

GATE CSE MADEEASY HAND NOTES-2020

**6.SUBJECT NAME-
DATA STRUCTURE**

GATEIESMENTOR

Data Structure

GATE NOTES

COMPUTER SCIENCE
MADEEEASY CLASS HAND NOTES

Page-129

Structures

classmate

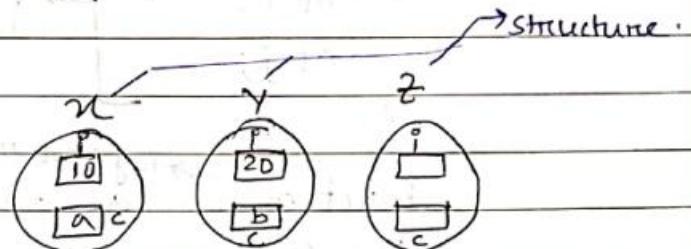
Date _____

Page _____

Introduction of Structure:

struct
{
 int i;
 char c;
};
u, v, z;

→ declaration



$$\begin{array}{l|l} n.i = 10 & v.i = 20 \\ n.c = 'a' & v.c = 'b' \end{array}$$

→ member operators

struct ex → Tag
{
 int i;
 char c;
};

struct ex u, v, z;

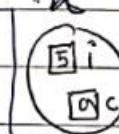
struct ex n={5, 'a'};

struct ex1 *

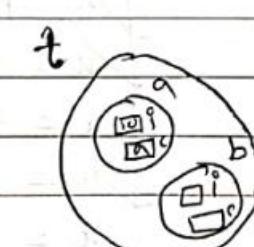
{
 struct ex a;
 struct ex b;
};

struct ex t;

n



→ declaration.



$$\begin{array}{l} t.a.i = 10 \\ t.a.c = 'a' \end{array}$$

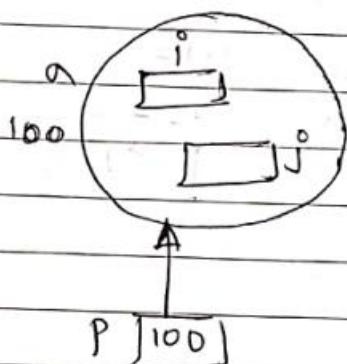
→ define.

- Example code on structures, arrays and pointers =

```
struct node
{
    int i;
    int j;
};
```

declaration (no memory allocated)

```
struct node a, *p;
p = &a;
```



$(*p).i$ → access of member of structure using pointer.
 $a.i$ → access by name of structure.

$(p \rightarrow i)$ same as $(*p).i$

→ structures can be passed to a function as well as returned by a function.

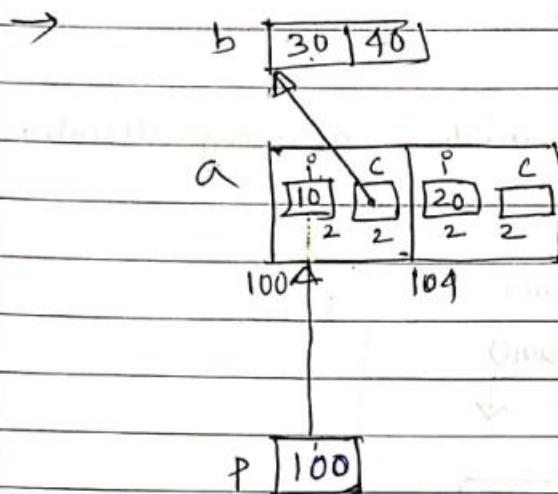
struct node fun(struct node n₁, struct node n₂);

Example :

```
struct node
{
    int i;
    int *c;
};
```

```
struct node a[2], *p;
int b[2] = {30, 40};
p = &a[0];
a[0].i = 10; a[1].i = 20;
a[0].c = b;
```

$\checkmark n = ++p \rightarrow i$
 $\checkmark n = (++p) \rightarrow i$
 $\checkmark n = (p++) \rightarrow i$
 $\checkmark n = *p \rightarrow c$
 $\checkmark n = *p \rightarrow c++$
 $\checkmark n = (*p \rightarrow c)++$
 $\checkmark n = *p++ \rightarrow c$.



$\checkmark \quad n = (++(p \rightarrow i)) \quad / \quad n = 11$

$\checkmark \quad n = (++p) \rightarrow i \quad / \quad n = 20$

$\checkmark \quad n = (p++) \rightarrow i \quad / \quad \begin{array}{l} \text{new } p \rightarrow i \\ \text{means } p = p+1 \end{array} \quad / \quad n = 10$

$\checkmark \quad n = (*p \rightarrow c) \quad / \quad n = 30$

$\checkmark \quad n = (*p \rightarrow c)++ \quad / \quad n = 30 \quad (\text{Post post increment } n=31)$

$\checkmark \quad n = (*p \rightarrow c)++ \quad / \quad n = 30$

$\checkmark \quad n = (*((p++) \rightarrow c)) \quad / \quad n = 30$

- Self referential structures:

struct ex

```
{ int i; }
```

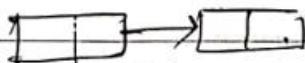
struct ex *link;

```
{; }
```

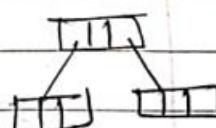
struct ex abc;

example of self referential structure =

(1) Link list :

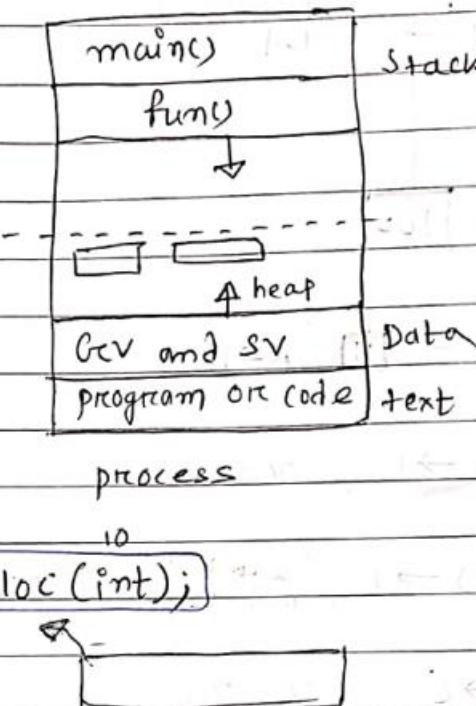


(2) tree



• Malloc =

→ static and global variables: memory allocated before run the program.



void * malloc (int);

malloc function call make a space ^{of process} ~~in~~ Dynamically
In the heap and then return the starting address
of this ~~space~~ location.

int *p = (int *) malloc (2);

void * malloc (sizeof(int)); — use this one

int *p = (int *) malloc (sizeof(2))

↳ type casting

'syntax of (which we use to make struct and get pointer)

Struct node
{ int i; }

Struct node * p = (Struct node *) malloc (sizeof(Struct node))

Struct node * l;

};

malloc(sizeof(Struct node)),

Linked List

classmate

Date _____

Page _____

Introduction of single linked list:

→ single linked list contain nodes, and this node generally structured w/ and which are self referential/referential.

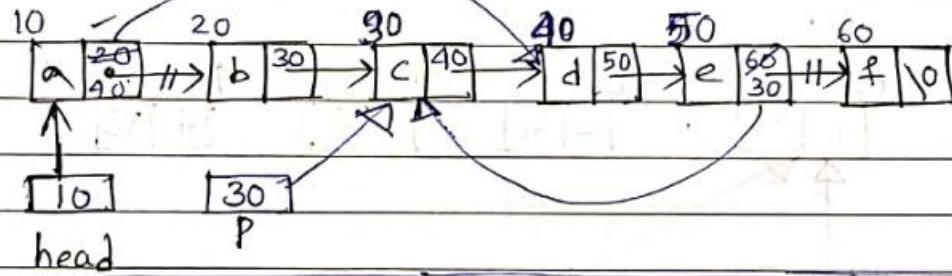
struct node

{

char data;

struct node *link;

};



→ Every node has only one link (pointer) so, that this is called single linked list.

struct node (*head);

→ linked list are sequential access.

→ going to any particular number structures (linked list) take $O(n)$ time.

struct node *p;

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

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

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

$\text{pf} ("abc", \text{head} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{data});$

Output: "d"

~~Traversing~~

In link list generally perform three operations -

- Traversing. (traverse the entire list).
- inserting. (create a new node and insert it).
- delete. (delete a node).

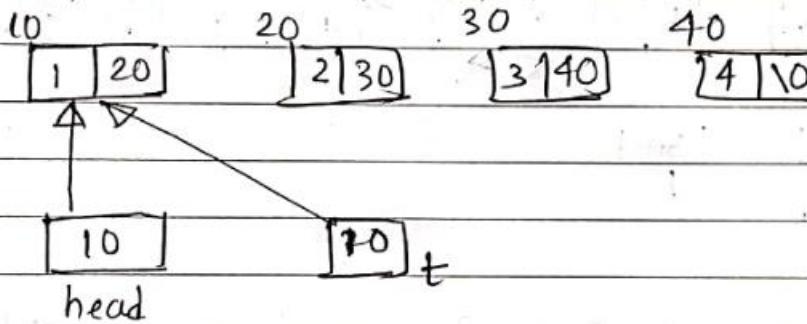
struct node

{

int i;

struct node *link;

};



① Traversing:

struct node *t;

t = head;

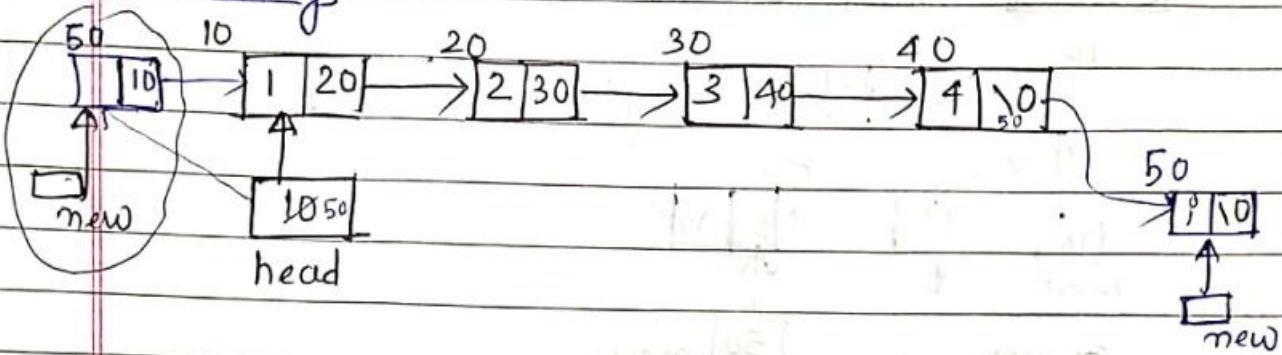
while ($t \neq \text{NULL}$) \equiv while (t)

{ printf ("%d", t->i)

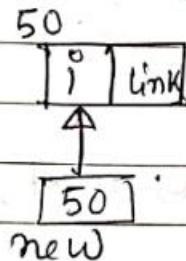
$t = t \rightarrow \text{link}$. }

Output: 1 2 3 4

(2) Inserting :



`struct node *new = (structnode *) malloc (sizeof (structnode))`



[Case-1] Insert at the beginning =

`new->link = head`

`head = new`

[Case-2] Insert at the end =

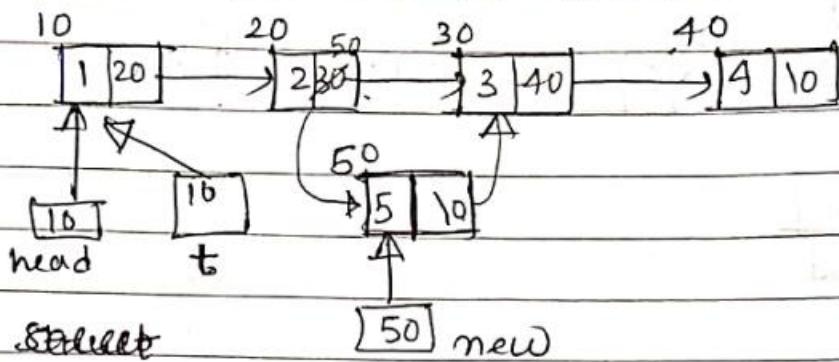
`struct node *t = head;`

`while (t->link != NULL) = while (t->link)`

{ `t = t->link;` }

`t->link = new;`

`new->link = NULL;`

Case - 3 Insert at the middle.

struct node *t; = head;

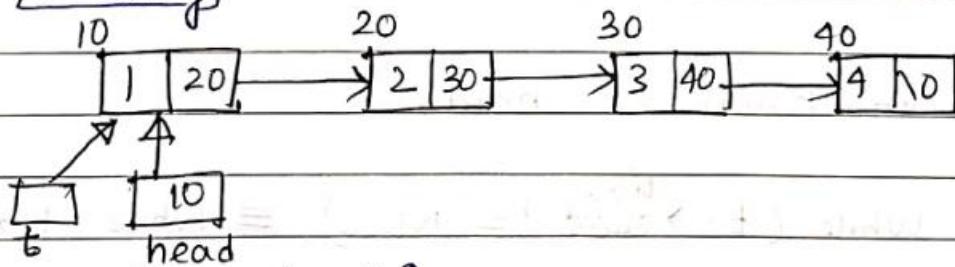
while ($t \rightarrow i \neq 2$)

~~* \Rightarrow {~~ $t = t \rightarrow$ ~~next~~ next; }

~~temp = $t \rightarrow$ next~~
~~link~~

$new \rightarrow link = t \rightarrow link$

$t \rightarrow link = new;$

③ Deleting

delete from head :-

struct node *t = head;

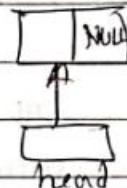
head = head \rightarrow ~~next~~ link;

free(t);

(to delete ~~first~~ first one)

Preconditions Condition Checking =

```
if (head == NULL)
    return
if (head → next == NULL)
    free(head)
```



before deletion check this one. (head are null or not)

(head → link is null or not)

- delete from tail :

```
struct node *t = head;
while (t → next → next != NULL)
    while (t → link → link != NULL)
        { t = t → next; }
```

free (t → list)

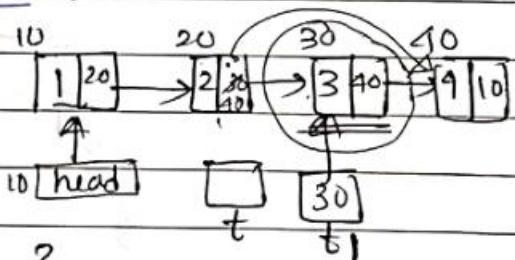
t → list = null

- delete node which contain 3 : i=3

```
struct node *t = head;
```

while (t → list → i != 3)

{ t = t → next; }



struct node *t1 = t → next;

~~t → next = t → next →~~

~~t → list = t → list → list;~~

Free (t1);

Crude, 2005-2008

classmate

Date _____

Page _____

Question -1

struct node

{

int val;

struct node *next;

}

void rearrange (struct node *list)

{

struct node *p, *q;

int temp;

(or)

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

p = list; q = list -> next;

list == NULL

= !list

while (q)

= (while (q != NULL))

{

temp = p -> val; p -> val = q -> val;

q -> val = temp; p = q -> next;

q = p ? p -> next : 0;

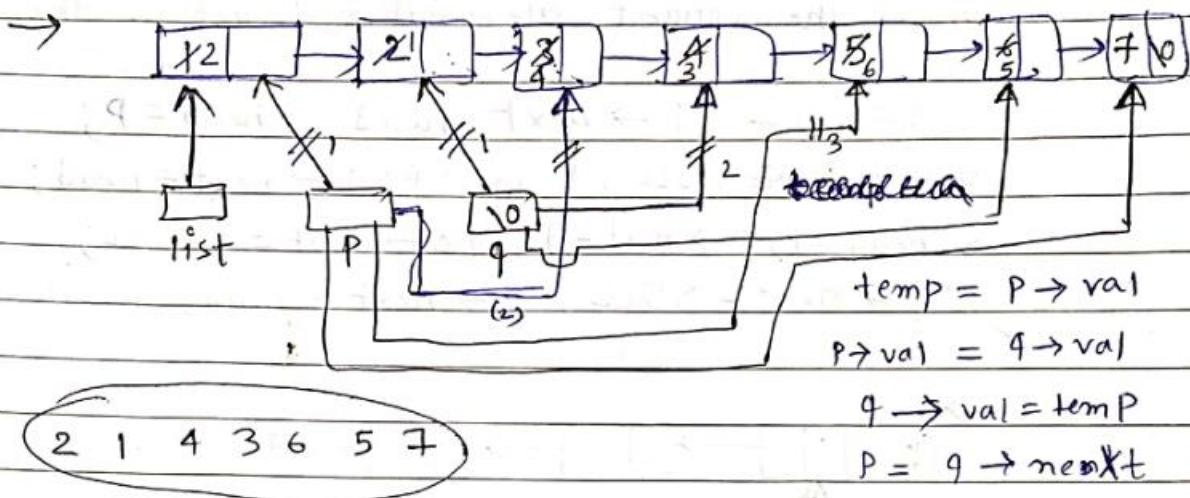
}

prep.

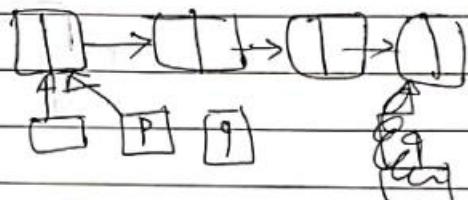
1 2 3 4 5 6 7

(A) 1, 2, 3, 4, 5, 6, 7. (C) 1, 3, 2, 5, 4, 7, 6.

(B) 2, 1, 4, 3, 6, 5, 7. (D) 2, 3, 4, 5, 6, 7, 1.

Question - 2

```
typedef struct node {
    int value;
    struct node *next;
} Node;
```



Node * move-to-front (Node * head)

```
Node *p, *q;
if (head == NULL) || (head->next == NULL)
    return head;
```

$q = \text{NULL}$;

$P = \text{head}$;

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

```
{   q = p;
```

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

}

return head;

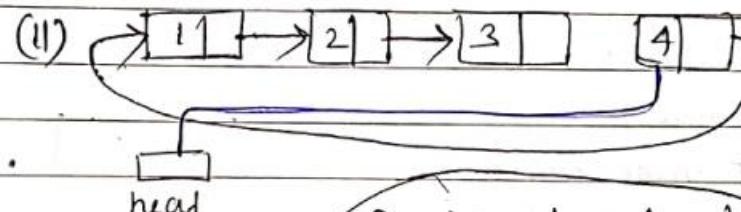
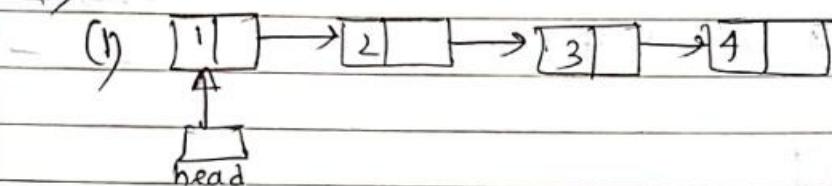
}

(blank)

Choose the correct alternative to replace the blank

- a) $q = \text{NULL} ; p \rightarrow \text{next} = \text{head} ; \text{head} = p ;$
- b) $q \rightarrow \text{next} = \text{NULL} ; \text{head} = p ; p \rightarrow \text{next} = \text{head} ;$
- c) $\text{head} = p ; p \rightarrow \text{next} = q ; q \rightarrow \text{next} = \text{NULL} ;$
- d) $q \rightarrow \text{next} = \text{NULL} ; p \rightarrow \text{next} = \text{head} ; \text{head} = p ;$

→



$$\begin{aligned}
 & p \rightarrow \text{next} = \text{head} \\
 & \text{head} = p \\
 & q \rightarrow \text{next} = \text{NULL}
 \end{aligned}$$

• printing the elements of single linkedlist using recursion

(1) `void f(struct node *p)`
{

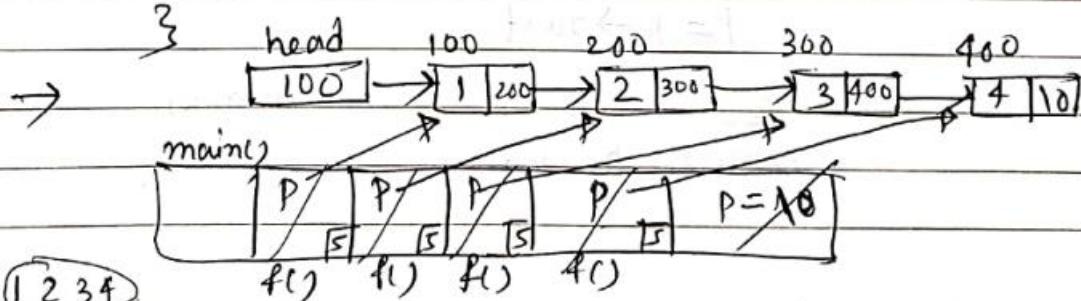
 if (p)

 { printf("%d", p->data);

 f(p->link);

 }

}



(2)

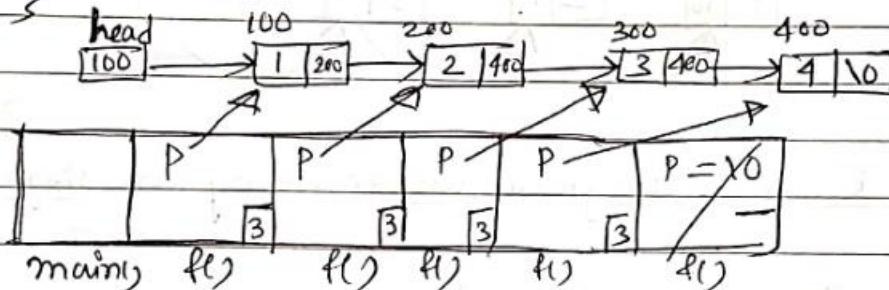
```
void f(struct node *P)
```

```
{  
    if (P)
```

```
{  
    f(P->link);
```

```
    printf ("%d", P->data);
```

```
}
```



O/p: 4 3 2 1

- Reversing a single linked list using iteration program

① Struct node

```
{
```

```
int i;
```

```
struct node *next;
```

```
};
```

```
struct node *reverse (struct node *cur)
```

```
{
```

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

```
while (cur) {
```

```
    nextNode = cur->next;
```

```
    cur->next = prev;
```

```
    prev = cur;
```

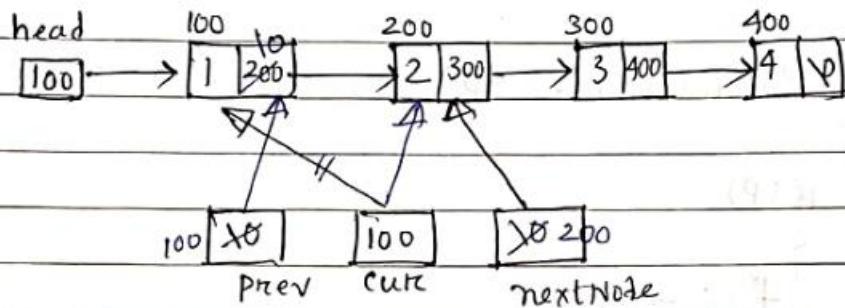
```
    cur = nextNode;
```

```
return prev; }
```

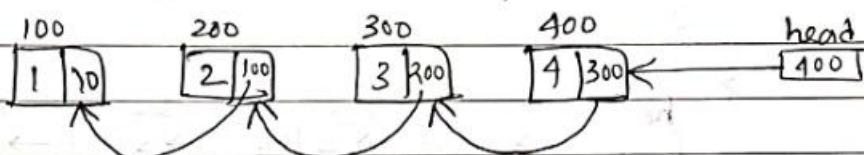
main() {

head = reverse(head);

? }



after operation



- Recursive program for reversing a single linked list =

struct node *head;

void reverse (struct node * prev, struct node * cur)

{
 ① if (cur)
 {

 ② reverse (cur, cur->link);
 ③ cur->link = prev;
 }

else

 head = prev;

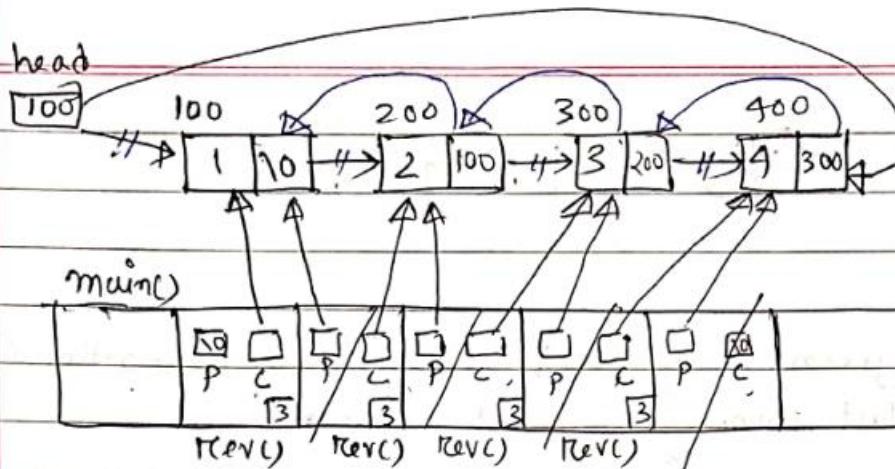
} }

void main()

{
 :

 reverse (NULL, head);

}



Q-2003)

Question -3

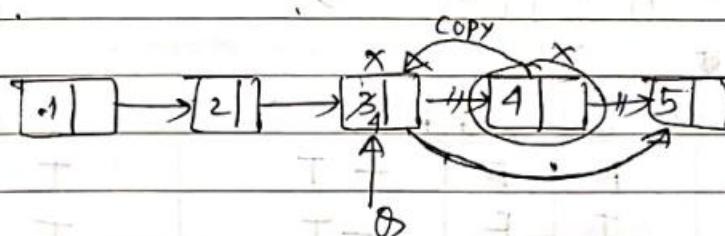
In the worst case, the number of comparisons needed to search a single linked list of length ' n ' for a given element is -

- (a) $\log_2 n$ (b) $n/2$ (c) $\log_2 n - 1$ (d) n .

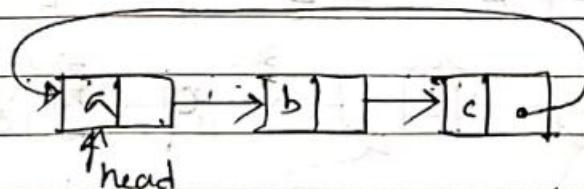
Q-2004)

Question -4

Let 'p' be a single linked list, let 'q' be a pointer to an intermediate node 'x' in the list. What is the worst case time complexity of the best known algorithm to delete the node 'x' from the list?

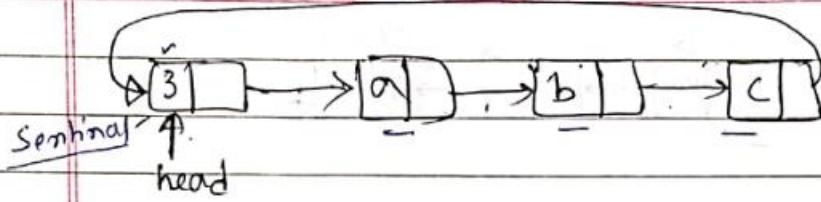
 $\rightarrow O(1)$.

- Circular linked list =



$$p = \text{head};$$

while ($p \rightarrow \text{next} \neq \text{head}$) \rightarrow to find the end of the list.



Adv → given any point we can search entire list.
→ last pointers are not wasted.

^{Ques 5}
[Question-5] (Checking if the list is in non decreasing)

Struct item {

int data;

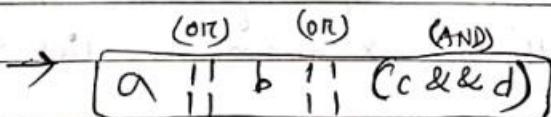
struct item *next;

};

int f(struct item *p)

{

return ((p == NULL) || (p->next == NULL) || (p->data <= p->next->data) && f(p->next)); }



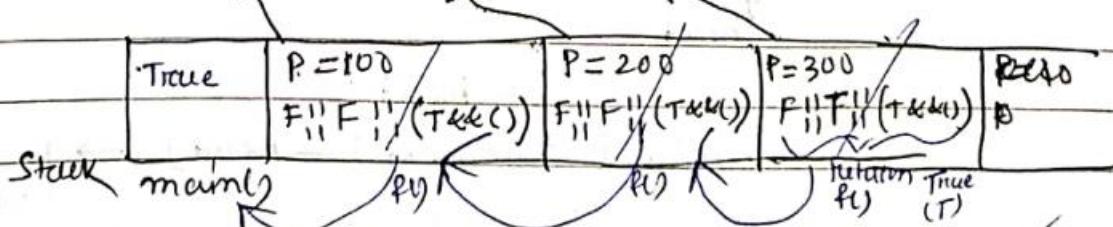
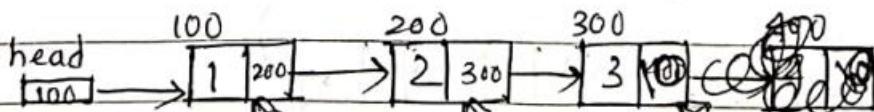
Q

a or b or (c and d)

T	F	F	-T
T	T	F	-T
T	T	T	-T
F	F	F	-F

c and d

T	T	-T
T	F	-F
F	T	-F
F	F	-F

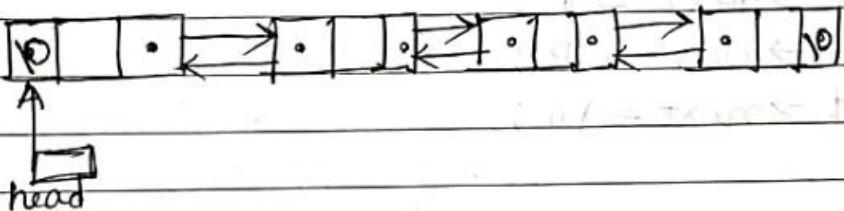


- Insertion into doubly linked list :-

Struct node

{

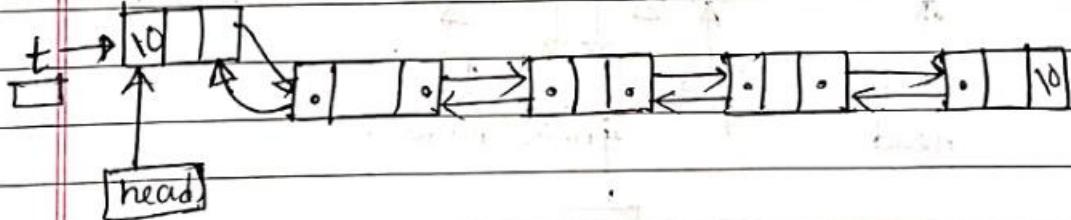
```
int i;
struct node *prev;
struct node *next;
};
```



beginning

- Insert a node at front :-

new node



struct node *t;

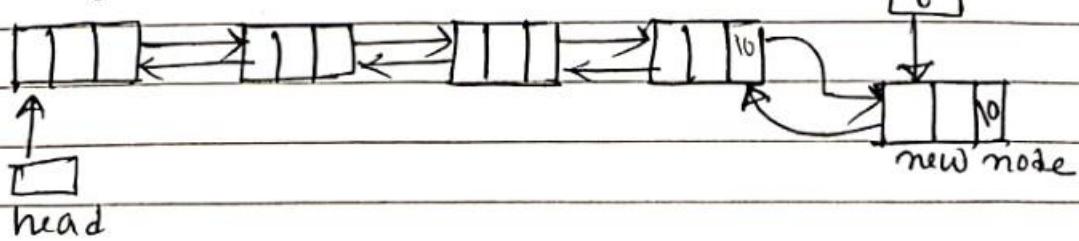
$t \rightarrow \text{next} = \text{head};$

$\text{head} = t;$

$t \rightarrow \text{prev} = \text{NULL};$

$\text{head} \rightarrow \text{next} \rightarrow \text{prev} = \text{head} \rightarrow \text{NULL};$

- Inserting at the end :-



~~while {~~

struct node *p;

p = head;

while (p → next)

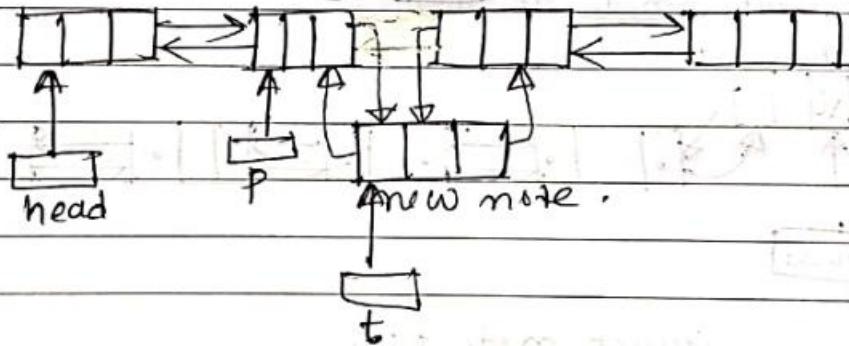
{ p = p → next; }

p → next = t;

t → prev = p;

t → next = 10;

- Inserting the node at intermediate :-



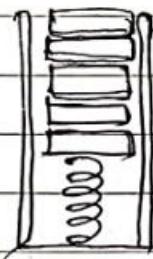
struct node *t;

(hold the link
and then modify)

{ t → prev = p;
t → next = p → next;
p → next = t;
p → next → prev = t;

Stacks and Queues

- Introduction to stacks :



→ push the plate from the top and take the plate from the top.

(stack) (FIFO, LIFO)

→ (pop)

- Applications of stack :

(i) Recursion.

(ii) IF → PF conversion.

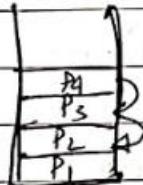
(iii) Parsing

(iv) Browsing.

(v) Editors (like)

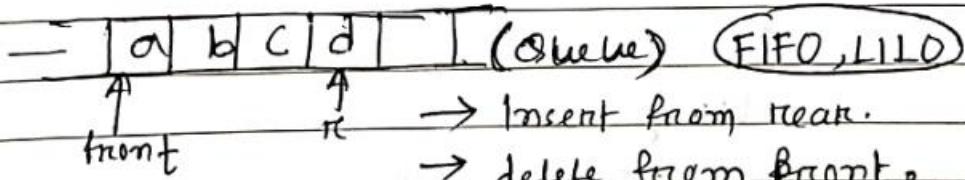
(vi) Tree traversals

and graph
traversals.



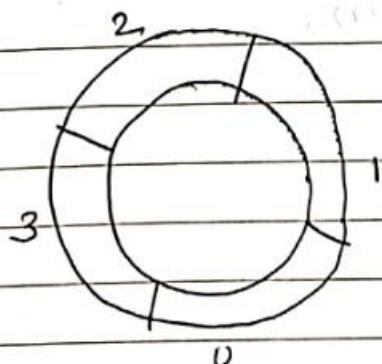
- Implementation of a Queue using circular array =

→ Insert the element from one side and delete the element from other side.



→ Insert from rear.

→ delete from front.

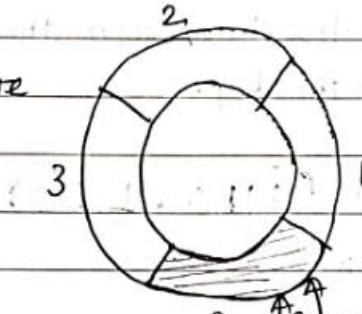


circular queue.

If \rightarrow in circular queue, if the size of an array ' n ' then we use ' $(n-1)$ '. amount of size.



~~one less~~



circular queue.

\rightarrow Should a blank space be

① Inserting an item in queue =

enqueue(item)

{

rear = (rear+1) mod n;

if (front == rear)

{ pf("Q is full");

if (rear == 0) rear = n - 1;

else

else rear = rear - 1;

return;

}

else {

q[rear] = item;

return;

}

}

② delete an item from queue

```

int Dequeue()
{
    if (front == rear)
    {
        pf ("Q is empty");
        return -1;
    }
    else
    {
        front = (front + 1) % n;
        item = q[front];
        return item;
    }
}

```

Over flow :
 $(\text{rear} + 1) \% n$
 $= \text{front}$

Under flow
 $\text{front} == \text{rear}$

• Linked List Implementation of stack

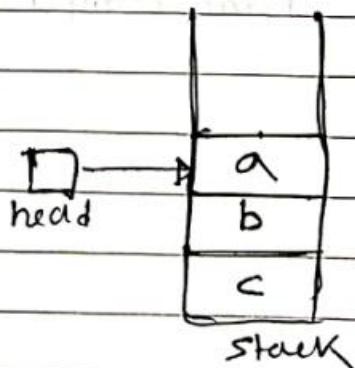
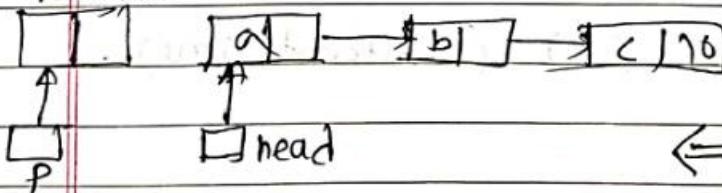
struct node

```

{
    int i;
    struct node *link;
}

```

new node



① Inserting a new item =

`push(int item)`

{

```
struct node *p = (struct node*) malloc(sizeof(struct node));
if (p == NULL) { pf("error of malloc"); return; }
```

`p->data = item;`

`p->link = head; NULL`

`p->link = head;`

`head = p;`

}

Time complexity = O(1) (constant time)

② deletion a item from stack =

Deleted code

`int pop()`

{ struct node *p;

`int item;`

```
if (head == NULL) { pf("Underflow"); return -1; }
```

`item = head->data;`

`p = head;`

`head = head->next;`

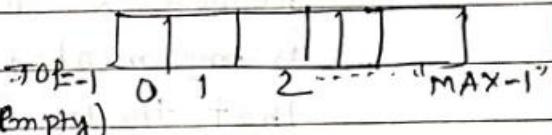
`free(p);`

Time complexity = O(1) (constant time)..

- Implementation of stack using array =

```
int stack[MAX];
```

```
int top = -1; // (stack empty)
```



- Pushing a item =

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

```
{
```

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

```
pf(" overflow");
```

```
else {
```

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

```
stack[top] = item; }
```

```
}
```

```
return;
```

```
}
```

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

- Popping a item =

```
int pop()
```

```
{ int temp;
```

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

```
{ pf(" underflow");
```

```
return -1;
```

```
}
```

```
else
```

```
{
```

```
temp = stack[top]; }
```

```
top = top - 1;
```

```
}
```

```
return temp;
```

```
}
```

```
temp = stack[top - 1]; }
```

→ Time complexity (push, pop) = O(1) Time.

(n-2) 2012

Question -1]

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 using 'REAR' and 'FRONT' as array index variables respectively. Initially, $\text{REAR} = \text{FRONT} = 0$. The conditions to detect queue with full and queue empty are =

→ Queue Full:

$$\text{FRONT} \oplus (\text{REAR} + 1) \bmod n == \text{FRONT}$$

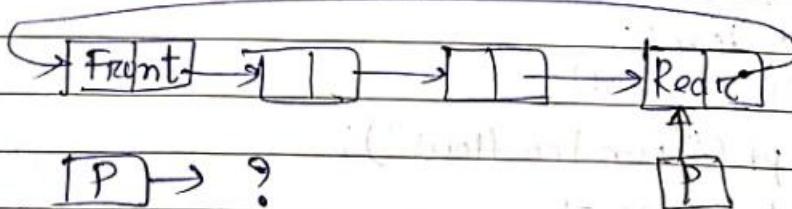
Queue Empty:

$$\text{REAR} = \text{FRONT}$$

(n-2) 2014

Question -2]

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 to such that both the operations enqueue and dequeue can be performed in constant time?



- a) Read mode
- b) FRONT node
- c) not possible.
- d) node next to Front.

→

Question-3

Which of the following permutations can be obtained in the o/p (In the same order) using a stack assuming that the i/p is the sequence.

1, 2, 3, 4, 5 in the order?

a) 3, 4, 5, 1, 2. ✓ b) 3, 4, 5, 2, 1.

c) 1, 5, 2, 3, 4. ✓ d) 5, 4, 3, 2, 1, 2.

→ a)

O/P	5	I/P	1
	4		2
	3		3
3, 4, 5, ..	2	1, 2, 3, 4, 5	5
	1		

(X)

✓ b)

O/P	5
	4
	3
3, 4, 5, 2, 1	2
	1

✗ c)

O/P	5
	4
	3
1, 5, ..	2
	X

✗ d)

O/P	5
	4
	3
5, 4, 3,	2
	1

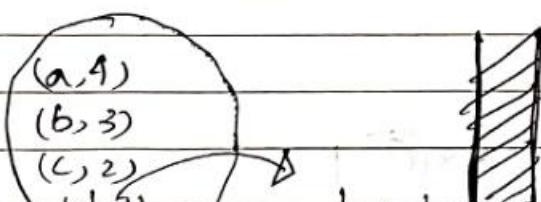
GATE-1997

[Question - 4]

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 an appropriate integer chosen by the implementation. pop is implemented as $\text{DELETE MIN}(Q)$. For a sequence of push operations, the keys chosen are in:

- a) Non-decreasing order.
- b) Non-increasing order.
- c) strictly increasing order.
- d) strictly decreasing order.

→



[a] b] c] d]

GATE-2003

[Question - 5]

Let 'S' be a stack of size $n \geq 1$. Starting with an empty stack, suppose we push the first n natural numbers in sequence and then perform n pop operations. Assume that push and 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 elapsed from end of $\text{push}(m)$ to the start of the pop operation that removes ' m ' from S. The average stack-life of an element of this stack is:

→

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array} \rightarrow y \quad \begin{array}{|c|c|} \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow y$$

$$\begin{array}{|c|c|} \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow y + x + y + x + y$$

$$= 2x + 2y + y$$

$$\begin{array}{|c|c|} \hline 3 & \\ \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow y$$

$$\begin{array}{|c|c|} \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow 2x + 2y + y$$

$$\begin{array}{|c|c|} \hline 1 & \\ \hline \end{array} \rightarrow 1x + 1y + y$$

$$\begin{array}{|c|c|} \hline 4 & \\ \hline 3 & \\ \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow y$$

$$\begin{array}{|c|c|} \hline 3 & \\ \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow 2x + 2y + y$$

$$\begin{array}{|c|c|} \hline 2 & \\ \hline 1 & \\ \hline \end{array} \rightarrow 4x + 4y + y$$

$$\begin{array}{|c|c|} \hline 1 & \\ \hline \end{array} \rightarrow 6x + 6y + y$$

for n elements,

$$\rightarrow 2(n-1)x + 2(n-1)y + ny$$

average life time,

$$2 * \left(\sum_{i=1}^{n-1} i \right) x + 2 \left(\sum_{i=1}^{n-1} i \right) y + ny$$

$$2 * \frac{(n-1)(n-1+1)}{2} * x + 2 * \frac{(n-1)(n-1+1)}{2} * y + ny$$

$$\Rightarrow \frac{n(n-1)x + n(n-1)y + ny}{n}$$

$$\Rightarrow (n-1)x + (n-1)y + y$$

$$\Rightarrow (n-1)(x+y) + y$$

$$\Rightarrow n(x+y) - x - y + y$$

$$\Rightarrow [n(x+y) - x]$$

16-2006

Question - 6

Queue is implemented using two stacks S_1 and S_2 as given below:

```
void insert(&S; x) {
    push(S1, x);
}
```

```
void delete(&S) {
    if (stack-empty (S2)) then
        if (stack-empty (S1)) then {
            printf ("S is empty");
            return;
        }
}
```

```
else while (! (stack-empty (S1))) {
```

```
    x = pop(S1);

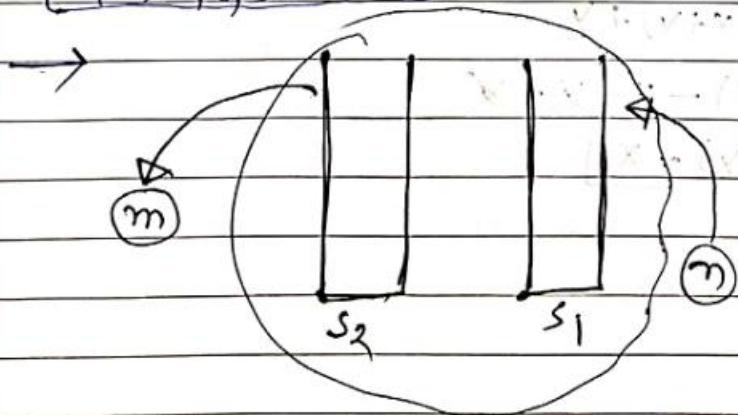
```

```
    push(S2, x);
}
```

```
x = pop(S2);
}
```

$n \rightarrow$ Inserts
 $m (\leq n) \rightarrow$ deletes
 $x \rightarrow$ pushes
 $y \rightarrow$ pops

What is relation b/w all this?



In Best case :

$$\text{PUSH} : 2m + (n-m) \\ = m+n$$

$$\text{POP} : 2m$$

Worst case :

$$\text{PUSH} : 2m$$

$$\text{POP} : 2(m+n)$$

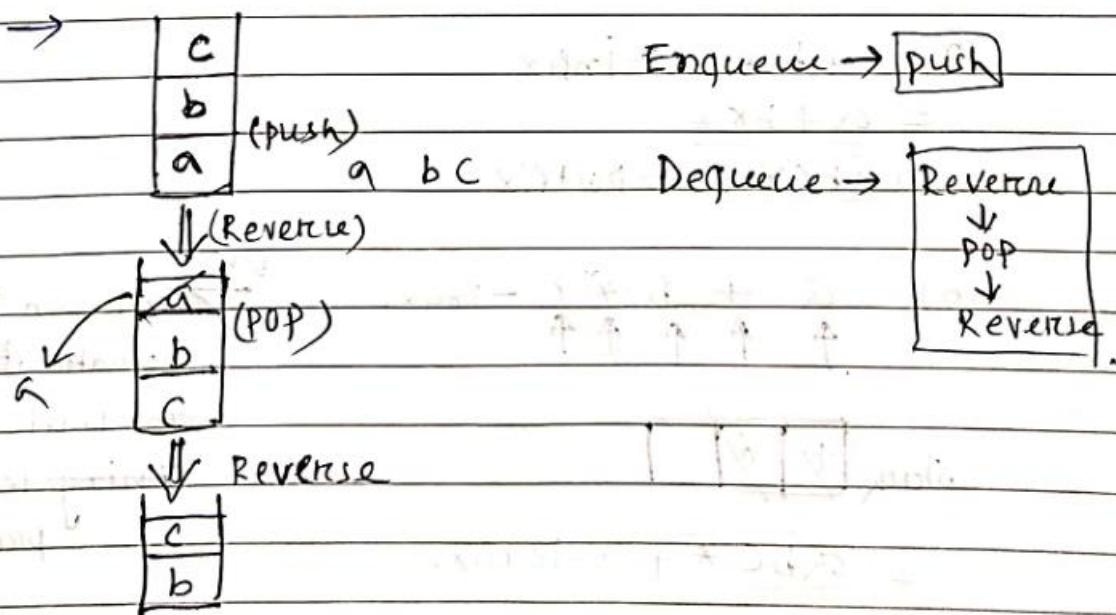
$$m+n \leq x \leq 2m \\ 2m \leq y \leq n+m$$

gate-2000

Question - 7

Suppose a stack implementation supports, in addition to push, and pop, an operation "REVERSE" 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 3 operations.



Infix \rightarrow postfix

- Infix to postfix Conversion Algorithm :- (operator stack)

→ postfix related anything stack is required.

example:

$$\textcircled{1} \quad a+b \rightarrow \text{Infix expression}$$

$$ab+ \rightarrow \text{postfix expression.}$$

$$\textcircled{2} \quad a+b*c \rightarrow \text{Infix exp}$$

$$= \underline{a+b*c}$$

$$= abc*+ \rightarrow \text{postfix expression.}$$

$$\textcircled{3} \quad a+b-c$$

$$= \underline{ab+-c}$$

$$= \boxed{ab+c-}$$

+ < . (
(< . +)

$$\textcircled{4} \quad a+(b-c) - \text{Infix}$$

$$= \underline{a+bc-}$$

$$= abc-+ - \text{postfix.}$$

$$\textcircled{5} \quad a+b*c - \text{infix}$$

$$= \underline{a+b*c*}$$

$$= \boxed{abc*+} - \text{postfix}$$

$$\textcircled{6} \quad a + b \times c - \text{infix.}$$

↑ ↑ ↑ ↑ ↑ ↑

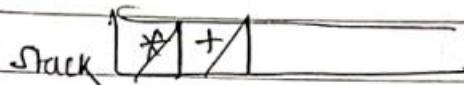
→ you can push an operator, only when the inside one is having lower lesser precedence.

Stack

+	*		
---	---	--	--

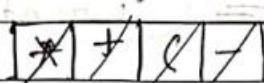
$$= \boxed{abc\times+} - \text{Postfix.}$$

⑦ $a * b + c$ — Infix expression.



$a b * c +$ — postfix expression.

⑧ $a * b + (c - d)$ — Infix



$a b * c d - +$ — postfix expression.

Algorithm

a) Create a stack.

b) For each character 't' in the i/p.

{
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 o/p)

else // 't' is an operator or left parenthesis

{
 Pop and o/p tokens until one of lower priority than 't' is encountered or a left parenthesis is encountered or the stack is empty.
 3 push t

c) pop and o/p all the tokens until the stack is empty.

space
Time complexity — $O(n)$
Data structure used — stack
On push only — operators.

- Postfix evaluation algorithm = Operand stack

~~3+2*1~~ — Infix

3+21*

3 2 1 * + — postfix
 $\uparrow \uparrow \uparrow \uparrow$

3 1 2 | 1 2 | 5 |

(2 * 1)
 op_1

(3 + 2)
 op_2

push — only operand.

Algorithm for postfix evaluation =

1. Scan the postfix string from left to right.
2. Initialize an empty stack.
3. Repeat steps 4 and 5 till all the characters are scanned.
4. If the scanned character is an operand, push it onto the stack.
5. If the scanned character is an operator, and if the 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 from the stack apply the operator to those popped elements. push the result on to the stack.

- 6) After all the elements are scanned, the result will be in the stack.

1

TREES

classmate

Date _____

Page _____

• Introduction to tree traversals:

We want to see
• Traversing means → what is information present in
Every node.

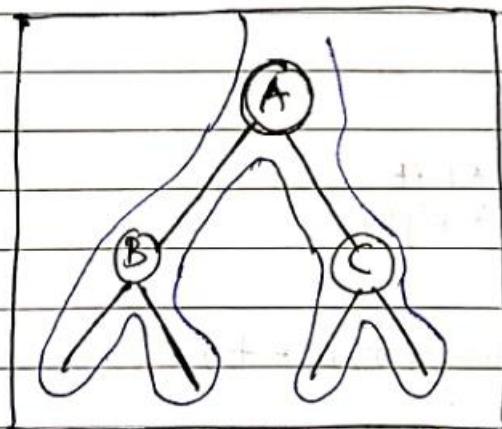
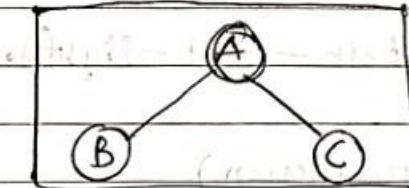
• Searching means → searching for a particular node.

Methods of traversing

2nd time → Inorder traversal (Left - Root - Right) - BAC

1st time → preorder traversal (Root - Left - Right) - ABC

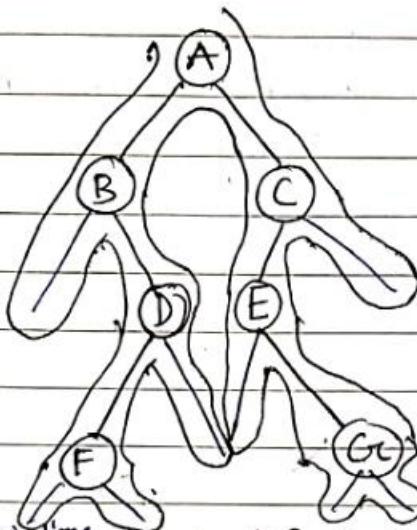
3rd time → postorder traversal (Left - Right - Root) - BCA



1st time - ABC - (pre)

2nd time - BAC - (Inorder)

3rd time - BCA - (post)

example :

- Inorder (2nd time) → A B F D A E G C
- post pre order (1st time) → A B D F G E C
- post order (3rd time) → F D B G E C A

- Implementation of traversals and time and space analysis

Inorder traversing → (left — root — Right).

program : (using recursion)

```

struct node
{
    int data;
    struct node *Left;
    struct node *Right;
}
  
```

void Inorder (struct node * t)

{

if (t!=NULL)

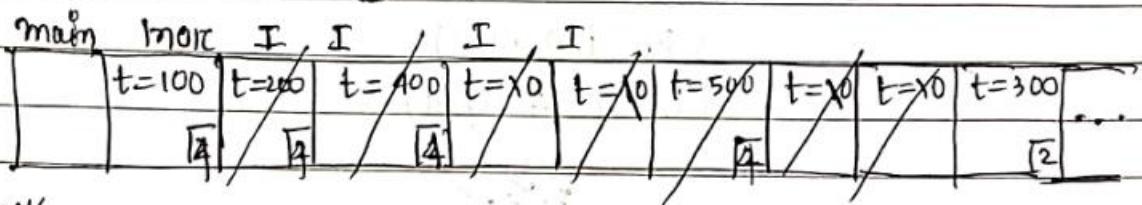
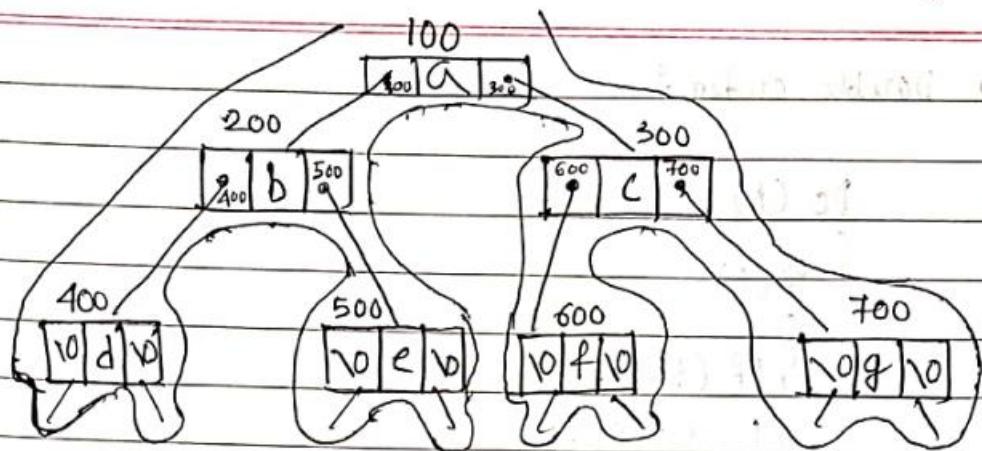
{

1. Inorder (@to t→ left.);

2. printf ("%c", t→ data);

3. Inorder (t→ Right);

}



O/P: abeafcg

In/pre/post order traversal all take
Time and space complexity = $O(n)$.

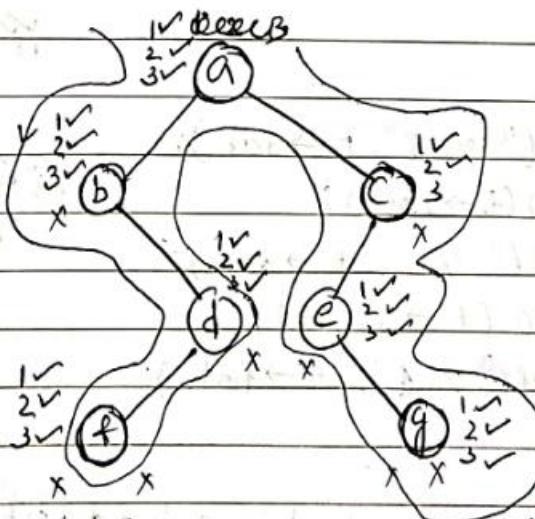
{ Inorder \rightarrow 1, 2, 3 (lines) }
 { pre order \rightarrow 2, 1, 3 }
 { post order \rightarrow 1, 3, 2 }

• Double order traversal:

(one other method)

example: (Inorder)

(root as input)



INORDER(t)

{

1. INORDER($t \rightarrow \text{left}$)

2. Pf

3. INORDER($t \rightarrow \text{right}$)

}

O/P: b f d a e g c

- Double orders:

Do(t)

{ PF(t)

{ PF(t → data)

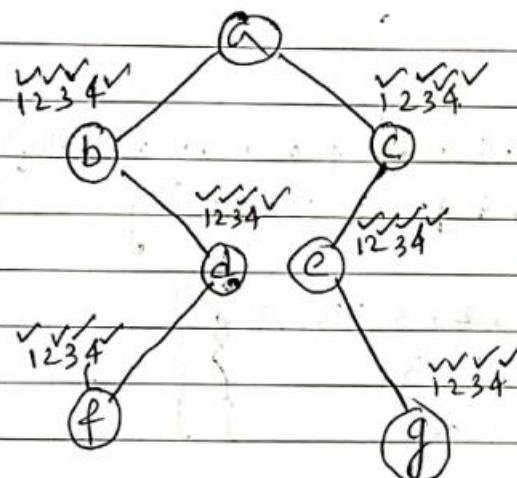
2 Do(t → left)

3 PF(t → data)

4 Do(t → right)

}

1 2 3 4



Output:

a b b d f f d a c e e g g e

- Triple order traversal:

void TO (struct node *t) ⁽¹⁰⁰⁾

{

if (t)

{ PF(" %o %o ", t → data);

2. TO(t → left);

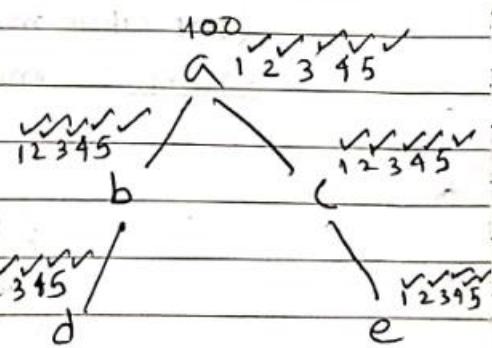
3. PF(" %o %o ", t → data);

4. TO(t → right);

5. PF(" %o %o ", t → data);

}

}



Output:

a b d d d b b c c e e e c a

- Indirect recursion on trees:-

```
void A (struct Node *t)
{
```

```
    if(t)
```

```
        {1. pf("odd", t->data);
```

```
        2. B (t->left);
```

```
        3. B (t->right);
```

```
}
```

```
void B (struct node *t)
```

```
{
```

```
    if(t)
```

```
{
```

```
    1. A (t->left);
```

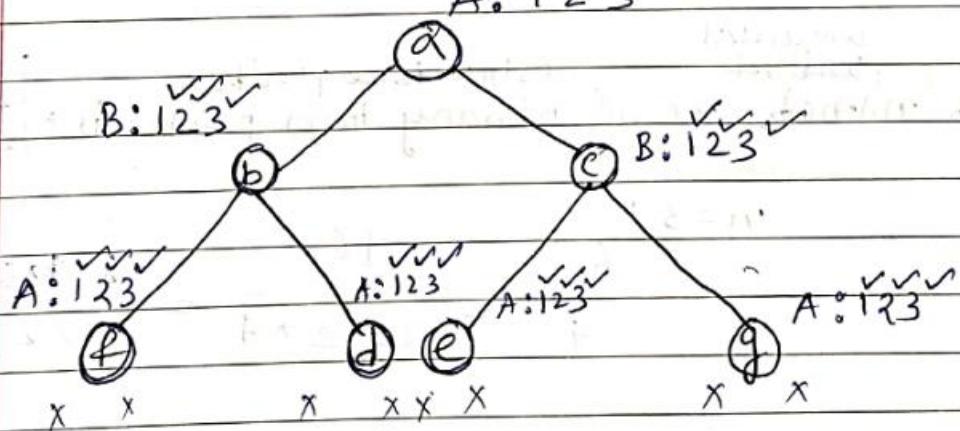
```
    2. pf("odd", t->data);
```

```
    3. A (t->right);
```

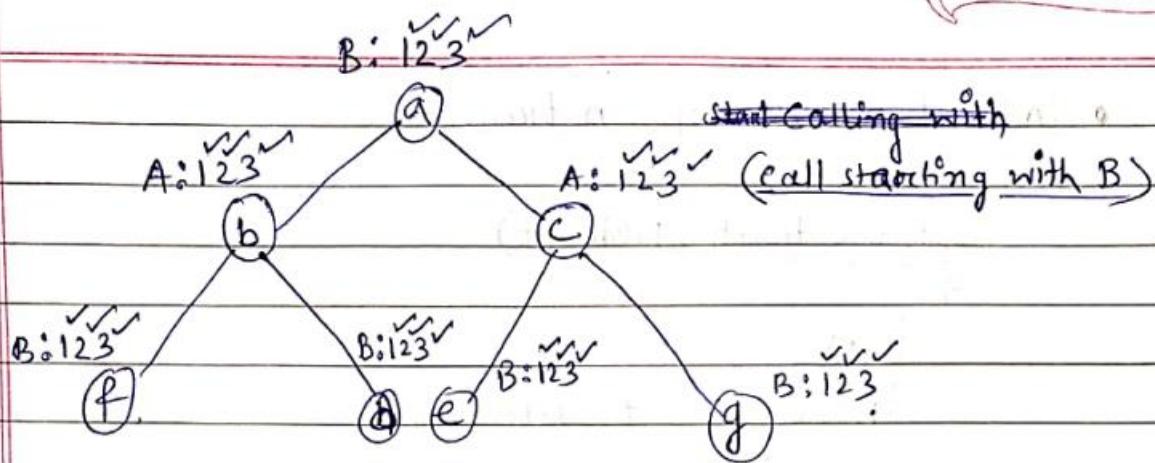
```
}
```

```
}
```

(call starting from with A)



af b d e c g — output.



bfdac eg → output.

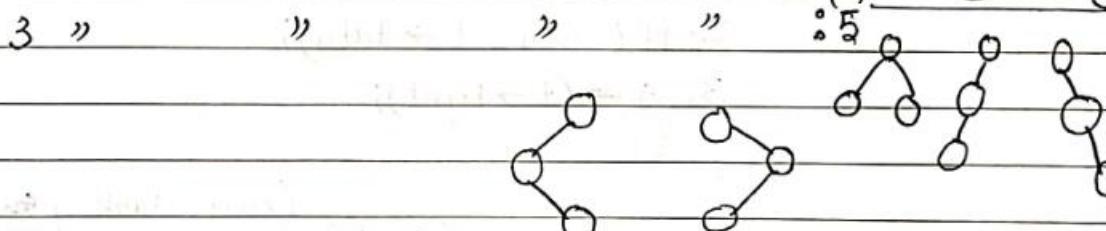
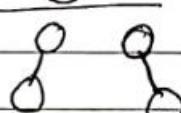
(structures)

- Number of Binary trees possible:

• Number of Binary tree possible with unlabelled node =

1 node no. of binary trees possible is : one

2 " " " " " " : two

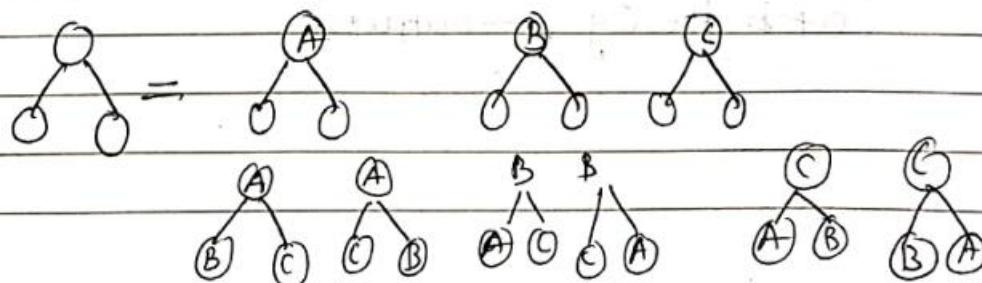


with m unlabelled node no. of Binary trees possible is ; $\frac{2^n C_n}{(n+1)}$ (structure possible)

n=3

$$\frac{6 C_3}{4} = \frac{16}{[3][3 * 4]} = \frac{8 * 5 * 4}{8 * 2 * 4} = 5$$

- Number of Binary tree possible with labelled node :



With 'n' labelled nodes no. of binary tree possible are =

$$= \frac{2^n C_n * n!}{(n+1)}$$

meas

Question type =

{How many node possible?}

Question:

All the Binary Trees with 3 nodes A, B and C which have preorder ABC (1st time) How many binary tree possible?



With 3 unlabelled node trees possible : 5

,, 3 labelled ,,, ,,, $\therefore 5 \times 1^3 = \$30.$

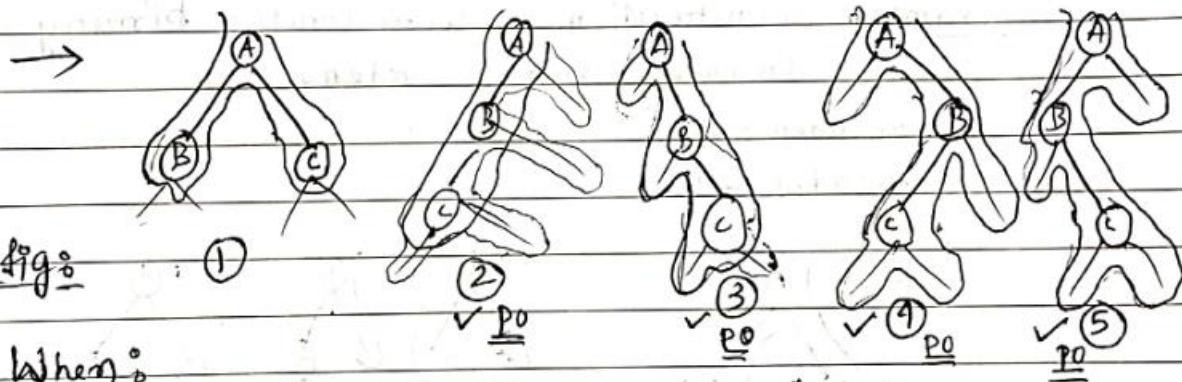


fig:

When:

preorder "ABC" & postorder is "CBA" =
(3rd time)

fig: ② ③ ④ ⑤ (4-trees possible)

when: more pre "ABC" & post "CBA" & inorder "BCA" =
(1) (3) (2nd time visit)

fig: ① ⑤ (only - one tree possible)

(hence one tree that satisfy all the condition)

→ no. of tree possible with node
given {n} nodes

{ Preorder - }

$$\text{no. of trees possible is } = \frac{2^m C_n}{(m+1)}$$

→ with given Preorder and Postorder you may get more than '1' binary tree.

* → with given Pre, Post, and Inorder we always get only 'one' binary tree.

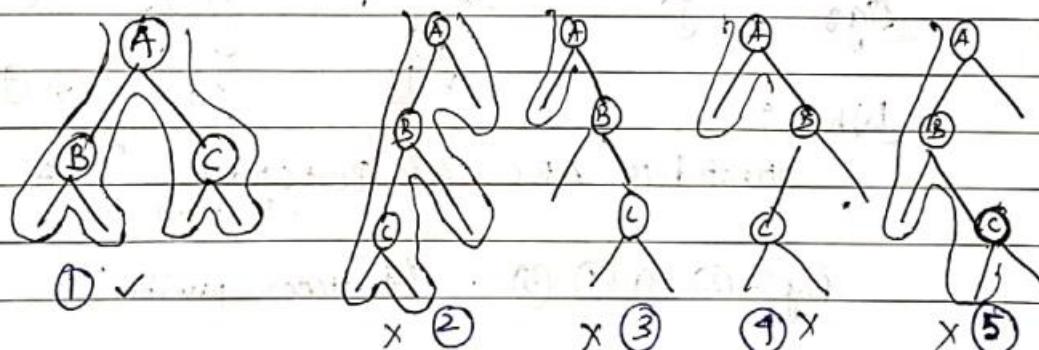
(unique)
unique

example: Construction of certain unique binary tree using Inorder and Preorder.

(1st) Preorder = ABC

(2nd) Inorder = BAC

→



→ 5 binary tree possible with Preorder: ① ② ③ ④ ⑤ .

→ ~~Ø~~ 1 BTS possible with (In and Pre both orders) fig - ① .

IN + PRE
IN + POST

possible
POST + PRE → not possible to find IN.

classmate

Date _____

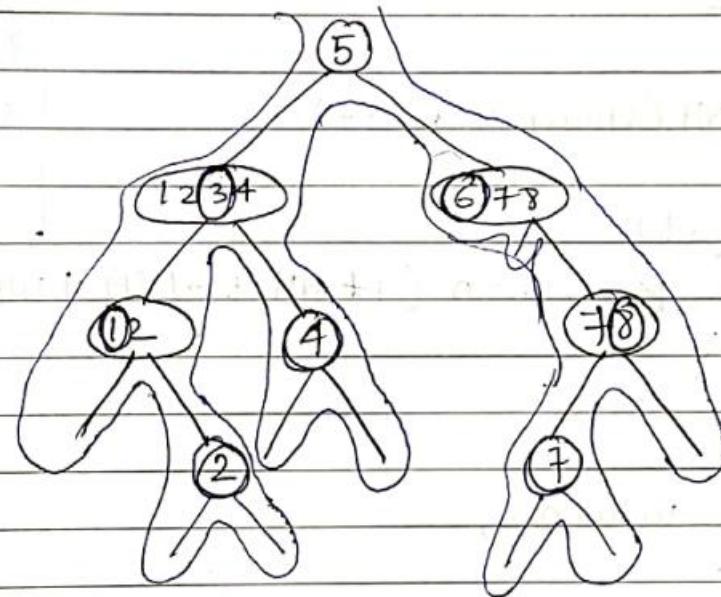
Page _____

Question:INORDER: 1, 2, 3, 4, 5, 6, 7, 8.PREORDER: 5, 3, 1, 2, 4, 6, 8, 7.

POSTORDER: ?

IN - L Root RPre - Root L R

+ 2 3 4 5 6 7 8



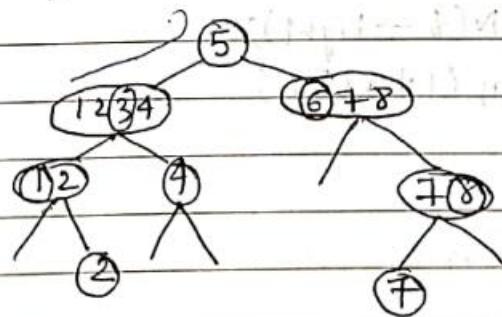
POSTORDER: 2 1 4 3 7 8 6 5

Question:

INORDER: 1 2 3 4 5 6 7 8 (L Root R)

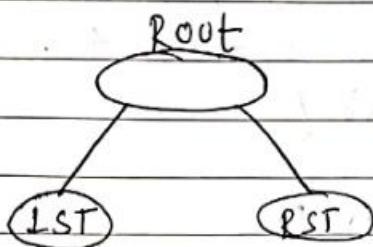
POST: 2 1 4 3 7 8 6 5 (L R Root) ←

PRE: ?



PRE: 5 3 1 2 4 6, 8 7.

- Recursive program to count the number of nodes



$$\text{NN}(T) = 1 + \text{NN}(\text{LST}) + \text{NN}(\text{RST})$$

= 0; if T is NULL.

Program:-

```
int NN (struct node *t)
```

{

if (t)

{ ~~return~~ return (1 + NN($t \rightarrow \text{left}$) + NN($t \rightarrow \text{right}$));

}

else

return 0;

}

or

```
int NN (struct node *t)
```

{

if (t)

int l, R;

1. $l = \text{NN} (t \rightarrow \text{left});$

2. $R = \text{NN} (t \rightarrow \text{right});$

3. $\text{return} (1 + l + R);$

}

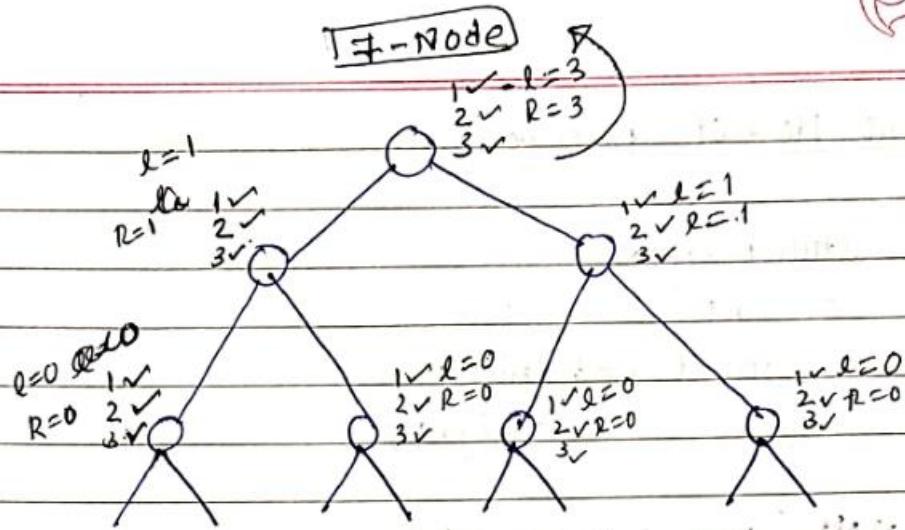
else

return 0;

}

struct node
{ int i;

struct node *
 *left;
 *right;



- Recursive program to count the no. of leaves and non-leaves =

- Count No. of leaves :

```
struct node {
```

```
    int i;
```

```
    struct node *left;
```

```
}
```

```
*right;
```

```
int NL(struct node *t)
```

```
{
```

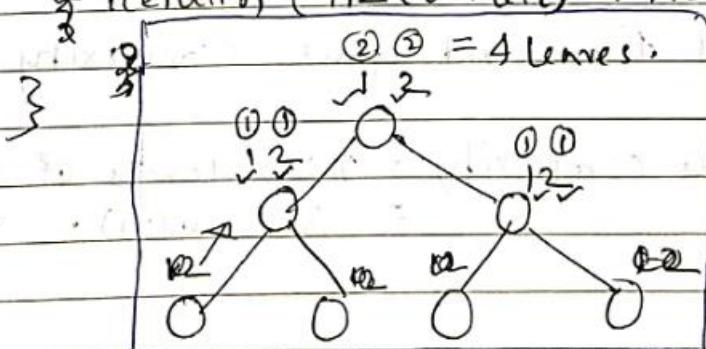
```
    if (t == NULL) return 0;
```

```
    if (t->left == NULL && t->right == NULL)
```

```
        return 1;
```

```
    else
```

```
        return (NL(t->left) + NL(t->right));
```



• Count No. of non leaves :

struct node

{ int i;

struct node *left,

};

*right; }

int NNL(struct node *t)

{

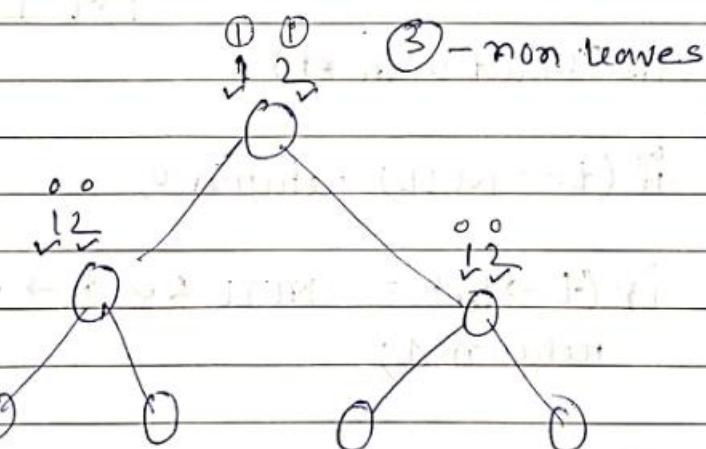
if ($t == \text{NULL}$) return 0;

if ($t \rightarrow \text{left} == \text{NULL} \& \&$
 $t \rightarrow \text{right} == \text{NULL}$)
 return 0;

else

return (1 + NNL($t \rightarrow \text{left}$) + NNL($t \rightarrow \text{right}$));

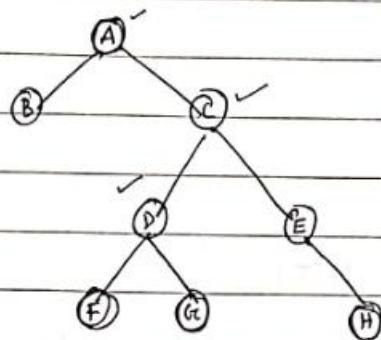
}



→ Total time and space complexity = $O(n)$.

→ Space complexity = no. of levels of tree.
 $= 3 = O(n)$.

- Recursive program to find the full nodes =



Total Full node : 3 (A,C,D)

Total node : 4 (A,C,D,E).

$$\begin{aligned}
 \text{return } & \\
 \text{FN}(T) = 0 & ; T = \text{NULL} \\
 = 0 & ; T \text{ is a leaf} \\
 = \text{FN}(T \rightarrow \text{LST}) + \text{FN}(T \rightarrow \text{RST}) & ; \text{if } T \text{ has only one child} \\
 = \text{FN}(T \rightarrow \text{LST}) + \text{FN}(T \rightarrow \text{RST}) + 1 & ; \text{if } T \text{ is a full node}.
 \end{aligned}$$

Program :

int FN (struct node *t)

{

 if ($\nexists t$) return 0;
 if ($\nexists t \rightarrow \text{left} \& \& \nexists t \rightarrow \text{right}$)
 return 0;

 return ($\text{FN}(t \rightarrow \text{left}) + \text{FN}(t \rightarrow \text{right}) + (\nexists t \rightarrow \text{left} \& \& \nexists t \rightarrow \text{right})$)

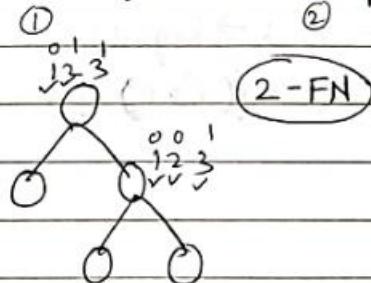
}

①

②

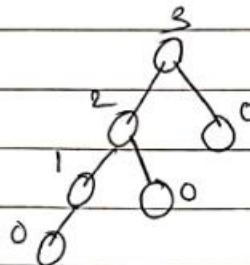
③

? 1 : 0;



→ Total Time complexity is = $O(n)$.
 && space

- Recursive program to find the height of a tree :-



Recursive equation, $H(T) = \begin{cases} 0, & T \text{ is empty.} \\ 0, & T \text{ is leaf.} \\ 1 + \max(H(LST), H(RST)), & \text{otherwise.} \end{cases}$

Program:

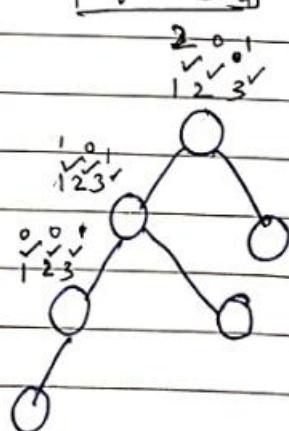
```
struct node
{
    int i;
    struct node *left;
    * Right;
}
```

```
int struct node *t;
int H(struct node *t)
{
    int l, r;
    if (!t) return 0;
    if ((!(t->left)) && (!(t->right)))
        return 0;
```

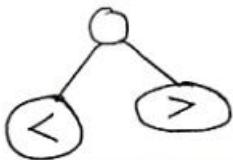
Height = 3

for

1. $l = H(t \rightarrow \text{left})$
2. $r = H(t \rightarrow \text{right})$
3. $\text{return } (1 + ((l > r) ? l : r))$



→ Time and space complexity is $O(n)$.

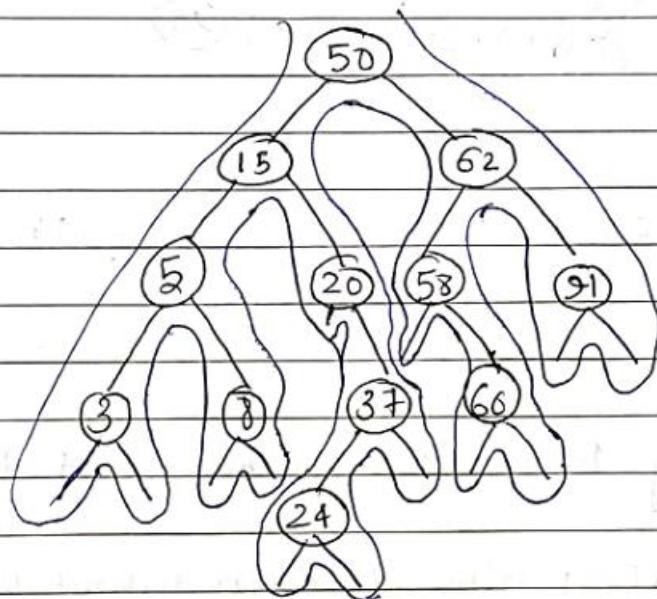


- Introduction of BST (Binary searchTree) :-

→ Binary search tree are a special kind of Binary tree. → Inorder is sorted order. (adv).

g-¹⁹⁹⁶ [Question] - ①

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



→ The Inorder traversal of a BST is going to give sorted order of element.

Inorder: 3, 5, 8, 15, 20, 24, 37, 50, 58, 60, 62, 91.
(2nd time) (ascending order)

g-²⁰⁰⁵ [Question] - ②

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

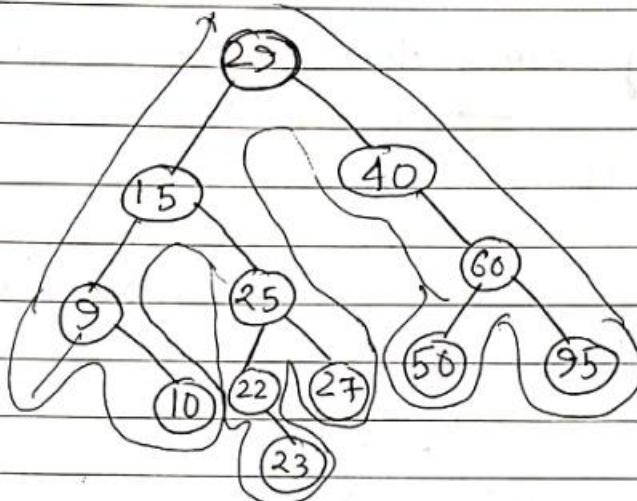
Inorder: ?

→ Inorder: 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95.
(ascending order)

then find post pre order: ?

Root

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



Pre order : [29, 15, 9 , 10, 25, 22, 23, 27 , 40, 60, 50, 95]

^{g-2005} Question - ③

How many BSTs are possible with 4 distinct keys.

value

→ No. of BST possible with n distinct keys are,
→ (same as Binary Unlabelled tree).

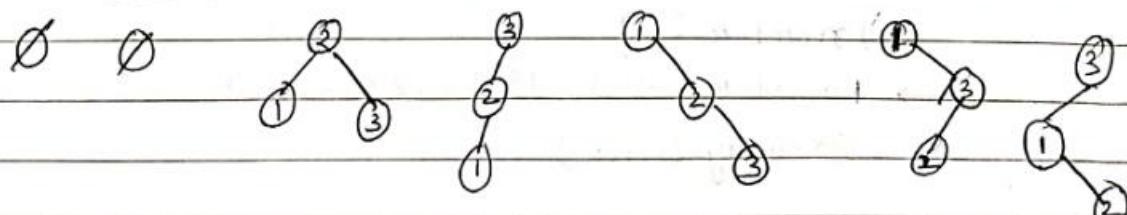
$$\frac{2n C_n}{n+1}$$

$$\frac{8 C_4}{4+1} = 14 \cdot 13 \cdot 5$$

$$\begin{aligned} &= 8 \times 7 \times 6 \times 5 \\ &= 4 \times 3 \times 2 \times 5 \end{aligned}$$

= 14 (BST) possible.

∅ BSTs 3 distinct keys = 1 2 3

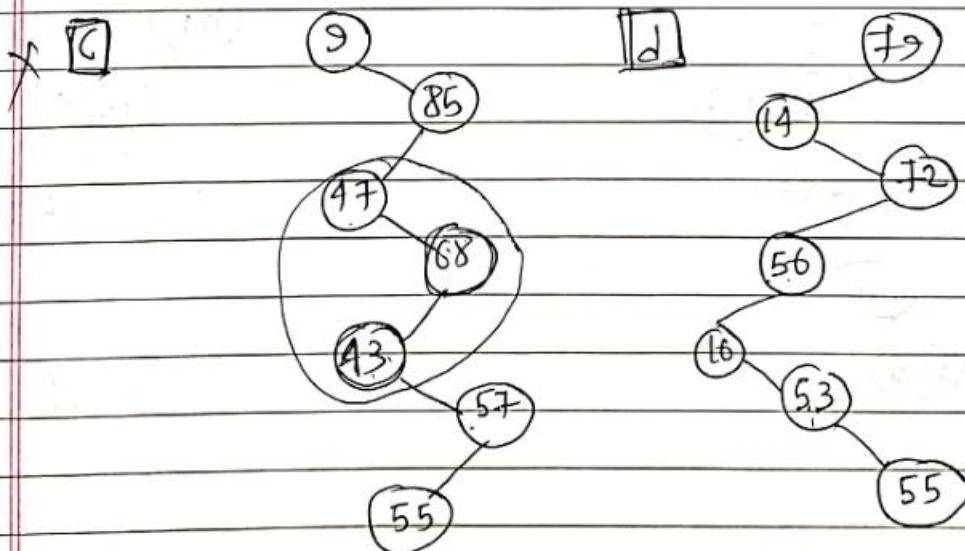
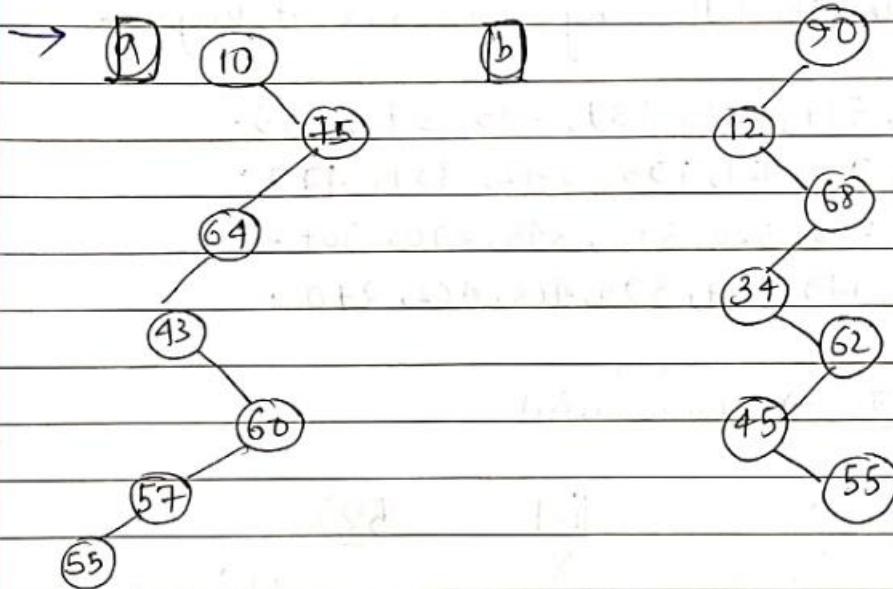


9-2006

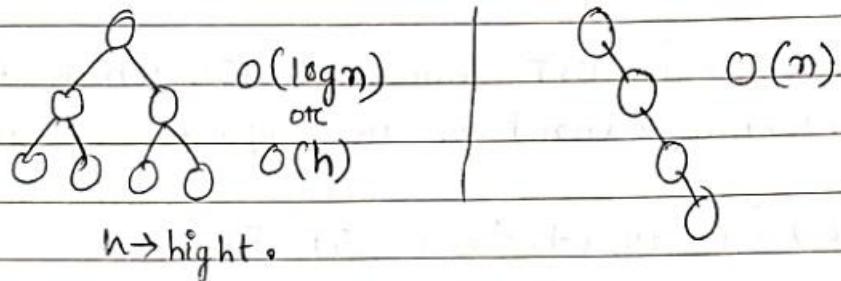
Question - 4

(1 - 100) in BST search for 55 which of the following sequences cannot be the sequence of nodes examined?

- a) {10, 75, 64, 43, 60, 57, 55}
- b) {90, 12, 68, 34, 62, 15, 55}
- c) {9, 85, 47, 68, 43, 57, 55} In Inorder traversal
ascending order
- d) {79, 14, 72, 56, 16, 53, 55}



→ Time Complexity of BST :



gate - 2008

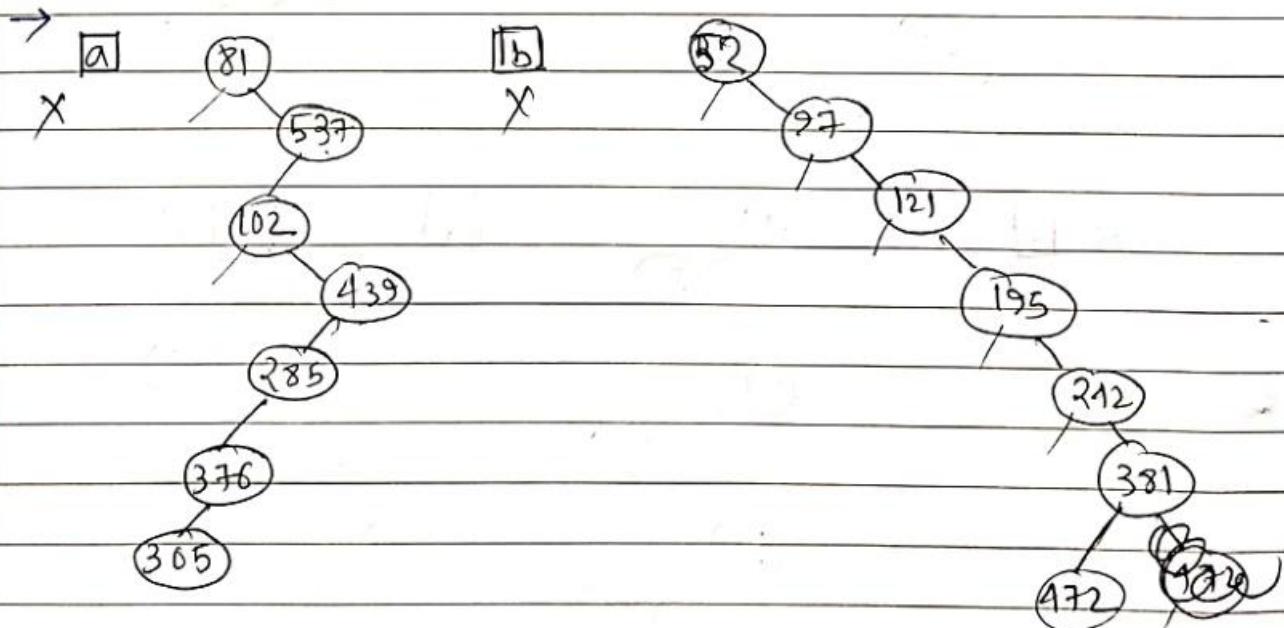
Question - ⑤

A BST stores values in the range 37 to 573.

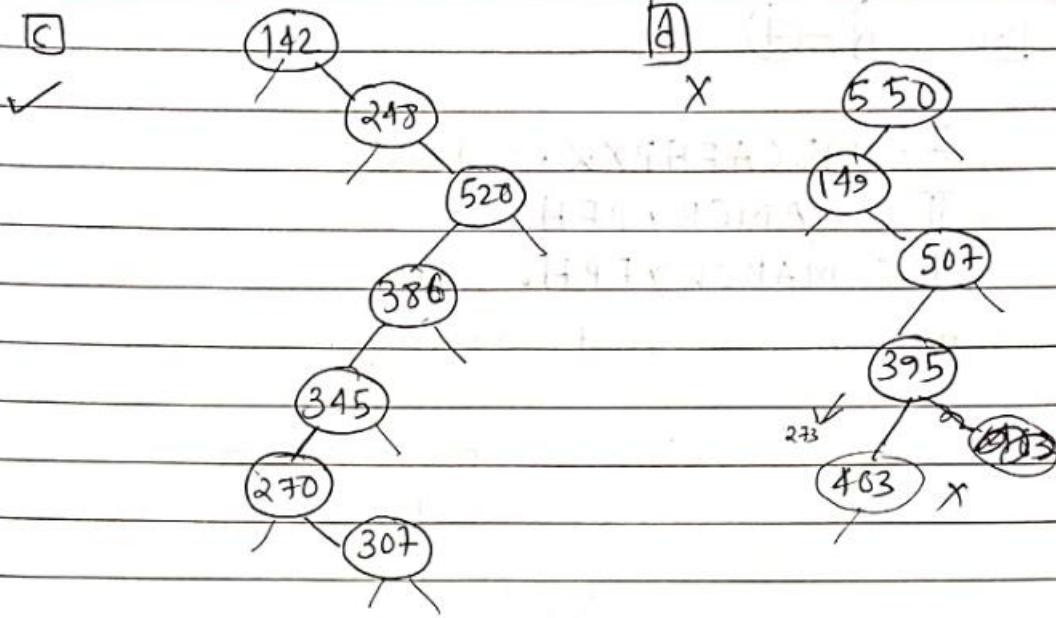
Consider the following sequence of keys -

- x a) 81, 537, 102, 439, 285, 376, 305.
- x b) 52, 97, 121, 195, 242, 381, 472.
- ✓ c) 142, 248, 520, 386, 345, 270, 307.
- x d) 550, 149, 507, 395, 463, 402, 270.

273 → ^{search} unsuccessful



273



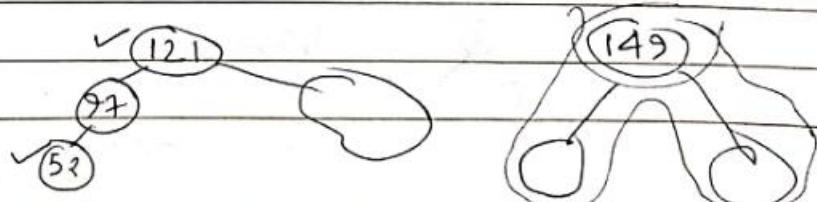
^{Q' 2008}
Question - (6)

A BST stores values in the range 37 to 573
consider the following sequence of key -

- X I) 81, 537, 102, 439, 285, 376, 305.
- II) 52, 97, 121, 195, 242, 381, 472.
- III) 142, 298, 520, 386, 345, 270, 307.
- X IV) 550, 149, 507, 395, 463, 402, 270.

Which of the following is true?

- X (a) I, II and IV are Preorder sequences of 3 diff BSTs.
- X (b) I is preorder sequence of some BST with 439 as the root.
- ✓ (c) II is an inorder of some BST with 121 as root and 52 as leaf.
- X (d) IV is a postorder sequence of some BST with 149 root.



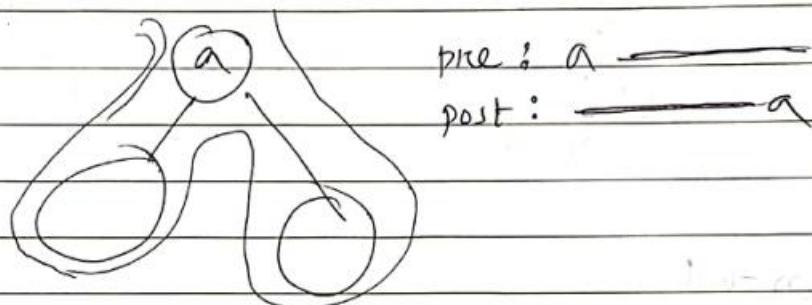
g-²⁰⁰⁸ Question - 7

I: MBCAFHPYK. — post

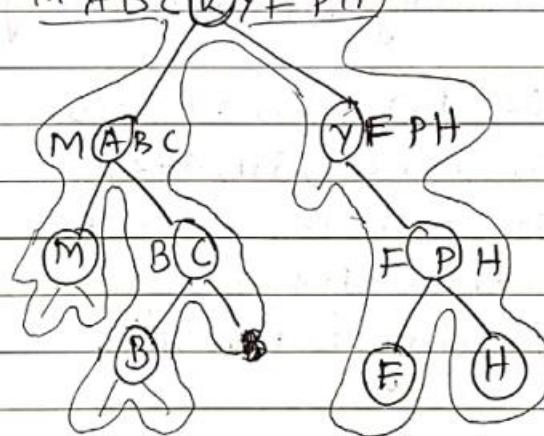
✓ II: KAMCBYPFH. — pre

III: MABCKYFPH. — in

which one is what order?



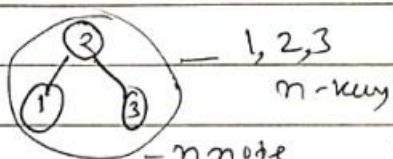
in: M A B C K Y F P H



g-²⁰⁰⁸ Question - 8

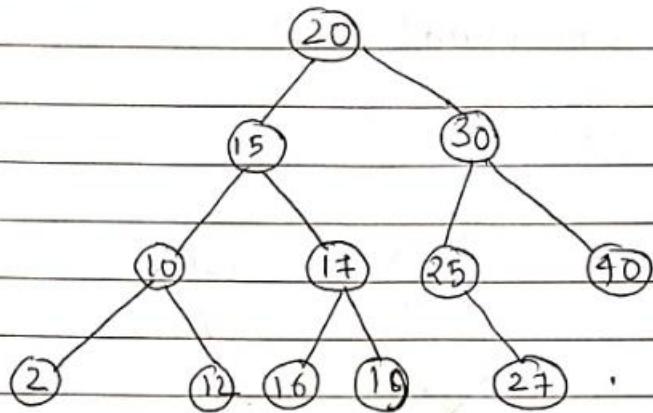
We are given set of n distinct elements and an unlabeled binary tree with ' n ' nodes. In how many ways can we populate the tree with the given set so that it becomes a BST =

- a) 0 b) 1 c) $n!$ d) $\frac{2^n n!}{n+1}$



→ only one way possible to become (BST),

- Deleting a node from BST (Binary search tree) :-

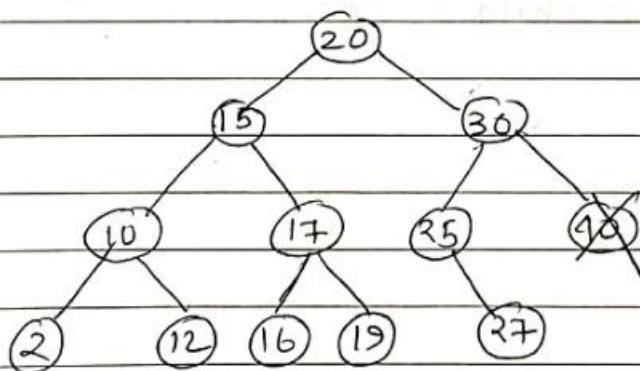


~~The Three types of deletion~~

Three types of element deletion possible in BST =

- leaf -
- Non-leaf
- | → one child.
- | → two children.

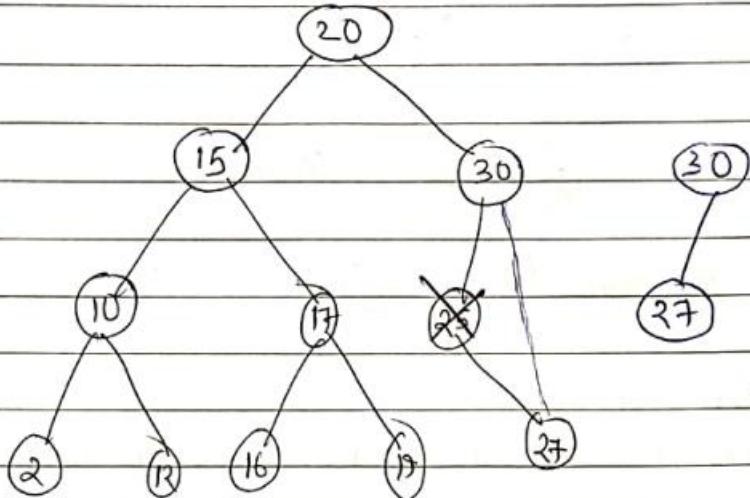
- Deleting leaf :- 10



→ When deleting a leaf you did not worry about anything

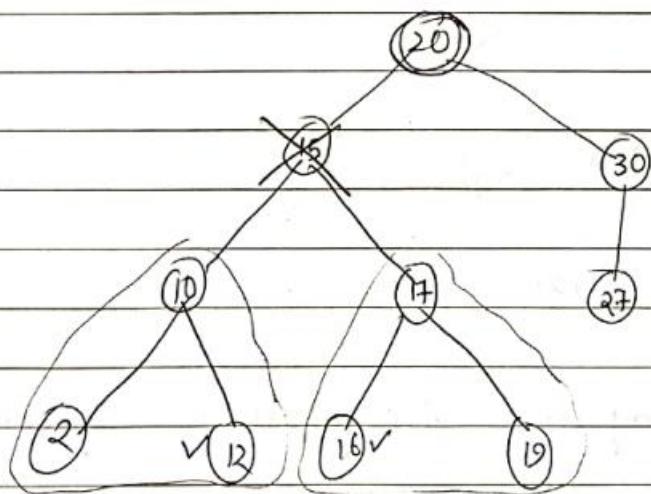
- Deleting Non-leaf:

(i) having one child: ②₅



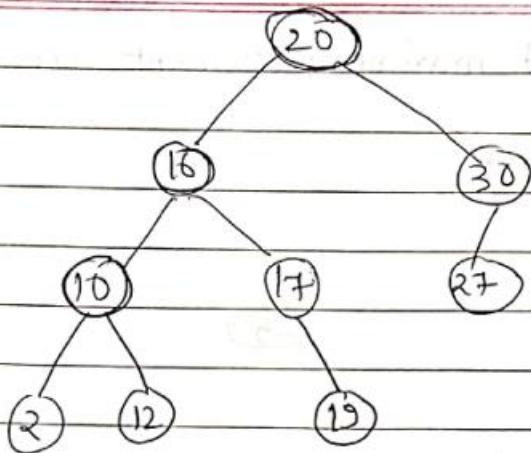
→ whenever a node have one child and you are going to delete the node then make that child point to the grand-parent.

(ii) having two child: ①₅

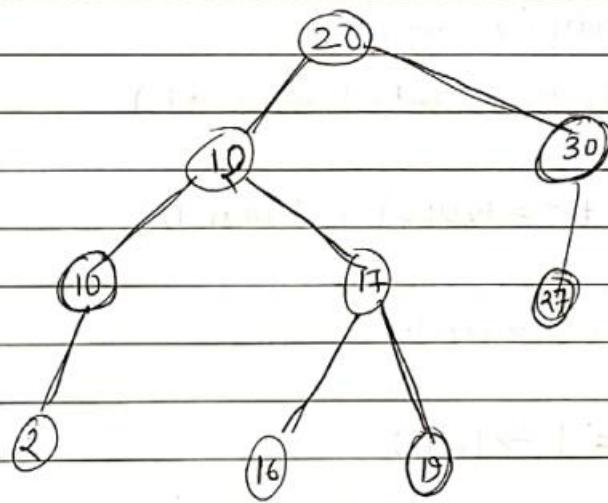


here 2-method:

→ go to the Right sub tree take the first element and then put in place of deleted node.



→ other method is, go to the left subtree. Then take the largest node element and then put it in place of deleted node.



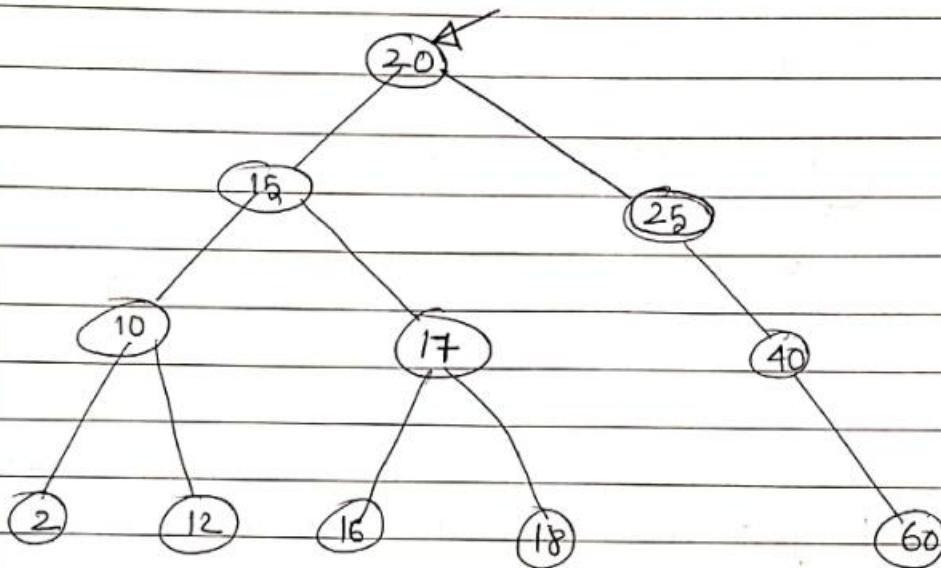
→ By doing deletion never increase the height of the tree.

→ By doing deletion insertion height of the tree might increase.

→ Inorder Successor: ^{least} ~~greatest~~ element in Right-sub-tree

→ Inorder predecessor: ^{greatest} ~~least~~ element in Left-sub-tree.

- Finding minimum and maximum element in a BST =



- find out least element of the BST :-
(min / inorder predecessor)

find-inord-pre (struct note *t)
 { -Min
 if ($t == \text{NULL}$) return 0;

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

{
 }
 t = $t \rightarrow \text{left}$;

- find greatest element of the BST :-

find-max (struct note *t).

{
 if (t) return 0

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

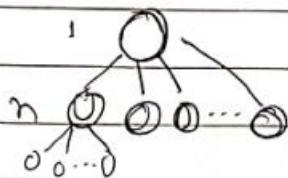
{
 }
 t = $t \rightarrow \text{right}$;

g' 1998

[Question] - ①

A complete n -ary tree is one in which every node has ' 0 ' or ' n ' sons. If ' x ' is the number of internal nodes of complete n -ary tree, the number of leaves in it is given by —

- a) $x(n-1)+1$ b) $xn-1$ c) $xn+1$ d) $x(n+1)$.



1st mode — n leaves.

2nd mode — $(n-1)+n$

3rd mode — $(n-2)+n+n = 3n-2$

4th mode — $(n-3)+n+n+n = 4n-3$

\vdots
x mode — $xn-(x-1)$

$$= xn-x+1$$

$$= x(n-1)+1 \text{ (leaves)}$$

g' 2002
[Question] - ②

The number of leaf nodes in a rooted tree of ' n ' nodes, each with each node having 0 or 3 children is —

- a) $n/2$ b) $(n-1)/3$ c) $(n-1)/2$, d) $(2n+1)/3$.

\rightarrow ~~deeper~~ $I \rightarrow$ internal node.

$n \rightarrow$ total node.

$L \rightarrow$ leaf.

$$I + L = n \quad -(1) \quad I = L(3-1) + 1 \quad -(2)$$

$$2I - 1 = -1$$

$$I + L = n$$

$$3I = n - 1$$

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

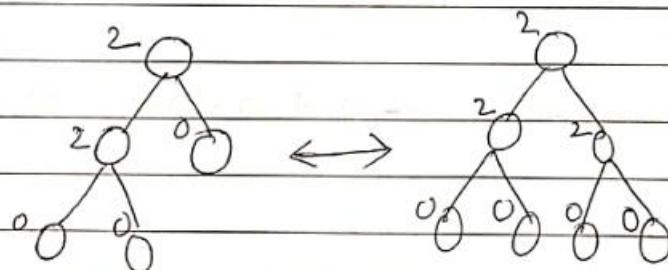
$$L + I = m$$

$$L + \frac{(n-1)}{3} = m$$

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

$$= \frac{3m - n + 1}{3} \Rightarrow \left(\frac{2m + 1}{3} \right)$$

- Recursive program on testing whether a tree is complete binary tree



both are complete binary tree.

IsComplete

int IsComplete (struct node *t)

{

if ($t == \text{NULL}$) return 0;

if ($(!(t \rightarrow \text{left})) \& \& (!(t \rightarrow \text{right}))$)
return 1;

else if ($t \rightarrow \text{left} \& \& t \rightarrow \text{right}$)

return (IsComplete ($t \rightarrow \text{left}$) $\&\&$ IsComplete ($t \rightarrow \text{right}$));

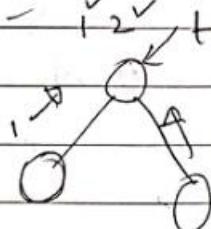
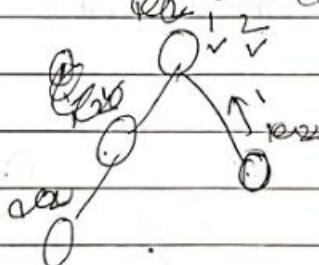
3

①

②

else

returning 0;

 $\therefore \text{if } 0 = 0 \text{ then true.}$  $\therefore 0 = 0 \text{ false (not com true)}$ 

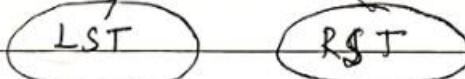
- Introduction to AVL tree =

AVL is an ~~balanced~~ balancing search tree.

Balanced factor (BF) = Height (LST) - Height (RST).

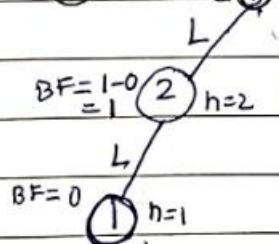
-1, 0, 1 - allowed (balanced).

-2 or 2 unbalanced

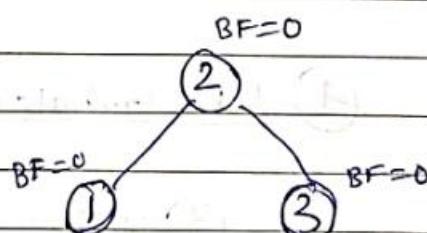


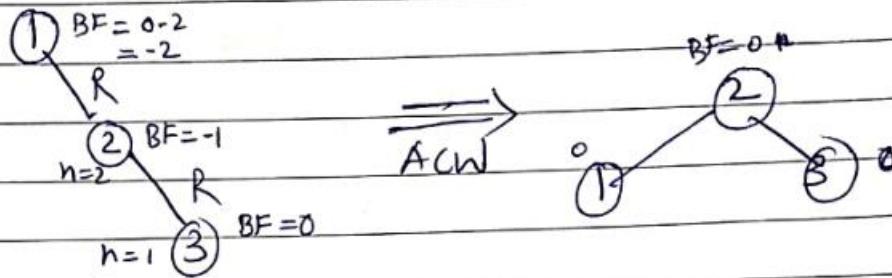
LL-Imbalanced

① BF = 2
= 2 ③ h = 3

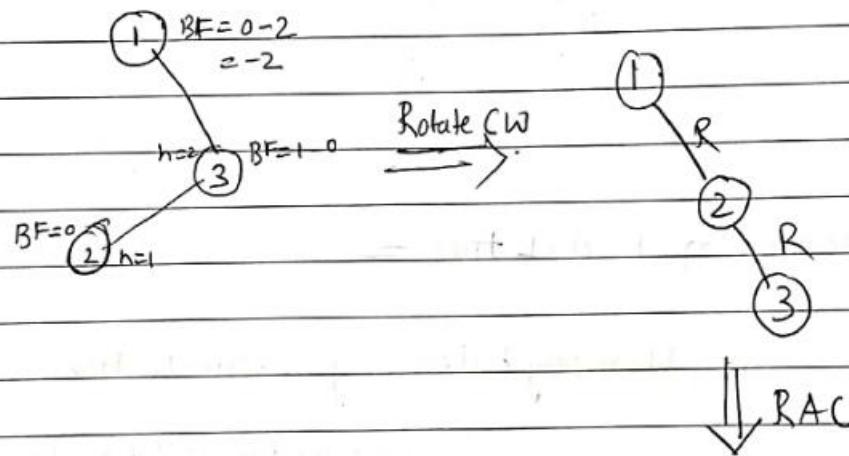


rotate
clockwise

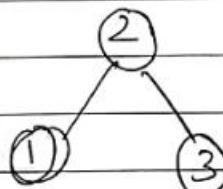


(3) RR - Imbalanced :(3) Right-left (RL) Imbalanced :

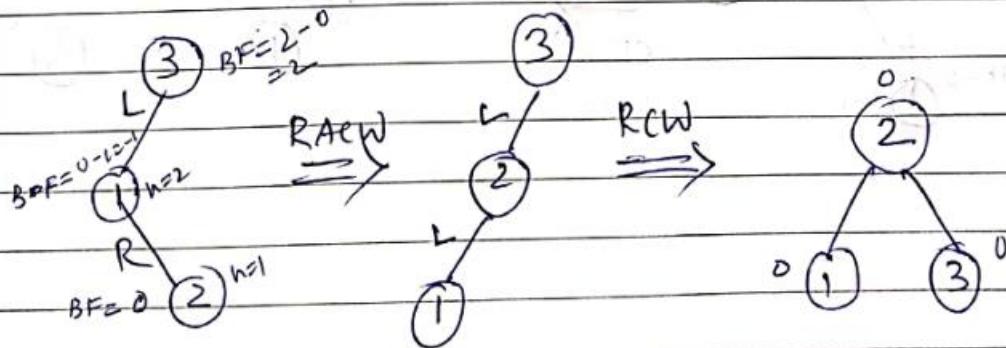
BF
= Balanced
factor
Should
($< 0, 0$)



↓ RAC



→ result will be BST otherwise something wrong.

(4) LR Imbalanced :

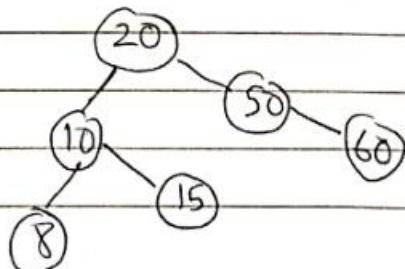
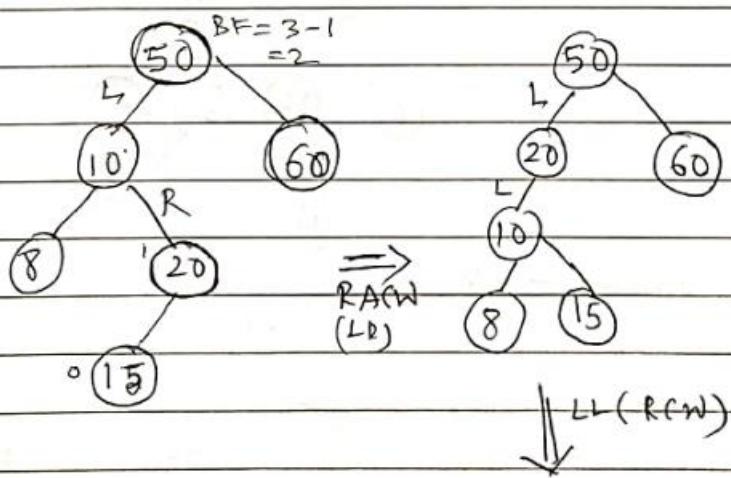
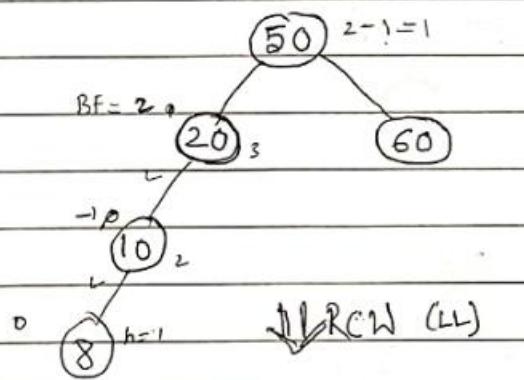
✓ After inserting a newnode you should travel from newly inserted node toward root to see on the path, if there is any imbalanced.

If there is any imbalanced node then start from imbalanced node, ^{and start going} toward ^{added} newnode to them you understand what type of imbalanced (LL, RR, LR, RL). then rotate them accordingly.

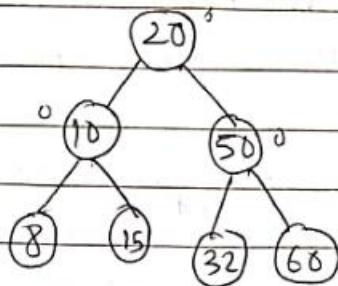
- Constructing AVL trees and time complexity analysis =

- Example - 1

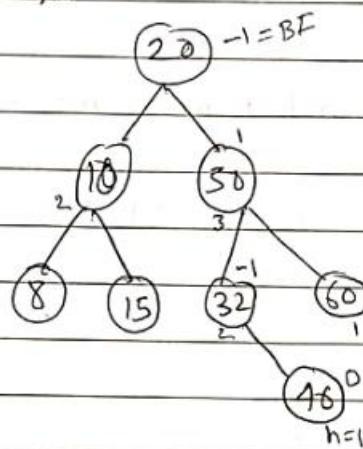
50, 20, 60, 10, 8, 15, 32, 46, 11, 18.



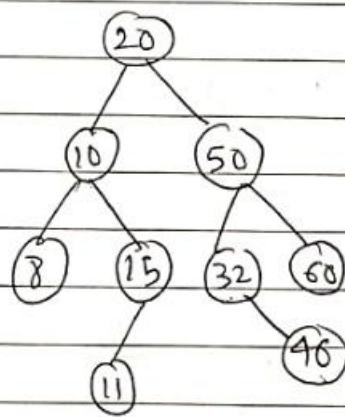
then insert 32.



Insert - 46

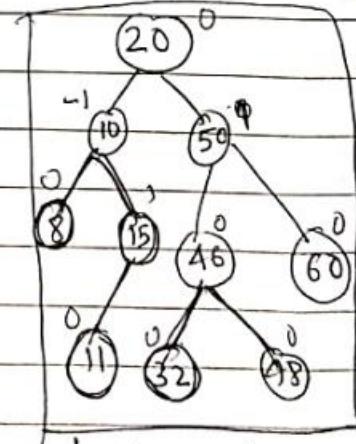
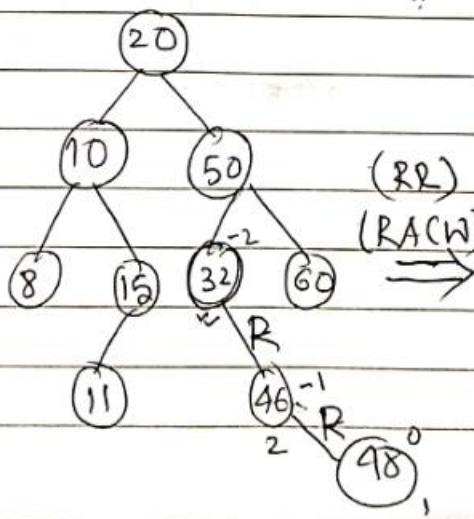


insert - 11



$(-1, 0, 1) = BF$

Insert - 48



↳ Balanced Binary Search

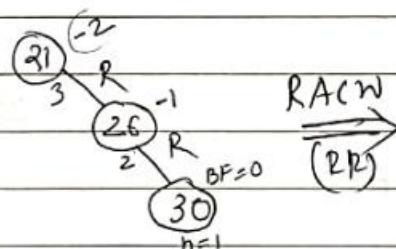
• Complexity analysis =

	BST	(balanced) BBST (AVL)
search -	$O(n)$	$O(\log n)$
height -	$O(n)$	$O(\log n)$
insertion -	$O(n)$	$O(\log n) + O(\log n) + c \Rightarrow O(\log n)$

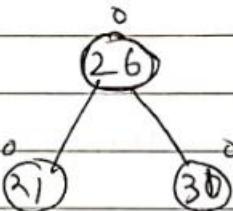
• Construction of AVL tree =

• Example - ②

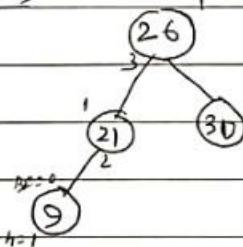
21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7.



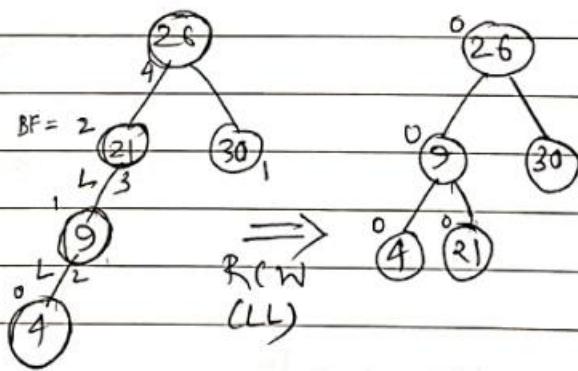
RACW
(RR)



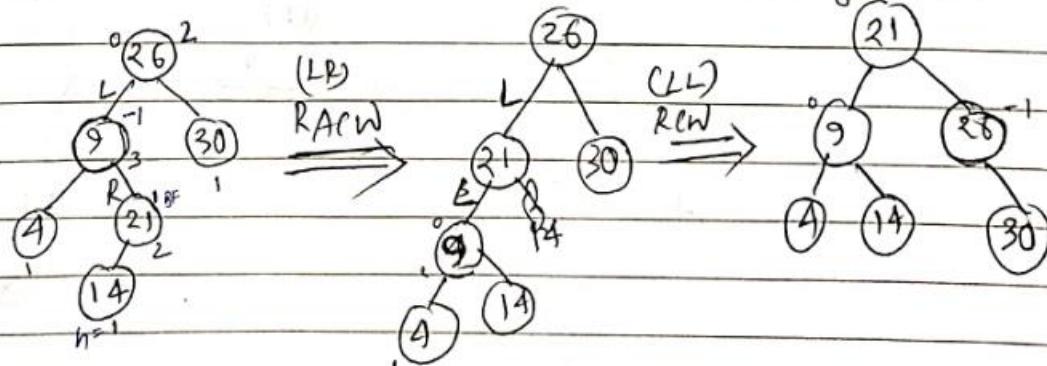
Insert - 9

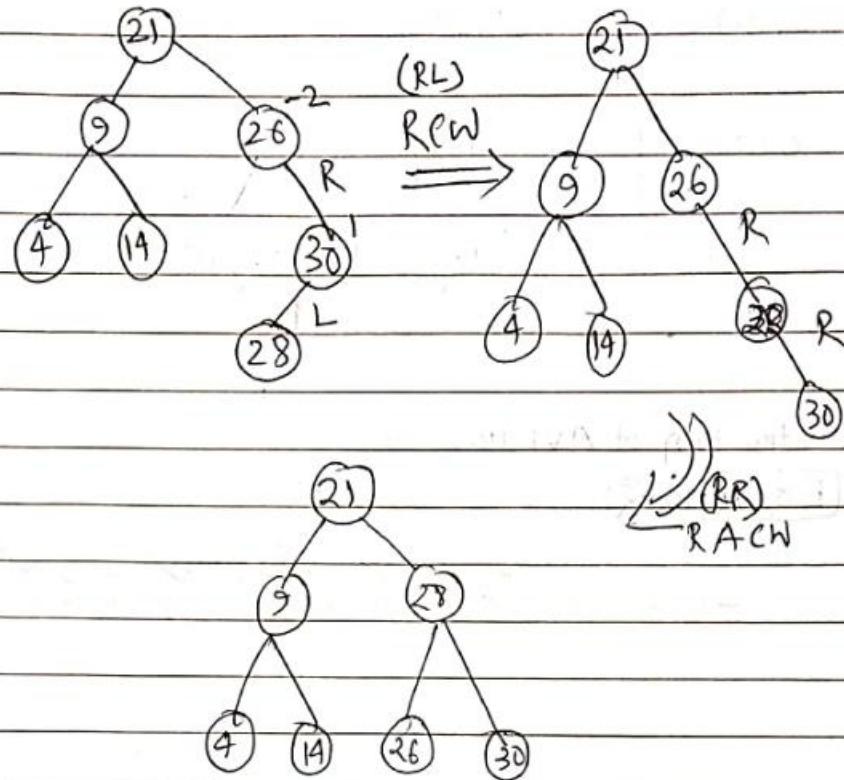
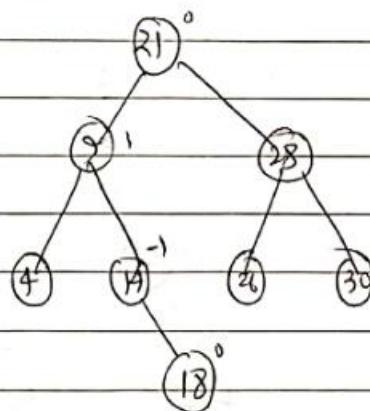
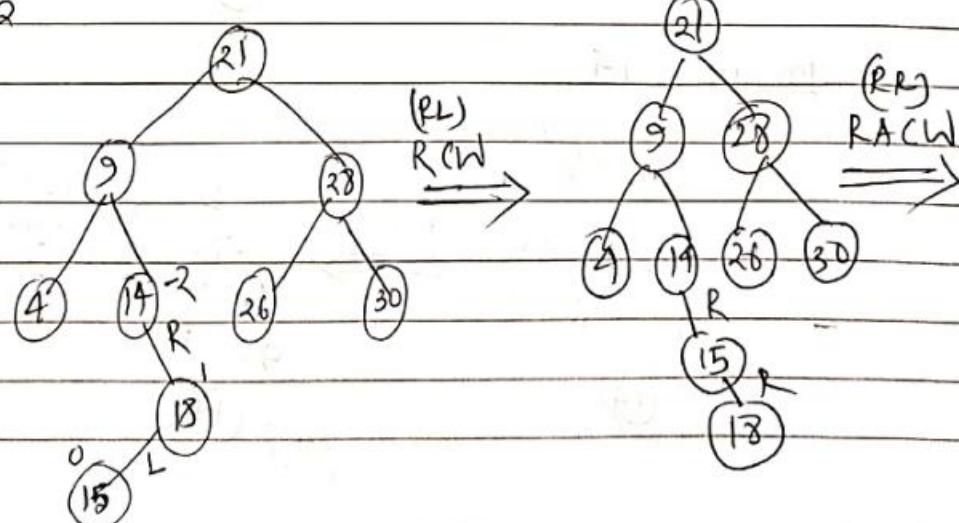


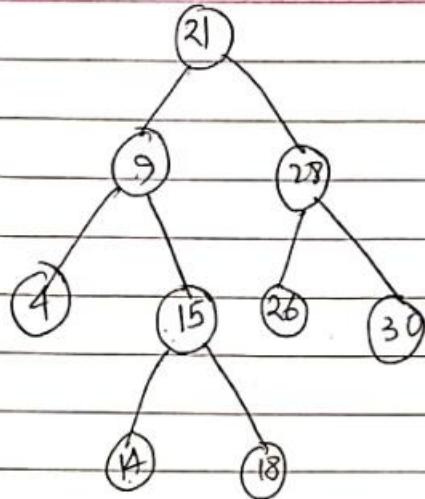
Insert - 4



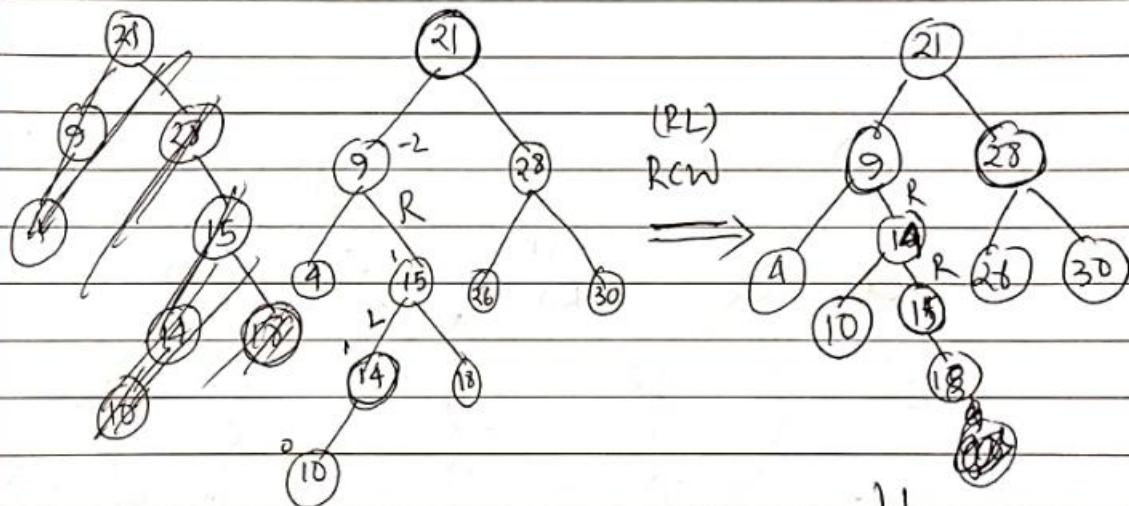
Insert - 14



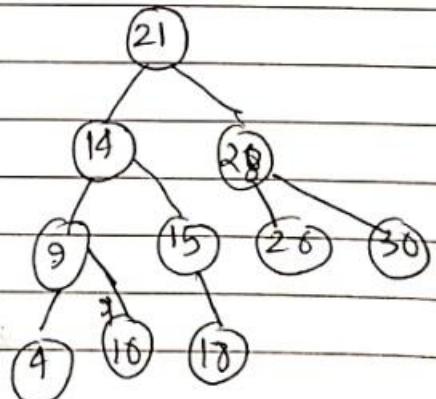
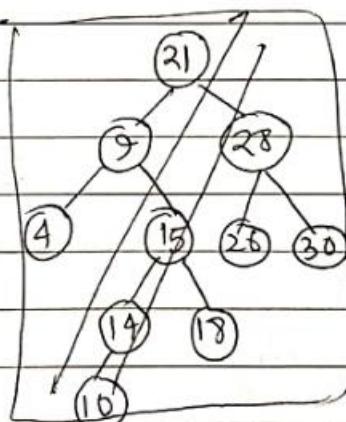
Insert - 28then, Insert - 18Insert - 15

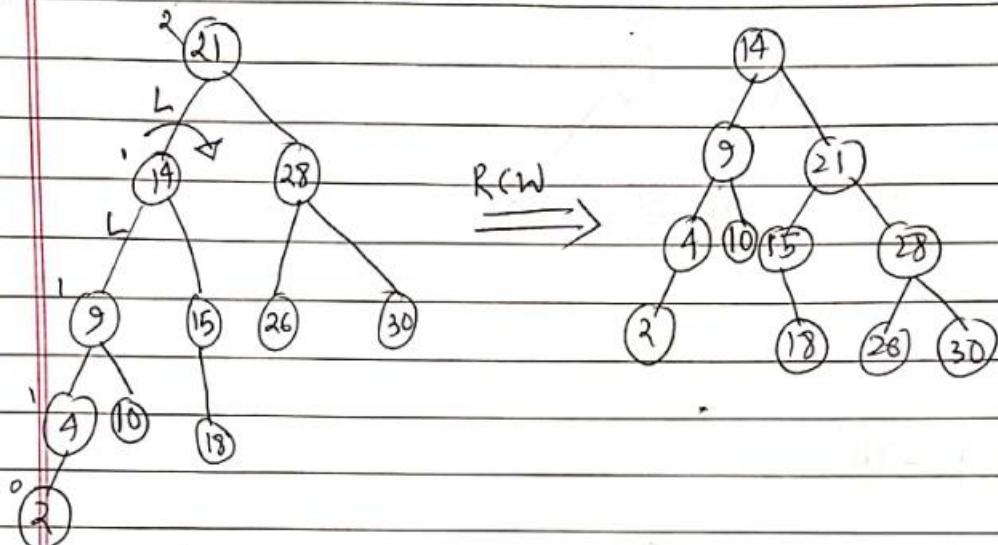
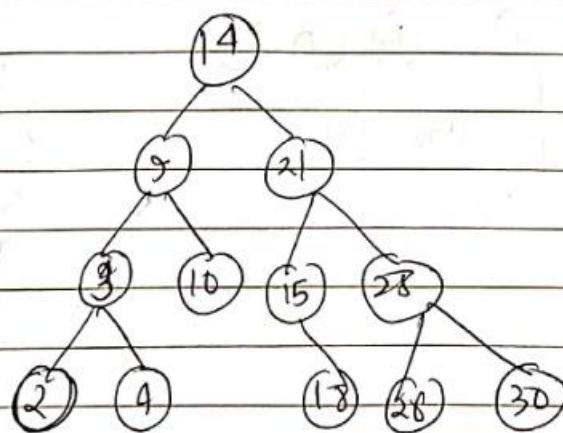
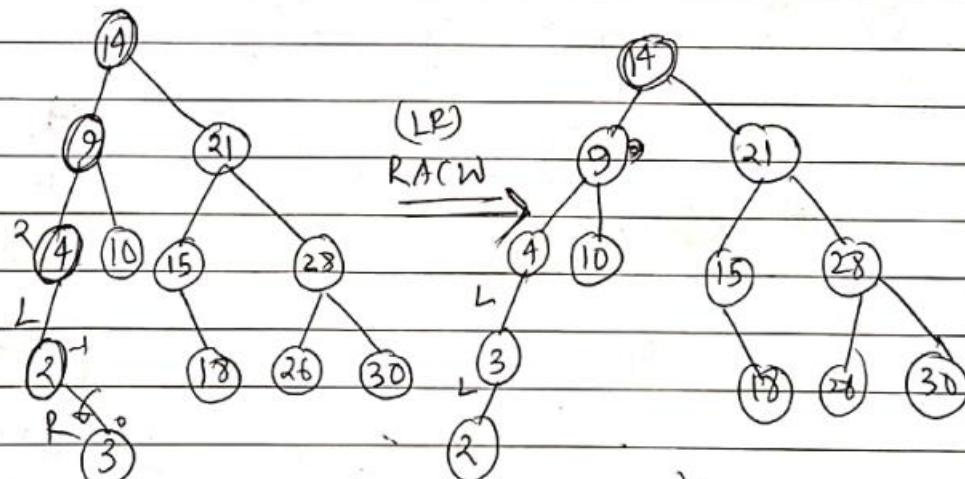


Insert - 10

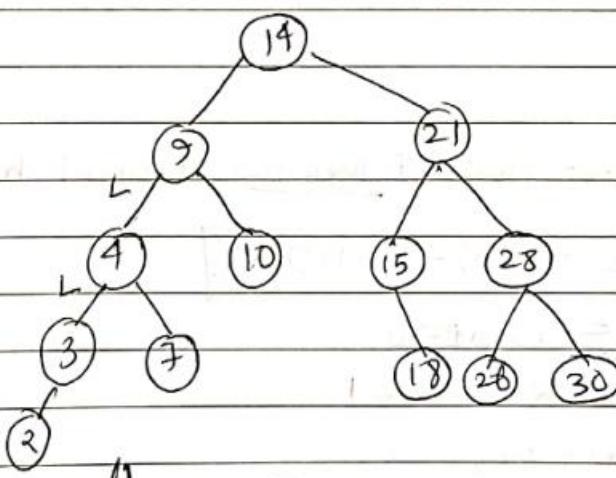
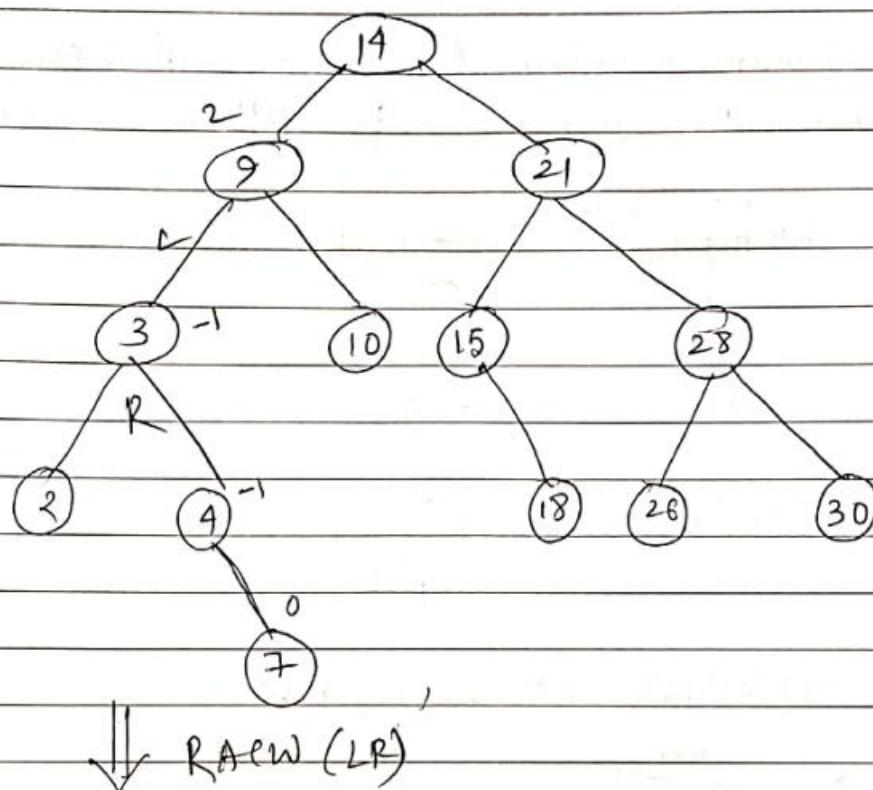


RACW
(RR)

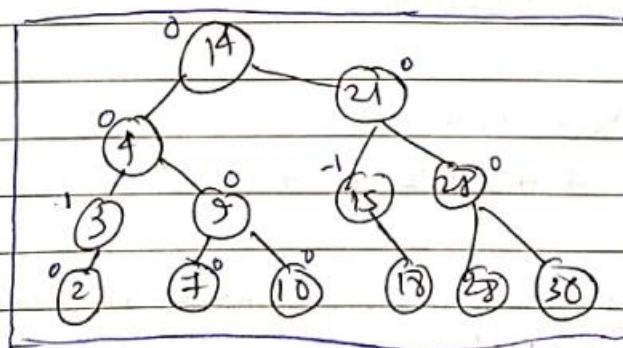


insert - 2insert - 3

Insert next last element in input - 7



↓ RAEW (LL)



→ Final balanced tree

- Minimum and Maximum nodes in an AVL tree of height h -

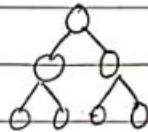
→ Maximum number of node present (in AVL tree or normal binary tree) is $\boxed{F(2^{h+1}-1)}$ // $h = \text{height of the tree.}$

no. of node
 for height node height
 1 0

$$1 + 2^1$$



$$1 + 2^1 + 2^2$$



$$1 + 2^1 + 2^2 + 2^3 + \dots + 2^h \text{ for height } h$$

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

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

→ Minimum no. of node of AVL tree of height h is,

$$\boxed{N(h) = N(h-1) + 1 + N(h-2)}$$

$$= 1; h=0$$

$$= 2; h=1$$

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

$$= 2 + 1 + 1$$

$$= 4 \text{ node(min)}$$

(With height 2 minimum 4 node possible)

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

$$= 4 + 1 + 2 = 7 \text{ (min node)}$$

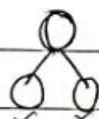
→ minimum no. of node can present in binary tree with height h $\boxed{=(h+1)}$

8-1995

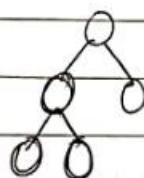
[Question] - ①

A binary tree 'T' has ' n ' leaf nodes. The no. of nodes of degree '2' in 'T' is -

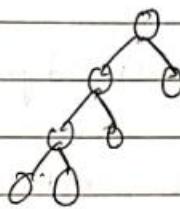
- a) $\log_2 n$ b) $n-1$ c) n d) 2^n



1 node of degree 2 = \rightarrow 2 leaf.



$$2 \rightarrow n = 3 \rightarrow$$



$$3 \rightarrow " " = 4 \text{ leaf.}$$

n node with degree 2 = $n+1$ leaf.

$$\text{node leaf}$$

$$n = (n+1)$$

when n leaf = $(n+1)$ node of degree 2.

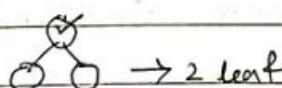
$(n+1)$ node with deg 2 = n leaf.

8-2005

[Question] - ②

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

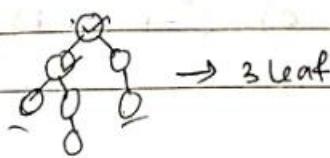
- a) 10 b) 11 c) 12 d) 15



\rightarrow 2 leaf

10 node with deg 2 = $10+1$

$$= 11$$



\rightarrow 3 leaf

gate-

2088

Question)-3

which of the following is true about search times —

AVL

- a) $\Theta(\log n)$
- b) $\Theta(1 \log n)$
- c) $\Theta(n)$
- d) $\Theta(n \log n)$

BST

- $O(n)$
- $\Theta(n \log n)$
- $\Theta(\log n)$
- $O(n)$

gate-

1998

Question)-4

Which of the following statement is false?

a) a tree with 'n' nodes has $(n-1)$ edges.

b) a labeled rooted binary tree can't be uniquely constructed given its post order and pre order traversal.

c) a complete binary tree with 'n' internal nodes has $(n+1)$ leaves.

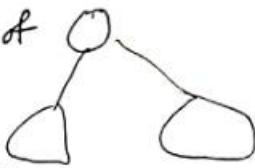
d) The max no. of nodes in a binary tree of height 'h' is $(2^{h+1}-1)$.

gate-
2005
1995Question)-5

In a binary tree, for every node the difference b/w the number of nodes in the left and right subtrees is at most 2. If the height of the tree is $h > 0$, then the minimum no. of nodes in the tree is —

- a) 2^{h-1}
- b) $2^{h-1} + 1$
- c) $2^h - 1$
- d) 2^h .

$N(h) \rightarrow$ no. of nodes of height h .



classmate

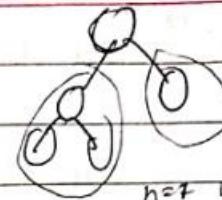
Date _____

Page _____

Precurrence equation -

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

$$[N(h) = 2N(h-1) - 1] \quad \text{--- (1)}$$



$h=1 \quad h=2$

$$\begin{aligned} N(2) &= 2 \cdot N(1) - 1 \\ &= 2 \cdot N(1) - 1 \\ &= 2(2) - 1 = 3, \quad h=2 \end{aligned}$$

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

$$N(h-1) = 2N(h-2) - 1 \quad \text{--- (11)}$$

$$N(h-2) = 2N(h-3) - 1 \quad \text{--- (111)}$$

(11) and (111) put into (1)

$$N(h) = 2^3 N(h-3) - 2^2 - 2 - 2$$

:

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

$$= 2^K N(h-K) - (2^{K-1} + 2^{K-2} + \dots + 1)$$

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

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

$h-K=2$
 $K=h-2$

$$= 2^{h-2} * 3 - (2^{h-2} - 1)$$

$$= 3 \cdot 2^{h-2} - 2^{h-2} + 1$$

$$= 2 \cdot 2^{h-2} + 1$$

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

\rightarrow here we substitution method by drawing the tree, according to question.

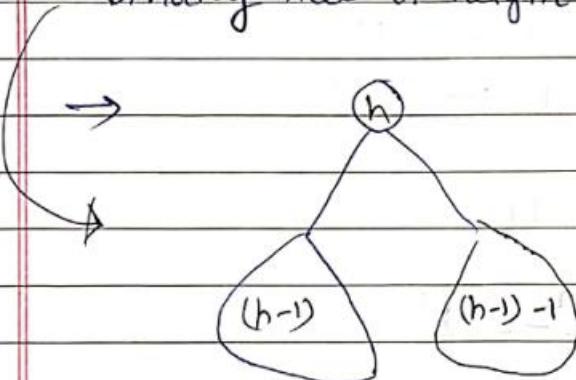
gate-1997

Question - 6

A size balanced binary tree is a binary tree in which for every node, the difference b/w the no. of nodes in the left and right subtrees is at most 1. The diff distance of a node from the root is the length of the path from the root to the node. The height of a binary tree is the maximum distance of a leaf node from the root.

Q)

Prove by induction on h , that a size-balanced binary tree of height ' h ' contains at least 2^h nodes.



$N(h) \rightarrow$ min no. of node that can present in tree of height h .

$$N(h) = \underbrace{N(h-1)}_{\text{LST}} + 1 + \underbrace{N(h-1)}_{\text{Root}} + \underbrace{N(h-1)}_{\text{RST}}$$

($n=0, \min \text{node} = 1$)

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

$$\boxed{N(h) = 2N(h-1)} \quad \text{--- (1)}$$

$h=0 \quad h=1$

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



$$N(h-2) = 2N(h-3) \quad \text{--- (3)}$$

Back substitution-

$$N(h) = 2^3 N(h-3)$$

$$h-k=0$$

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

$$k=h$$

$$= 2^h N(0)$$

$$\boxed{N(h) = 2^h}$$

• Can ask question on this topic

(1) What is the min no. of node required to make in order to make height h ?

(2) Given node then how much height build.

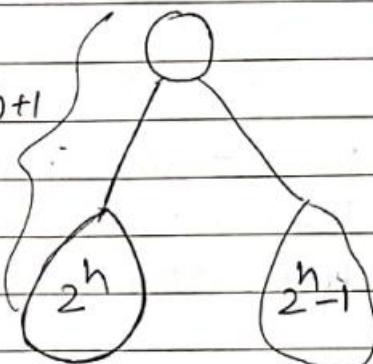
$$2N(h) = 2^h$$

By induction -

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

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

$$\text{height} = h+1$$



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

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

g^{2014}

[Question] - ⑦ (Balanced binary search tree)

Suppose we have a balanced 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 b/w 'l' and 'H'. Suppose there are 'm' such numbers. If the tightest upper bound on the time to complete the sum is $O(n^a \log^b n + m^c \log^d m)$, the value of $a + 10b + 100c + 1000d$.



$$O(m \log n)$$

~~$$O(n^a \log^b n + m^c \log^d n)$$~~

$$a = 0, b = 0$$

$$c = 1, d = 1$$

$$a + 10b + 100c + 1000d$$

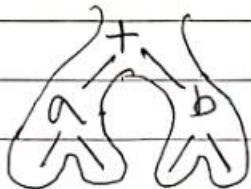
$$\Rightarrow 0 + 10 \times 0 + 100 \times 1 + 1000 \times 1$$

$$\Rightarrow 1100$$



• Expression trees =

① $a+b$



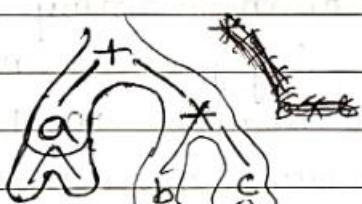
PRE : +ab (prefix expression)

IN : a+b (infix expression)

POST : ab+. (postfix expression)

→ On the expression tree if you perform pre, IN, and post order traversal, then it will give you prefix, infix and postfix expression respectively.

② $a+b*c$

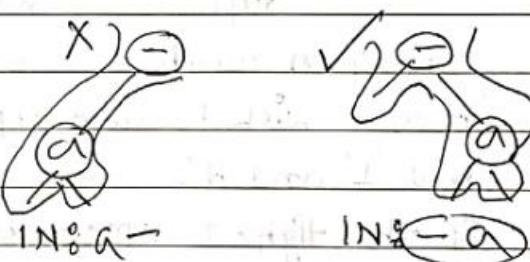


PRE : +a*bc

IN : a+b*c

POST : abc*+

③ $-a\sqrt{b}$



IN : a-

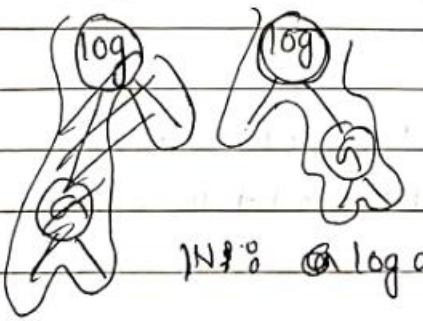
IN : -a

④ $a!$



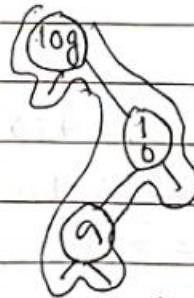
IN : a!

⑤ $\log a$



IN : @log a

⑥ $\log(a!)$

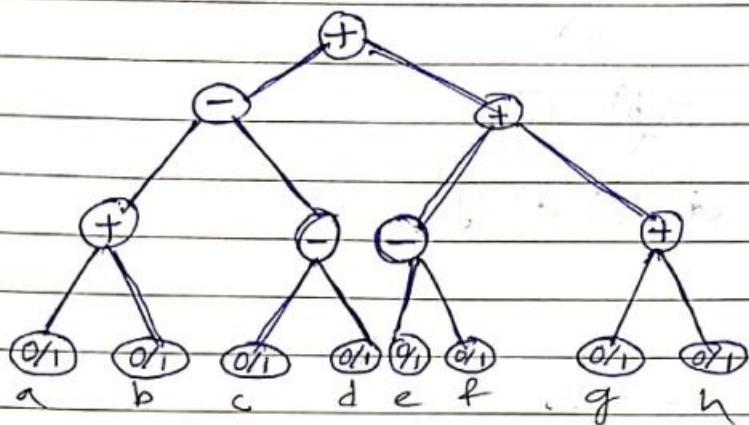


IN order : log(a!)

→ By travelling In order, find out the tree.

→ When combination of operators, then build tree step by step.

g-2014

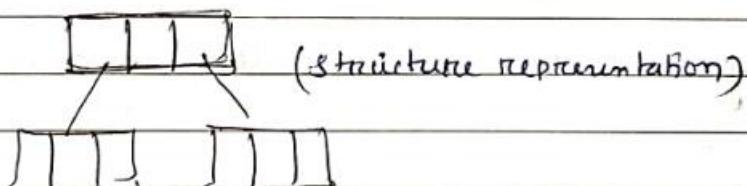
Question - ①

When every operand can take '0' or '1' then what will be the max value of expression.

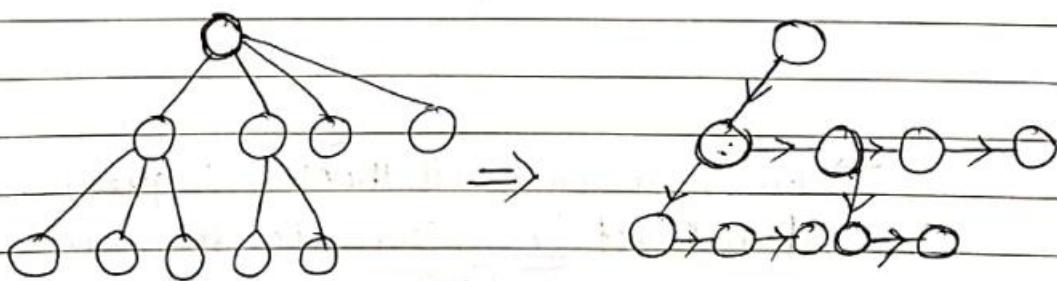
$$\rightarrow ((\underset{1}{a} + \underset{1}{b}) - (\underset{0}{c} - \underset{1}{d})) + ((\underset{1}{e} - \underset{0}{f}) + (\underset{1}{g} + \underset{1}{h}))$$

$$\rightarrow 3 + 3 \rightarrow \boxed{6}$$

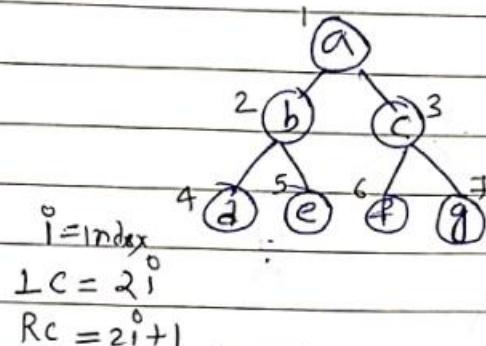
- Various tree representations =



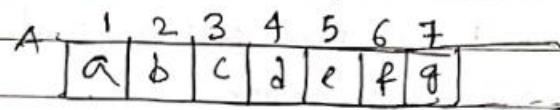
- Left child Right sibling representation =
(LCRS)



- array representation of Binary tree -



$$\text{parent}(x) = x/2.$$

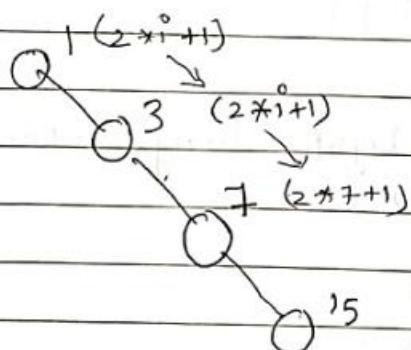


adv:

- finding out parent easy, and finding out right child and left child is easy by array representation.
- traversing easy.

→ widely used in implementation of heaps.

disadv:



→ In worst case (if the tree is skewed) and if has got ' n ' elements then the size of array required is $(2^n - 1)$

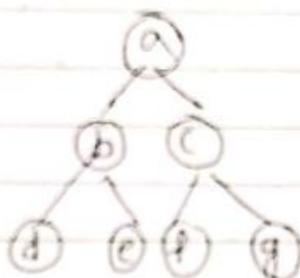
- Nested representation =



$(ba\bar{c}) \quad (\bar{a} b \bar{c})$

L int R Lnt L R

Ex:



= $a(b\bar{d}\bar{e})(\bar{c} \bar{f} \bar{g})$

 | | |
 Lnt L R R

- 4 - representation =

i) → Normal (pointer, struct)



ii) → LCRS (n-ary, n>>2)

iii) → Array (heaps, ^{using} all must complete BT)

iv) → Nested.

gateiesmentor ^{100%}
[Question] - ② $\frac{\text{Root}}{L \text{ int } R}$

(xyz) , y and z can be NULL,

which of the following does represent binary tree -

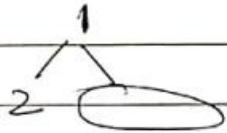
a) $(1(2(4 5 6)7))$

b) $(1((2 3)4)56)7)$

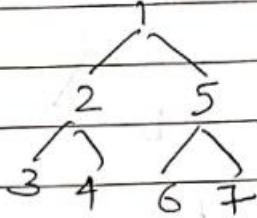
c) $(1(2 3)4)(5(6))7)$

d) $(1(2 3 \text{NULL})(4 5))$

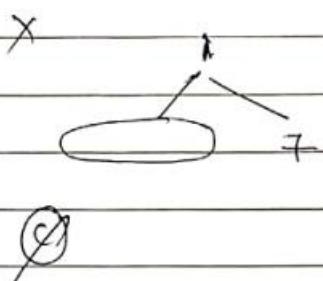
Ⓐ $(12(\underline{45}67))$



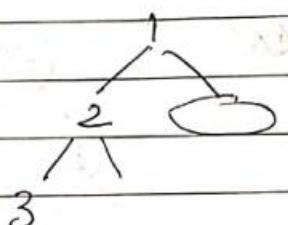
(6) $(1\ (2\ 3\ 4)\ (5\ 6\ 7))$



(b) $(1 \underline{(234)} 56) \exists$



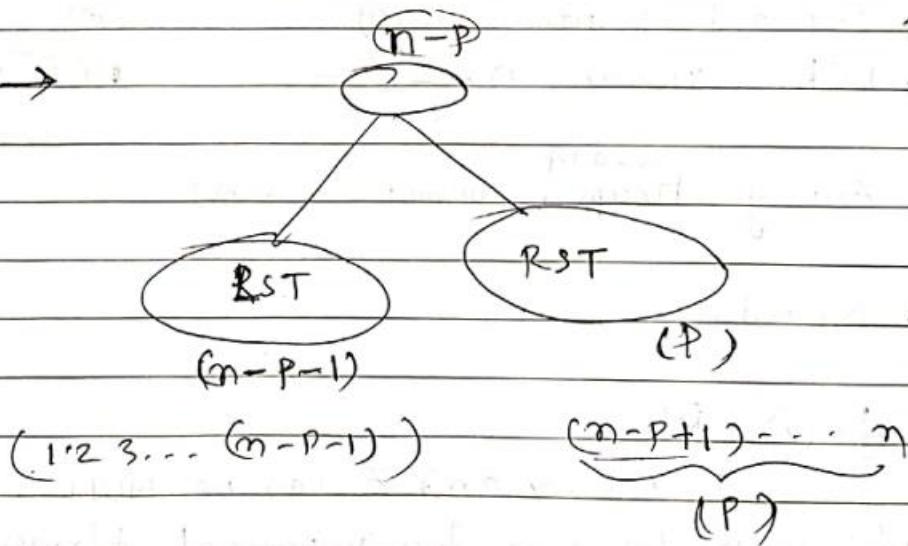
d) (1(23NULL)(45))



gate 2005 question - ③

The numbers 1, 2, ..., 7 are inserted into a BST in some order. In the resulting tree, the right subtree of the root contains p nodes. The first number to be inserted in the tree must be -

- (a) p (b) p+1 ~~(c) n-p~~ (d) n-p+1.



Date
2006

Question - ④

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

④-1) The ~~fix~~ index of the parent of element $X[i]$, $i \neq 0$, is

$$\textcircled{1} [i/2]$$

$$\textcircled{2} [i/2]$$

$$\textcircled{3} [(i-2)/2]$$

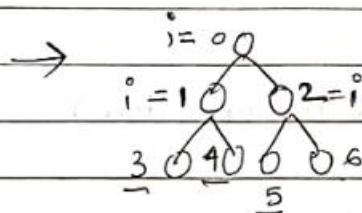
$$\textcircled{4} [i/2] - 1$$

both correct.

Value of i

by substitute them in option

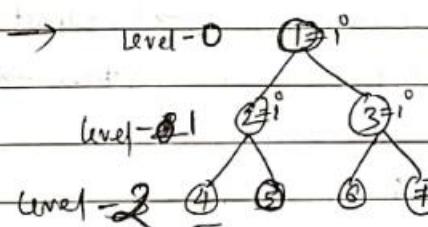
option find current option.



④-2)

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

$$\textcircled{1} \lceil \log_2 i \rceil \quad \textcircled{2} \lceil \log_2 (i+1) \rceil \quad \textcircled{3} \lceil \log_2 (i+1) \rceil \quad \textcircled{4} \lceil \log_2 i \rceil$$



Ques 8

$$\lceil \log_2 8 \rceil = 3$$

$$\lceil \log_2 9 \rceil = 4$$

by using substitution method find founded current option.

9-2014

Question] - ⑤

(LCRS)

leftmost child - right sibling rep is used -
Each node of tree is of type treeNode.

```
typedef struct treeNode *treeptr;
```

```
struct treeNode
```

{

};

```
treeptr leftMostChild, rightSibling;
```

};

```
int DoSomething (treeptr tree)
```

{

```
int value = 0;
```

```
if (tree != NULL)
```

{

```
if (tree → leftMostChild == NULL)
```

```
value = 1;
```

else

```
value = DoSomething (tree → leftMostChild);
```

```
value = value + DoSomething (tree → rightSibling);
```

};

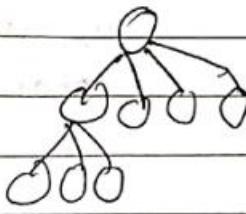
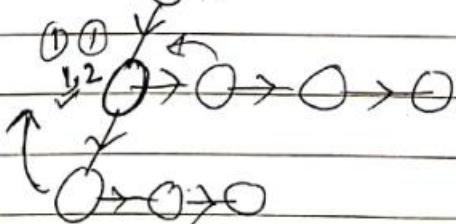
```
return (value);
```

};

n



$$\text{tree} \xrightarrow{\text{②}} \begin{cases} 0 & \text{if } 0 \\ 2 & \text{if } 2 \end{cases} = 2$$



Following

^ Which of option is false -

- (a) no. of internal node are counted.
- (b) height of the tree counted.
- (c) parent node node without right sibling.

Q3 no. of leaf node are counted.

GATEIESTMENTOR

classmate

Date _____

Page _____

GATEIESTMENTOR

RECURSION

classmate

Date _____

Page _____

- Tracing the recursion =

[Ex: 1]

A(n)

{

1. if ($n \geq 0$)

2. { printf ("0/d", n-1); }

3. A(n-1)

4. }

}

main()

{

A(3);

i) }

3

• (using stack method)

A(3) A(2) A(1) A(0)

main() n=3 n=2 n=1 n=0

stack

O/P: 1 2 1 0

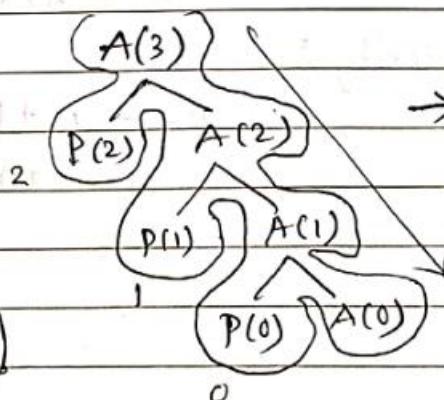
→ (5 activation record are pushed)

• Space complexity, $A(n) = n+1$
 $= \underline{\underline{O(n)}}$

$$A(3) = 3+1 = 4 \\ = O(3).$$

• Time complexity, $T(n) = C + T(n-1)$
 $= O(n).$

• (using tree method)



→ max stack record
will be created is =
depth of the tree.
(4)

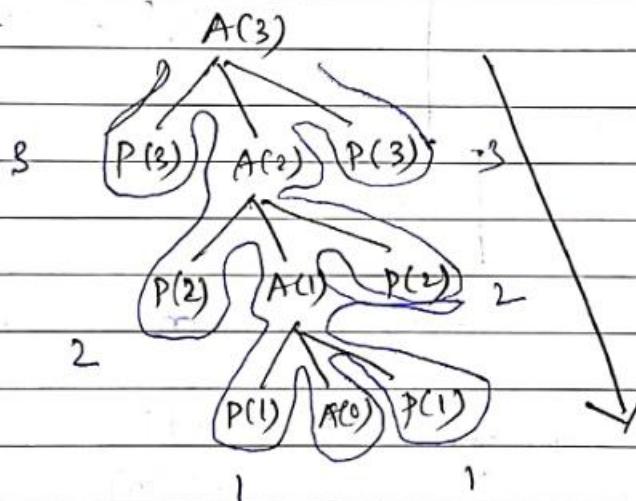
ex:2

```

A(n)
{
- If (n>0)
{
- pf(n);
A(n-1);
- pf(n);
}
}

```

using tree method find output when n=3.



O/P: 321123

Time Complexity:

$$\begin{aligned}
 T(n) &= C + T(n-1) \\
 &= O(n).
 \end{aligned}$$

\rightarrow constant time required.

Space Complexity:

$$\left. \begin{aligned}
 A(n) &= n+1 \\
 &= O(n)
 \end{aligned} \right\} \quad \left. \begin{aligned}
 A(3) &= 4 \\
 &= O(3)
 \end{aligned} \right.$$

Ex: 3

 $A(n)$

{

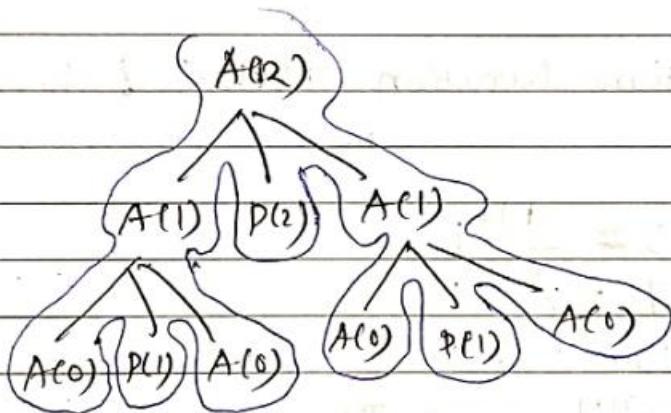
if($n > 0$)

{

 $A(n-1);$ $PF(n);$ $A(n-1);$

{

what will be output?

O/P : 1 2 1(total record made in stack = depth of
the tree)
(3)

	A(0)	A(0)	A(0)
	A(1)	A(1)	
	A(0)		

When first time exist push and last from visited pop.

complete

→ '3' record are created to execute this.

→ If there are less no. of statement go with tree method.

→ " " more " " stack method.

Important

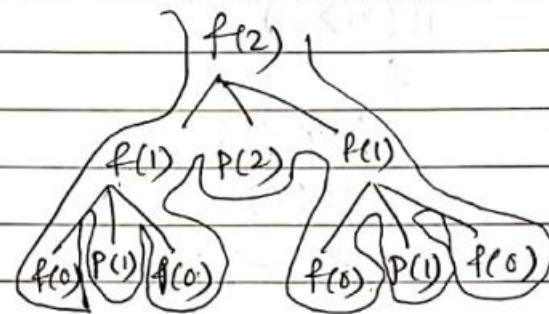
Question asked on recursion.
→ Trace and find out output.

- Analyzing recursion =

$f(n)$
 $\left\{ \begin{array}{l} \text{if } (n == 0) \\ \quad \text{return } n; \\ \quad f(n-1); \\ \quad p(f(n)); \\ \quad f(n-1); \end{array} \right.$

$n=2$

tree method -



output : 1 2 1

$f(2) = 7$ (times
function
invoked)

• how many time function $f(10) = ?$ invoked
When $f(10)$?

→

$$f(1) = 3 = 2^{1+1} - 1$$

$$c(1) = 1$$

$$f(2) = 7 = 2^{2+1} - 1$$

$$c(2) = 2$$

$$f(3) = 15 = 2^{3+1} - 1$$

$$c(3) = 10$$

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

$$\begin{aligned} f(10) &= 2^{10+1} - 1 \\ &= 2047 \text{ (no. of time functions are invoked).} \end{aligned}$$

other way -

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

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

$$F(n-2) = 2F(n-3) + 1 \rightarrow (3)$$

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

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

$$= 2^3 F(n-3) + 2^2 + 2^1 + 2^0$$

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

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

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

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

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

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

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

$$= 2^{10+1}-1 \Rightarrow 2^{11}-1 \Rightarrow 2047.$$

(Total no. of fun call required).

Time complexity,

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

$$= O(\infty) \cdot O(2^n).$$

Space complexity,

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

create - 2001
B1

void abc(char *s)

{

if ($s[0] == '0'$)

return;

abc($s+1$);

abc($s+1$);

pf("0%c", $s[0]$);

}

main()

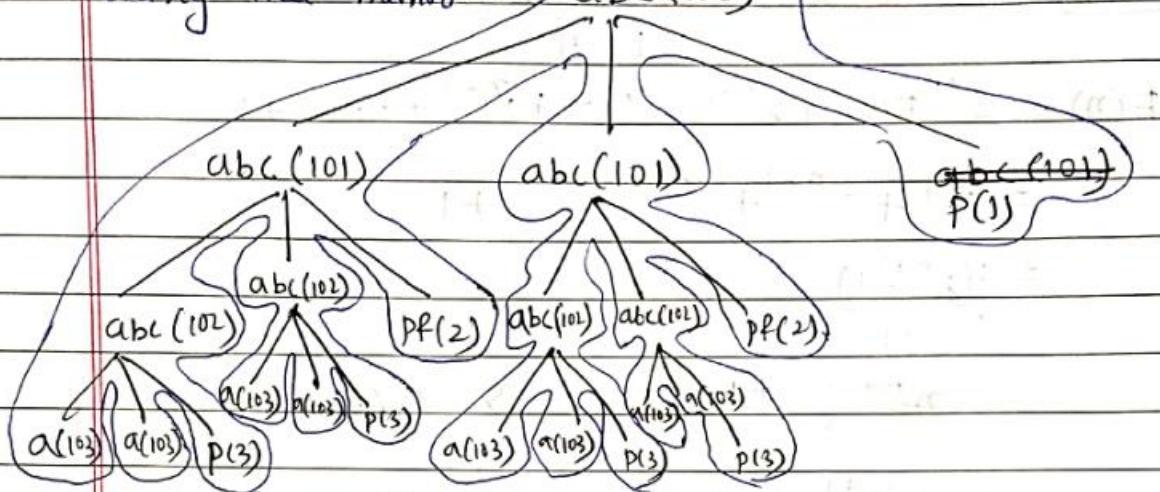
{ abc("123");

}

1	2	3	10
---	---	---	----

100 101 102 103

using tree method = abc(100)



(3 3 2 3 3 2 - 1) - output.
(7-characters printed)

B2

```
void abc(char *s)
{
    if (*s == '\0') return;
    abc(s+1);
    abc(s+1);
    printf("%c", s[0]);
}
```

main()
{

abc("123");
}

If abc(s) is called with a null-terminated string of length 'n' characters (not counting the null ('\0') character), how many characters will be printed by abc(s)?



$$c(i) = 1$$

no. of characters printed.

$$C(n) = 2C(n-1) + 1 \rightarrow (1)$$

$$C(n-1) = 2C(n-2) + 1 \rightarrow (2)$$

$$C(n-2) = 2C(n-3) + 1 \rightarrow (3)$$

$$C(1) = 1$$

$$C(n) = 2^3 \Rightarrow C(n-3) + 2^2 + 2^1 + 1$$

$$\begin{matrix} n-i=1 \\ i=n-1 \end{matrix}$$

$$C(n) = 2^{\overset{\circ}{n}} \cancel{C(n-i)} + 2^{\overset{\circ}{i-1}} + 2^{\overset{\circ}{i-2}} + \dots + 1$$

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

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

$$= 1 \frac{(2^n - 1)}{2 - 1} \quad (\text{G.P})$$

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

$$= 2^3 - 1$$

$$= 8 - 1$$

date 2004
Q3



int rec(int n)

{

if (n == 1)

return 1;

else

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

}

Time complexity = ?

- a) O(n) b) O(n log n) c) O(n²) d) O(2ⁿ) .

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

$$(n-i=1)$$

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

$$T(1) = 1$$

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

$$T(n) = 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 1 = 2^n - 1 \Rightarrow O(2^n)$$

call 2205
84

void foo (int n, int sum)

{

1. int K=0, j=0;

2. if (n==0) return;

3. K = n % 10, j = n/10;

4. sum = sum + K;

5. foo (j, sum);

6. printf ("%d", K);

}

int main()

{

int a = 2048, sum = 0;

foo (a, sum);

printf ("%d", sum);

}

→

main	foo()	foo()	foo()	foo()	foo	foo()
a = 2048	n = 2048	n = 204	n = 20	n = 2	n = 0	n = 0
sum = 0	sum = 8	sum = 8+4	sum = 12	sum = 12+2	sum = 14	sum = 14
	K = 8	K = 4	K = 0	K = 2	K = 0	K = 0
	j = 2048	j = 2040	j = 204	j = 2	j = 0	j = 0

bottom stack

pop

Ans 2, 0, 4, 8, 0 - output

Date: 2005/2010

(85)

```
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};
```

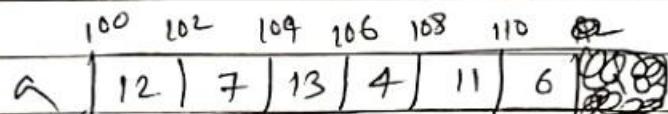
```
    printf("%d", f(a, 6));
```

```
    return 0;
```

{

*

- When function contain many lines, then go with the stack method.
- When function contain only recursive function call then go with the tree method



$f(100, 6)$

↓
next page.

$f(100, 6)$

$$15 \quad 12 + f(102, 5) = 15$$

$$3 \quad 7 - f(104, 4)$$

$$4 \quad 13 - f(106, 3)$$

$$9 \quad 4 + f(108, 2)$$

$$5 \quad 11 - f(110, 1)$$

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

output: 15



- Analysing the recursion program of Towers of Hanoi =

① $\text{TOH}(n, x, y, z)$

{

if ($n \geq 1$)

{

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

Move top 'x' to 'y':

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

}

{

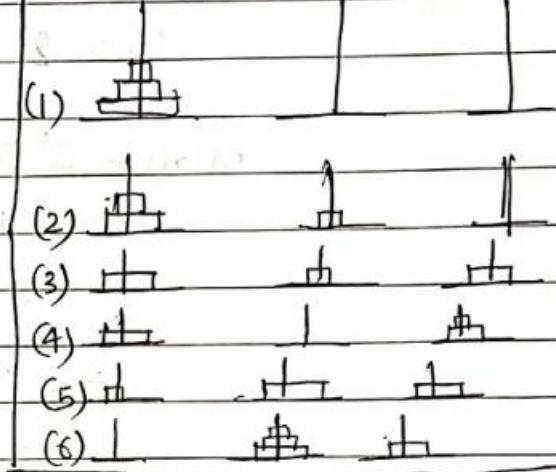
disk $x \rightarrow y$ using z.

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

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

$n \rightarrow y$

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



② function

no. of invocation required for 'n' disk =

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

$$\text{For } 3 \text{ disk, } I(3) = 2^{3+1} - 1$$

$$= 15$$

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

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

$$I(n-1) = 2I(n-2) + 1 \rightarrow (II)$$

$$I(n-2) = 2I(n-3) + 1 \rightarrow (III)$$

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

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

$$= 2^3 I(n-3) + 2^2 + 2^1 + 2^0$$

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

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

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

$$\text{Orp} = 1 \frac{(2^n - 1)}{2-1} = [2^{n+1} - 1] = O(2^n)$$

(3) Total disk movement required for n disk = $M(n) = 2^{n-1}$

for 3 disk = 2^{3-1}

= 7 Movement.

$$M(0) \cdot M(n) = 2M(n-1) + 1$$

$$M(n) = 0, n=0$$

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

$$n-1=0$$

$$M(n-2) = 2M(n-3) + 1$$

$$n=1$$

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

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

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

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

(3) Time complexity required = $T(n) = (2^{n+1}-1)$

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

$$T(n) = 2T(n-1) + 1 \xrightarrow{\text{constant time}}$$

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

$$\begin{cases} n-1=0 \\ i=n \end{cases}$$

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

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

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

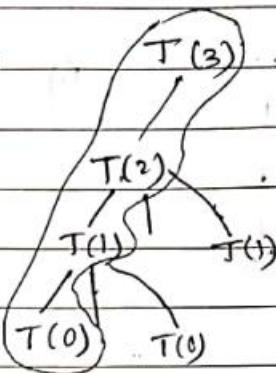
$$= (2^n)2^1 - 1$$

$$= O(2^n)$$

(4) space complexity with 'n' disk = $S(n) = (n+1) = O(n)$.

→ depth of the tree.

$$\boxed{S(3) = 3+1} \\ = 4$$



② $TOH(n, x, y, z) \rightarrow$ modification of previous Algo,
 { if ($n > 1$)

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

 move top 'n' to 'y';

$TOH(n-1, z, y, n);$

}

else if ($n == 1$)

 move "x → y";

}

now no. of function invocation required -

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

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

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

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

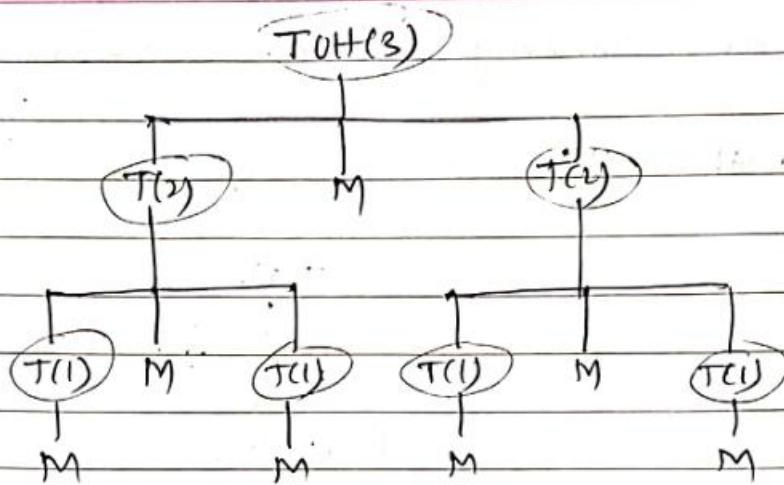
$$n-i=1$$

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

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

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

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



hence
total function
invocation is 7.

GRAPHS

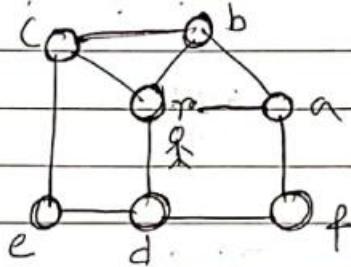
classmate

Date _____

Page _____

- Introduction of Graphs =

→ set of vertices and edges is called graph.



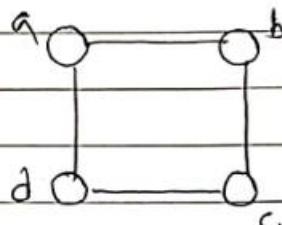
(friend)
- adjacency of $\text{so} - (\underline{\text{dcba}})$
(friend
list)

- Two popular representation of graph =
 - adjacency matrix.
 - adjacency list.

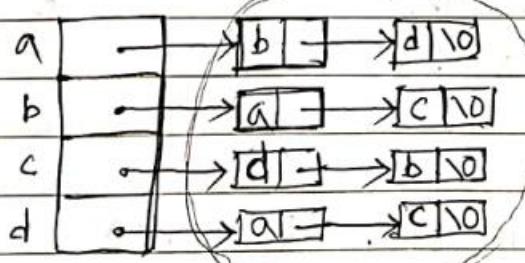
- Adjacency matrix =

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

space required
 $O(v^2)$



- Link list Adjacency list =



space required
 $O(v+2E)$
 $= O(v+E)$.

→ When the graph is Dense (if the no. of edges very very high) then go with matrix representation.

$$E = O(v^2)$$

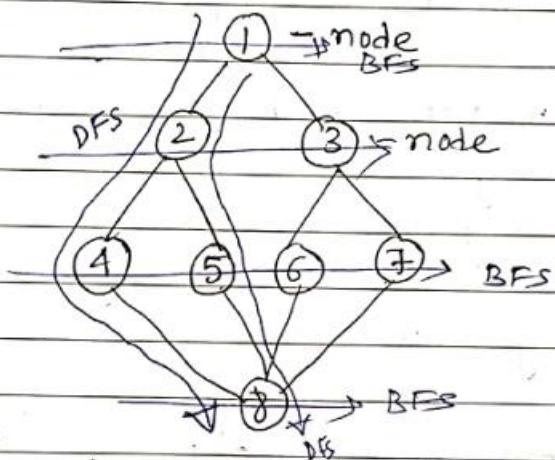
→ When the graph is sparse (few edges), then go with edge-adjacency list representation.

$$E = O(v)$$

- Introduction of BFS and DFS =

↓ ↓

breadth depth
first first
search search



Search all the node in the given graph by using Breadth first search and depth first search.

vertices 3-types -

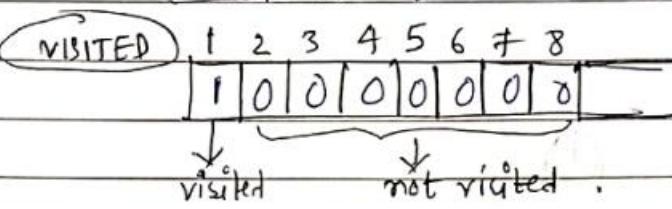
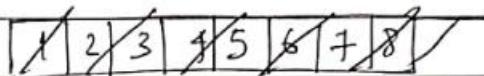
→ visited, not-visited, Explorad.

↓
seen that
vertex

↓
seen it and road
seen all vertex adj
with ~~to~~ that vertex.

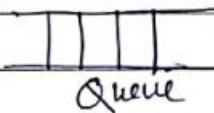
	<u>VISITED</u>	<u>EXPLORED</u>	
case-1	0	0	→ not visited, not explored.
c-2	01	0	→ visited, but not explored.
c-3	1	1	→ visited and explored.

→ Keep track of this two things (~~visited, Explored~~) both the algorithm maintain an array -



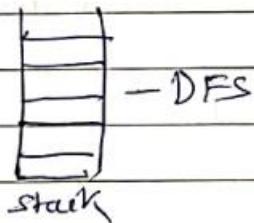
✓ space com = $O(n)$.

→ If the Algo which use the queue to keep track of all unexplored vertex is called - **BFS**.



— BFS ✓ space complexity = $O(n)$

→ The algo which use stack to keep track of all unexplored nodes is called - **DFS**.



✓ space complexity = $O(n)$.

• BFS algorithm = $\text{BFS}(v) \rightarrow$ address of 1st node

// The graph 'G' and array `visited[]` are global;
`visited[]` is initialized to 0.

{
 $u = v$; `visited[v] = 1`;

repeat

{

 for all vertices w adj to u

 { if (`visited[w] == 0`)

 { add w to queue; $\uparrow \text{pf}(w)$
 `visited[w] = 1`; } }

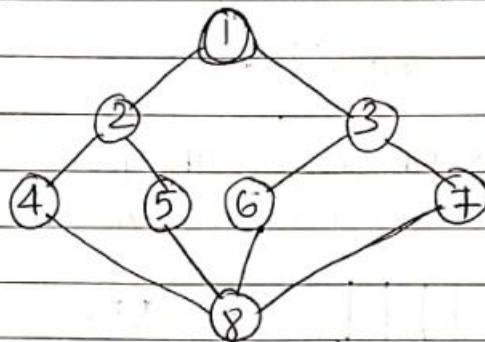
if queue is empty then return;

Delete the next element, v , from queue and add to U ;

$\{$

$\}$

example =



array	1	2	3	4	5	6	7	8
	∅	2	∅	4	5	6	7	∅

$U = \{X, 2, 3, 4, 5, 6, 7, X\}$ \rightarrow now all vertices visited

$v = 1$

$w = 2, 3$ ✓

$v = 2$

$w = 1, 4, 5$ ✓

$v = 3$

$w = 1, 6, 7$ ✓

Front Queue

X	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

$v = 4$

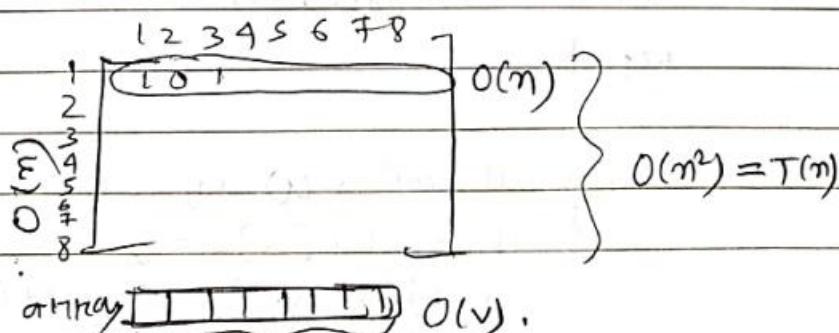
$w = 2, 8$

BFS analysis on adjacency matrix implementation =

$$\text{Time complexity} \Rightarrow T(n) = O(n^2) \\ = O(v^2)$$

$v \rightarrow \text{vertices}$

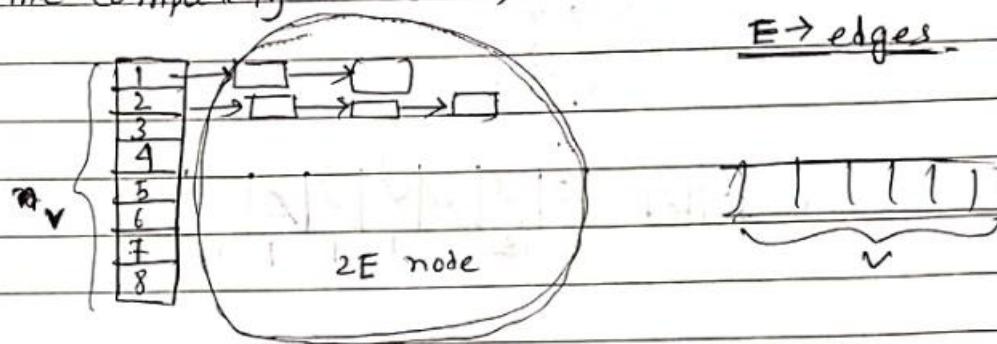
Space Complexity = $O(v)$



- BFS analysis in case of linked list implementation of graph =

✓ Space complexity in worst case = $O(n)$
 $= O(N)$

✓ Time complexity = $O(V+E)$



- Breadth First Traversal using BFS =

BFT (G, n)

{

for $i=1$ to n do

 visited [i] = 0

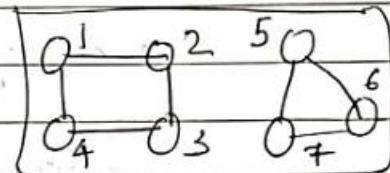
 for $j=1$ to n do

 if (visited [j] == 0) then

 }

$i = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8$
 visited $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

 BFS (i);



Time complexity = $O(n^2) \ O(E + V)$

Space complexity =

→ Time and space complexity of BFT are same as BFS.

- DFS algorithm =

DFS (v)

{

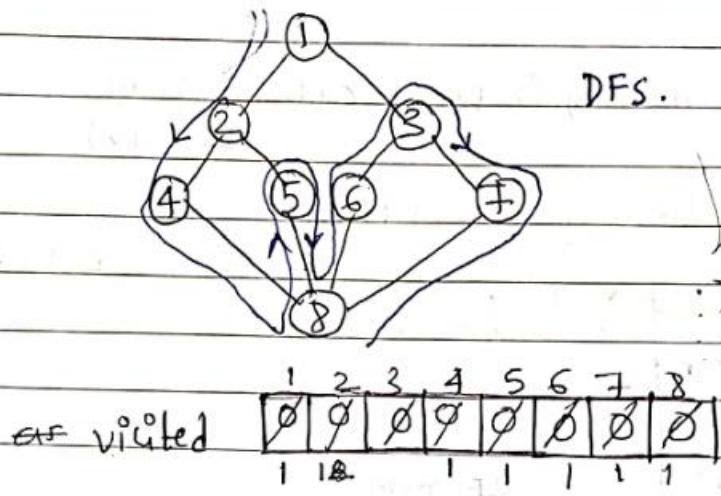
 visited [v] = 1;

 for each vertex w adj to v do

 if (visited [w] == 0) then

 DFS (w);

}

Example -

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

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

- Analysis of DFS and DFT =

→ In case of Adj^o Matrix -

Time complexity = $O(v^2)$

Space complexity = $O(v)$

→ In case of Adj List -

Time complexity = $O(v+E)$

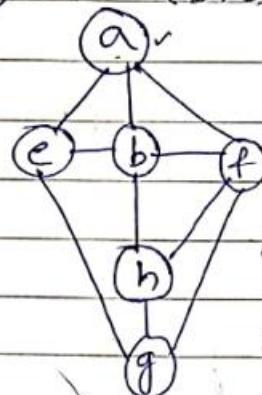
Space complexity = $O(N)$.

→ Space and Time complexity are same in both the case of DFS and DFT traversal.

g-2003

Question - 1

(DFS) - depth first search -



✓ I. abeghf. (possible)

✗ II. abfehg. (not possible)

✓ III. abfhge. (possible)

✓ IV. afghbe. (possible)

Which of the following sequence
does not possible?



$v=a$	$v=e$	$v=b$
$w=\{e,b,f\}$	$w=\{a,b,g\}$	

I.

$v=a$	$v=b$	$v=e$	$v=g$	$v=h$	$v=f$
$w=\{e,b,f\}$	$w=\{a,e,f,h\}$	$w=\{a,b,g\}$	$w=\{e,f,g\}$	$w=\{b,g\}$	$w=\{a,b,h,g\}$

abeghf

II & III.

$v=a$	$v=b$	$v=f$	$v=h$	$v=g$	$v=e$
$w=\{e,b,f\}$	$w=\{a,e,f,h\}$	$w=\{a,b,g\}$	$w=\{b,f,g\}$	$w=\{e,h,f\}$	$w=\{a,b,g\}$

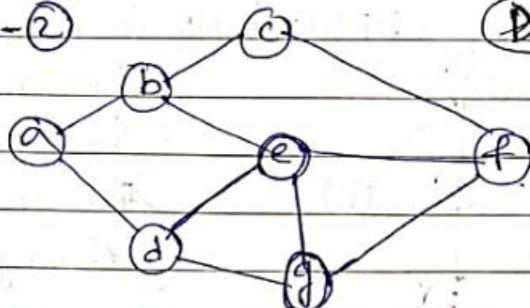
IV.

$v=a$	$v=f$	$v=g$	$v=h$	$v=b$	$v=e$
$w=\{e,b,f\}$	$w=\{a,b,h,g\}$	$w=\{e,h,f\}$	$w=\{b,f,g\}$	$w=\{a,e,v\}$	$w=\{a,b,g\}$

gak
2008

Question - ②

DFS



which of the following sequence are possible -

X¹) abefdg.c. (not possible)✓²) abefcg.d.✓³) adgebcf.X⁴) adbcgef. (not possible)

→

D²)

$v=a$	$v=b$	$v=e$	$v=f$	$v=c$	$v=g$
$W=\{b, d\}$	$W=\{a, c, e\}$	$W=\{b, d, f\}$	$W=\{c, e, g\}$	$W=\{b, f\}$	$W=\{a, e, g\}$

3)

→

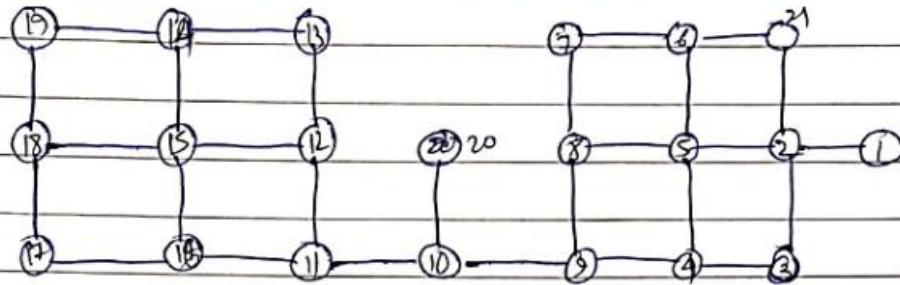
$v=a$	$v=d$	$v=g$	$v=e$	$v=b$	$v=c$
$W=\{b, d\}$	$W=\{a, e, g\}$	$W=\{a, e, g\}$	$W=\{b, d, f\}$	$W=\{a, c, e\}$	$W=\{b, f\}$

4)

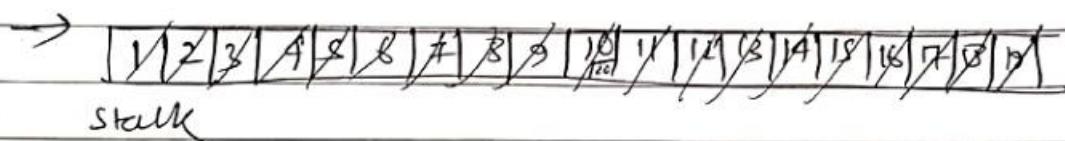
→

$v=a$	$v=d$
$W=\{b, d\}$	$W=\{a, e, g\}$

g-2014
[Question]-③ (DFS)



start visiting from any node - , and find out how many ^{max} node will prevent at worst at a time in stack .



(19 node can prevent at same time on the stack)

gate
2006

[Question]-④

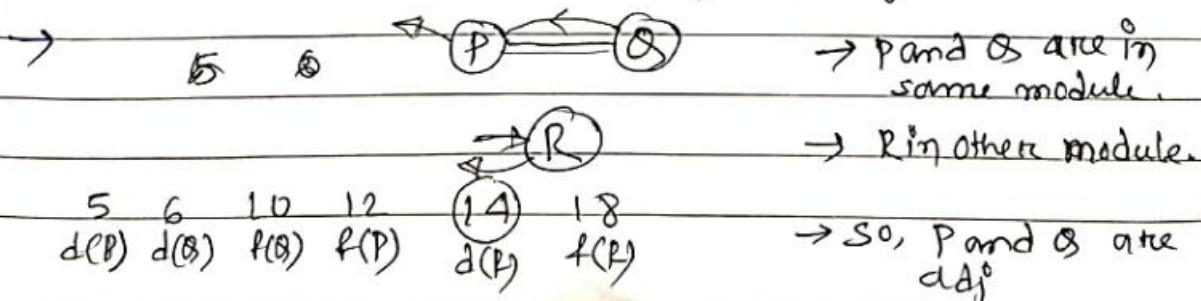
consider a DFS of an undirected graph with 3 vertices P, Q, R . let discovery time $d(u)$ represent the time instant when the vertex 'u' is first visited, and finish time $f(u)$ represent the time instance when the vertex 'u' is last visited given that -

$$d(P) = 5 \text{ units} \quad f(P) = 12 \text{ units}$$

$$d(Q) = 6 \text{ units} \quad f(Q) = 10 \text{ units}$$

$$d(R) = 14 \text{ units} \quad f(R) = 18 \text{ units}$$

What is true about the graph ?



Hashing

- Direct Address Table =
(DAT)

Searching time of some DS -

- (i) unsorted array - $O(n)$
- (ii) sorted array - $O(\log n)$
- (iii) linked list - $O(n)$.
- (iv) Binary tree(BT) - $O(n)$
- (v) Binary search tree - $O(n)$
- (vi) Balanced BST - $O(\log n)$.
- (vii) priority queue(min & max heap) - $O(n)$.

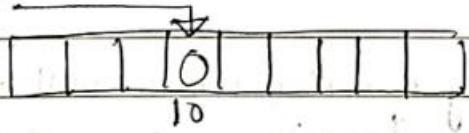
→ If we used hashing technique then on average time taken to search an element is - $O(1)$.

Hashing is used to minimize the search time.

Before using hashing technique people used one more technique called as DAT (direct address table).

DAT: It is similar to Array.

Key - {1, 2, 3, ..., 100}



→ The first DS which has supported the searching in - $O(1)$ is called DAT (direct address table). In this table we are have a array and we are going to place an element according to key value.

→ This method is useful only when you know the no. of keys which are going to in small range and

if the

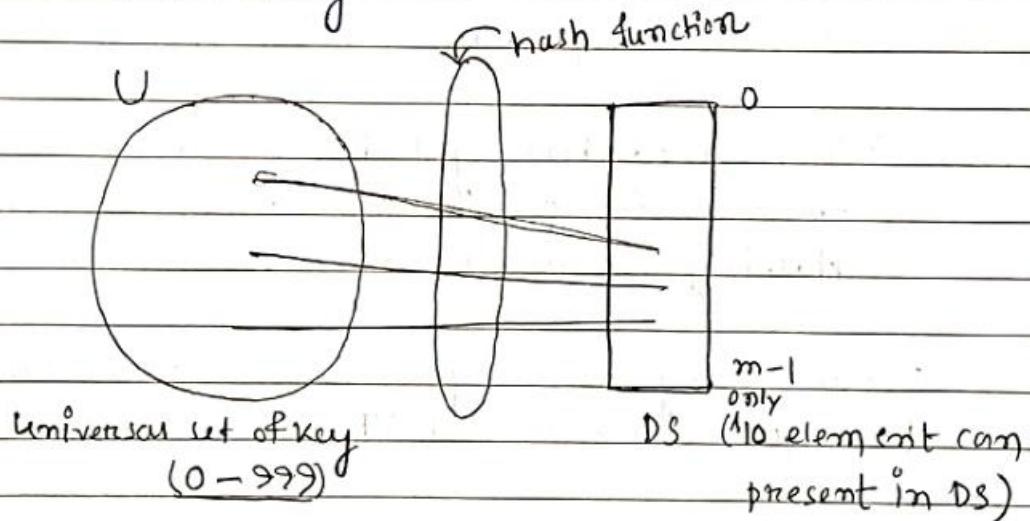
In case keys used are very large numbers, but the total no. of keys are less in number.

Key - {100000, 100001, 100002, ...}

	100000	100001	...
--	--------	--------	-----

so, that the array size will be 1M, just to store 10 values. In such case DAT fail. (Waste of space).
→ solve this problem we use hashing.

- Introduction to hashing =



hash function :

$$h(U) \rightarrow (0-(m-1))$$

$$h(0-999) \rightarrow (0-9)$$

hash function is any function which will take a key from U and it will map it to one of the values from 0 to $m-1$.

example :

(0-999)

set of Key - (121, 145, 132, 999)

hash function - ($\text{mod } 10$)

0	1	2	3	4	5	6	7	8	9
NULL	121	132	NULL	N	145	N	N	N	999

(size-10)

When searching for an element - 150, 145

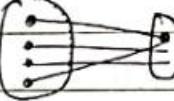
$150 \% 10$
= 0

element is not present.

$145 \% 10$
= 5 element is present.

→ In hashing technique, there is a problem called deletion and insertion, and searching is going to take constant time by using the hashing technique.

→ In hashing technique, there is a problem called collision.

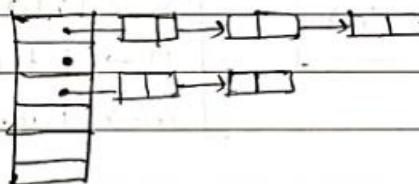
→ Collision problem = 

→ When two key or two element being hash into same cell that condition is called collision.

→ Solving collision problem is -

(i) Better hash function.

(ii) Chaining.



(For deletion chaining is better)

(iii) Open addressing. (i,s)

(probing - searching for an empty space to insert an element)

(i) Linear probing
.....
.....

(For deletion open addressing is not better)

(ii) Quad probing
.....
.....

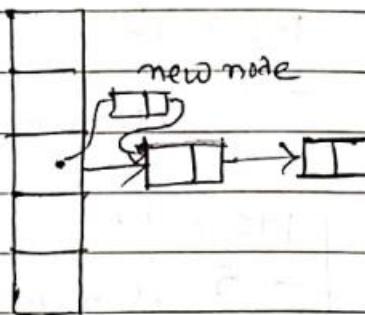
(iii) Double hashing

- Collision solving technique : chaining.

$K \rightarrow$ Key value

$T[h(K)]$

T



Time taken

hash table

→ Insertion In case of chaining time taken = $O(1)$.

→ In worst case time for searching = $O(n)$.

(n element inserted in the table, all the element might map $\Theta(m)$ in worst case to a single entry (cell) and the list of entry might contain as many as n)

→ In worst case Deletion time = $O(n)$.

$$\alpha = \frac{n}{m}$$

$\alpha \rightarrow$ load factor.

$n \rightarrow$ total element present in table.

$m \rightarrow$ total size of table

✓ In this case average search time = $\Theta(1 + \frac{n}{m})$

$$n = K m^{\text{constant}}$$

$\rightarrow \Theta(1 + \alpha)$

$$\frac{n}{m} = K(\alpha)$$

$\rightarrow \Theta(1)$

Average deletion time = $\Theta(1)$.

↳ Insertion ↳

Adv.: Adv.

→ deletion is easy in chaining.

disadv.:

→ pointers (space wasted).

g-1996
9-2014

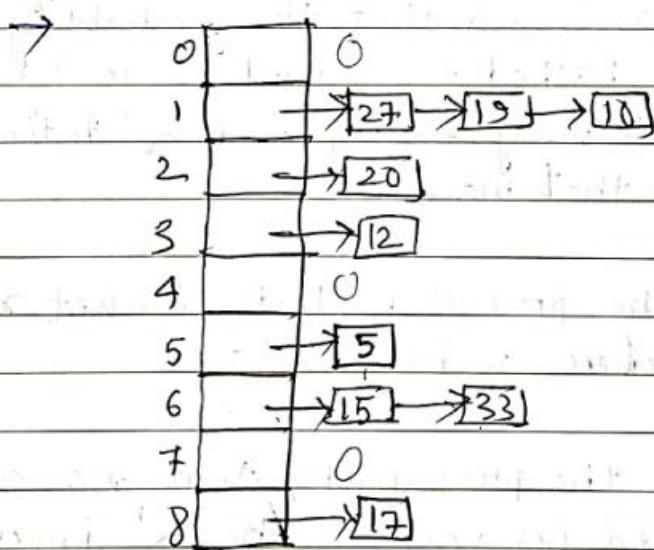
Question-①

An advantage of chain hash table over open addressing scheme is -

- worst case complexity of search is less.
- space used is less.
- deletion is easier.
- none of the above.

g-2014
Question-②

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



✓ max chain length → 3

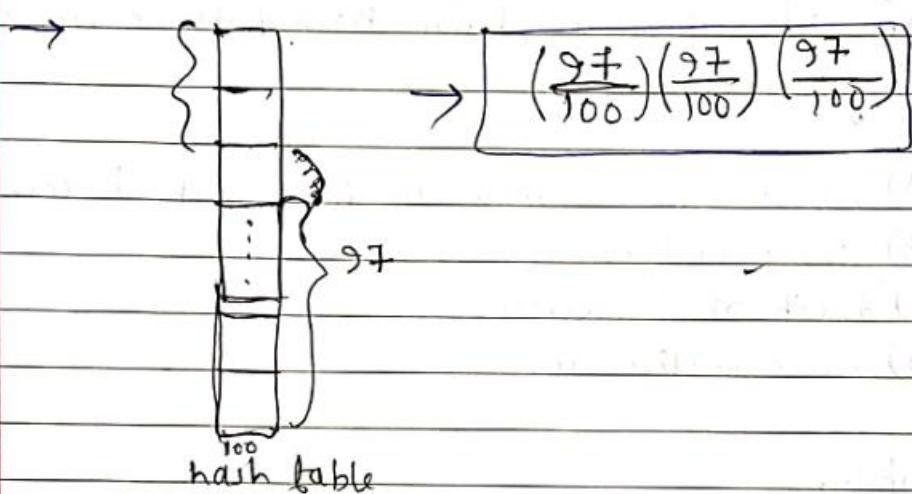
✓ min " " → 1

✓ avg " " → $\frac{0+3+1+1+0+1+2+0+1}{9} \rightarrow 1$.

gate-
2014

[Question] - ③

consider a hash table with 100 slots. collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions?

gate
1997

[Question] - ④

consider a hash table with 'n' buckets 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 $\frac{1}{n}$. The hash table is initially empty and 'K' distinct values are inserted in the table.

a) What is the probability that bucket no. 1 is empty after K insertions.

b) What is the probability that no collision has occurred in any of the 'K' insertions?

c) What is the prob that first collision occurs at the Kth insertion.

(a) $\rightarrow \gamma_n$

$$(1 - \gamma_n)$$

$$= \left(\frac{n-1}{n}\right)$$

$$\left(\frac{n-1}{n}\right) \left(\frac{n-1}{n}\right) \cdots \left(\frac{n-1}{n}\right) = \left(\frac{n-1}{n}\right)^K.$$

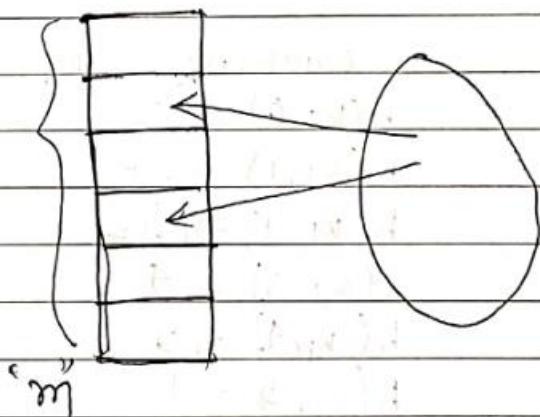
(b) $\left(\frac{n}{n}\right) \left(\frac{n-1}{n}\right) \left(\frac{n-2}{n}\right) \cdots \left(\frac{n-(K-1)}{n}\right)$
 1st ele : 2nd ele 3rd ele ... Kth element.

(c) $\frac{(K-1)}{n}$

* Collision Solving Technique: [Open addressing]
 or (closed hashing)

$$\text{load factor, } \alpha = \frac{n}{m}$$

$$0 \leq \alpha \leq 1$$



$$h: U \rightarrow \{0, 1, \dots, m-1\}$$

$$h: (U \times \{0, 1, 2, \dots, m-1\}) \rightarrow \{0, 1, \dots, m-1\}$$

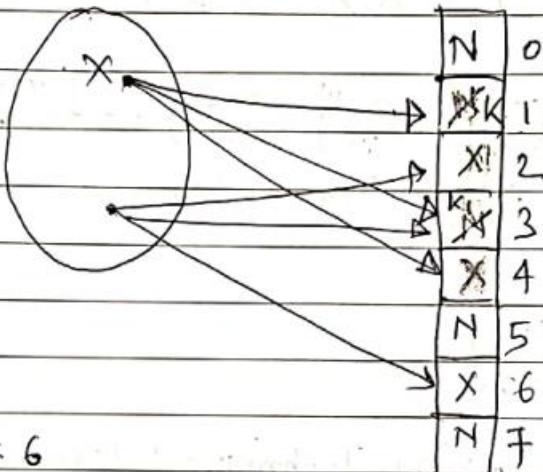
probe Sequence:

Whenever I insert a key then follow a sequence of examinations, In case if you find any empty cell in this sequence you are going to insert the element there, if you don't find any sequence empty cell in that sequence then declared

that entire ^{table} is full, you are not able to insert.

worst time take to searching - $O(m)$

X



$$h(K) = 1$$

$$h(K_1) = 2$$

$$h(K_1, 1) = 6$$

$$h(K_1, 2) = 3$$

Searching for K_1

$$h(K_1, 0) = 2$$

$$h(K_1, 1) = 6$$

$$h(K_1, 2) = 3$$

$$h(K_1, 3) = 4$$

$$h(K_1, 4) = 5$$

$$h(K_1, 5) = 1$$

$$h(K_1, 6) = 7$$

$$h(K_1, 7) = 0$$

→ In case of full hash table and you are going to unsuccessful search the time complexity for a search is - $O(m)$.

* In Average search time complexity - $O(1)$.

- Insert an new element K_1 ,
- $h(K_1, 0) = 2 \checkmark$
 - $h(K_1, 1) = 6 \checkmark$
 - $\checkmark h(K_1, 2) = 3 \checkmark$
 - $h(K_1, 3) = 4 \checkmark$
 - $h(K_1, 4) = 5 \checkmark$
 - $h(K_1, 5) = 1$
 - $h(K_1, 6) = 7$
 - $h(K_1, 7) = 0$

0	N
1	N
2	a
3	c
4	d
5	K ₁ b
6	b
7	N

→ We are going to stop & either when we find the searching element or empty cell.

Then Delete and an element 'c' from table.
after deleting c search for

K_1 .

0	N
1	N
2	a
3	N
4	d
5	K ₁
6	b
7	N

→ Stop searching here.
and declare
that K_1 is not
present even
when K_1 is
present.

→ In case of deletion open addressing
Create such a problem.

To solve this problem =

We are going to mark deleted place in such a
way that it will indicate that some element
are earlier there but now it is deleted.

0	N
1	N
2	✓
3	D
4	d
5	(K1)
6	b
7	N

D indicate that, earlier there was an element but now it deleted.

- Various technique which used in Open addressing =

(i) Linear probing :

hash function =

$$h: U \rightarrow \{0, \dots, m-1\}$$

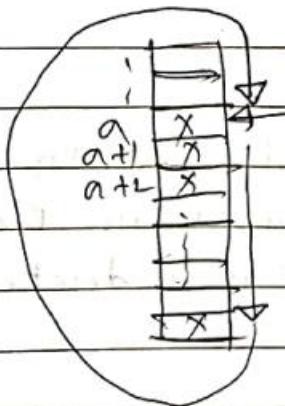
$$h(k) = a$$

$$h'(k, i) = (h(k) + i) \bmod m$$

$$\begin{aligned} h'(k, 1) &= (h(k) + 1) \bmod m \\ &= (a+1) \bmod m. \end{aligned}$$

$$\begin{aligned} h'(k, 2) &= (h(k) + 2) \bmod m \\ &= (a+2) \bmod m. \end{aligned}$$

$$h'(k, 3) = (h(k) + 3) \bmod m = (a+3) \bmod m$$



Probe sequence →

$(0, 1, 2, 3, \dots, m-1)$ always.

$(1, 2, 3, \dots, m-1, 0)$ search for

$(2, 3, 4, \dots, m-1, 0, 1)$ 'm' element

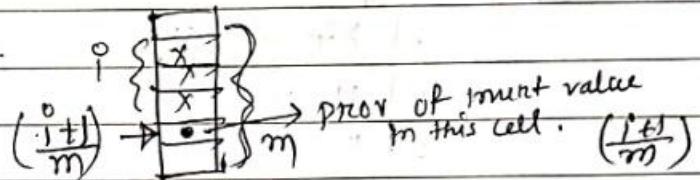
$(3, 4, 5, \dots, m-1, 0, 1, 2)$ to insert

element.

so, probe sequence is possible is "m".

Secondary clustering =

In case of two elements first probe is found in same value then both of them follow going to follow same probe sequence, this call secondary clustering.

Primary clustering =

disadvantage: Linear probing suffer two things (secondary and primary clustering)

Cverage search time taken to search an element = $O(2.5)$.
Worst case = $O(n)$.

gate 2008

Question - ①Key: 12, 18, 13, 2, 3, 23, 5 and 15. $HST = 10$ (hash table size) $h(k) = k \bmod 10$ and linear probing. What is resultant hash table?

→

0	12	\leftarrow get into this $(\frac{9}{10} \text{ prov})$
1		
2	12	$h(k_i) h(i, 0) = (h(k) + i) \bmod 10$
3	13	$h(13, 0) = (h(12) + 1) \bmod 10$
4	2	$= (2 + 1) \bmod 10$
5	3	$= 3$
6	23	$h(23, 1) = (2 + 2) \bmod 10 = 4$
7	5	
8	8	
9	15	

Primary cluster.

gate
2008[Question] - ②

$$HTS = 11 \quad (0-10) \quad \text{using}$$

$h(k) = k \bmod 11$, ^{1°} linear probing method.

Keys : 43, 36, 92, 87, 11, 4, 71, 13, 14.

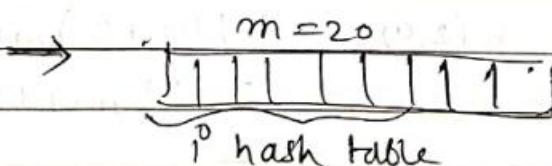
What is the index into the last record is inserted?

0	87	→ 11
1	11	
2	13	
3	36	→ 14
4	92	→ 9
5	4	→ 71
6	71	
7	14	
8	4	→ probability that this cell get next insertion
9		
10	43	→ 87

gate
2007[Question] - ③

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

- a) 5 b) 6 c) 7 d) 10.



$$\frac{i}{m} > 0.5$$

$$\frac{i}{20} > 0.5 \Rightarrow i > 10$$

gate
2010Question - ④ $h(k) = k \bmod 10$, linear probing.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

A)

A) After insertion, which will be look like give sequence -

x a) 46, 42, 34, 52, 23, 33x b) 34, 42, 23, 52, 33, 46✓ 46, 34, 42, 23, 52, 33x d) 42, 46, 33, 23, 34, 52.

B) How many insertion sequences are possible, to get the give table sequence -

 $\rightarrow \underbrace{(42-23-34)}_{3!} - 52 - 33$

46 can place take place any of the blank space.

 $3! \times 5 = 30$ (no insertion) sequences possible.

(ii) Quadratic probing =

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

Ex :-

$m = 10$ (size of hash table).

(0 --- 9)

$c_1 = 1, c_2 = 1$

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

0		
1	X	←
2		
3	X	←
4		
5		
6		
7	X	
8		
9		

$$h(k_1) = 1$$

$$h'(k_1, 1) = 1 + 1 \cdot 1 + 1 \cdot 1^2 \\ = 3$$

$$h(k_1, 2) = 1 + 2 + 4 \\ = 7$$

$$h(k_1, 3) = 1 + 3 + 9 \\ = (13) \bmod 10 = 3$$

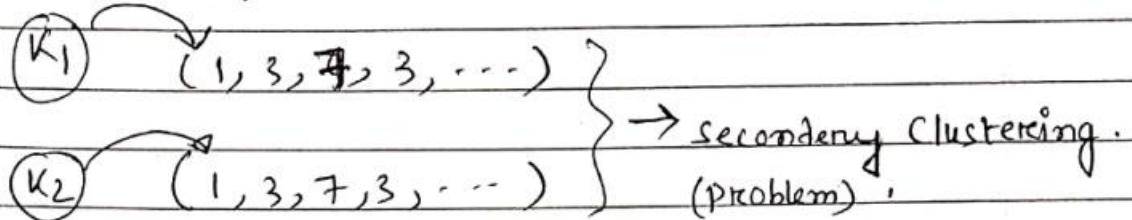
$$h(k_1, 4) = 1 + 4 + 16 = 21 \bmod 10 \quad "m"$$

after 10 probs = 1

In this case you are not able to examine all the location by using 10 probes of hash table. This is the problem with quadratic probing.

You should choose c_1, c_2 and m value very well. Carefully. Otherwise it will lead problems.

probing sequence -



→ If two keys happen to have the same initial probe location then they are going to have similar probe sequence.

two key's

→ A probe sequence will same when Initial probe is same.

different

→ The no. of A probe sequence possible is - "m" -
| (every probe sequence depends on initial probe sequence)
| "m" probe sequence possible.

best probing (iii) [Double hashing]
technique.

In Double hashing no. of probe sequence is possible is $= m^2$
(where " m " → size of hash table) $= O(m^2)$

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

→ there are no secondary and primary clustering problems.

$\begin{matrix} < m \\ \text{(i)} \\ \uparrow & \uparrow \\ h_2(k) & , m \\ \downarrow & \downarrow \\ \text{odd} & 2^k \end{matrix}$
prime.

→ hashing is more better compare to other DS. when the most popular operation is ^{only} searching and Data is static).

atlantis

Date _____
Page _____