = Height of (LST) - Height (RST)

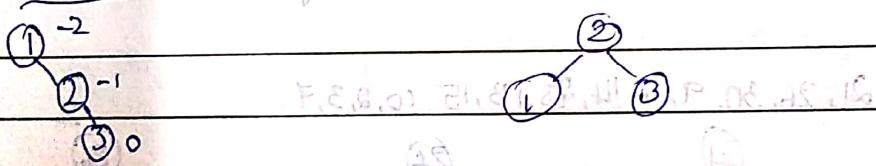


It is called LL imbalance

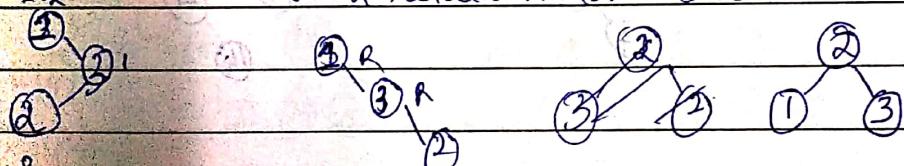
Whenever L-L imbalance, rotate clockwise



R-R imbalance: In order to balance, rotate anticlockwise

R-L imbalance

Do 2 rotations. Rotate clockwise. Rotate anticlockwise

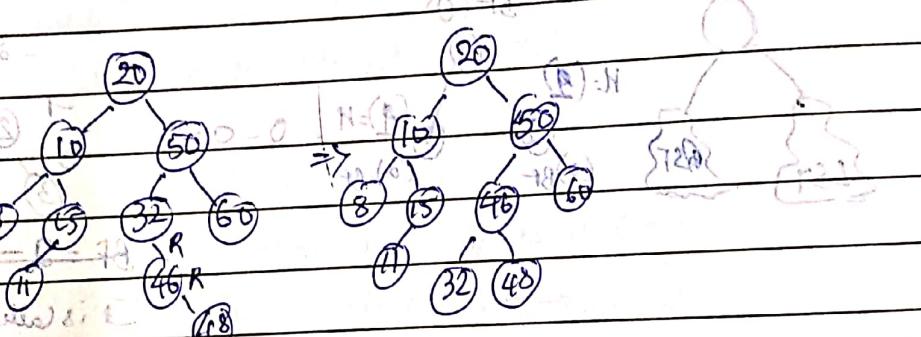
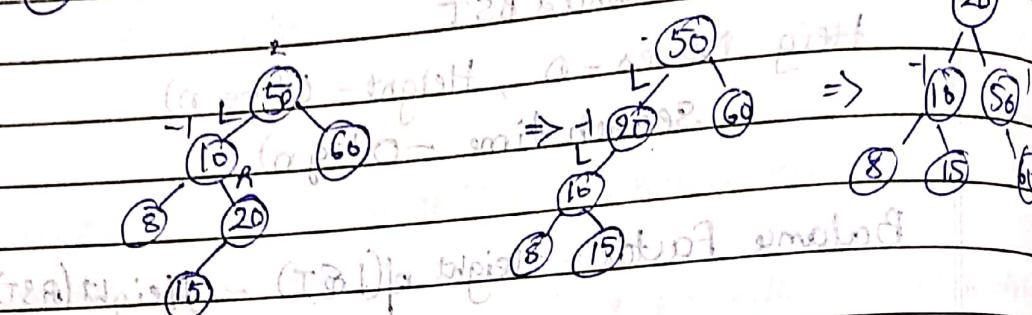
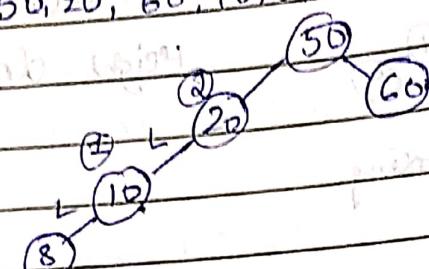


After this, result obtained is Binary Search Tree.

③ 2 This is LR imbalance. Rotate clockwise.

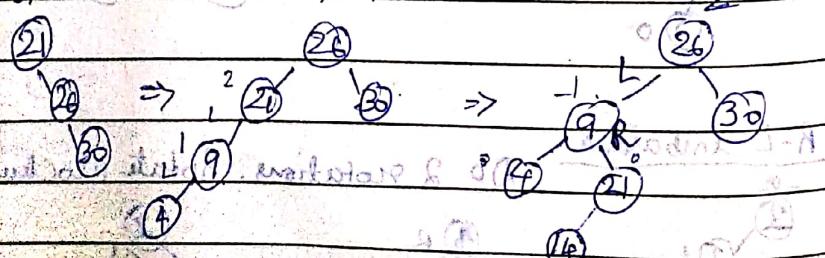
① ② ③ Rotate clockwise.

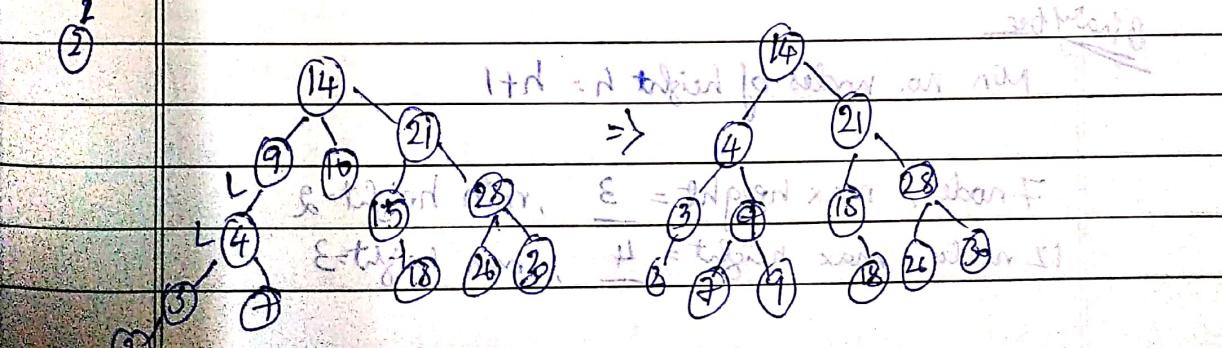
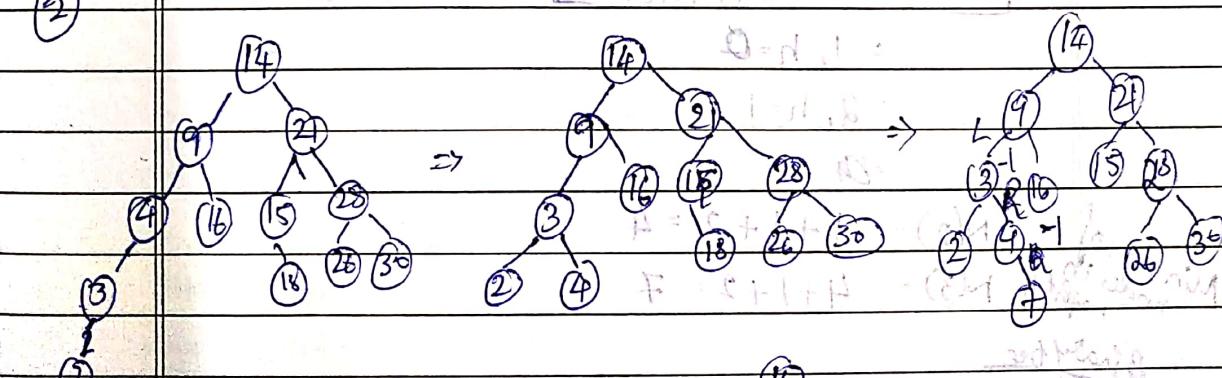
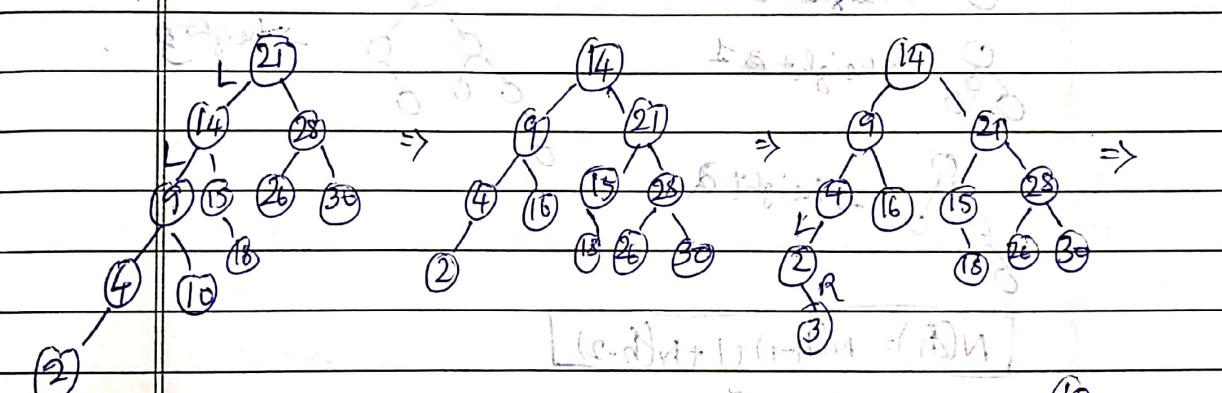
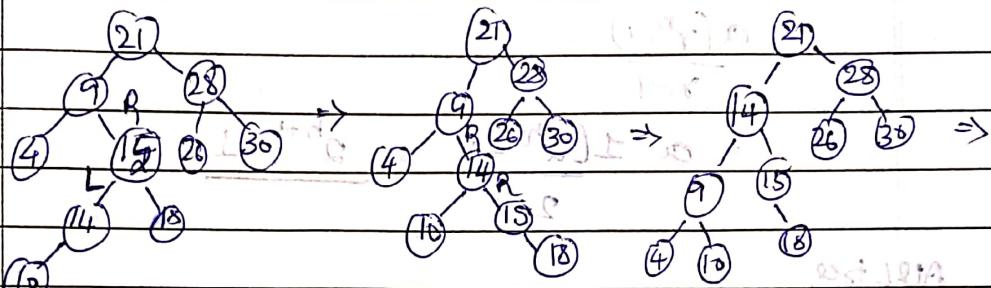
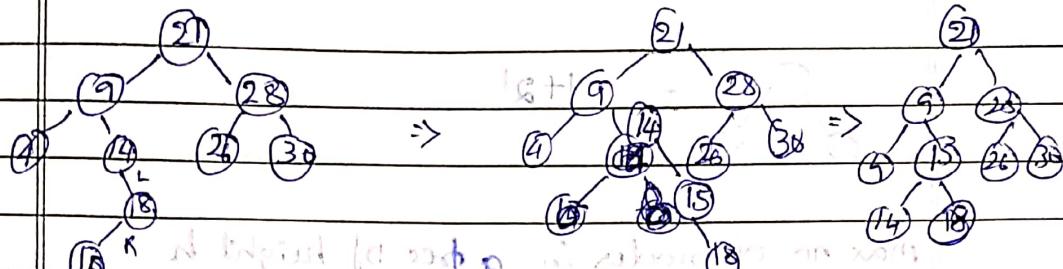
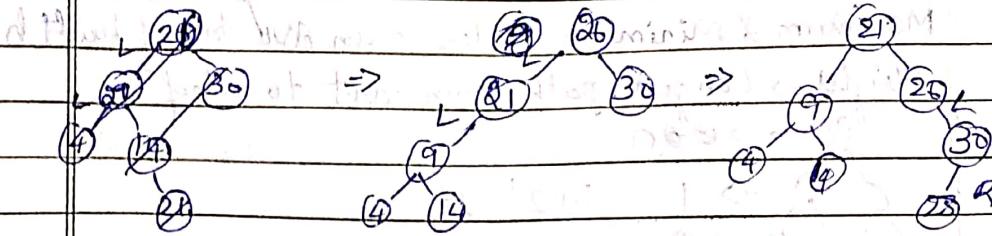
① 50, 20, 60, 10, 8, 15, 32, 46, 11, 47



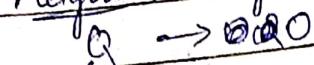
Operations	BST	BBST = 78
Search	$O(n)$	$O(\log n)$
Height	$O(n)$	$O(\log n)$
Insertion	$O(n)$	$O(\log n) + \text{Balancing time}$

21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

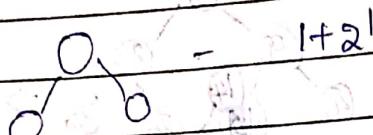




Maximum & Minimum nodes in an AVL tree of height  $h$   
Height  $\rightarrow$  Longest path from root to a leaf



$$1 \rightarrow 1+2^1+2^2 \\ 1+2^1+2^2 \rightarrow 2$$



Max no. of nodes in a tree of height  $h$

$$1+2^1+2^2+2^3+\dots+2^h$$

$$\alpha(r^{n-1})$$

$$\frac{r^n - 1}{r - 1}$$

$$= \alpha \frac{1(2^{h+1} - 1)}{2 - 1} = 2^{h+1} - 1$$

AVL tree

0  $\rightarrow$  height 0

0  $\rightarrow$  height 1

0  $\rightarrow$  height 2

0  $\rightarrow$  height 3

$$N(h) = N(h-1) + 1 + N(h-2)$$

$$= 1, h=0$$

$$= 2, h=1$$

$$= 4$$

$$N(2) = 1 + 1 + 2 = 4$$

$$\text{Min no. of nodes in } h \text{ th level } N(3) = 4 + 1 + 2 = 7$$

Binary tree

Min no. nodes of height  $h = h+1$

7 nodes, max height = 3, min height = 2

12 nodes, max height = 4, min height = 3

GATE 2006

In a binary tree, the number of internal nodes of degree 1 is 5, and number of nodes of degree 2 is 10. The number of nodes in the binary tree is

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

$$N(1) = 2$$

$$N(2) = (2-1) + 2 = 1 + 2 = 3$$

$$= N(1) - 1 + 2 = 3 = 3 \times 2^0$$

$$N(3) = 3 - 1 + 2 = 4$$

$$\boxed{N(n) = N(n-1) + 1}$$

$$(1+2+ \dots + N(n-2) + 1 + 1)$$

$$N(n-3) + 1 + 1 + 1$$

$$N(n) = N(n-k) + k \quad \text{where } n-k=1$$

$$N(1) = N(1) + N(n-1) = 1 + (N(n-1) - 1) \quad \boxed{[k=n-1]}$$

$$N(1) = 2 + 0$$

BST Degree 2 n = Degree n+1

$$10 = 10 + 1$$

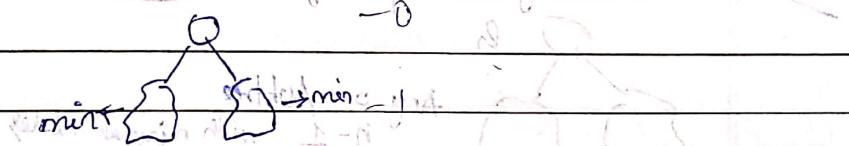
$$\therefore 11$$

Answer will be 11. (Ans) ~~for 11 or 10~~

GATE 2005 Ques No 11. In a binary tree, the minimum number of nodes in the left and right subtrees is at most 2.

In a binary tree, for every node, the difference between the number of nodes in the left and right subtree is at most 2.

If the height of tree is  $h > 0$ , then minimum number of nodes in the tree is  $2^h - 1$ .



$$N(n) = N(n-1) + 1 + (N(n-1) - 2) \Rightarrow N(h) = 2N(h-1) - 1$$

$$n=1, m=3, \text{ so } (1-2) = -1 \quad \text{Hence } 2^h - 1$$

$$\text{Substituting } n=1 \quad ((1-1) + 1 + (1-1) - 1 = N(1) = 2N(1) - 1 = 2 \times 2 - 1 = 3$$

$$a^{h-1} = a^0 = 2^0 = 1 \quad \times$$

$$b) 2^{h-1} + 1 = 2^0 + 1 = 2 \quad \checkmark, \quad 2^0 + 1 = 3 \quad \times \quad \text{Hence } 2^h - 1$$

$$c) 2^{h-1} - 2^0 - 1 = 1 \quad \times \quad (2-1) + 1 = N(1) = 2N(1) - 1$$

$$d) 2^h = 2^1 = 2 \quad \checkmark \quad (2-1) + 1 = N(1) = 2(2N(h-2) - 1) - 1$$

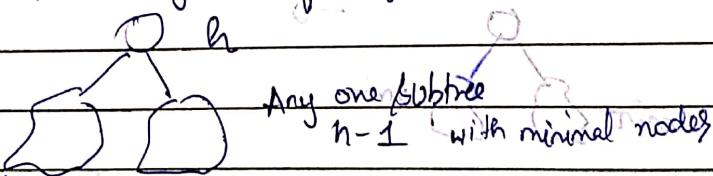
$$= 2^h (N-1) 2^{h-2} - 2^1 = 2^h (N-1) 2^{h-2} - 2^1 = 2^h (2N(h-2) - 1)$$

$$\begin{aligned}
 N(h) &= 2N(h-1) - 1 \\
 N(h-1) &= 2N(h-2) - 1 \\
 N(h-2) &> 2N(h-3) - 1 \\
 N(h) &\geq 2^6 N(h-2) - 1 - (h-6)u + (h-6)u \\
 &= 2^2 N(h-2) - 2 - 1 \\
 &= 2^2(2N(h-3) - 1) - 2 - 1 \\
 &= 2^3 N(h-3) - 2^2 - 2 + 1 - (h-7)u \\
 N(h) &= 2^k : \\
 N(h) &= 2^k N(h-k) - 2^{k-1} - 2^{k-2} - \dots - (2-1) \\
 &= 2^k N(h-k) - (2^{k-1} + 2^{k-2} + \dots + 2 + 1) \\
 &= 2^k N(h-k) - (2^{k-1}) + (k-1)u \\
 &= 2^{h-2} N(2) - (2^{h-2} - 1) + (h-1)u - (h-2)u \\
 &= 2^{h-2} - (2^{h-2} - 1)u + (h-1)u - (h-2)u \\
 &= 2 \cdot 2^{h-2} + 1 \\
 &= 2^{h-1} + 1
 \end{aligned}$$

GATE 1997 and GATE 2005

A size balanced binary tree, for every node the difference between the number of nodes in the left and right subtrees is at most 1. If the height of the tree is  $h > 0$ , then the minimum number of nodes in the tree is  $n$  and the maximum of height of a leaf node (from root) of trees is  $2^h$ .

A binary tree of height  $h$  has  $2^h$  nodes.



$1 - (1-\alpha) \text{ nodes } n-1 \text{ nodes } n-1 \text{ of LST } \leq (1-\alpha)u + 1 + (1-\alpha)u - (n)u$

$$N(h) = h(N(h-1)) + 1 + (N(h-1) - 1) \quad // \text{no. of nodes decreased}$$

$$AVL \text{ tree } N(h) = N(h-1) + 1 + N(h-2) \quad // \text{no. of height decreased}$$

$$N(h) = 2N(h-1) - 1$$

$$1 - (1-\alpha)N(h-1) \leq 2N(h-2) \quad // \text{height decreased}$$

$$N(h) \leq 2N(h-2)$$

$$1 - (1-\alpha)N(h-2) \leq 2^2 N(h-3) \quad // \text{height decreased}$$

$$1 - (1-\alpha)N(h-3) \leq 2^3 N(h-4) \quad // \text{height decreased}$$

$$1 - (1-\alpha)N(h-4) \leq 2^4 N(h-5) \quad // \text{height decreased}$$

$$N(h) = 2^k N(h-k) \quad \text{where } h-k=1$$

$\therefore k=h-1$

maximum height with 4 nodes

$$4 = 2^h$$

(Ans)  $\boxed{h=2}$  max height of 2 nodes

becomes balanced & each node has unit weight

∴ max height with 8 nodes

$$8 = 2^h \quad \text{so } h=3 \text{ max height of 3 nodes}$$

$\boxed{h=3}$  add more nodes after 8 nodes, max height

$$N(h) = 2^h$$

$$N(h+1) = 2^{h+1}$$



∴ at most  $h+1$  levels in tree, each level has  $2^h$  nodes

∴  $N(h+1) = 2^{h+1} + 2^h + 2^{h-1} + \dots + 2^0$  max height of  $h+1$

∴ max height  $= 2^{h+1} - 1$  (if  $h+1$  levels are formed)

∴ max height =  $2^{h+1} - 1$  (if  $h+1$  levels are formed)

∴ max height =  $2^{h+1} - 1$  (if  $h+1$  levels are formed)

∴ max height =  $2^{h+1} - 1$  (if  $h+1$  levels are formed)

∴ max height =  $2^{h+1} - 1$  (if  $h+1$  levels are formed)

∴ Gate No 14 of binary search tree A (Ans)

Q1 Suppose we have an unbalanced binary search tree 'T' holding

n numbers. We are given two numbers 'L' and 'H', and

wish to sum up all the numbers in 'T' that lie between L

and H. Suppose there are m such numbers in T. If the tightest

upper bound time to compute this sum is  $O(n^a \log n + m^c \log m)$ , the

value of  $a + 10b + 100c + 1000d$  is (Ans)

$$\begin{aligned}
 & \text{Diagram: A binary search tree with root } 100. \text{ Left child is } 100-199, \text{ right child is } 200. \\
 & \text{Left child } 100-199 \rightarrow O(m \log n) \quad (\text{since } m=100, n=1000) \\
 & \text{Right child } 200 \rightarrow O(m \log n) \quad (\text{since } m=100, n=1000) \\
 & \text{Total sum: } 100 + 10(100) + 1000(100) \\
 & \quad = 110000
 \end{aligned}$$

Other methodInorder  $O(n)$ 

$$T \quad [ \quad ] \quad O(\log n)$$

$\uparrow \quad \uparrow$   
 $L \quad H$

In worst case for finding  $L$  and  $H$  it will take  $O(\log n)$  time as the given tree is balanced binary search tree. Now there are  $m$  elements between  $L$  and  $H$ . So to traverse  $m$  element it will take  $O(m)$ .

Time (traversal algorithm given below)

Total  $(m + \log n)$ 

$$\Rightarrow a=0, b=1, c=1, d=0$$

$$= 0 + 10(1) + 100(1) + 1000(0)$$

$$= 110$$

To find all the numbers from  $L$  to  $H$ , we can do an inorder traversal from root and skip (avoid) all elements before  $L$  and after  $H$ . But this has  $O(n)$  time complexity. So we can do a modification to inorder traversal and combine with Binary search as follows:

- ① Find  $L$  using binary search and keep all nodes encountered in search using stack.
- ② After finding  $L$ , add it to stack as well and initialize sum = 0.
- ③ Now, for all nodes in stack, do an inorder traversal starting from their right node & adding the node value to sum. If  $H$  is found, stop the algorithm.

Expression tree

(Root + left) \* (right - 1) = P1 - 001

Left = 0, Right = 1 = Pre tab

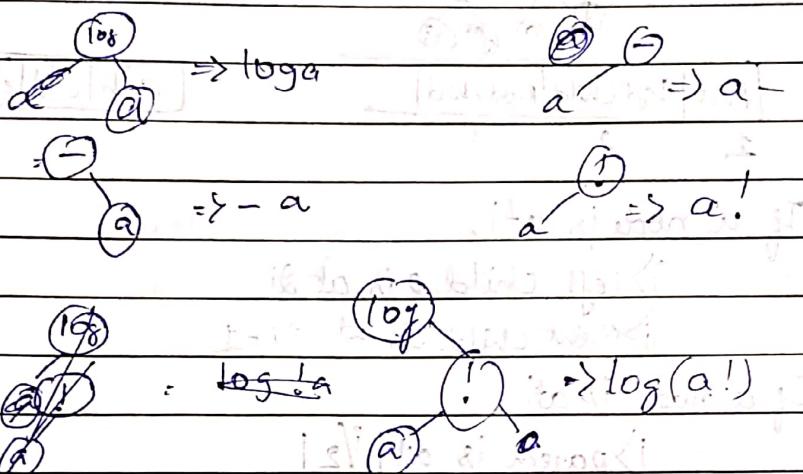
(a) (b) In (left + right) = Post ab+1

$$a + b * c$$

+      ~~a \* b~~

~~a~~      ~~b~~      ~~c~~

\* → high precedence

Pre  $+ a * bc$ In  $a + b * c$ Post ~~a b + c \*~~ abc \* +GATE 2005

The numbers  $1, 2, \dots, n$  are inserted into a BST in some order  $n$ . In the resulting tree, the right subtree contains  $p$  nodes. The first number to be inserted in the tree must be  $(p+1)$ .

8) BST - element ( $p+1$ ) // for BBST & BST

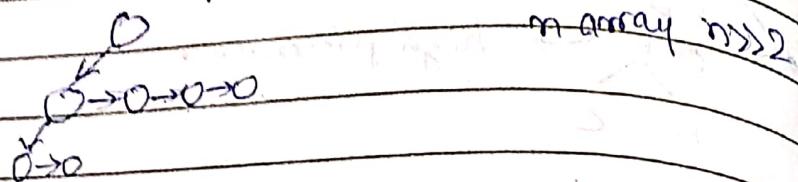
LST -  $n-p+1 = n-p-1$ Root -  $n-p(p+1)(n-p-1)$ GATE 2014

max value to be found

$$\begin{aligned}
 & ((a+b) - (c-d)) + \\
 & \Rightarrow ((e-f) + (g+h)) \\
 & = (i+j) - (o-n) + (l-m) \\
 & \quad + (k+l) \\
 & = 2 + 1 + 1 + 2 = 6
 \end{aligned}$$

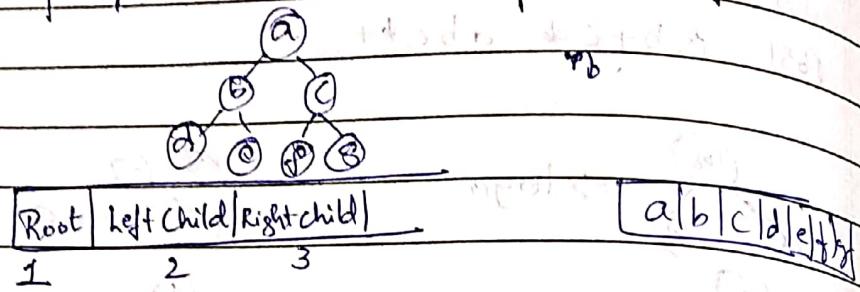
Various representations of a tree

① LCBS  $\rightarrow$  Left child Right Sibling



in array form

② Array representation  $\rightarrow$  Mainly for Heaps



If a node is at  $i$ ,

i) left child is at  $2i$

ii) right child is at  $2i+1$

If a node is at  $i$ ,

i) parent is at  $\lfloor i/2 \rfloor$

elements,  $2^n - 1 \rightarrow$  array size

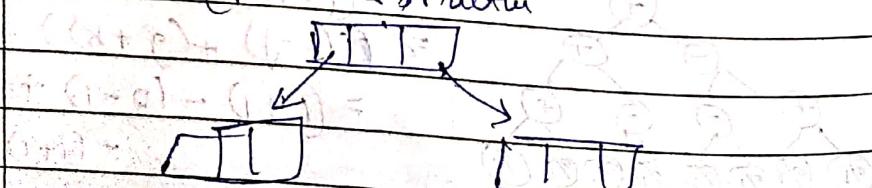
③ Nested representation

(a b a c) (b) (Root Left Right)

(a b c) (a) (b) (c)

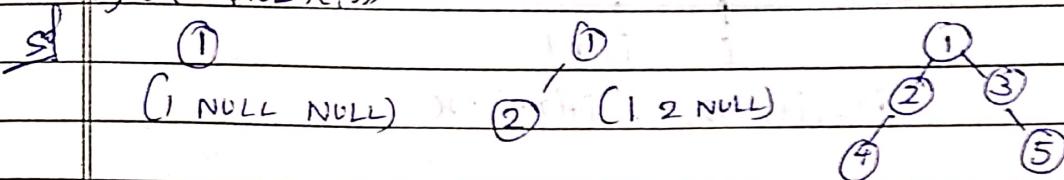
a  
b' c  
d' c' g  
a(b(c))((d)e)(f(g))

④ Normal - Pointer & structure

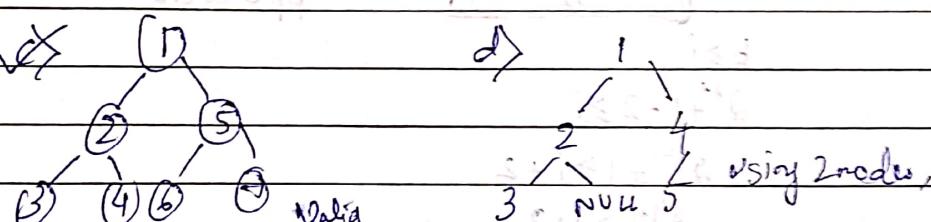
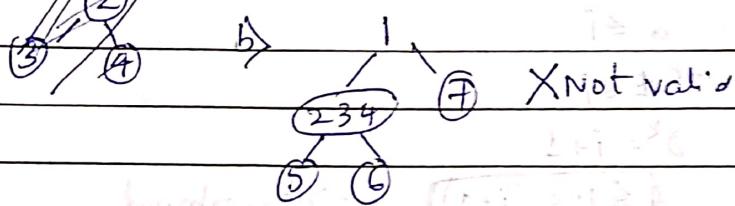


GATE 2000

- 1) Consider the following nested representation of binary tree:  
 $(xyz)$  indicates  $y$  &  $z$  are the left & right subtrees, respectively,  
of node  $x$ . Note that  $y$  &  $z$  may be NULL, or further nested.  
which of the following represents a valid binary tree?
- $(1_2(4567))$
  - $((234)56)7$
  - $(1(234)(567))$
  - $(1(23\text{NULL})(45))$



a) ~~(1) 4 elements~~ 1 ~~(2) (1) (2) (2) No 4 elements~~  $(1(24)(35))$   
~~(2) (1) (2) (2) No 4 elements~~  $(1(24\text{NULL})(3\text{NULL}5))$

GATE 2006

In array  $X$  of  $n$  distinct integers is interpreted as a complete binary tree. The index of the first element of the array is 0.

- 1) The index of the parent element  $X[i]$ ,  $i \neq 0$  is  
 $\lfloor i/2 \rfloor$

2) If only the root node does not satisfy the heap property, the algorithm to convert the binary heap into heap has the best asymptotic time complexity of  $O(\log n)$

Because we have to heapify whole heap i.e.,  $O(\log n)$  is height of heap

3) If the root node is at level 0, the level of element  $X[i], i \neq 0$

Let  $i$  at level ' $k$ '

$$2^0 + 2^1 + 2^2 + \dots + 2^{k-1} = a \quad 1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^{k-1}} = b$$

$$\frac{i}{2^k - 1} = a \quad (a \leq b)$$

$$a = 2^k - 1$$


$$i = 0, i+1, i+2, \dots, i+(2^k-1) = 2k$$

$$(a+2^k)b = a+2^{k-1}$$

$$(a+2^k)(b-1) = 2^k - 1 + 2^{k-1}$$

$$b = 2^{k+1} - 2$$

$$a \leq i$$

$$2^k \leq i+1$$

$$2^k = i+1$$

$$k \leq \log_2(i+1) \rightarrow \text{Upperbound}$$

$$b \geq i$$

$$2^{k+1} - 2 \geq i$$

$$2^{k+1} + 1 \geq i+2$$

$$2^{k+1} \geq i+2$$

$$k+1 \geq \log_2(i+2) - 1$$

$$k \geq (\log_2(i+2) - 1) \rightarrow \text{Lowerbound}$$

There can be only one integer b/w these bounds

$$\lceil \log_2(i+1) \rceil \text{ and } \lceil \log_2(i+2) - 1 \rceil$$

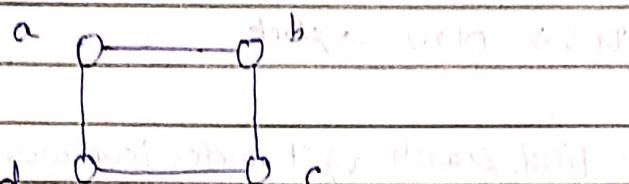
Level

## Geography

Date \_\_\_\_\_ Page \_\_\_\_\_

- ▷ World Wide Web is represented as a graph
  - ▷ Social Network
    - ▷ Person  $\rightarrow$  Nodes, Friend  $\rightarrow$  Edge b/w a person & his friend
    - Friends  $\rightarrow$  Nodes
  - ▷ Webpage  $\rightarrow$  Node
  - Links  $\rightarrow$  Edge

## Representation of Graphs



## Adjacency matrix

Adjacency List → useful for large/dense graphs

	a	b	c	d	
a	0 1 0 1	a	$\rightarrow [b]$	$\rightarrow [d] \vee 0$	
b	1 0 1 0	b	$\rightarrow [a]$	$\rightarrow [c] \vee 0$	
c	0 1 0 1	c	$\rightarrow [b]$	$\rightarrow [d] \vee 0$	
d	1 0 1 0	d	$\rightarrow [a]$	$\rightarrow [c] \vee 0$	

Dense graph: If the no. of edges in a graph is equal to number order of nodes

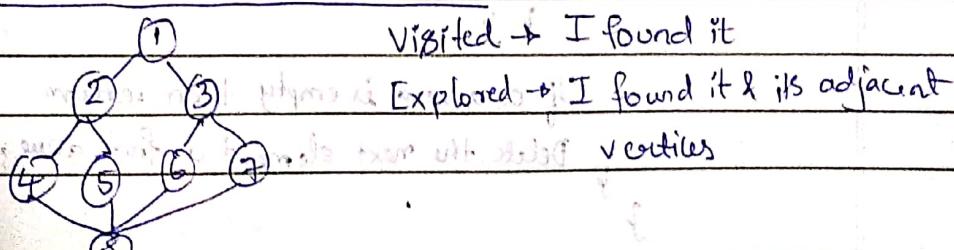
No. of elements =  $2 \times E$  in undirected graph in Adjacency List

$E$  in directed graph

$= O(V+E) \rightarrow$  Adjacency List

$O(\sqrt{2}) \rightarrow$  Adjacency matrix

## Introduction to BFS and DFS



Visited      Boolean array      Both BFS & DFS uses

0 → Not visited      array to keep track of

1 → Visited      visited nodes

DFS algorithm keeps track of unexplored nodes  
using stack

BFS algorithm

using queue

Space complexity → at least  $O(n)$

BFS →  $O(n)$  → queue

DFS →  $O(n)$  → stack

Breadth first search visit nodes levelwise (horizontally)

Depth first search visit nodes depthwise (vertically)

BFS algorithm

// BFS(v)

// The graph G and array visited[] are global

visited[] is initialized to 0

graph G;

u = v;

repeat

{

  for all vertices w adjacent to u do

    if (visited[w] == 0) {

      start printing (v, w)

      add w to queue;

      visited[w] = 1;

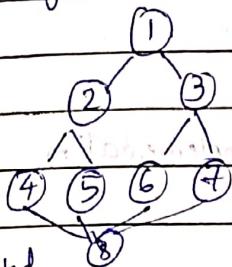
    } // if loop

  } // for all vertices w adjacent to u

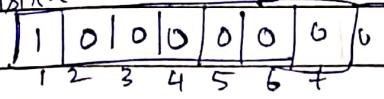
  if (queue is empty) {

    Delete the next element, u from queue;

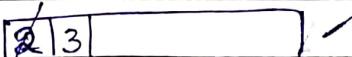
Tracing



Visited

 $u=1$  $w = \{2, 3\}$ 

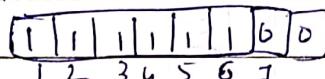
Queue



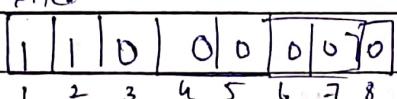
(u) &lt;= t [8] = 1

 $\rightarrow O(n)$  $(n-1) \rightarrow O(n)$ 

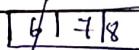
Visited



Visited



Queue

 $u=6$  $v = \{3, 8\}$ 

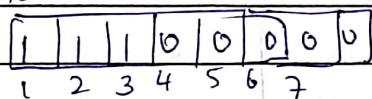
Queue

 $u=2$  $w = \{1, 4, 5\}$ 

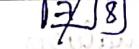
Queue



Visited



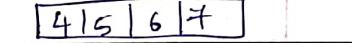
Queue

 $u=4$  $w = \{3, 8\}$ 

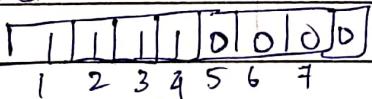
Queue

 $u=3$  $w = \{1, 6, 7\}$ 

Queue



Visited



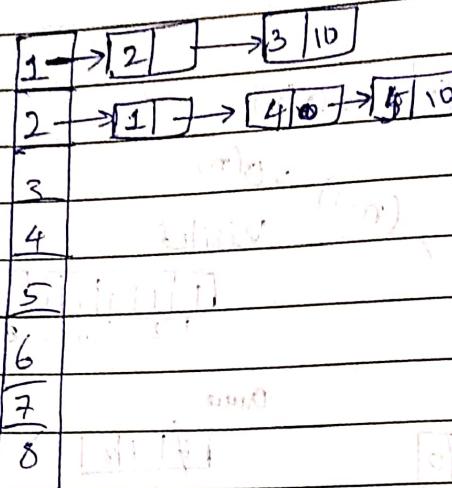
Queue

 $u=8$

BFS analysis in case of linked list

Space complexity =  $O(n)$

Time complexity - depends on implementation



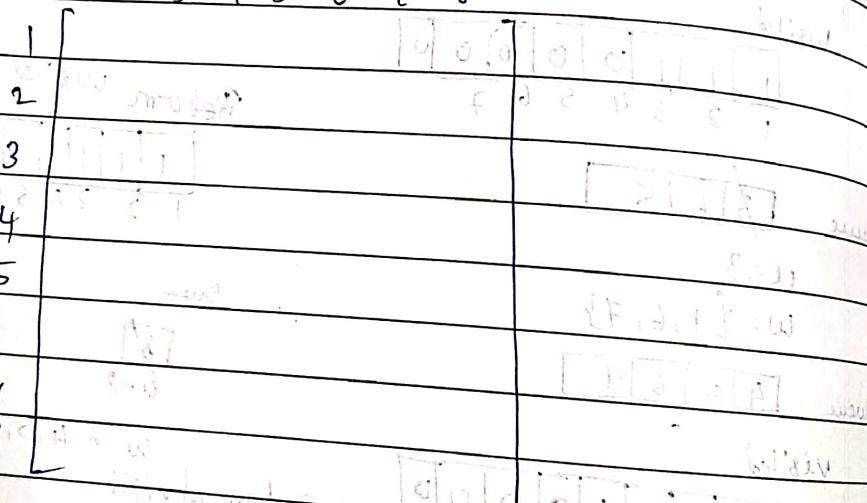
$O(2E) \rightarrow$  undirected graph

$O(E) + O(v) \rightarrow$  Time Complexity

Space complexity  $\rightarrow O(v)$

BFS analysis in case of Adjacency matrix

1 2 3 4 5 6 7 8



Space complexity  $\rightarrow O(v)$

Time complexity  $\rightarrow O(n^2)$

$\rightarrow O(v^2)$

Initialization  $\rightarrow O(v^2 + v) \rightarrow O(v^2)$

Breadth First Traversal using BFS

BFT( $G, n$ )

{

for  $i=1$  to  $n$  do

visited[i] = 0;

for  $i=1$  to  $n$  do

if (visited[i] = 0)

then BFS(i);

}

1

2

3

4

5

6

7

visited [ ] 0 0 0 0 0 0 0

1 2 3 4 5 6 7

visited [ ] 1 1 1 1 0 0 0

call BFS(5)

visited [ ] 1 1 1 1 1 1 1

Time complexity -  $O(EV)$

Space complexity -  $O(V)$

DFS algorithm

DFS(v)

{

visited[v] = 1;

for each vertex adj to v do

{

if (visited[w] = 0) then

DFS(w);

}

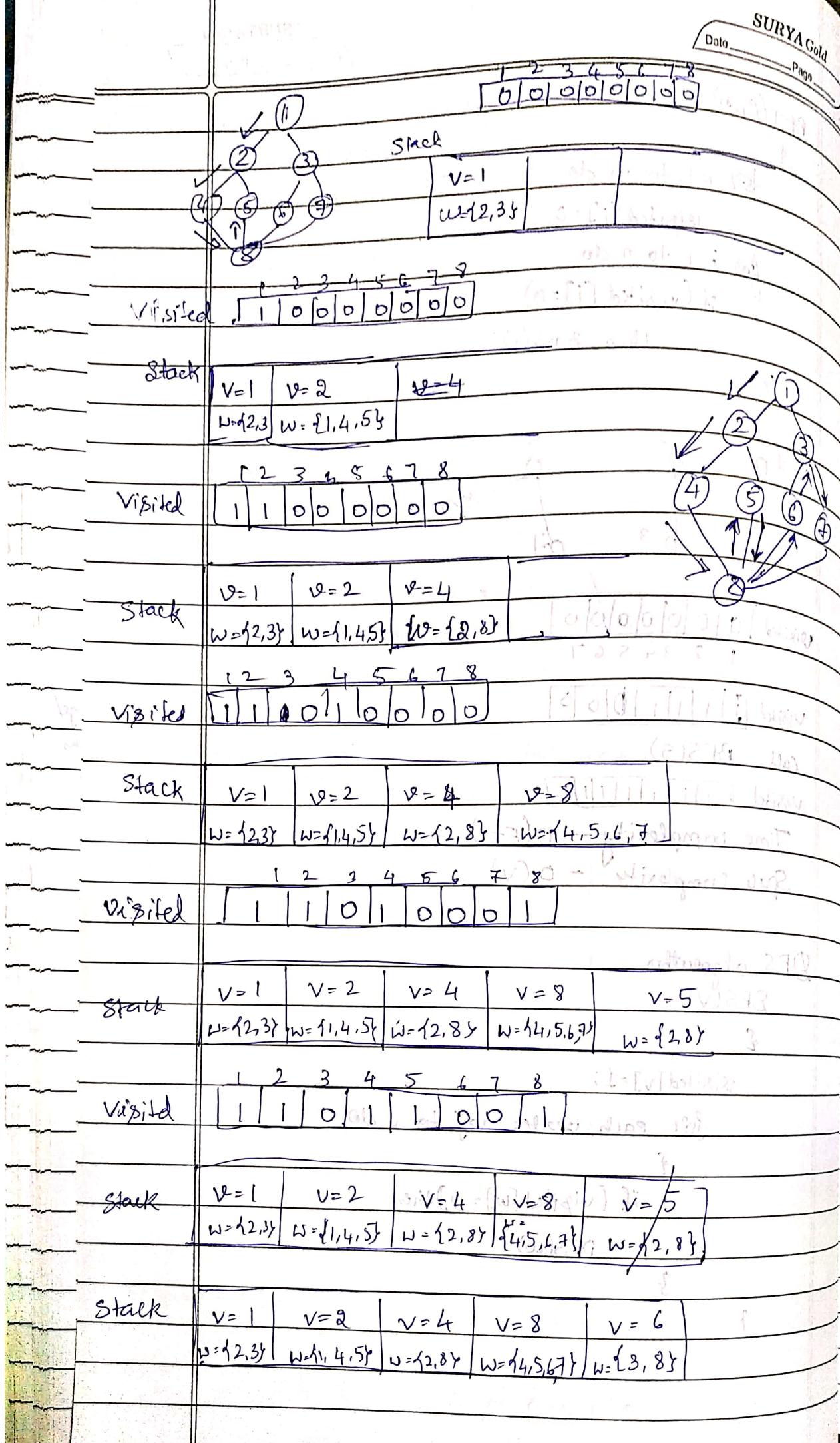
}

}

}

}

}



Visited

	1	2	3	4	5	6	7	8
Visited	1	1	0	1	1	1	0	1

Stack

$v=1$	$v=2$	$v=4$	$v=8$	$v=6$	$v=3$
$w=\{2, 3\}$	$w=\{1, 4, 5\}$	$w=\{2, 8\}$	$w=\{4, 5, 6, 7\}$	$w=\{3, 8\}$	$w=\{1, 6, 7\}$

Visited

	1	2	3	4	5	6	7	8
Visited	1	1	1	1	1	1	0	1

Stack

$v=1$	$v=2$	$v=4$	$v=8$	$v=6$	$v=3$	$v=7$
$w=\{2, 3\}$	$w=\{1, 4, 5\}$	$w=\{2, 8\}$	$w=\{4, 5, 6, 7\}$	$w=\{3, 8\}$	$w=\{1, 6, 7\}$	$w=\{3, 8\}$

Visited

	1	2	3	4	5	6	7	8
Visited	1	1	1	1	1	1	1	1

Stack

$v=1$	$v=2$	$v=4$	$v=8$	$v=6$	$v=3$	$v=7$
$w=\{2, 3\}$	$w=\{1, 4, 5\}$	$w=\{2, 8\}$	$w=\{4, 5, 6, 7\}$	$w=\{3, 8\}$	$w=\{1, 6, 7\}$	$w=\{3, 8\}$

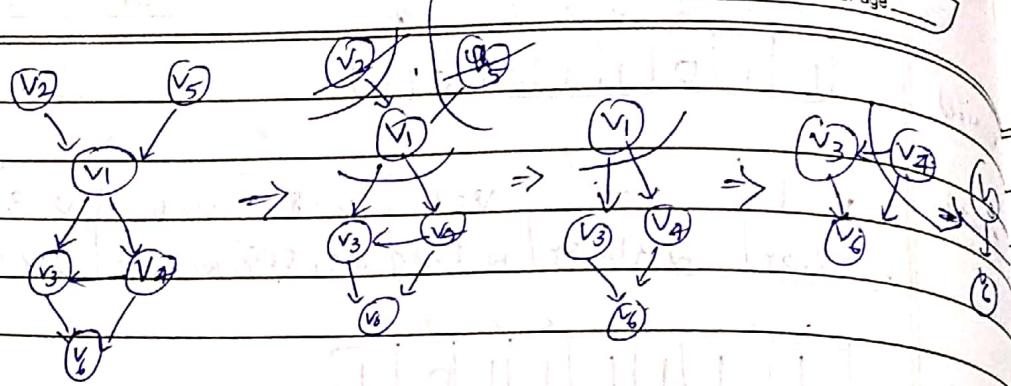
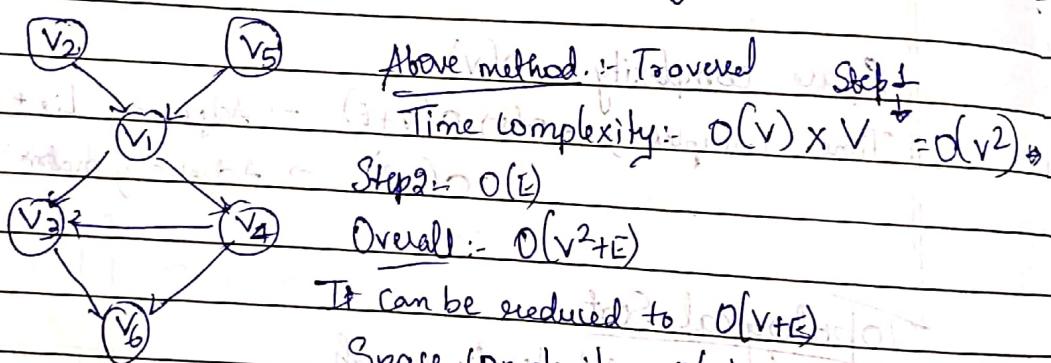
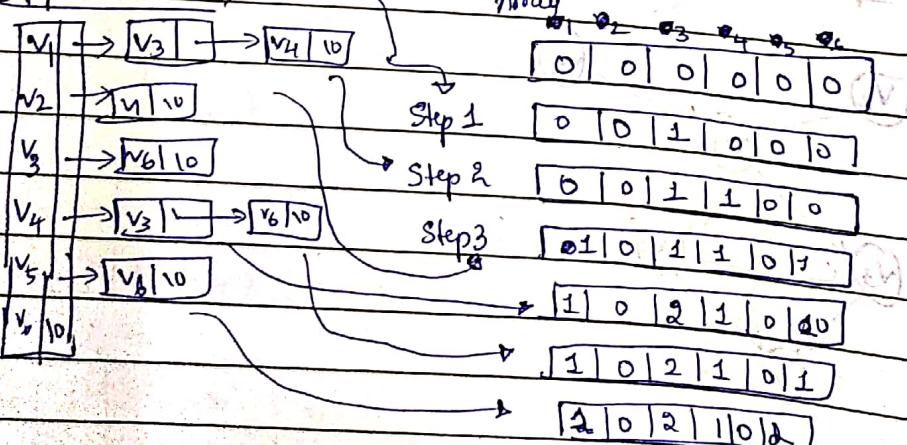
① ⑥ ④ ⑤ ③ ② ⑦ ① Popping order

Space complexity  $\sim O(V)$ Time complexity  $\sim O(E) O(V+E) \rightarrow$  Adjacency List + $O(V^2) \rightarrow$  Adjacency matrix

Topological Sort

Input: Directed Acyclic graph

Output:  $i \rightarrow j$ ,  $j$  should be first in order i.e., sorted①  $V_2$  ②  $V_1$  ③  $V_5$ ④  $V_6$  ⑤  $V_7$  ⑥  $V_3$ ⑦  $V_4$  ⑧  $V_8$  ⑨  $V_9$ ⑩  $V_{10}$  ⑪  $V_{11}$  ⑫  $V_{12}$ ⑬  $V_{13}$  ⑭  $V_{14}$  ⑮  $V_{15}$ ⑯  $V_{16}$  ⑰  $V_{17}$  ⑱  $V_{18}$ ⑲  $V_{19}$  ⑳  $V_{20}$  ⑳  $V_{21}$ ⑳  $V_{22}$  ⑳  $V_{23}$  ⑳  $V_{24}$ ⑳  $V_{25}$  ⑳  $V_{26}$  ⑳  $V_{27}$ ⑳  $V_{28}$  ⑳  $V_{29}$  ⑳  $V_{30}$ ⑳  $V_{31}$  ⑳  $V_{32}$  ⑳  $V_{33}$ ⑳  $V_{34}$  ⑳  $V_{35}$  ⑳  $V_{36}$ ⑳  $V_{37}$  ⑳  $V_{38}$  ⑳  $V_{39}$ ⑳  $V_{40}$  ⑳  $V_{41}$  ⑳  $V_{42}$ ⑳  $V_{43}$  ⑳  $V_{44}$  ⑳  $V_{45}$ ⑳  $V_{46}$  ⑳  $V_{47}$  ⑳  $V_{48}$ ⑳  $V_{49}$  ⑳  $V_{50}$  ⑳  $V_{51}$ ⑳  $V_{52}$  ⑳  $V_{53}$  ⑳  $V_{54}$ ⑳  $V_{55}$  ⑳  $V_{56}$  ⑳  $V_{57}$ ⑳  $V_{58}$  ⑳  $V_{59}$  ⑳  $V_{60}$ ⑳  $V_{61}$  ⑳  $V_{62}$  ⑳  $V_{63}$ ⑳  $V_{64}$  ⑳  $V_{65}$  ⑳  $V_{66}$ ⑳  $V_{67}$  ⑳  $V_{68}$  ⑳  $V_{69}$ ⑳  $V_{70}$  ⑳  $V_{71}$  ⑳  $V_{72}$ ⑳  $V_{73}$  ⑳  $V_{74}$  ⑳  $V_{75}$ ⑳  $V_{76}$  ⑳  $V_{77}$  ⑳  $V_{78}$ ⑳  $V_{79}$  ⑳  $V_{80}$  ⑳  $V_{81}$ ⑳  $V_{82}$  ⑳  $V_{83}$  ⑳  $V_{84}$ ⑳  $V_{85}$  ⑳  $V_{86}$  ⑳  $V_{87}$ ⑳  $V_{88}$  ⑳  $V_{89}$  ⑳  $V_{90}$ ⑳  $V_{91}$  ⑳  $V_{92}$  ⑳  $V_{93}$ ⑳  $V_{94}$  ⑳  $V_{95}$  ⑳  $V_{96}$ ⑳  $V_{97}$  ⑳  $V_{98}$  ⑳  $V_{99}$ ⑳  $V_{100}$  ⑳  $V_{101}$  ⑳  $V_{102}$ ⑳  $V_{103}$  ⑳  $V_{104}$  ⑳  $V_{105}$ ⑳  $V_{106}$  ⑳  $V_{107}$  ⑳  $V_{108}$ ⑳  $V_{109}$  ⑳  $V_{110}$  ⑳  $V_{111}$ ⑳  $V_{112}$  ⑳  $V_{113}$  ⑳  $V_{114}$ ⑳  $V_{115}$  ⑳  $V_{116}$  ⑳  $V_{117}$ ⑳  $V_{118}$  ⑳  $V_{119}$  ⑳  $V_{120}$ ⑳  $V_{121}$  ⑳  $V_{122}$  ⑳  $V_{123}$ ⑳  $V_{124}$  ⑳  $V_{125}$  ⑳  $V_{126}$ ⑳  $V_{127}$  ⑳  $V_{128}$  ⑳  $V_{129}$ ⑳  $V_{130}$  ⑳  $V_{131}$  ⑳  $V_{132}$ ⑳  $V_{133}$  ⑳  $V_{134}$  ⑳  $V_{135}$ ⑳  $V_{136}$  ⑳  $V_{137}$  ⑳  $V_{138}$ ⑳  $V_{139}$  ⑳  $V_{140}$  ⑳  $V_{141}$ ⑳  $V_{142}$  ⑳  $V_{143}$  ⑳  $V_{144}$ ⑳  $V_{145}$  ⑳  $V_{146}$  ⑳  $V_{147}$ ⑳  $V_{148}$  ⑳  $V_{149}$  ⑳  $V_{150}$ ⑳  $V_{151}$  ⑳  $V_{152}$  ⑳  $V_{153}$ ⑳  $V_{154}$  ⑳  $V_{155}$  ⑳  $V_{156}$ ⑳  $V_{157}$  ⑳  $V_{158}$  ⑳  $V_{159}$ ⑳  $V_{160}$  ⑳  $V_{161}$  ⑳  $V_{162}$ ⑳  $V_{163}$  ⑳  $V_{164}$  ⑳  $V_{165}$ ⑳  $V_{166}$  ⑳  $V_{167}$  ⑳  $V_{168}$ ⑳  $V_{169}$  ⑳  $V_{170}$  ⑳  $V_{171}$ ⑳  $V_{172}$  ⑳  $V_{173}$  ⑳  $V_{174}$ ⑳  $V_{175}$  ⑳  $V_{176}$  ⑳  $V_{177}$ ⑳  $V_{178}$  ⑳  $V_{179}$  ⑳  $V_{180}$ ⑳  $V_{181}$  ⑳  $V_{182}$  ⑳  $V_{183}$ ⑳  $V_{184}$  ⑳  $V_{185}$  ⑳  $V_{186}$ ⑳  $V_{187}$  ⑳  $V_{188}$  ⑳  $V_{189}$ ⑳  $V_{190}$  ⑳  $V_{191}$  ⑳  $V_{192}$ ⑳  $V_{193}$  ⑳  $V_{194}$  ⑳  $V_{195}$ ⑳  $V_{196}$  ⑳  $V_{197}$  ⑳  $V_{198}$ ⑳  $V_{199}$  ⑳  $V_{200}$  ⑳  $V_{201}$ ⑳  $V_{202}$  ⑳  $V_{203}$  ⑳  $V_{204}$ ⑳  $V_{205}$  ⑳  $V_{206}$  ⑳  $V_{207}$ ⑳  $V_{208}$  ⑳  $V_{209}$  ⑳  $V_{210}$ ⑳  $V_{211}$  ⑳  $V_{212}$  ⑳  $V_{213}$ ⑳  $V_{214}$  ⑳  $V_{215}$  ⑳  $V_{216}$ ⑳  $V_{217}$  ⑳  $V_{218}$  ⑳  $V_{219}$ ⑳  $V_{220}$  ⑳  $V_{221}$  ⑳  $V_{222}$ ⑳  $V_{223}$  ⑳  $V_{224}$  ⑳  $V_{225}$ ⑳  $V_{226}$  ⑳  $V_{227}$  ⑳  $V_{228}$ ⑳  $V_{229}$  ⑳  $V_{230}$  ⑳  $V_{231}$ ⑳  $V_{232}$  ⑳  $V_{233}$  ⑳  $V_{234}$ ⑳  $V_{235}$  ⑳  $V_{236}$  ⑳  $V_{237}$ ⑳  $V_{238}$  ⑳  $V_{239}$  ⑳  $V_{240}$ ⑳  $V_{241}$  ⑳  $V_{242}$  ⑳  $V_{243}$ ⑳  $V_{244}$  ⑳  $V_{245}$  ⑳  $V_{246}$ ⑳  $V_{247}$  ⑳  $V_{248}$  ⑳  $V_{249}$ ⑳  $V_{250}$  ⑳  $V_{251}$  ⑳  $V_{252}$ ⑳  $V_{253}$  ⑳  $V_{254}$  ⑳  $V_{255}$ ⑳  $V_{256}$  ⑳  $V_{257}$  ⑳  $V_{258}$ ⑳  $V_{259}$  ⑳  $V_{260}$  ⑳  $V_{261}$ ⑳  $V_{262}$  ⑳  $V_{263}$  ⑳  $V_{264}$ ⑳  $V_{265}$  ⑳  $V_{266}$  ⑳  $V_{267}$ ⑳  $V_{268}$  ⑳  $V_{269}$  ⑳  $V_{270}$ ⑳  $V_{271}$  ⑳  $V_{272}$  ⑳  $V_{273}$ ⑳  $V_{274}$  ⑳  $V_{275}$  ⑳  $V_{276}$ ⑳  $V_{277}$  ⑳  $V_{278}$  ⑳  $V_{279}$ ⑳  $V_{280}$  ⑳  $V_{281}$  ⑳  $V_{282}$ ⑳  $V_{283}$  ⑳  $V_{284}$  ⑳  $V_{285}$ ⑳  $V_{286}$  ⑳  $V_{287}$  ⑳  $V_{288}$ ⑳  $V_{289}$  ⑳  $V_{290}$  ⑳  $V_{291}$ ⑳  $V_{292}$  ⑳  $V_{293}$  ⑳  $V_{294}$ ⑳  $V_{295}$  ⑳  $V_{296}$  ⑳  $V_{297}$ ⑳  $V_{298}$  ⑳  $V_{299}$  ⑳  $V_{300}$ ⑳  $V_{301}$  ⑳  $V_{302}$  ⑳  $V_{303}$ ⑳  $V_{304}$  ⑳  $V_{305}$  ⑳  $V_{306}$ ⑳  $V_{307}$  ⑳  $V_{308}$  ⑳  $V_{309}$ ⑳  $V_{310}$  ⑳  $V_{311}$  ⑳  $V_{312}$ ⑳  $V_{313}$  ⑳  $V_{314}$  ⑳  $V_{315}$ ⑳  $V_{316}$  ⑳  $V_{317}$  ⑳  $V_{318}$ ⑳  $V_{319}$  ⑳  $V_{320}$  ⑳  $V_{321}$ ⑳  $V_{322}$  ⑳  $V_{323}$  ⑳  $V_{324}$ ⑳  $V_{325}$  ⑳  $V_{326}$  ⑳  $V_{327}$ ⑳  $V_{328}$  ⑳  $V_{329}$  ⑳  $V_{330}$ ⑳  $V_{331}$  ⑳  $V_{332}$  ⑳  $V_{333}$ ⑳  $V_{334}$  ⑳  $V_{335}$  ⑳  $V_{336}$ ⑳  $V_{337}$  ⑳  $V_{338}$  ⑳  $V_{339}$ ⑳  $V_{340}$  ⑳  $V_{341}$  ⑳  $V_{342}$ ⑳  $V_{343}$  ⑳  $V_{344}$  ⑳  $V_{345}$ ⑳  $V_{346}$  ⑳  $V_{347}$  ⑳  $V_{348}$ ⑳  $V_{349}$  ⑳  $V_{350}$  ⑳  $V_{351}$ ⑳  $V_{352}$  ⑳  $V_{353}$  ⑳  $V_{354}$ ⑳  $V_{355}$  ⑳  $V_{356}$  ⑳  $V_{357}$ ⑳  $V_{358}$  ⑳  $V_{359}$  ⑳  $V_{360}$ ⑳  $V_{361}$  ⑳  $V_{362}$  ⑳  $V_{363}$ ⑳  $V_{364}$  ⑳  $V_{365}$  ⑳  $V_{366}$ ⑳  $V_{367}$  ⑳  $V_{368}$  ⑳  $V_{369}$ ⑳  $V_{370}$  ⑳  $V_{371}$  ⑳  $V_{372}$ ⑳  $V_{373}$  ⑳  $V_{374}$  ⑳  $V_{375}$ ⑳  $V_{376}$  ⑳  $V_{377}$  ⑳  $V_{378}$

Orders:- $v_2, v_5, v_1, v_4, v_3, v_6$  $v_5, v_2, v_1, v_4, v_3, v_6$ Approach-1Step 1: Identify vertices that has no incoming edgesStep 2:- Delete this vertex of indegree 0 and all its outgoing edges from the graph. Place it in the outputStep 3:- Repeat step ① & ② until graph is emptyApproach 2Time complexity:  $O(V+E)$  → No. of entries in A.L.Step 1: Point highest incoming degree

Step 2 :- Get into vertices previously connected deleted  
Repeat & 2 until graph is empty

1	2	3	4	5	6
1	-1	2	1	0	2

 $v_2$ 

1	2	3	4	5	6
0	-1	2	1	0	2

 $v_2, v_3$ 

0	-1	2	1	-1	2
---	----	---	---	----	---

 $v_2, v_5, v_1$ 

-1	-1	1	0	-1	2
----	----	---	---	----	---

 $v_2, v_5, v_1, v_4$ 

-1	-1	0	-1	-1	1
----	----	---	----	----	---

 $v_2, v_5, v_1, v_4, v_3$ 

-1	-1	-1	-1	-1	0
----	----	----	----	----	---

 $v_2, v_5, v_1, v_4, v_3, v_6$ 

-1	-1	-1	-1	-1	-1
----	----	----	----	----	----

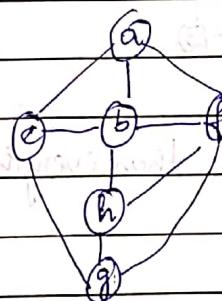
Time complexity  $\rightarrow O(V+E) \rightarrow$  Topological sort.

$$= O(V^2)$$

(1)

GATE 2003

DFS on



I) abeghf

II) abfegh

III) abfghed

IV) afghabe

So I, III, IV

are DFS

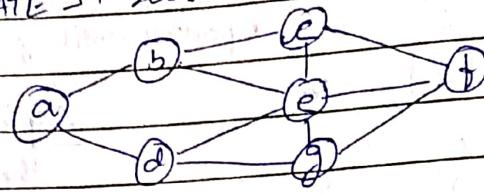
I)	$v=a$ $w=f, b, h, e, g, f$	$v=b$ $w=a, e, f, h$	$v=c$ $w=a, g, b, f$	$v=d$ $w=e, h, f, g$	$v=e$ $w=d, b, f, g$	$v=f$ $w=d, b, f, g$
----	-------------------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------

II) not possible	$v=a$ $w=d, b, f, h$	$v=b$ $w=a, e, f, h$	$v=f$ $w=d, b, f, g$	$v=h$ $w=d, b, f, g$	$v=g$ $w=e, h, f, g$	$v=e$ $w=d, a, g, b, f$
------------------	-------------------------	-------------------------	-------------------------	-------------------------	-------------------------	----------------------------

III)	$v=a$ $w=d, b, f, h$	$v=f$ $w=a, b, f, g$	$v=g$ $w=a, e, f, h$	$v=h$ $w=d, b, f, g$	$v=b$ $w=d, a, e, f, h$	$v=e$ $w=a, g, b, f$
------	-------------------------	-------------------------	-------------------------	-------------------------	----------------------------	-------------------------

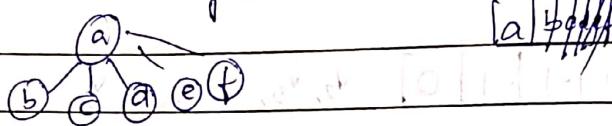
(2)

GATE IT 2008



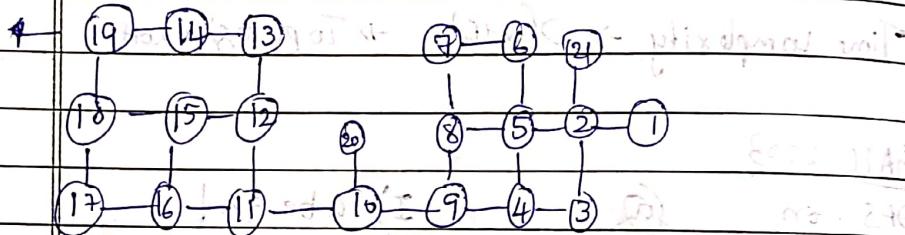
(direct)

- (a) abefdcg → Check connectivity of graph ✓
- (b) abefcgda → ✓
- (c) adgebcf → ✓
- (d) adabcgef → X

DFS → n-1 of elements BFS queue of 2<sup>n-1</sup>

(3)

GATE 2014 → A graph without number given! We shall number it



Depth of recursion stack → 19, then everything gets popped off

(4)

GATE 2006

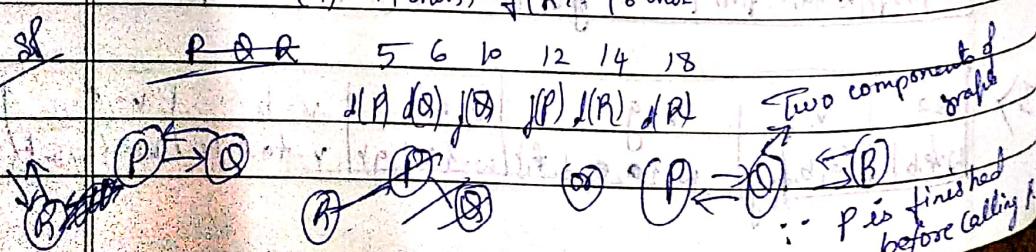
Consider a DFS of an undirected graph with 3 vertices P, Q, R. Let discover time  $d(u)$  represent the time instant when the vertex 'u' is first visited and finish time  $f(u)$  represent the time instant when vertex 'u' is last visited. Given that  $d(P)=5$  units,  $f(P)=12$  units.

$d(Q)=6$  units,  $f(Q)=10$  units

$d(R)=14$  units,  $f(R)=18$  units

P Q R 5 6 10 12 14 18

$d(P)d(Q)f(R) f(P)d(R)f(Q)$  Two components of graph



∴ P is finished before calling R

# Hashing

SURYA Gold

Date \_\_\_\_\_

Page \_\_\_\_\_

## Searching (Worst case)

Unsorted Array	$O(n)$
Sorted Array	$O(\log n)$
Linked List	$O(n)$
Binary Tree	$O(n)$
Binary Search Tree	$\Theta(\log n) \Rightarrow O(n)$
AVL Tree	$O(\log n)$
Priority Queue	$O(n)$ $O(1) \rightarrow \min \max$

Hashing reduces search time. It is order  $O(1)$

## Direct Address Table

Each record has unique key you have to know key value.

We can store it in array

Separate arrays for record & key to be maintained

Fails when values are large

## Introduction to Hashing

→ No restriction in no. of elements

Requires more space since we maintain a linked list

## Open Addressing

Inserts within table, No. element must be of table size

Chaining → Deletions easier

OA → Insertion, Searching

In insertion, deletion, searching - constant time in hashing

Chaining → Worst Case -  $O(n)$

Average Case -  $\Theta(1 + \frac{n}{m})$

Load factor  $\alpha = \frac{n}{m}$  = Elements in Hash table / Size of hash table

Optimal load factor =  $K_1 \# m$  where  $K_1$  is a constant

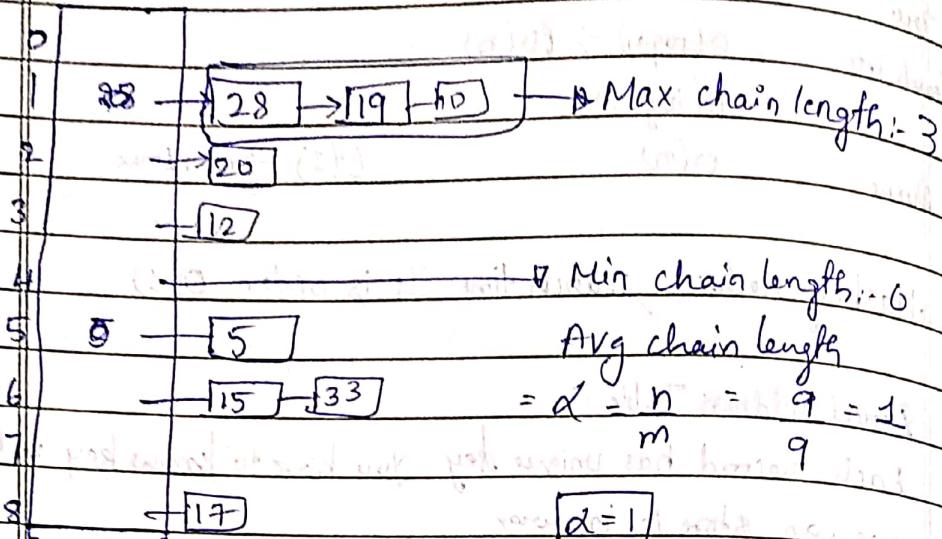
$$\alpha = K_1$$

GATE 2014

(1)

$h(K) = k \bmod 9$ . Hash Table has 9 slots, chaining is used.  
 Keys: 5, 28, 19, 15, 20, 33, 12, 17, 10. Then max, min & average chain lengths in hash table.

Sol:



(2)

Consider a hash-table with 100 slots. Collisions are resolved using Chaining. Assume simple uniform hashing. What is the probability that the first 3 slots are unfilled after 3 insertions?

Sol:

$$100 - 3 = 97$$

$$\left(\frac{97}{100}\right) \left(\frac{97}{100}\right) \left(\frac{97}{100}\right) = \frac{95 \times 96 \times 97}{100^3}$$

(3)

Consider a hash-table with  $n$  buckets, where chaining is used to resolve collisions. The hash function is such that the probability that a key value is hashed to a particular bucket is  $1/n$ . The hash-table is initially empty & 'k' distinct values are inserted in the table.

a) What is the probability that bucket number '1' is empty after 'k' insertions?

b) What is the probability that no collision has occurred in any one of  $k$  insertions?

c) What is the probability that first collision occurs at  $k^{th}$  insertion?

8 (a)  $\binom{n-1}{n} \binom{n-1}{n} \dots \binom{n-1}{n}$   
 $= \binom{n-1}{n}^k$

(b)  $\binom{n}{n} \binom{n-1}{n} \binom{n-2}{n} \dots \binom{n-(k-1)}{n}$

(c)  $\binom{n}{n} \frac{k-1}{n}$

Open Addressing

Time Complexity -  $O(n)$   $\rightarrow$  Worst case

$O(1) \rightarrow$  Average case

Deletions is troublesome

$$h(k) = (a + \sum_{i=0}^{k-1} m) \mod m$$

Linear Probing

0	$h(k) : U \rightarrow \{0, \dots, m-1\}$
1	$h(k) = a_2 + (k - (0, 1))m$
$a+1$	$h'(k, i) \in (h(k) + i) \mod m$
$a+2$	$h'(k, 1) = (a + 1) \mod m$
$m-1$	$h'(k, 2) = (h(k) + 2) \mod m$
	$= (a + 2) \mod m$

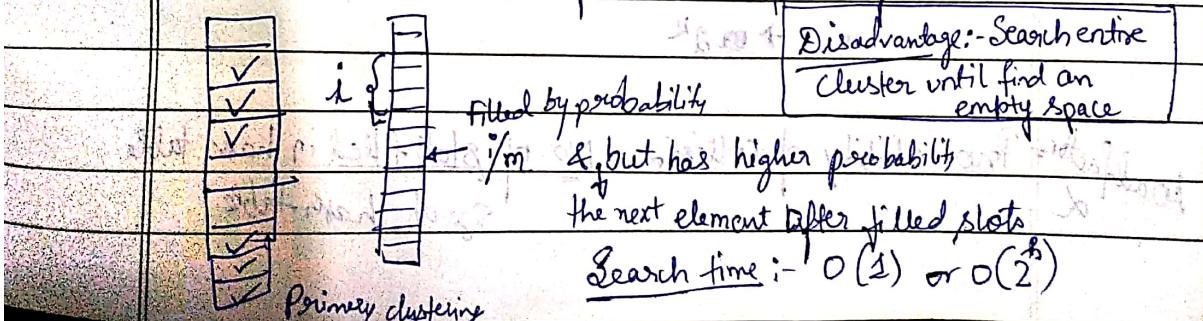
We search till we get free slot

$i = (0, 1, 2, \dots, m-1)$  provided  $(i) \neq \text{free}$

We probe only  $m$  times if there are  $m$  slots in the table

Problem: ① Secondary clustering  $\rightarrow$  It is a problem in which both elements start from same  $i$  & then probing is same for both

② Primary clustering  $\rightarrow$  It is a problem in which many elements cluster in a particular part of hash table



Quadratic Probing

$$h'(k, i) = (h(k) + c_1 i + c_2 i^2) \bmod m$$

Next probe position is quadratic

Probe position increases quadratically

Avoids ① Primary clustering

Primary clustering is a process in which a block of data is formed in the hash table when collision is occurred.

Eg:  $m=10$ ,  $(0 \dots 9)$ ,  $c_1=1$ ,  $c_2=1$ ,

$$h(k) = k \bmod 10$$

1	x
2	
3	x
4	
5	
6	x
7	
8	
9	
10	

$$h'(k_1, 1) = 1 + 1 + 1 \quad h'(k_1, 5) = 1 + 1 + 25$$

$$= 3 \quad = 27$$

$$h'(k_1, 2) = 1 + 2 + 4 \quad h'(k_1, 6) = 1 + 1 + 36$$

$$= 6 \quad = 37$$

$$h'(k_1, 3) = 1 + 1 + 9 \quad h'(k_1, 7) = 1 + 1 + 49$$

$$= 11 \quad = 51$$

$$h'(k_1, 4) = 1 + 1 + 16 \quad h'(k_1, 8) = 1 + 1 + 64$$

$$= 18 \quad = 66$$

$$h'(k_1, 9) = 1 + 1 + 81 = 81$$

Problem :- Whole (Entire) table is not examined

Widely choose  $c_1, c_2, m$

Make  $c_1 \otimes c_1 + c_2 i^2$  dependent on key, then it's better

in form  $(s + t)$ .

Double Hashing

Avoids ① Primary clustering ② Secondary clustering

Successive probes depends on keys and does not

follow same sequence if initial probe position is same

$$h'(k) = (h_1(k) + i h_2(k)) \bmod n$$

Choose  $m, n$  are relatively prime

$$h_2(k) \rightarrow \text{odd } < m$$

$$m \rightarrow \text{odd}$$

Load factor  
 $\alpha$

Probability of collision = No. of slots filled in hash table

Size of hash table

- ① Consider a hash table that distributes keys uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5

$$\text{S.f. : } \frac{i}{20} \geq 0.5 \quad i \geq 0.5 \times 20 = 10$$

- ② How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table shown above?

0		52, 23, 34, 46
1		None
2	42	4 elements 42, 23, 34, 46 are in their position
3	23	are in their position
4	34	2 elements 52, 33 are probed
5	52	(42, 23, 34) 52, 33, 46
6	46	46 → 5 ways
7	33	42, 23, 34 → 3! ways
8		$3! \times 5 = 6 \times 5$
9		= 30 ways

# Recursion

```

(1) A(n)
    {
1. if (n > 0)
2. {
P   3. printf ("%d", n-1)
A   4. A(n-1)
5. }
}
main()

```

*Activation record is created on calling each function in stack*

Stack	A(3)	A(2)	A(1)	A(0)	Instruction pointer
main()	n=3	n=2	n=1	n=0	points to 5 which
	5	5	5	5	is next instruction to be executed
	main()   A(3)   A(2)   (A(1))   A(0)				n is local variable
	main()   A(3)   A(2)   A(1)				
	main()   A(3)   A(2)				
	main()   A(3)				
	main()				

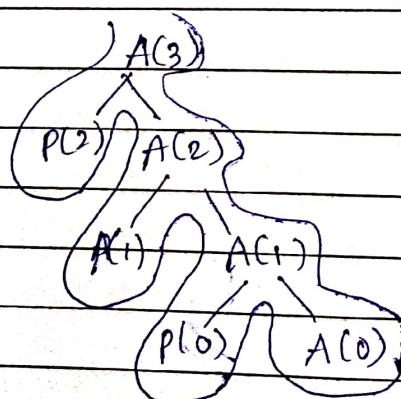
A(0) returns without doing anything. Then A(1) is called. we have to go to line no. 5 & same continues.

main() is called & gets finished.

Space complexity - Stack records =  $n+1$

Space =  $O(n)$

$$\begin{aligned}
 \text{Time complexity} \\
 T(n) &= C + T(n-1) && \text{5 Activation records are created} \\
 &= O(n)
 \end{aligned}$$



Depth of tree =  $n+1$

= Max length of stack records

(2) A(n)

{

if (n &gt; 0)

{

Pf(n);

A(n-1);

Pd(n);

}

}

}  $\rightarrow$   $A(2) = P(1) + A(1) + P(2)$ 

+ P(3)

}  $\rightarrow$   $A(3) = P(1) + P(2) + A(2) + P(3)$ }  $\rightarrow$   $A(n) = P(1) + P(2) + \dots + P(n) + A(n-1)$ }  $\rightarrow$   $A(n) = C + T(n-1)$ }  $\rightarrow$   $T(n) = C + T(n-1)$ }  $\rightarrow$   $T(n) = C + T(n-1) + T(n-2) + \dots + T(1) + T(0)$ }  $\rightarrow$   $T(n) = C + nT(1)$ }  $\rightarrow$   $T(n) = C + nC$ }  $\rightarrow$   $T(n) = C(1+n)$ }  $\rightarrow$   $T(n) = O(n)$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ }  $\rightarrow$   $O/p :- 3 2 1 1 2 3$ 

(3) A(n)

{

if (n &gt; 0) {

{

A(n-1);

Pf(n);

A(n-1)(a);

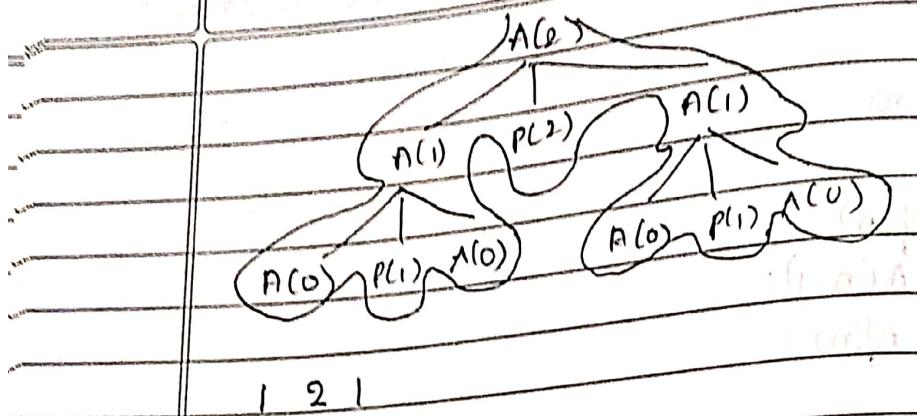
}

}

}

}

}



$A(0)$	$A(0)$	$A(0)$
$A(1)$	$A(1)$	$A(1)$
$A(2)$		

No. of stack records =  $n+1$   
 $= 2+1$   
 $= 3$

Dynamic programming saves function calls! Hence saving time

Analysing the recursion

$f(n)$   
{

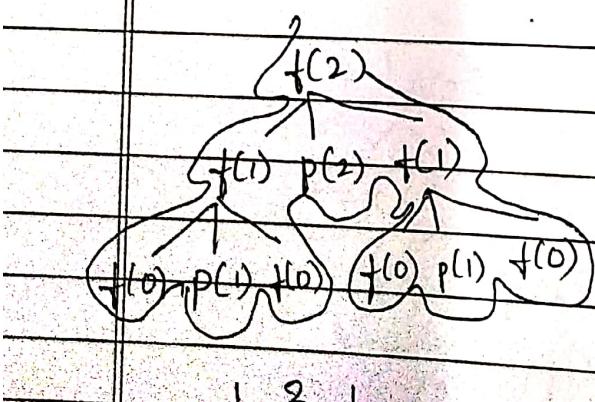
if( $n == 0$ )  
 return;

$f(n-1);$

printf(n);

$f(n-1);$

}



No. of function call

$$f(1) = 3$$

$$f(2) = 7$$

$$f(3) = 27$$

$$f(n) = 2^{n+1} - 1$$

$F(n) = \text{No. of times } f(n) \text{ called}$

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

1 - already calling

$$\begin{aligned}
 F(n) &= 2F(n-1) + 1 & F(n-1) &= 2F(n-2) + 1 \\
 &= 2[2(F(n-2) + 1) + 1] & F(n-2) &= 2F(n-3) + 1 \\
 &= 2[2[2(F(n-3) + 1) + 1] + 1] \\
 &= 2^2 F(n-2) + 2 + 1 \\
 &= 2^3 F(n-3) + 2^2 + 2 + 1 \\
 &= 2^i F(n-i) + 2^{i-1} + 2^{i-2} + 2^{i-3} + \dots + 1 \\
 F(n) &= 1, n=0 & i = 0 & \\
 && i=n & \\
 &= 2^n F(0) + 2^{n-1} + 2^{n-2} + \dots + 1 \\
 &= 2^n \cdot 1 + 2^{n-1} + 2^{n-2} + \dots + 1 \\
 &= 2^n + 2^{n-1} + 2^{n-2} + \dots + 1
 \end{aligned}$$

It is in Geometric Progression

$$a(r^{n+1} - 1)$$

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

$$= 1(2^{n+1} - 1)$$

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

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

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

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

$$= O(2^n)$$

Space Complexity =  $O(n)$ . Since depth of tree is Order of  $n$

Gate 2001

```
void abc(char *s)
```

{

```
if(s[0] == '\0') return;
```

```
abc(s+1);
```

```
abc(s+1);
```

```
printf("%c", s[0]);
```

```
}
```

main()

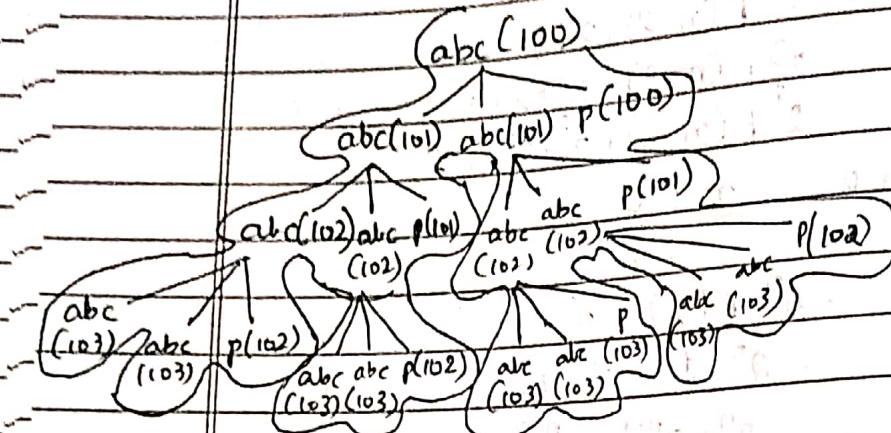
```
abc("123");
```

}

a)

'In memory it is stored as

	1	1	2	3	10
	100	101	102	103	

O/p:- 3 3 2 3 3 2 1b) If  $abc(s)$  is called with a null terminated string  $s$  of length ' $n$ ' characters (not counting null ('10') character).How many characters will be printed by  $abc(s)$ 

$c(n)$  - no. of characters printed by  $abc(s)$   
 giving input of  $n$  characters

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

Just above use back substitution method

$$c(n) = 2^1 \cdot 2^2 \cdot 2^3 c(n-3) + 2^2 + 2^1 + 2^0$$

$$= 2^i (n-i) + 2^{i-1} + 2^{i-2} + \dots + 2^1 + 2^0$$

$$n-i=1 \quad (\because c(1)=1)$$

$$= 2^{n-1} c(1) + 2^{n-2} + 2^{n-3} + \dots + 2^1 + 2^0$$

$$= \frac{a(r^{n+1}-1)}{r-1}$$

$$\frac{1}{r-1} (2^{n+1}-1) \quad (0/1 + 1 c(1) - 1 \text{ char})$$

$$c(n) = 2^n - 1 \quad (C(2) = 3 \text{ char})$$

(cross verify with a)

GATE 2004

int suc(int n)

{

if ( $n == 1$ )

return 1;

else

return (rec( $n - 1$ ) + rec( $n - 1$ ))

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

$$T(n) = 1 \text{ if } n=1$$

$$T(n) = 2^i T(n-i) + 2^{i-1} + 2^{i-2} + \dots + 1$$

$$\begin{aligned} n-i &= 1 & T(n) &= 2^{n-1} T(1) + 2^{n-2} + 2^{n-3} + \dots + 1 \\ &= 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 1 \\ &= 1(2^{n-1} - 1) \end{aligned}$$

$$T(n) = 2^n - 1$$

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

Ques 2

gate 2005

void foo(int n, int sum)

{

    int k=0, j=0;

    if (n==0) return

    k=n%10, j=n/10;

    sum = sum+k;

    foo(j, sum)

    printf("%d", k);

}

int main()

{

    int a = 2048, sum=0;

    foo(a, sum);

    printf("%d", sum);

}

main	for	for			
$a=2048$	$n = 2048$	$n = 204$	$m = 20$	$n = 2$	$n = 0$
$sum = 0$	$sum = 08$	$sum = 812$	$sum = 12$	$sum = 2$	$sum = 14$
$j = 0$	$j = 8$	$j = 20$	$k = 0$	$k = 2$	$k = 6$

2048 1024 512 128 2 14

Gate 2010

```

int f(int *a, int n)
{
    if (n <= 0)
        return 0;
    else
        if (*a % 2 == 0)
            return *a + f(a+1, n-1);
        else
            return *a - f(a+1, n-1);
}

```

```

int main()
{
    int a[] = {12, 7, 13, 4, 11, 6};
    printf("%d", f(a, 6));
    return 0;
}

```

100 102 104 106 108 110

12	7	13	4	11	6
----	---	----	---	----	---

f(a, 6)

$\Rightarrow f(100, 6)$

$\downarrow$   
 $12 \% 2 == 0$ ,

$12 + f(102, 5)$

$\downarrow$   
 $7 \% 2 == 0$

$7 - f(104, 4)$

$\downarrow$   
 $13 \% 2 == 0$

$13 - f(106, 3)$

$$4 \cdot 2 = 0$$

$$4 + f(108, 2)$$

↓

$$11 \cdot 2 = 0$$

$$11 - f(110, 1)$$

$$6 \cdot 2 = 0$$

$$6 + f(112, 0)$$

$$6 + 0 = 6$$

$$\cancel{12+7} - \cancel{13-4} \cancel{+ 11-6}$$

$$4 + 5$$

$$13-9$$

$$7-4$$

$$12+3$$

$$= 15$$

$$12 + (7 - (13 - (4 + (11 - (6 + (0)$$

$$11 - 6$$

$$4 + 5$$

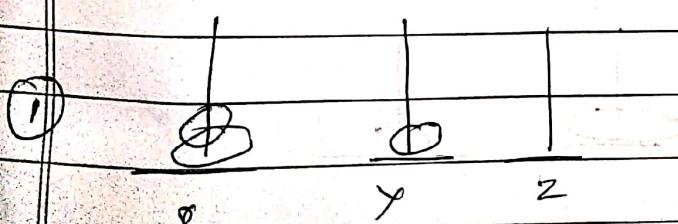
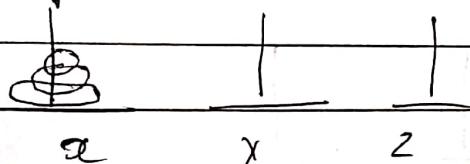
$$13-9$$

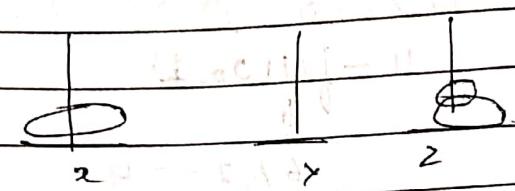
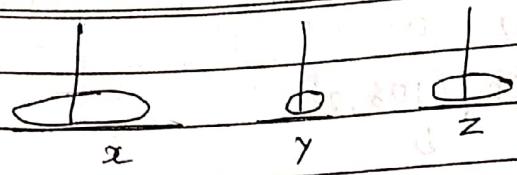
$$7-4$$

$$12+3$$

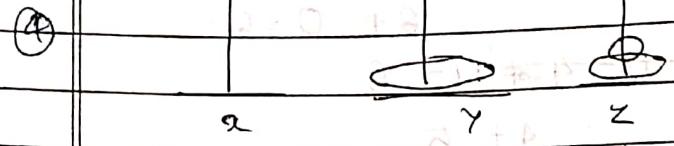
$$= 15$$

### Tower of Hanoi



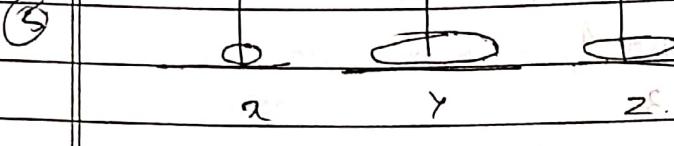


(x,y), z + 2



x + y, z + 2

P = 1



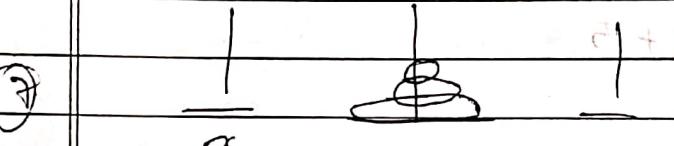
x + y, z + 2

P = 1



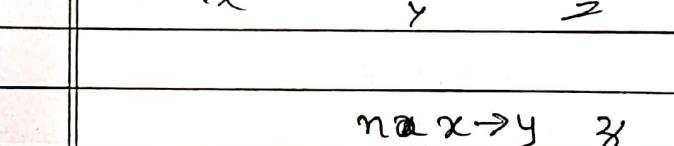
x + y, z + 2

P = 1



x + y, z + 2

P = 1



x + y, z + 2

P = 1

E + P

$\max(x \rightarrow y, z)$

$n-1, x \rightarrow y, z$

$x \rightarrow y$

$n-1$

$n-1, z \rightarrow y, x$

Algorithm for Towers of Hanoi

$\text{TOH}(n, x, y, z)$

{

if ( $n \geq 1$ )

{

$\text{TOH}(n-1, x, z, y)$

move top 'x' to y;

$\text{TOH}(n-1, z, y, x);$

}

}

$\text{TOH}(3, x, y, z)$

$\text{TOH}(2, x, y, z)$

$\text{TOH}(2, z, y, x)$

$\text{TOH}(1, x, y, z)$

$x \rightarrow y$

$\text{TOH}(1, z, y, x)$

$y \rightarrow z$

$\text{TOH}(1, x, z, y)$

$z \rightarrow y$

$\text{TOH}(1, y, z, x)$

$x \rightarrow z$

$\text{TOH}(1, x, y, z)$

$z \rightarrow y$

$\text{TOH}(1, z, y, x)$

$y \rightarrow z$

$\text{TOH}(1, x, z, y)$

$z \rightarrow x$

$\text{TOH}(1, y, x, z)$

$x \rightarrow y$

$\text{TOH}(1, x, y, z)$

$y \rightarrow z$

$\text{TOH}(1, y, z, x)$

$z \rightarrow x$

→ 3 disc : No. of function invocations = 15

No. of movements = 7

Analysis of tower of hanoi

$\text{TOH}(n, x, y, z) \quad I(n) = 1 + 2I(n-1) \quad \text{①} \quad \begin{matrix} \text{Invocation} \\ \text{required for } n \text{ disc} \end{matrix}$

{  $I(n) = 1, n=0$

if ( $n > 1$ )  $I(n-1) = 1 + 2I(n-2) \quad \text{②}$

{  $I(n-2) = 1 + 2I(n-3) \quad \text{③}$

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

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

$= 2^2(2I(n-3) + 1) + 2 + 1$

$= 2^3 I(n-3) + 2^2 + 2 + 1$

$$\begin{aligned}
 &= 2^i I(n-i) + 2^{i-1} + 2^{i-2} + \dots + 1 \\
 n-i=0, i=n &\Rightarrow 2^n I(0) + 2^{n-1} + 2^{n-2} + \dots + 1 \\
 &+ 2^n + 2^{n-1} + 2^{n-2} + \dots + 1 \\
 &= \frac{1(2^{n+1}-1)}{2-1}
 \end{aligned}$$

$$I(n) = 2^{n+1} - 1 \quad n=3, I(n) = 2^4 - 1 = 15$$

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

$$M(n) = 0, n=0 \quad M(n) = 1, n=1$$

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \dots + 1$$

$$n-i=0, i=n$$

$$= 2^n M(0) +$$

$$= 2^n 0 + 2^{n-1} + 2^{n-2} + \dots + 1$$

$$= 2^{n-1} + 2^{n-2} + \dots + 1$$

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

$$M(n) = 2^n - 1$$

Movement of disc  $M(n)$

$$T(n) = 2T(n-1) + 1 \quad (T(n) - \text{Time complexity})$$

$$T(n) = 0, n=0$$

$$T(n) = 2^n - 1 \quad (\text{Same as } I(n))$$

$$T(n) = 2 \cdot 2^n - 1$$

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

$$T(0), T(1), T(2), T(3)$$

based on result of recursion

$$T(1)$$

$$T(2)$$

$$T(3)$$

$$\text{Stack records} = n+1 + i \cdot (b/c)$$

$$= O(n)$$

Improved algo

$TOR(n, x, y, z)$

{

if ( $n > 1$ )

{

$TOR(n-1, x, z, y)$

more tor 'x' to 'y'

$\downarrow TOR(n-1, z, y, x)$

else

if ( $n == 1$ ) more 'x' to 'y'

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

$$I(n) = 1, n = 1$$

$$I(n) = 1, n = 0$$

$$I(n) = 2^i I(n-i) + 2^{i-1} + 2^{i-2} + \dots + 1$$

$$n-i = 1$$

$$\therefore i = n - 1$$

$$2^{n-1} \cdot 1 + 2^{n-2} + 2^{n-3} + \dots + 1$$

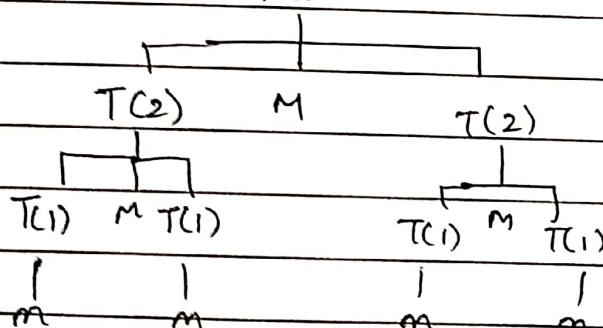
$$= 2^{n-1} + 2^{n-2} + \dots + 1$$

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

$$I(n) = 2^n - 1$$

$$n = 3, I(n) = 2^3 - 1 = 7$$

$T(3)$



No. of invocations = 7