WWW.GATENOTES.IN

GATE Hand written Notes

DATA STRUCTURES

GATE CSE NOTES

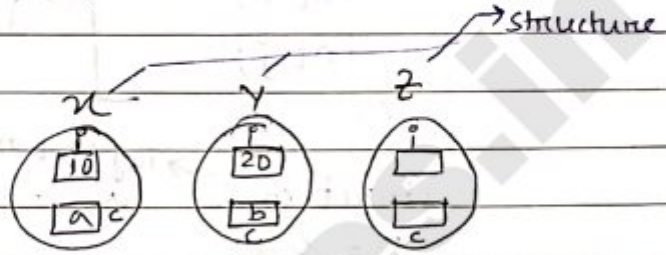
For more visit: WWW.GATENOTES.IN

Structures

- Introduction of Structure:

```
struct
{ int i;
  char c;
} x, y, z;
```

→ declaration



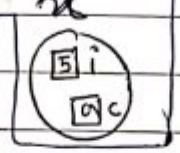
x.i = 10 | y.i = 20
x.c = 'a' | y.c = 'b'

→ member operaters

```
struct ex → Tag
{ int i;
  char c;
};
```

struct ex x, y, z;

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

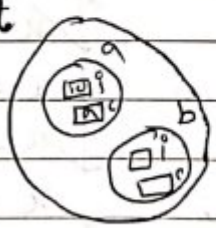



```
struct ex1 *
{
  struct ex a;
  struct ex b;
};
```

→ declaration.

struct ex1 t;

t.a.i = 10
a, a.c = 'a'
→ define.



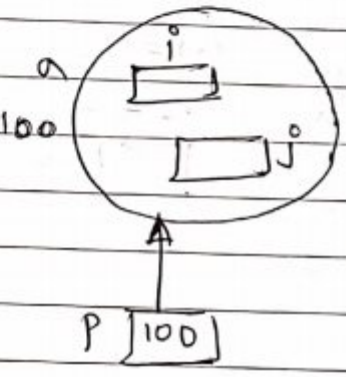
www.gatenotes.in

classmate
Date
Page

- Example on Structures, arrays and pointers —

```
struct node
{ int i;
  int j;
};
```
→ declaration (no memory allocated)

```
struct node a, *p;
p = &a;
```
node → Tag (not necessary)



(*p).i → access of member of structure using pointer.

a.i → access by name of structure.

p→i same as (*p).i

→ structures can be pass to a function as well as return by a function.

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

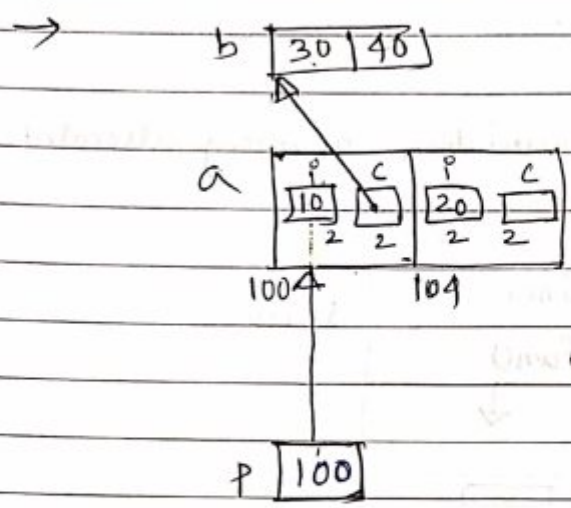
Example:

```
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;
```

- ++p→i
- x = (++p)→i
- x = (p++)→i
- x = *p→c
- x = *p→c++
- x = (*p→c)++
- x = *p++→c.

www.gatenotes.in

classmate
Date
Page



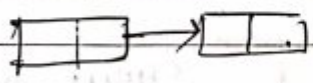
- $x = (++(p \to i))$ / $x = 11$
- $x = (++p) \to i$ / $x = 20$
- $x = (p++) \to i$ / acts $p \to i$ means $p = p+1$ / $x = 10$
- $x = (*(p \to c))$ / $x = 30$
- $x = (*(p \to c))++$ / $x = 30$ (For post increment $x = 31$).
- $x = (*p \to c)++$ / $x = 30$.
- $x = (*(p++) \to c))$ / $x = 30$

- **Self referential structures:**

```
struct ex
{ int i;
  struct ex *link;
};
struct ex abc;
```

example of self referential structure =

(1) Link list.



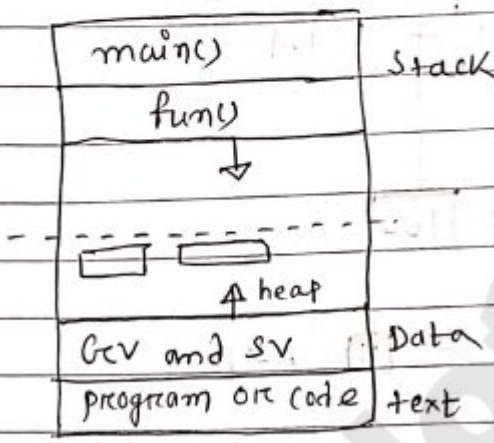
(2) tree

www.GateNotes.in

classmate
Date
Page

Malloc =

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



10

```
void * malloc (int);
```

malloc function call make a space in Dinamicly in the heap of process and then return the starting address of this location.

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

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

```
int*p=(int*) malloc (size of (2))
```

↳ type casting

syntax (which we to make struct and get pointer)

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

```
struct node* p = (struct node*) malloc(sizeof(struct node));
```

```
malloc(sizeof(struct node));
```

www.gatenotes.in

classmate
Date
Page

www.GateNotes.in

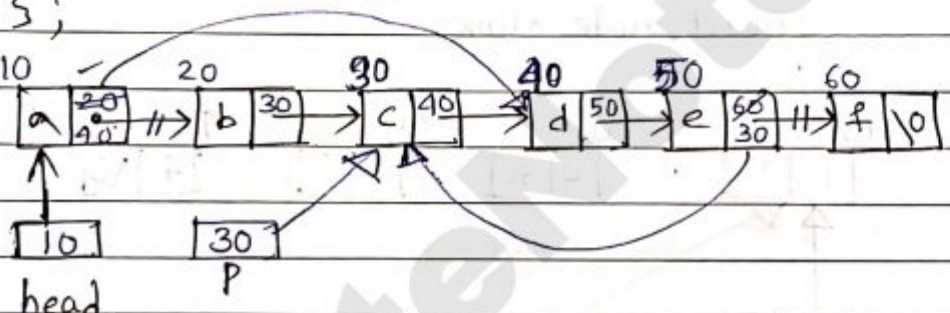
www.gatenotes.in

Linked-List

- Introduction of single linked list :

→ single linked list contain nodes, and this node generaly structured ~~is~~ and which are self ~~Referrential~~ Referrential.

```
struct node
{
  char data;
  struct node *link;
};
```



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

struct node *head;

→ linklist are sequential access.

→ going to any particular number structure (link list) take O(n) time.

```
struct node *p;
p = head → link → link ;
p → link → link → link = p;
head → link = p → link ;
pf ("%c", head → link → link → link → link → data);
```

Output : "d"

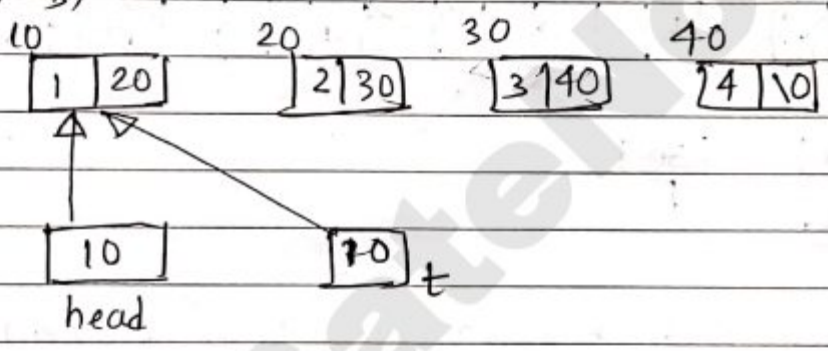
www.gatenotes.in

Traversing of LL

In link list generally perform three operations —

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

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



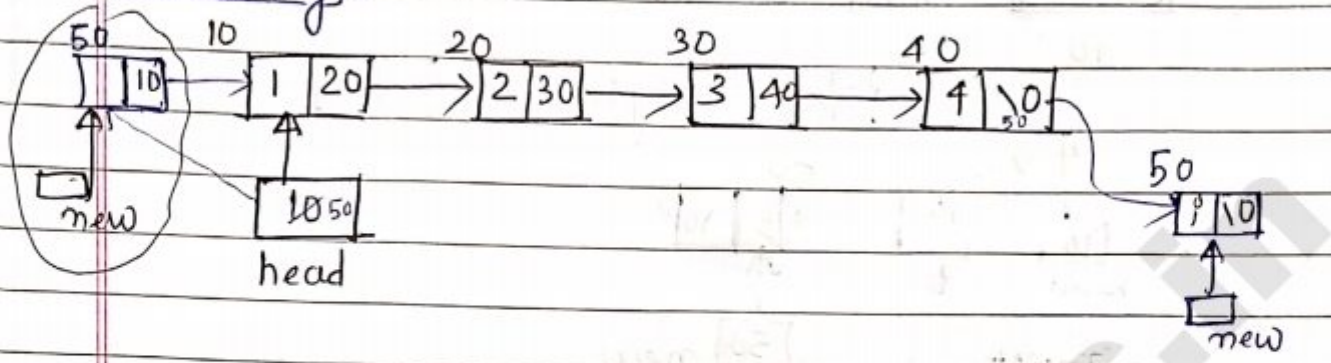
① Traversing :

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

while (t != NULL)   ≡ while (t)
{printf ("%d", t->i)

  t = t->link. }
```

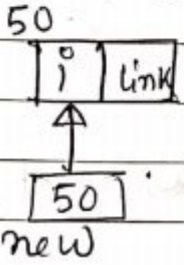
output : 1 2 3 4

www.Gatenotes.in

② Inserting :




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



Case-1 Insert at the ~~bg~~ beginning =

```
new → link = head
head = new
```

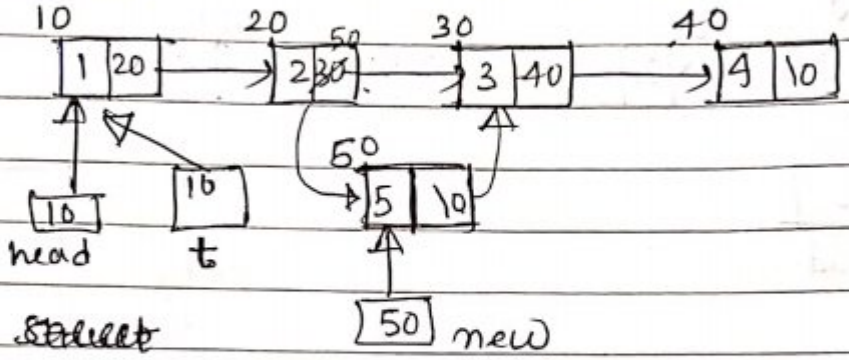
Case-2 Insert at the end =

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

www.gatenotes.in

classmate
Date
Page

Case-3 Insert at the middle.



Struct node *t; = head;

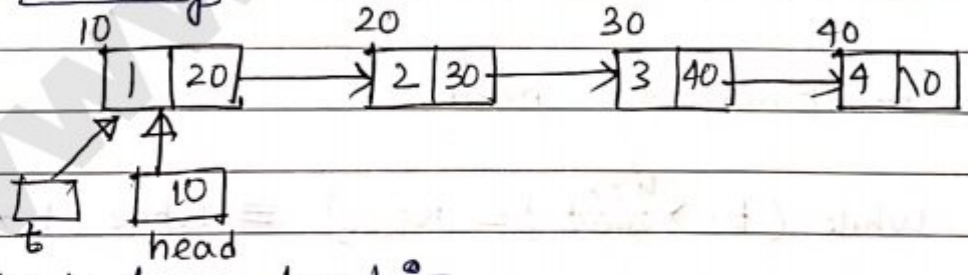
while (t → info != 2)

{ t = t → next; (link) }

t → link = new → link

new → link = t;

③ **Deleting**:



- Delete from head :-

Struct node *t = head;

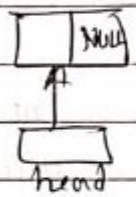
head = head → link; (to delete first one)

free(t);

www.gatenotes.in

Condition Checking =

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

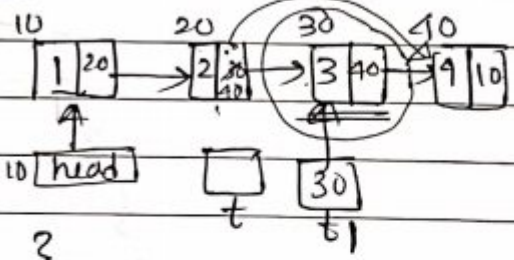


before deletion check this (head are null or not)
(head → link is null or not)

- delete from tail :

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

free (t → list)
t → list = null
```

- delete node which contain i = 3 :




```
struct node *t = head;

While (t → list → i != 3)
{ t = t → list; }

struct node *t1 = t → list;

t → list = t → list → list;

Free (t1);
```

www.gatenotes.in

Gate-2005-2018

Question - 1

```
struct node
  {
    int val;
    struct node *next;
  };

void rearrange (struct node *list)
  {
    struct node *p, *q;
    int temp;
    if (!list || !(list->next)) return;
    p = list; q = list->next;

    while (q)
    {
      temp = p->val; p->val = q->val;
      q->val = temp; p = q->next;
      q = p ? p->next : 0;
    }
  }
```

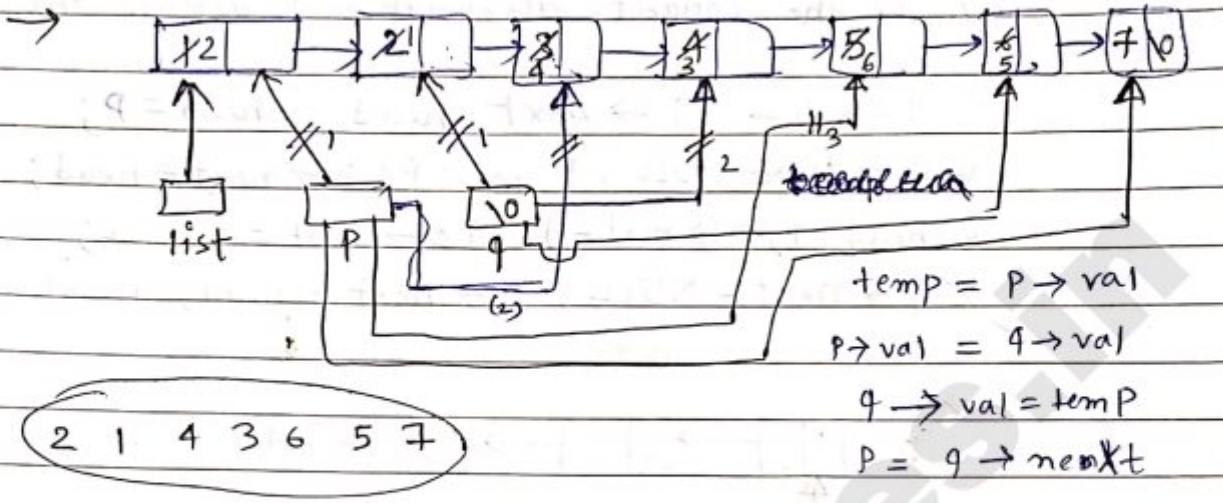
list == NULL ≡ !list

while (q) ≡ while (q != NULL)

1 2 3 4 5 6 7

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

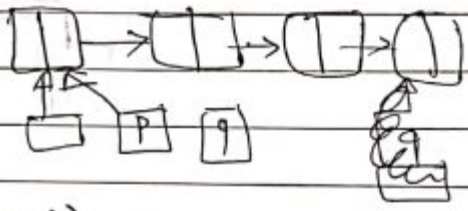
www.GateNotes.in



Gate-2010

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 = NULL;
  p = head;
  While (p → next != NULL)
  { q = p;
    p = p → next;
  }
  ____________ (blank)
  return head;
}
```

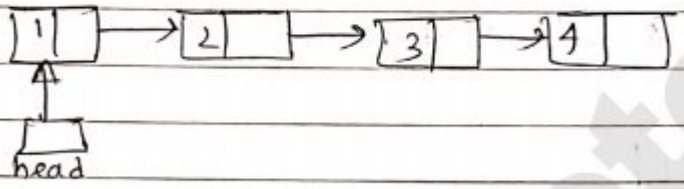
www.GateNotes.in

Choose the correct alternative to replace the blank

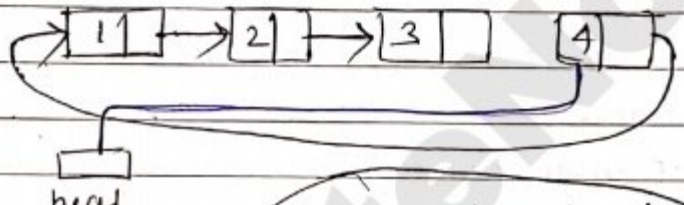
a) q=NULL; p→next=head; head=p;
b) q→next=NULL; head=p; p→next=head;
c) head=p; p→next=q; q→next==NULL;
d) q→next=NULL; p→next=head; head=p;

→

(I)



(II)

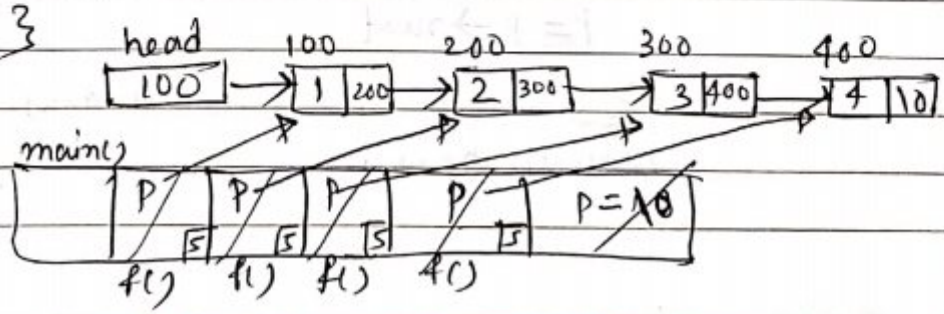


p→next = head
head = p
q→next = NULL

- printing the elements of single linkedlist using recursion

```
① void f(struct node *p)
  {
    if (p)
    { printf("%d", p→data);
      f(p→link)
      {
  }
```

→

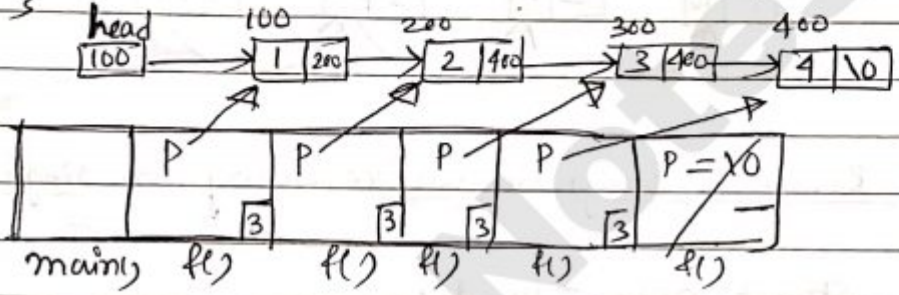


output: 1 2 3 4

www.gatenotes.in

(2)

```
void f(struct node *p)
{
1   if (p)
    {
2       f(p->link);
3       printf("%d", p->data);
    }
}
```



o/p: 4 3 2 1

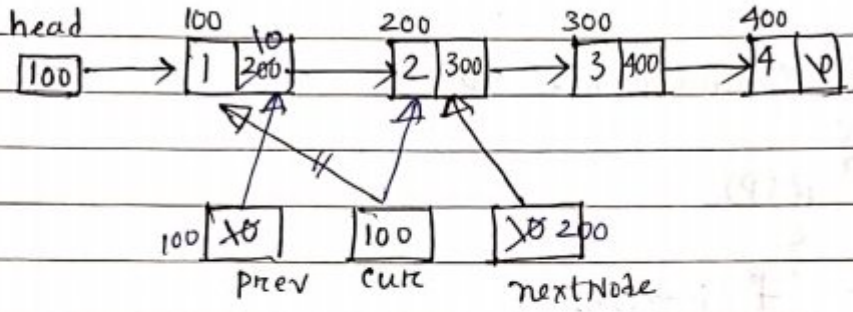
- Reversing an single linkedlist using iteration program

```
main(){
    head = reverse(head);
}
```

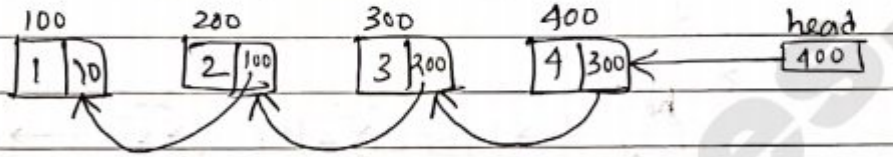
(1)

```
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; }
```

www.gatenotes.in



after operation

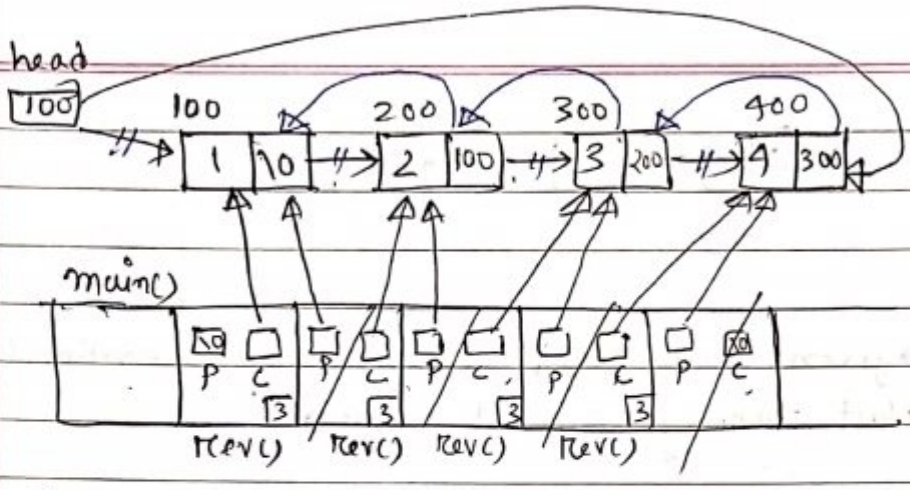


- Recursive program for reversing an single linked list =

```
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);
}
```

www.GateNotes.in

classmate
Date
Page



[GATE-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 -

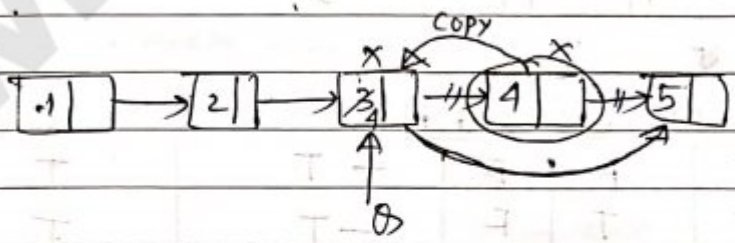
(A) $\log_2 n$ (B) $n/2$ (C) $\log_2 n - 1$ (D) n.

[GATE-2004]

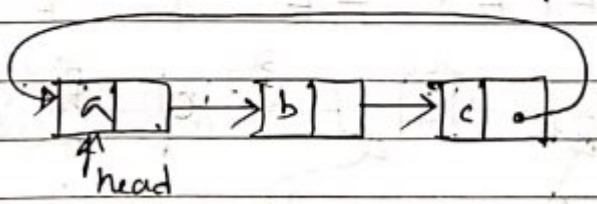
Question-4

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?

→ O(1).

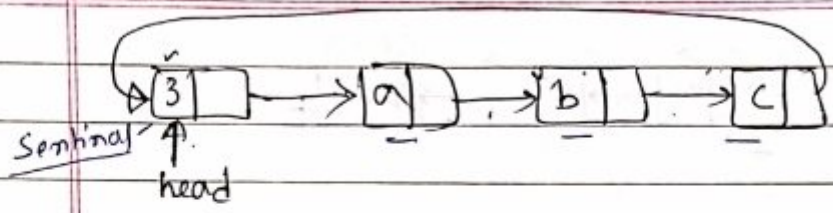


- Circular linked list =




```
p = head;
while (p->next != head)   → to find the end of the list.
{ : }
```

www.gatenotes.in

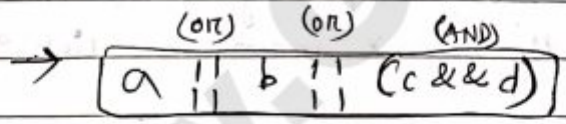


Adv:→ given any point we can search entire list.
→ last pointer are not wasted.

Q-2003

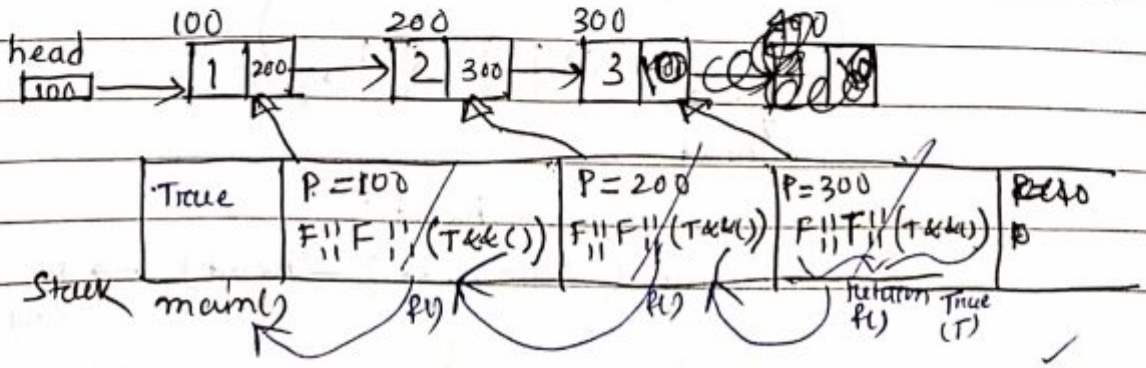
Question-5 (Checking if the list is in non decreasing)

```
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)));}
```



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

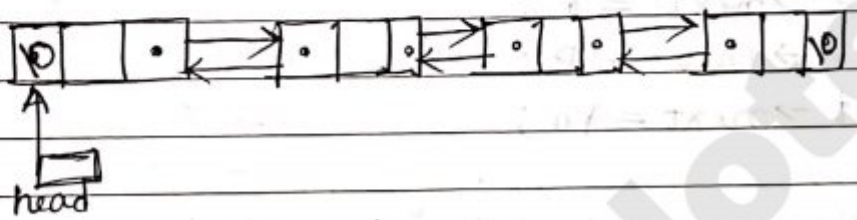


www.GateNotes.in

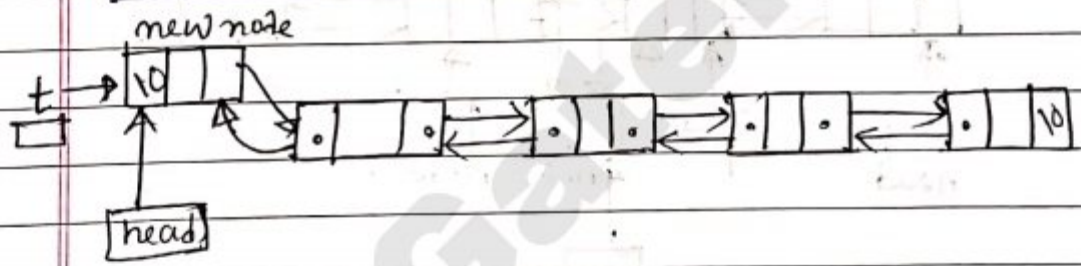
classmate
Date
Page

- Insertion into doubly linked list :

```
Struct node
{
  int i;
  Struct node *prev;
  Struct node *next;
};
```

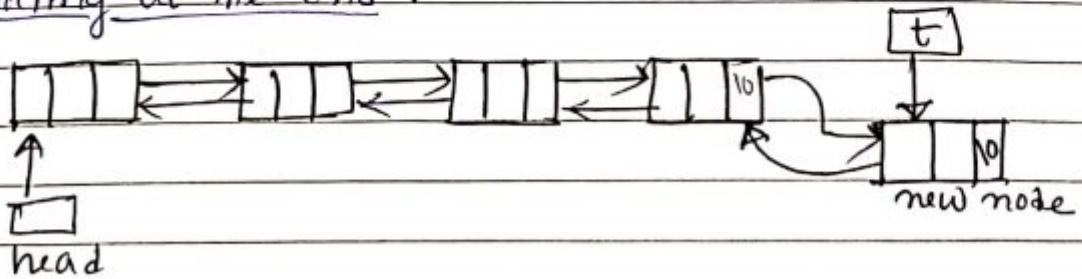


- Insert a node at ~~front~~ beginning :-




```
struct node *t;
t -> next = head;
head = t;
t -> prev = NULL;
head -> next -> prev = ~~t~~ head;
```

- Inserting at the end :-

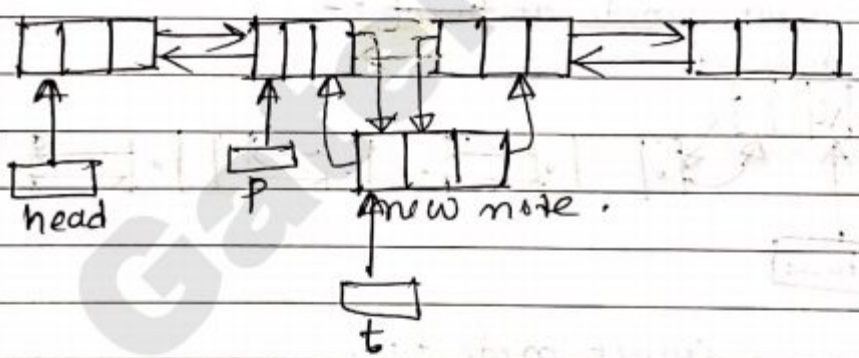


www.gatenotes.in

~~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 nod *t;
```

(hold the link and then modify)

```
t->prev = p;
t->next = p->next;
p->next = t;
p->next->prev = t;
```

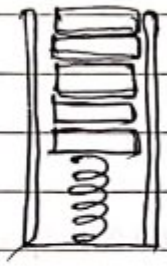
www.GateNotes.in

classmate
Date
Page

www.GateNotes.in

Stacks and Queues

- **Introduction to stacks :**

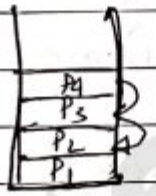


→ push the plate from the top (insert) and take the plate from the top. (pop)

(stack) (FILO, LIFO)

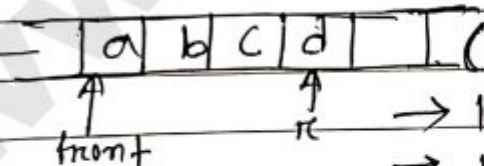
- **Applications of stack :**

(i) Recursion.
(ii) IF → PF conversion.
(iii) Parsing
(iv) Browsers. →
(v) Editors. (use)
(vi) Tree traversals and graph traversal.



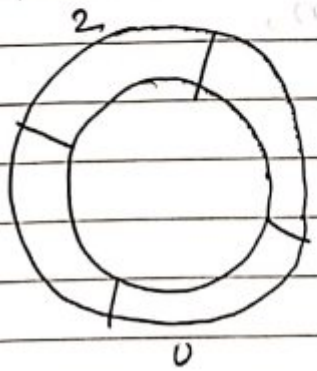
- **Implementation of a Queue using circular array :**

→ Insert the element from one side (enqueue) and delete the element from other side. (dequeue)



(Queue) (FIFO, LILO)

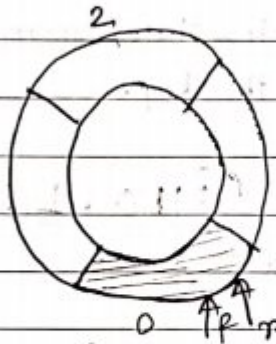
→ Insert from rear.
→ delete from front.



circular queue.

www.gatenotes.in

If → In circular queue, if the size of an array 'n' then we use '(n-1)' amount of size.



circular queue.

→ should a blank space.

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
    rear = rear - 1;
    return;
  }

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

www.gatenotes.in

② delete an item from queue :-

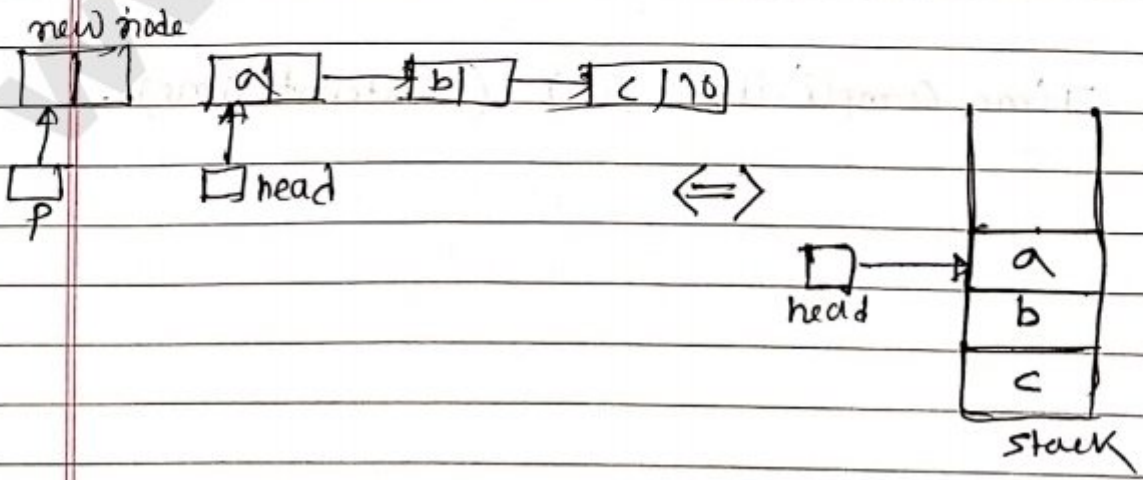
```
int Dequeue()
{
  if (front == rear)
    {
      Pf ("Q is empty");
      return -1;
    }
  else
    {
      front = (front+1) mod n;
      item = q[front];
      return item;
    }
}
```

Over flow :
(rear+1) mod n == front

under flow
front == rear

• Linked List Implementation of stack :

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



www.gatenotes.in

classmate
Date
Page

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 =

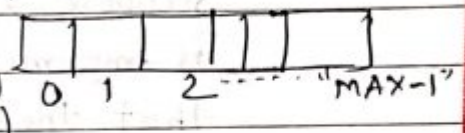
```
Pseudocode
int pop()
{ struct node *p;
  int item;
  if (head == NULL) { pf ("Underflow"); return -1;}

  item = head → i;
  p = head;
  head = head → next;
  free (p);
```

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

www.GateNotes.in

- Implementation of stack using array =

```
int stack[MAX];
int top = -1; //(stack Empty)
```



- puring a item =

```
Void push (int item)
{
    if (top == MAX-1)
        pf("overflow");

    else {
        top = top+1;
        stack[top] = item;
    }
    return;
}
```

stack[++top] = item;

- poping a item =

```
int pop()
{ int temp;
  if (top == -1)
  { pf("under flow");
    return -1;
  }
  else
  {
    temp = stack[top];
    top = top-1;
  }
  return temp;
}
```

temp = stack[top--];

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

www.GateNotes.in

GR-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, REAR=FRONT=0. The conditions to detect queue ~~with~~ full and queue empty are —

→

Queue Full:

(REAR+1) mod n == FRONT.

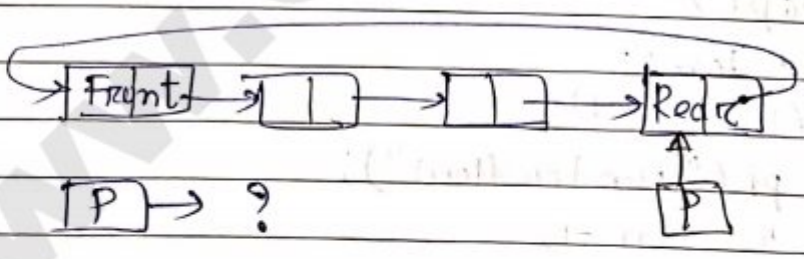
Queue Empty:

REAR = FRONT

GR-2004

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 node
b) FRONT node
c) not possible.
d) node next to Front.

→

www.gatenotes.in

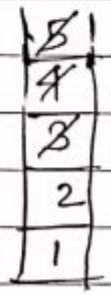
classmate
Date
Page

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?

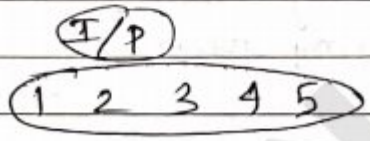
✗ a) 3,4,5,1,2. ✓ b) 3,4,5,2,1.
✗ c) 1,5,2,3,4. ✗ d) 5,4,3,2,1.

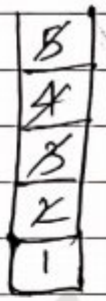
✗ (a)



o/p
3,4,5..

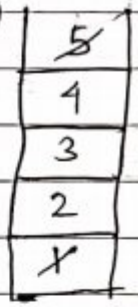
(b)





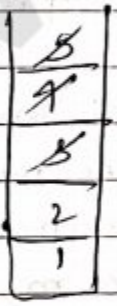
o/p
3,4,5,2,1

✗ (c)



1,5,.,..

✗ (d)



5,4,3,

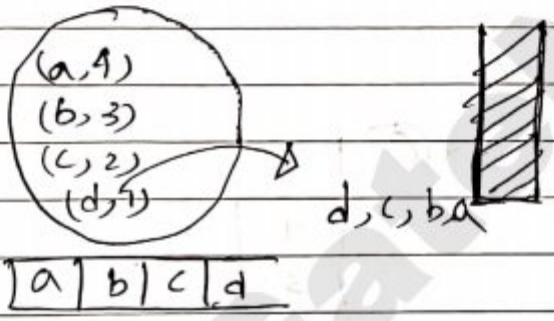
www.GateNotes.in

G-1997 **Question - 4**

A priority queue 'Q' is used to implement a stack 's' that stores characters PUSH(c) is implemented as ~~Insert~~ Insert (Q, c, K) where K is an appropriate integer key chosen by the implementation. POP is implemented as 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.

→



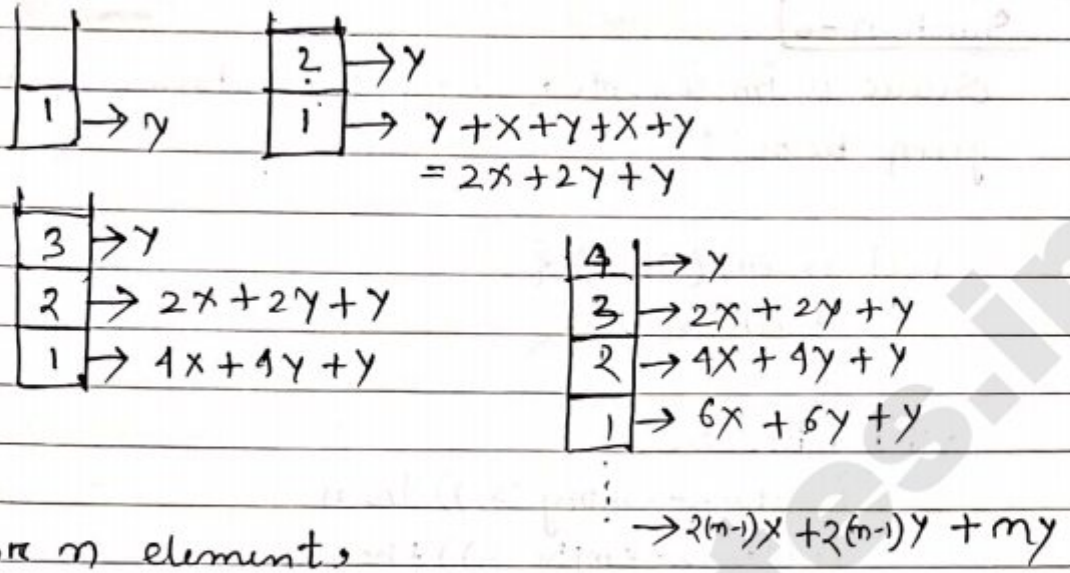
g-2003 **Question - 5**

Let 's' be a stack of size $n \geq 1$. Starting with the ~~ent~~ 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 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:

→

www.gatenotes.in

classmate
Date
Page



for n elements,

average life time,

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

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

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

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

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

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

$$\Rightarrow \boxed{n(x+y) - x}$$

www.GateNotes.in

06-2006

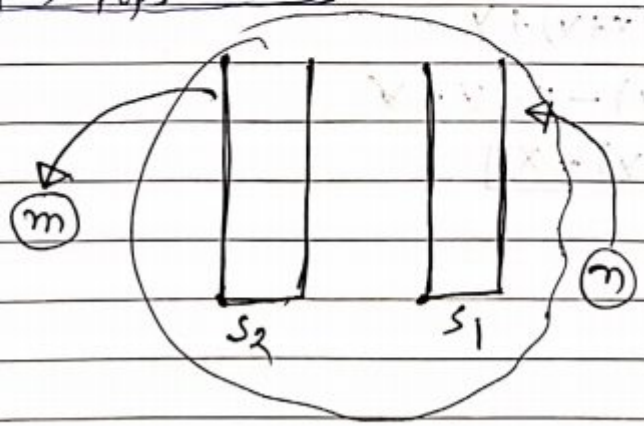
Question-6

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

```
void insert(Q, x) {
    push(S1, x);
}
void delete(Q) {
    if (stack-empty (S2)) then
    if (stack-empty (S1)) then {
        printf("Q is empty");
        return;
    }
    else while (!(stack-empty(S1))) {
        x = pop(S1);
        push(S2, x);
    }
    x = pop(S2);
}
```

n → Inserts
m (≤n) → deletes
x → Pushes
y → Pops

What is relation b/w all this?



www.gatenotes.in

In Best case:

PUSH : $2m + (n-m)$
$= m+n$

POP : $2m$

Worst case:

PUSH: $2n$

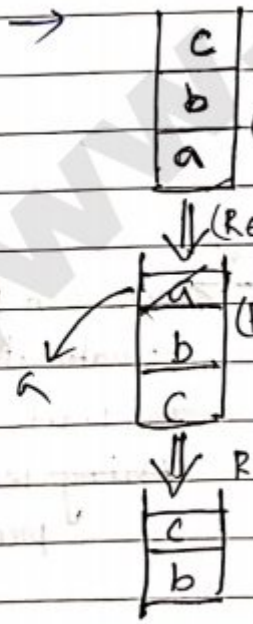
POP: $2(n+m)$

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

gate-2000 **Question - 7**

Suppose a stack implementation supports, in addition to PUSH, and POP, an operation "REVERSE" the order of the 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.



a b C

Enqueue → push

Dequeue → Reverse → POP → Reverse

www.gatenotes.in

classmate
Date
Page

Infix → postfix

- **Infix to postfix Conversion algorithm :-** (Operator stack)

→ postfix related anything stack is required.

example :

① a + b → Infix expression :
ab+ → postfix expression.

② a + b * c → Infix exp
= a + bc*
= abc*+ → postfix expression.

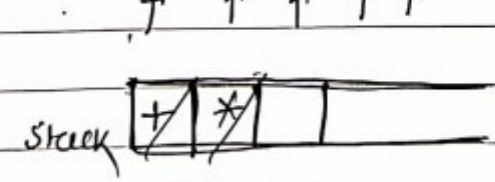
③ a + b − c
= ab+ − c
= ab+c−

+ <. (
(<. +

④ a + (b − c) − Infix
= a + bc−
= abc−+ — postfix.

⑤ a + b * c − infix
= a + bc*
= abc*+ — postfix

⑥ a + b * c − Infix.



= abc*+ - Postfix.

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

www.gatenotes.in

classmate
Date
Page

⑦ a * b + c — Infix expression.

Stack [*][+]

a b * c + — postfix expression.

⑧ a * b + (c − d) — Infix

[*][+][(][−]

a b * c d − + — postfix expression.

Algorithm

a) Creat 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 parenthe-
        -sis 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 encounted or a
          left parenthesis is encounted or
          the stack is empty.
        }
  push t
}
```

www.GateNotes.in

www.gatenotes.in

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

Space or Time complexity — O(n)
Data structure used — stack
on PUSH only — operators.

Postfix → Infix

- Postfix evaluation algorithm = (Operand stack)

3 + 2 * 1 — Infix

3 + 21*

321*+ — postfix

2 * 1 — 3 + 2

op1 op2

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

www.gatenotes.in

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.

www.GateNotes.in

classmate
Date
Page

www.GateNotes.in

TREES

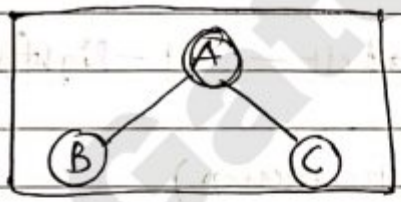
- **Introduction to tree traversals:**

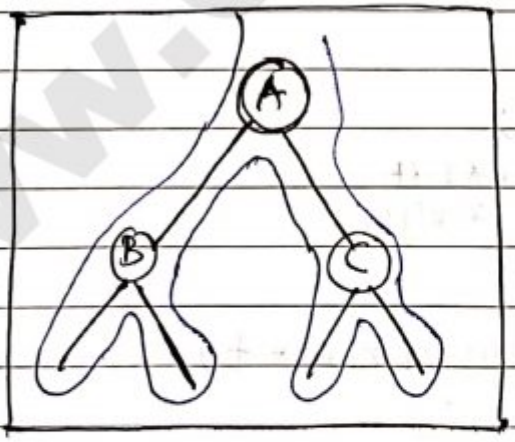
- **Traversing means** → We want to see what is information present in Every node.

- **Searching means** → searching for a particular node.

Methods of traversing

- 2nd time → Inorder traversing. (Left-Root-Right) – BAC
- 1st time → preorder traversing (Root-Left-Right) – ABC
- 3rd time → postorder traversing. (Left-Right-Root) – BCA

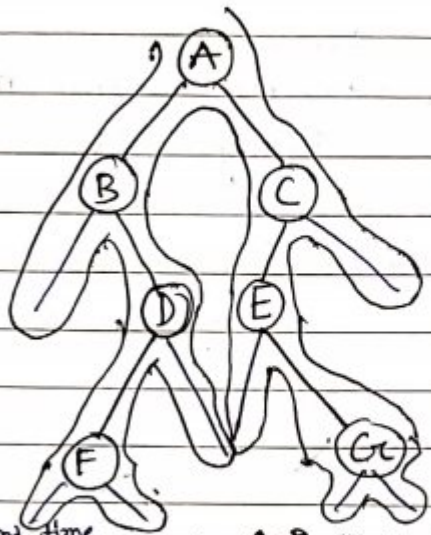




1st time – ABC – (pre)
2nd time – BAC – (Inorder)
3rd time – BCA – (post)

www.gatenotes.in

example :



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

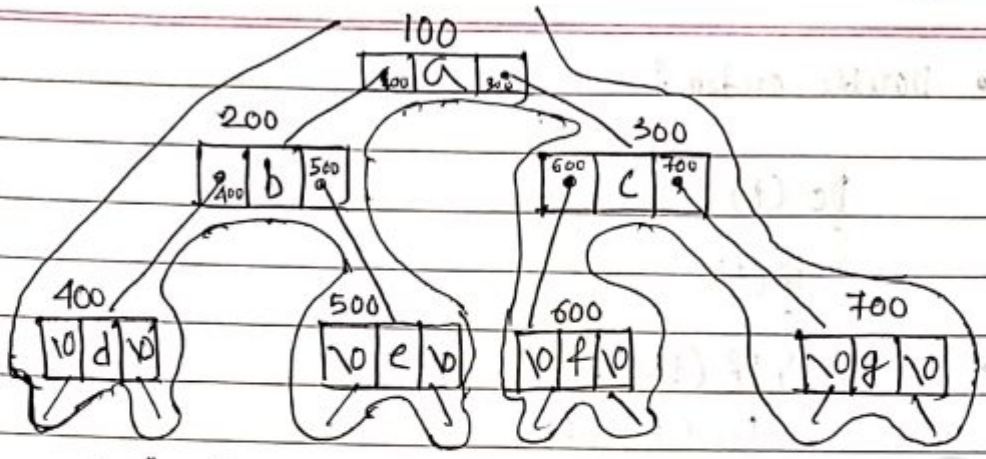
- Implementation of traversals and time and space analysis =

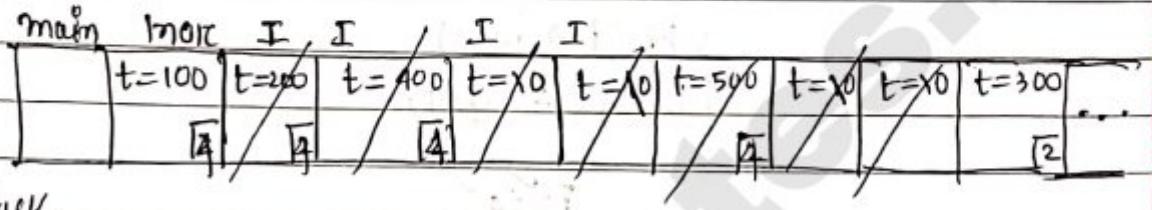
Inorder traversing → (left — root — Right).

Program : (Using recursion)

```
struct node
{  char data;
   struct node *left
               *Right;
}

void Inorder (struct node *t)
{
   if (t != NULL)
   {
   1. Inorder (t -> left);
   2. printf ("%c", t -> data);
   3. Inorder (t -> Right);
   }
}
```

www.GateNotes.in





Stack

O/p: dbeafcg

In/pre/post order traversal all takes

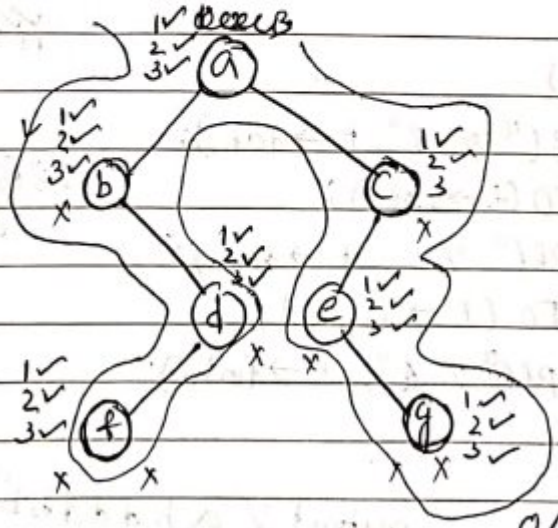
Time and space complexity = O(n).

{ Inorder → 1, 2, 3 (lines)
pre order → 2, 1, 3
post order → 1, 3, 2

- Double order traversal:

(one other method)

example: (Inorder) (root as Input)



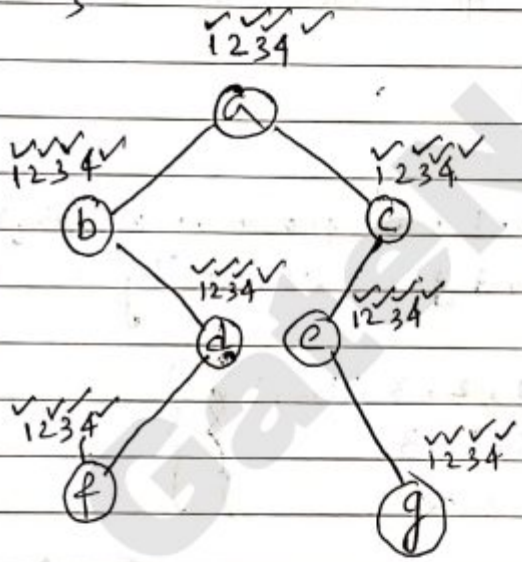

```
INORDER(t)
{
1. INORDER(t->Left)
2. pf
3. INORDER(t->Right)
}
```

O/p: b f d a e g c

www.gatenotes.in

- Double order :

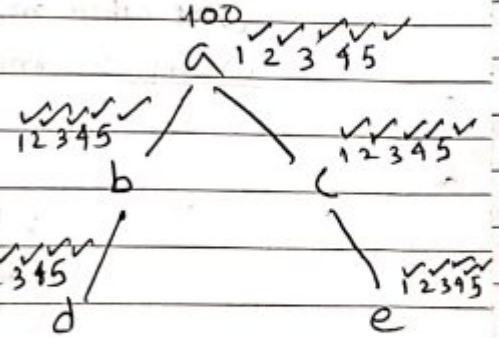
```
DO (t)
{ PF(t)
  {
  1 PF (t→data)
  2 DO (t→left)
  3 PF (t→data)
  4 DO (t→right)
  }
```



Output :
a b b d f f d a c e e g g c

- Triple order traversal :

```
                    (100)
void TO (struct node *t)
{
  if (t)
  {
  1. pf("%d", t→data);
  2. TO (t→Left);
  3. pf("%c", t→data);
  4. TO (t→Right);
  5. pf("%d", t→data);
  }
}
```



output : a b d d d b b a c e e e c c a

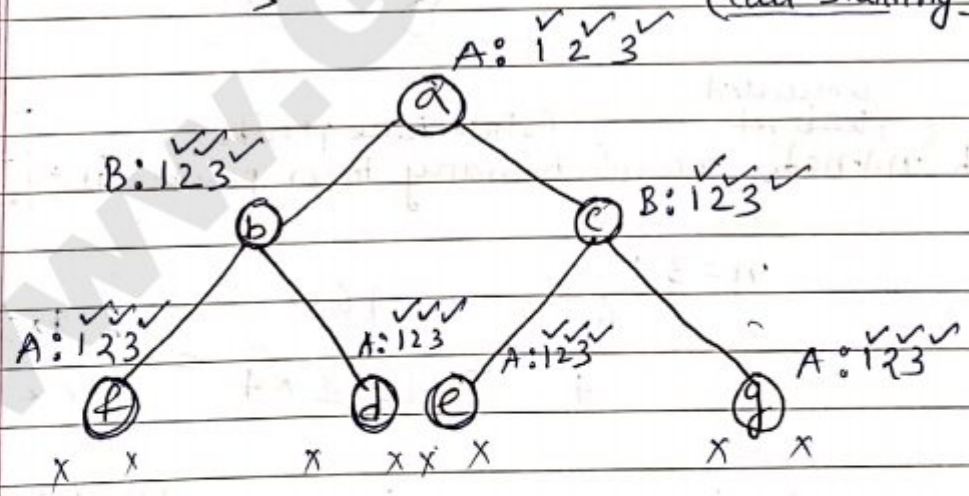
www.gatenotes.in

classmate
Date
Page

- Indirect recursion on trees:-

```
void A (struct Node *t)
{
  if(t)
  {1. pf("%d", t->data);
   2. B (t->left);
   3. B (t->right);
  }
}

void B (struct node *t)
{
  if(t)
  {
   1. A (t->left);
   2. pf("%d", t->data);
   3. A (t->right);
  }
}
```

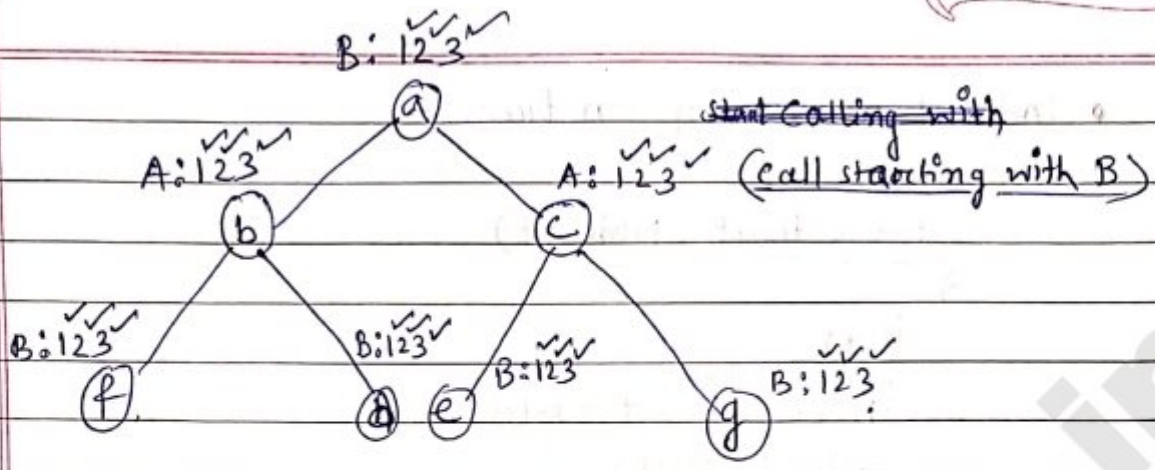
(call starting with A)



afbdecg — output.

www.gatenotes.in

classmate
Date
Page

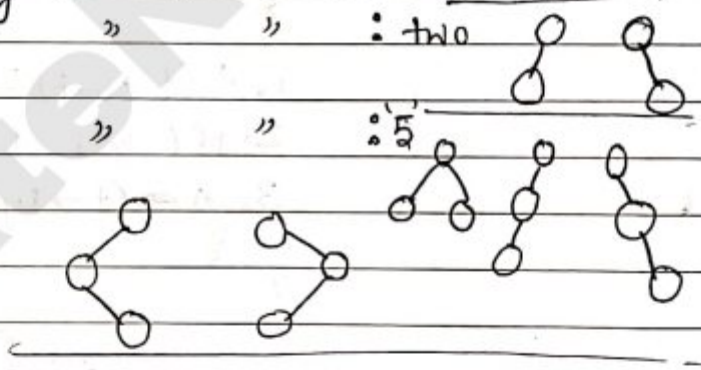


~~start Calling with~~ (call starting with B)

bfdaceg → output.

(structures)

- **Number of Binary trees possible:**
  - Number of Binary tree possible with unlabelled ~~unlevel~~ node =
    1 node no. of binary trees possible is: one
    2 " " " " : two
    3 " " " " : 5



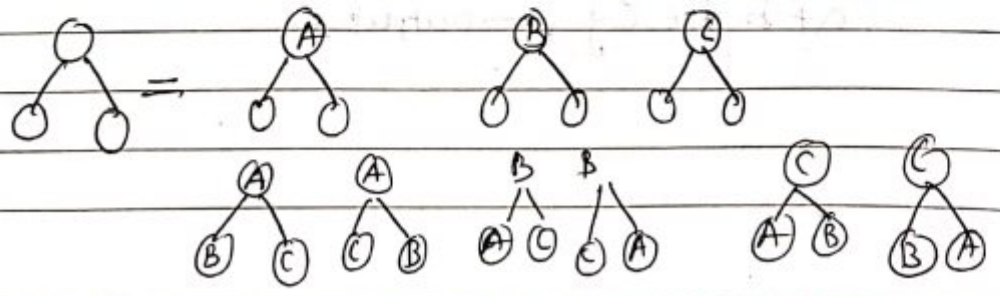
With 'n' unlabelled ~~unlevel~~ node no. of Binary trees possible is: (structure possible)

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

$n = 3$

$$\frac{6C_3}{4} = \frac{\lfloor 6}{\lfloor 3 \lfloor 3 * 4} = \frac{6 \times 5 \times 4}{3 \times 2 \times 4} = 5$$

- **Number of Binary tree possible with the ~~velevel~~ labelled node :=**



www.gatenotes.in

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

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

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 » » : 5 × 12 = 30.

→

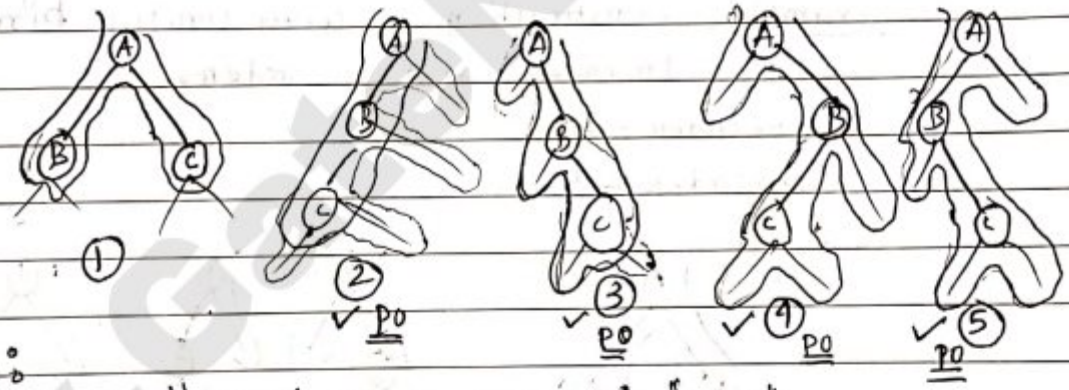


fig: ①

When:

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

fig: ② ③ ④ ⑤ : (4 trees possible)

When: pre "ABC" (1) && post "CBA" (3) && Inorder "BCA" (2nd time visit) =

fig: ~~①~~ ⑤ (only - one tree possible)

(here one tree that satisfy all the condition)

www.gatenotes.in

→ ~~no. of tree possible with node~~

given { 'n' nodes
preorder -

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

→ with given preorder and postorder you may get more than '1' binary tree.

*→ with given pre ~~am~~, post, and Inorder we always get only 'one' binary tree.

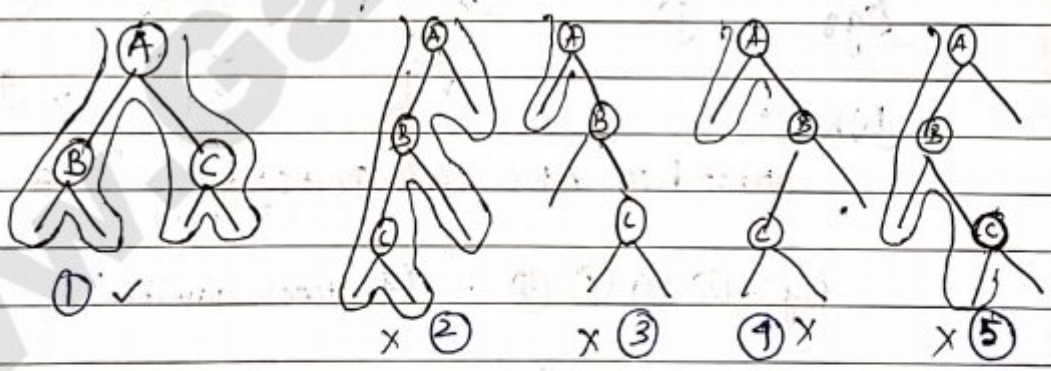
(~~unic~~) unique

<u>example</u>: construction of ~~uni~~ unique binary tree using inorder and preorder.

(1st) preorder = ABC

(2nd) Inorder = BAC

→



→ 5 binary tree possible with preorder ① ② ③ ④ ⑤.

→ ~~(2)~~ 1 BTS possible with (In and pre both order) fig - ①.

www.gatenotes.in

IN + PRE
IN + POST
> possible

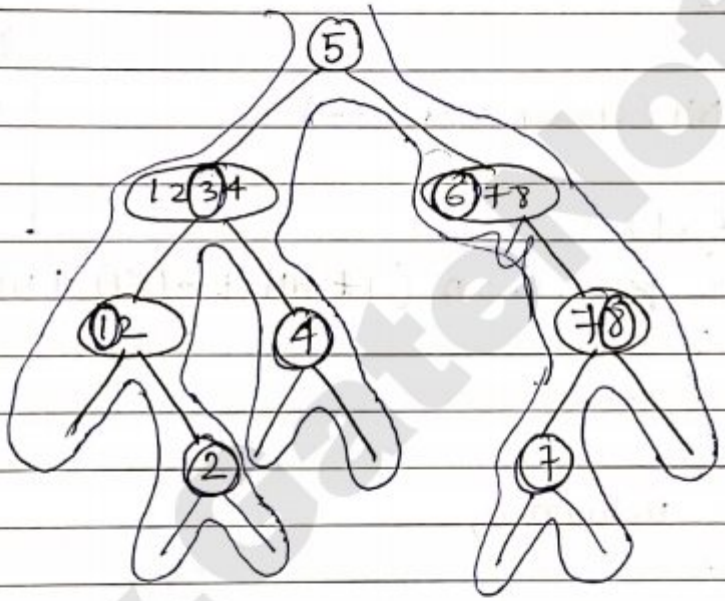
POST + PRE → not possible to find IN.

Question:

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

→

IN – L (Root) R
Pre – (Root) L R

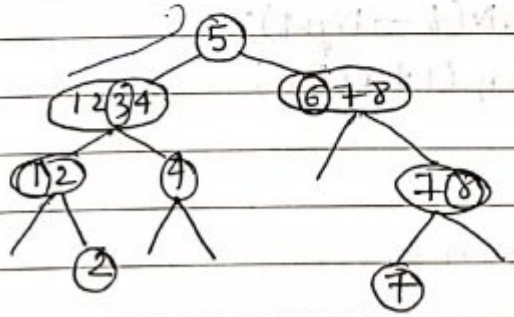


POSTORDER: 2 1 4 3 7 8 6 5

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: ?

→

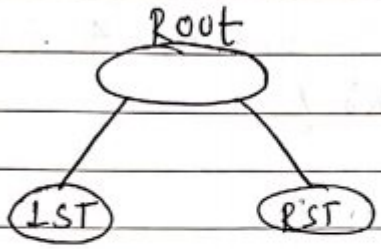


PRE: 5 3 1 2 4 6 8 7.

www.gatenotes.in

classmate
Date
Page

• Recursive program to count the number of nodes =



NN(T) = 1 + NN(LST) + NN(RST)
= 0; if T is NULL.

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

Program :-

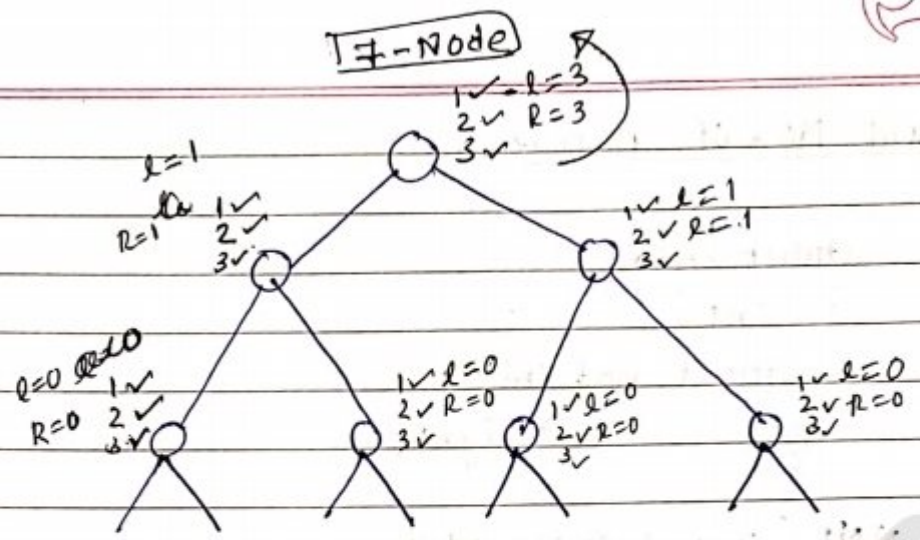
```
int NN(struct node *t)
{
  if(t)
  { return (1+NN(t->left)+NN(t->Right));
  }
  else
    return 0;
}
```

or

```
int NN(struct node *t)
{
  if(t)
  int l,R;
1. l = NN(t->left);
2. R = NN(t->Right);
3. return (1+l+R);
  }
else
  return 0;
}
```

www.gatenotes.in

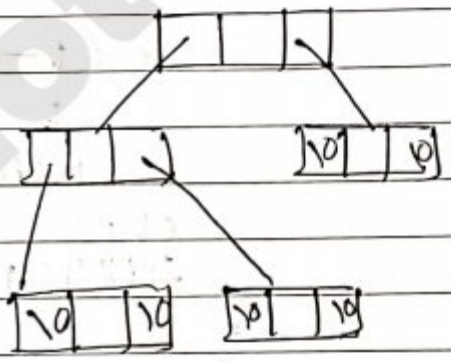
classmate
Date
Page



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

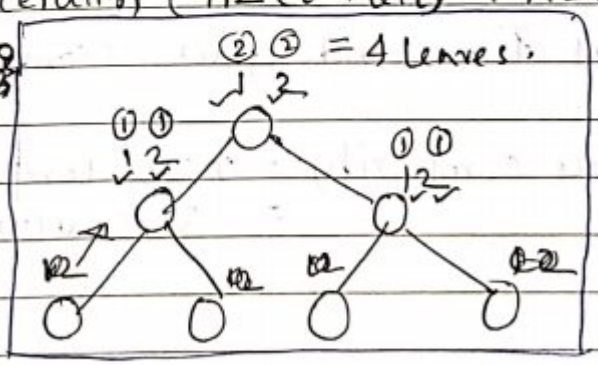
- Count NO. of leaves :

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



→ 3 leaves

```
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));
}
```



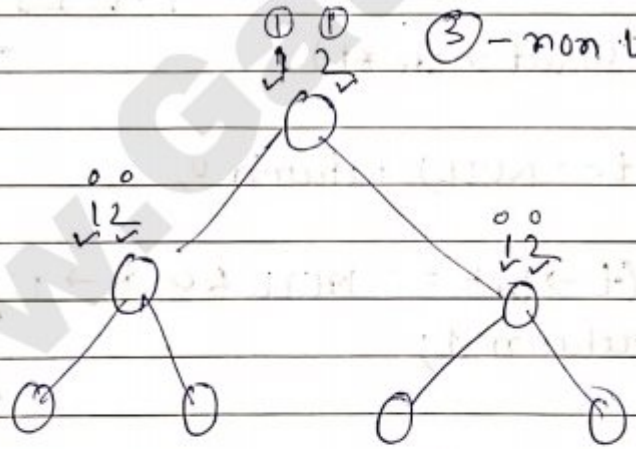
www.gatenotes.in

classmate
Date
Page

• Count No. of non leaves :

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

```
Int NNL(struct node *t)
{
  if (t == NULL) return 0;

  if (t->left == NULL &&
      t->Right == NULL)
    return 0;
  else
    return (1 + NNL(t->left) + NNL(t->Right);
}
```



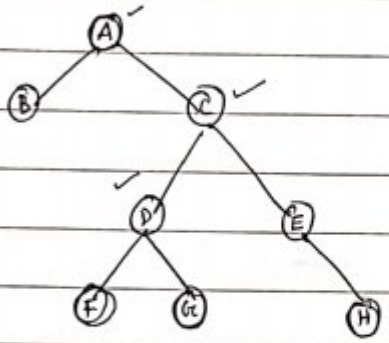
→ Total time and space compuxity = O(n).

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

www.gatenotes.in

classmate
Date
Page

- Recursive program to find the full nodes =



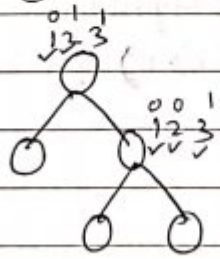
Total Full node : 3 (A,C,D)
Total node : 4 (A, C, D, E).

FN(T) = 0 (return) ; T = NULL.
= 0 ; T is a leaf.
= FN(T→LST) + FN(T→RgsT) ; if T has only one child.
= FN(T→LST) + FN(T→RST) + 1 ; if T is a full node.

Program :

```
int FN (struct node *t)
{
  if (!t) return 0;
  if (!t->left && !t->right)
     return 0;

  return (FN(t->left) + FN(t->right) + (t->left && t->right) ? 1 : 0);
}
```

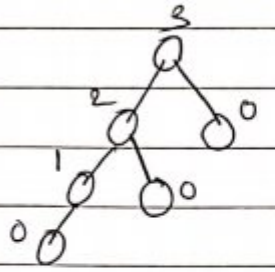
① ② ③



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

www.gatenotes.in

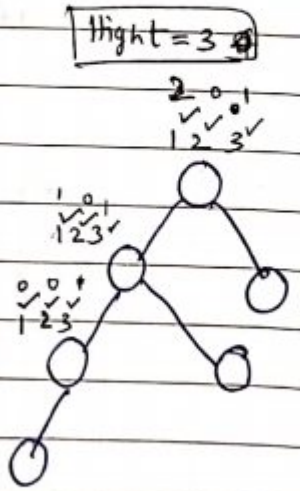
• Recursive program to find the height of a tree :-



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

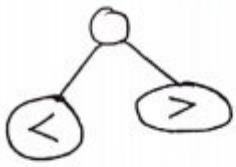
Program:

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

int H(struct node *t) {
    int l, r;
    if (!t) return 0;

    if ( (!(t->left))&&(!(t->Right)))
      return 0;

    Else
      1. l = H(t->left)
      2. r = H(t->Right)
    3. return (1+((l>r) ? l : r));
}
```



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

www.GateNotes.in



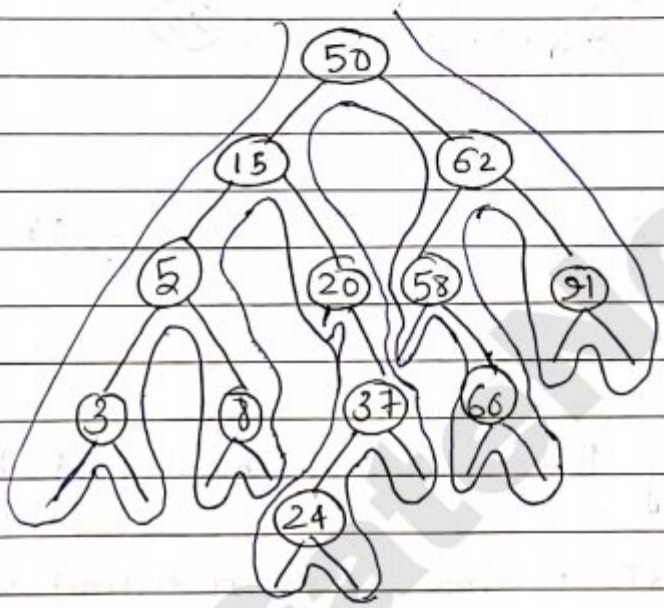
classmate
Date
Page

- **Introduction of to BST (Binary search Tree) =**

→ 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) (assending order)

g-2005 [Question] - (2)

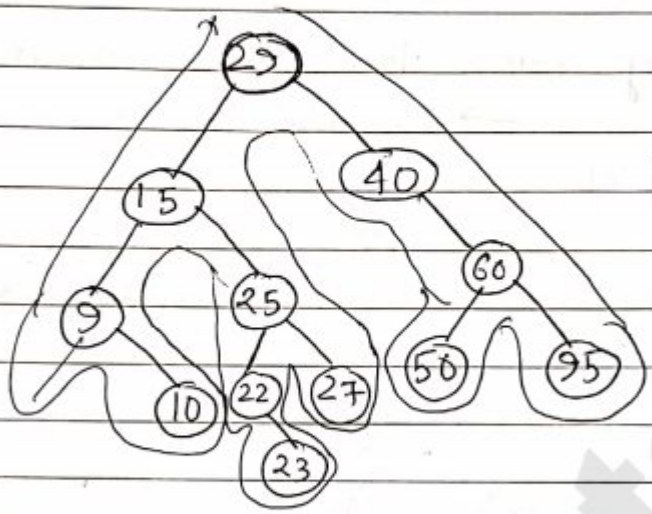
Post Order: 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29.
Inorder: ?
→ Inorder: 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95.
(assending Order)

then find ~~post~~ pre order: ?

www.gatenotes.in

classmate
Date
Page

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



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).

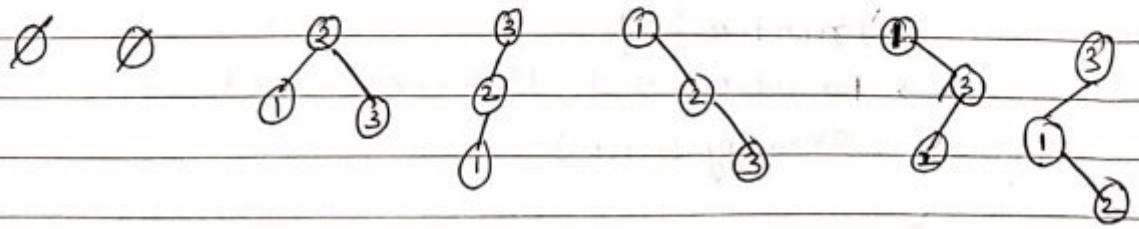
$$\frac{2n C_n}{n+1}$$

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

$$= \frac{8*7*6*5}{4*3*2*5}$$

= 14 (BST) possible.

∅ BSTs 3 distinct keys = 1 2 3



www.GateNotes.in

classmate
Date
Page

g-2006

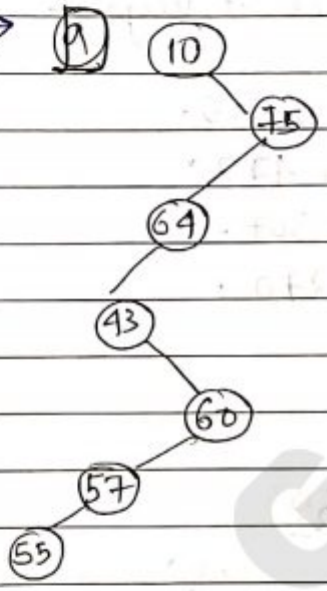
Question - (4)

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

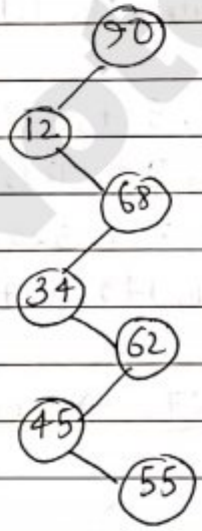
a) {10, 75, 64, 43, 60, 57, 55}
b) {90, 12, 68, 34, 62, 45, 55}
x c) {9, 85, 47, 68, 43, 57, 55}
d) {79, 14, 72, 56, 16, 53, 55}

(Inorder traversal ascending order)

→ [a]

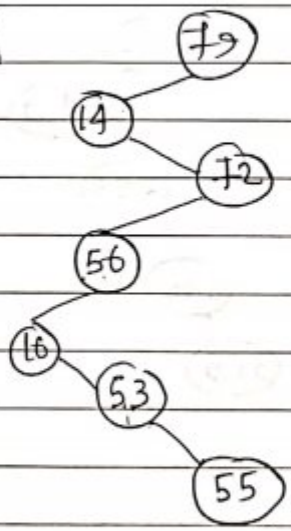


[b]



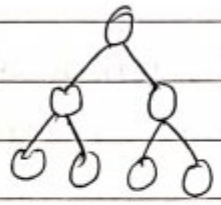
x [c]

[d]

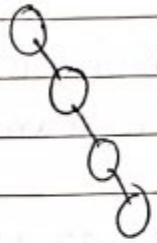


www.GateNotes.in

→ Time compurity of BST :



O(log n) or O(h)



O(n)

h → hight.

gate-2008

Question - (5)

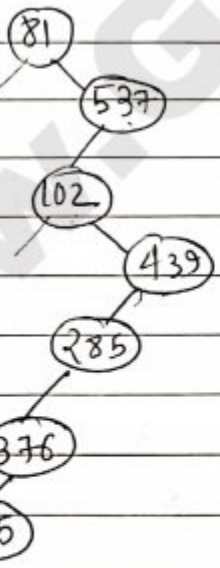
A BST stores values in the range 37 to 573. Consider the following sequence of keys —

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

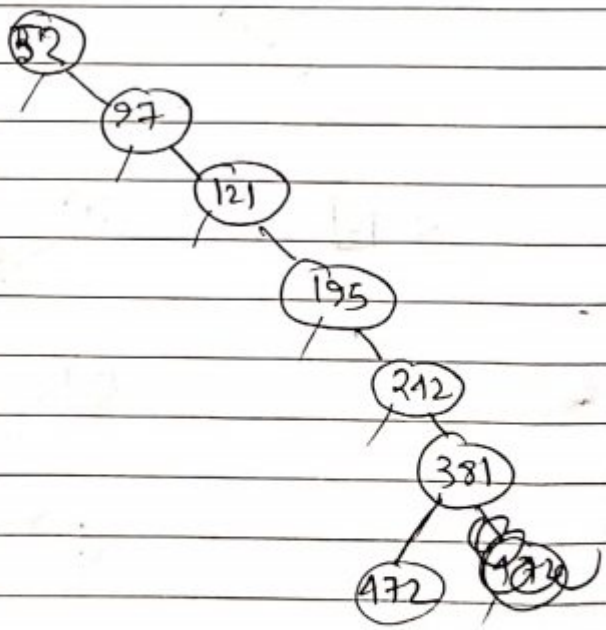
search
273 → Unsuccessful

→

[a]
X

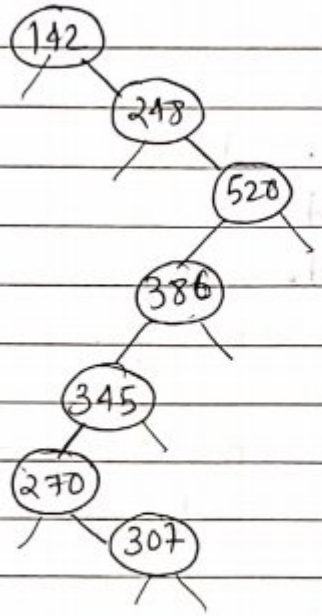


[b]
X

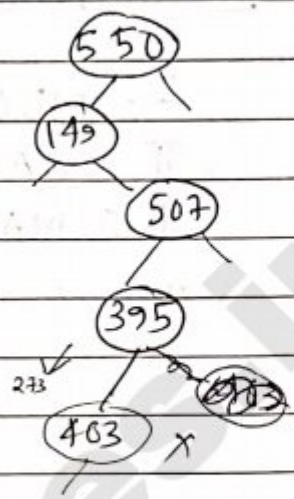


www.GateNotes.in

[C] ✓



[A] X



g-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, 248, 520, 386, 345, 270, 307.

X IV) 550, 149, 507, 395, 463, 402, 270.

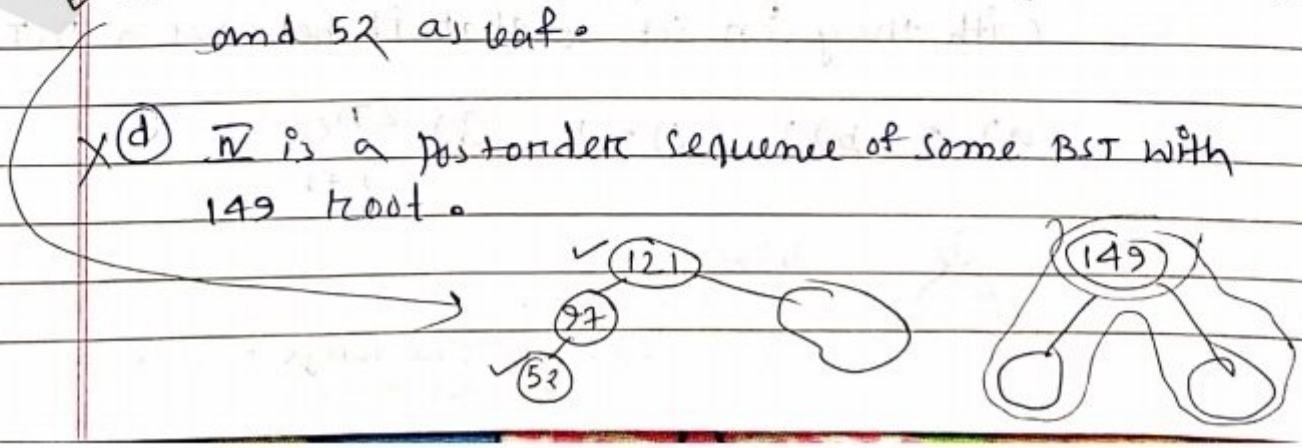
Which of the following is true ?

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.



www.gatenotes.in

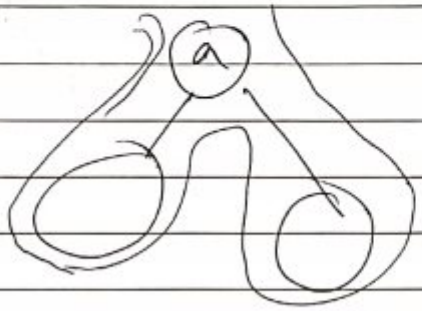
q-2008 **Question - 7**

I: MBCAFHPYK. — Post

✓II: KAMCBYPFH. — Pre

III: MABCKYFPH. — In

Which one is what order?



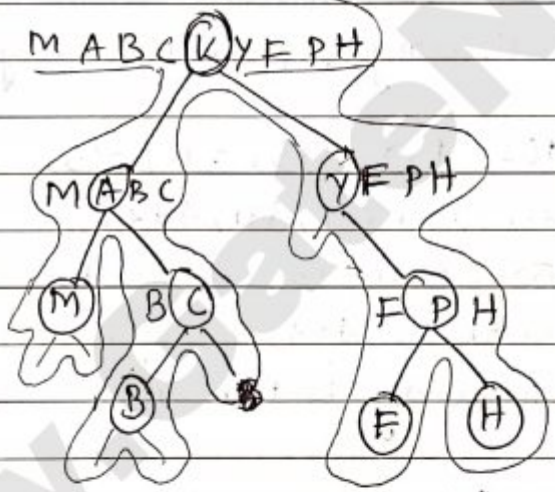
pre: A ——

post: —— A

In: L Root R

Pre: Root L R

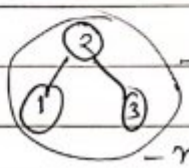
In: M A B C K Y F P H



q-2011 **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) $\frac{2n C_n}{n+1}$.



1, 2, 3

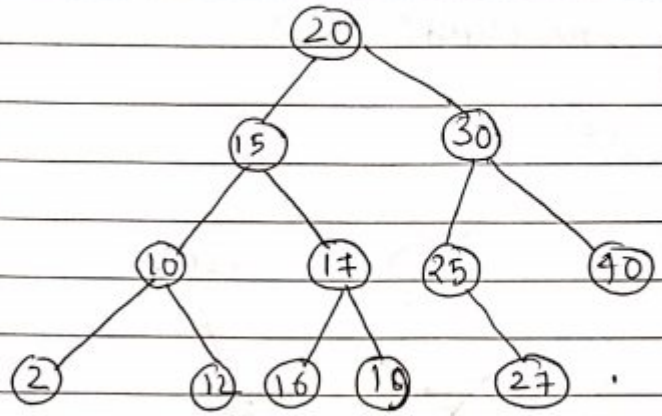
n-keys

n node

→ only one ways possible to becomes (BST).

www.gatenotes.in

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

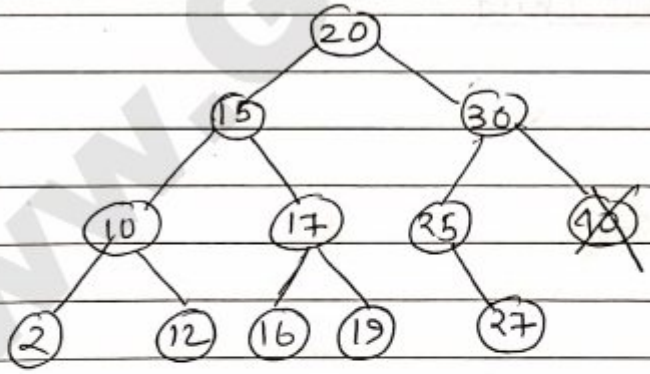


~~The Three types of deletion~~

Three types of element deletion possible in BST =

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

• Deleting leaf : 40



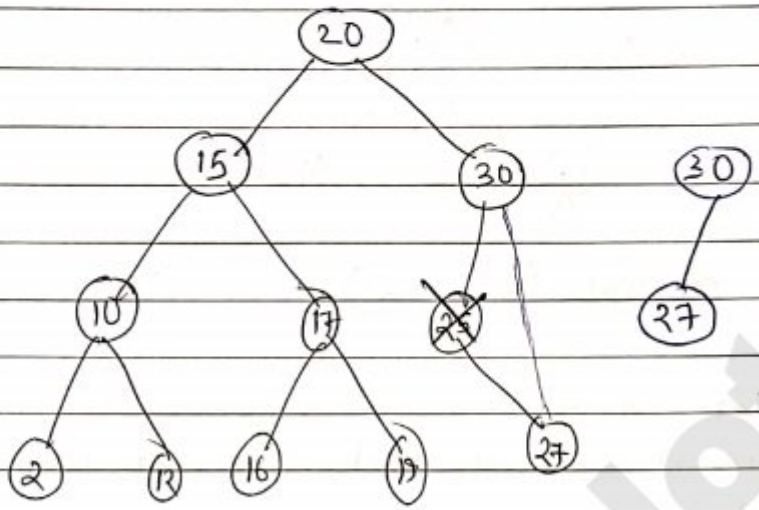
→ When deleting a leaf you did not worry about anything

www.GateNotes.in

classmate
Date
Page

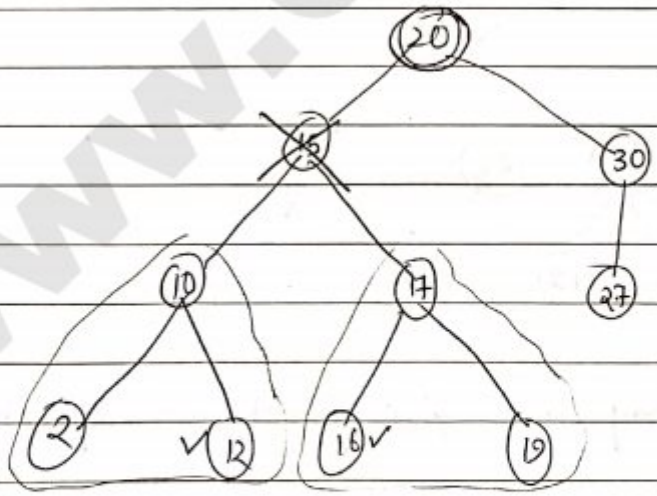
- Deleting Non-leaf :

(i) having one child : (35)



→ 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)



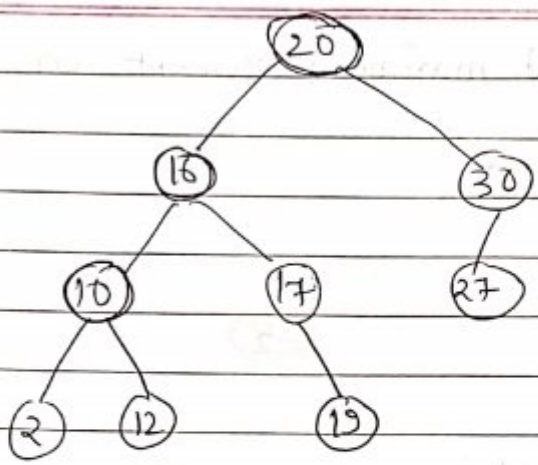
here 2-method :

⇒ go to the Right sub tree take the ~~tiit~~ least element and then put in place of deleted node.

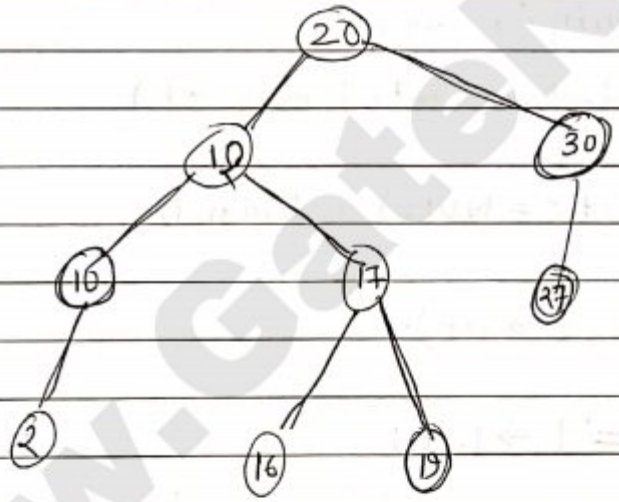
www.GateNotes.in

www.gatenotes.in

classmate
Date
Page



→ other method is, go to the Left Sub tree ~~then~~ take the largest ~~node~~ element and the put it in place of deleted node.



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

→ By doing ~~deletion~~ insertion hight of the tree might increase.

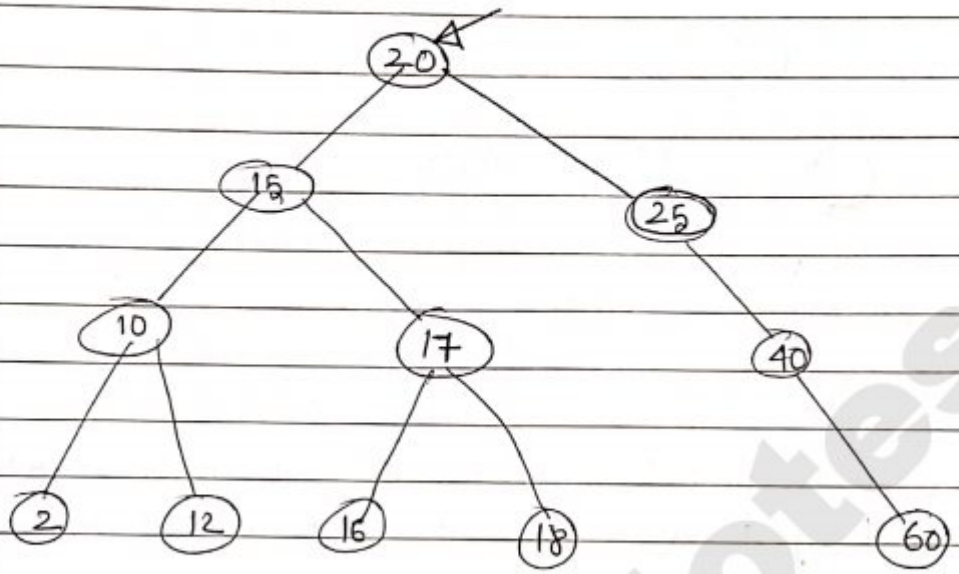
→ Inorder Sucessor: ~~Gratest~~ least ~~Grates~~ element in Right-sub-tree

→ Inorder predecessor: ~~least~~ grates element in Left-sub-tree.

www.GateNotes.in

classmate
Date
Page

- Finding minimum and maximum element in a BST =



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

```
find-inord-pre (struct node *t)
{ - Min
    if(t==NULL) return 0;

    while(t->left)
    {
      t = t->left;
    }
```

- find greates element of the BST :-

```
find-max (struct node *t).
{
    if (!t) return 0
    while (t->right)
    {
      t = t->right;
    }
}
```

www.gatenotes.in

classmate
Date
Page

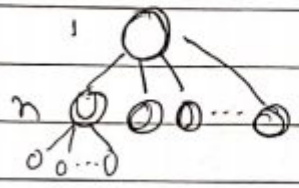
q-1998

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) $x(n+1)$.

→



1 internal node — n leaves.

2 node — $(n-1)+n$

3 internal node — $(n-2)+n+n = 3n-2$

4 node — $(n-3)+n+n+n = 4n-3$

⋮

x node — $xn-(x-1)$

$= xn - x + 1$

$= x(n-1)+1$ (leaves)

q-2002

Question - ②

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

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

→ I → internal node.

n → total node.

l → leaf.

$l+I = n$ ... (i) $l = I(3-1)+1$ ... (ii)

www.gatenotes.in

$2I - L = -1$

$I + L = n$

$3I = n - 1$

$I = (n-1)/3$

$L + I = n$

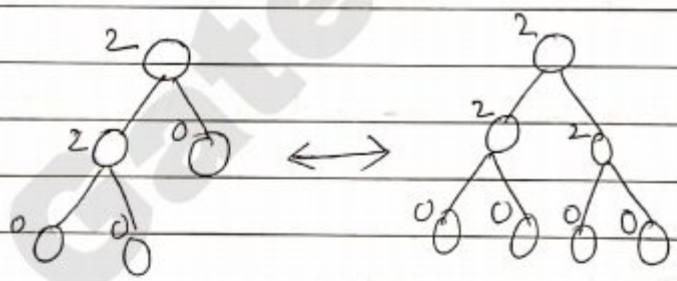
$L + (n-1)/3 = n$

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

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

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

→



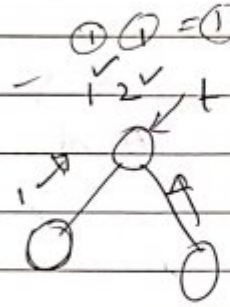
both are complete binary tree.

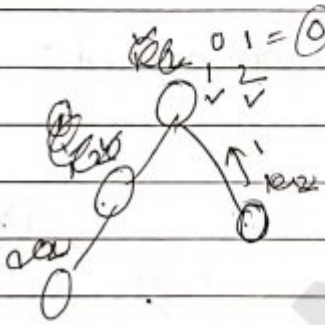
```
int iscomplete (struct node *t)
{
    if (t == NULL) return 0;

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

    else if (t->left && t->right)
        return (iscomplete (t->left) && iscomplete (t->right));
}
```

① ②

www.gatenotes.in

else
return 0;
}





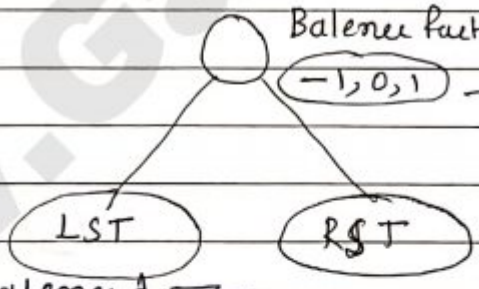
- Introduction to AVL tree =

AVL is an ~~Blanding~~ balencing search tree.

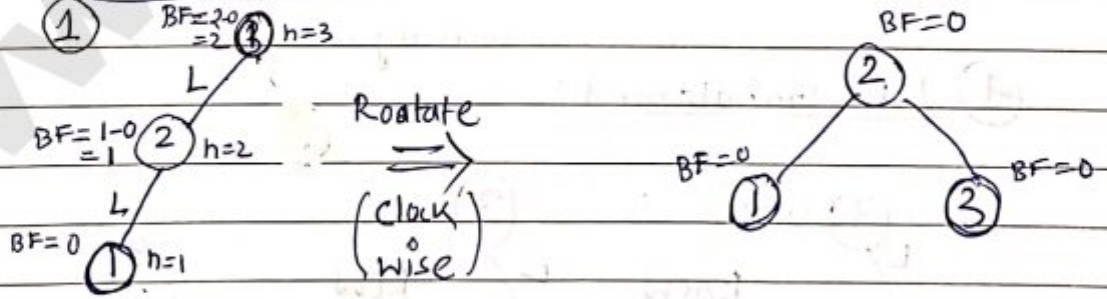
Balence Factor (BF) = Hight (LST) - H (RST).

(-1, 0, 1) - allow (balenced.

(-2 or 2) unbalenced



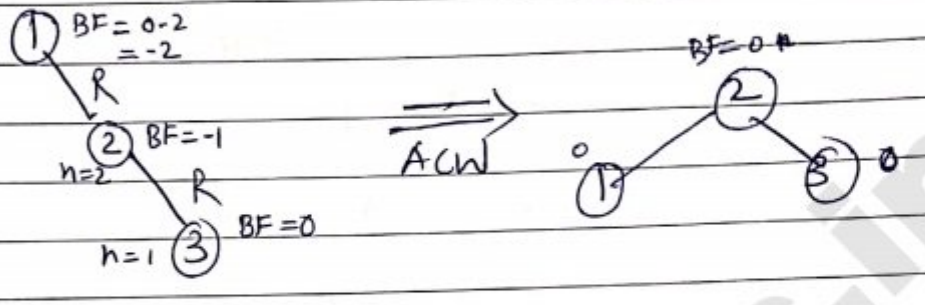
LL - Imbalenced =



www.GateNotes.in

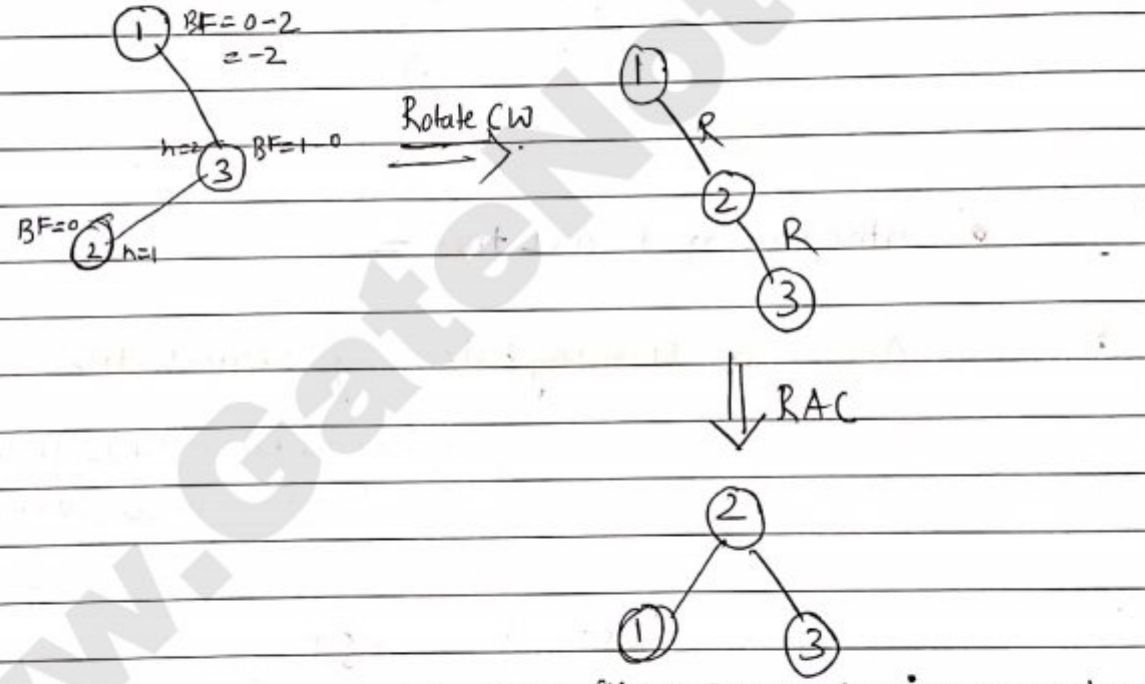
classmate
Date
Page

(2) RR - Imbalanced :



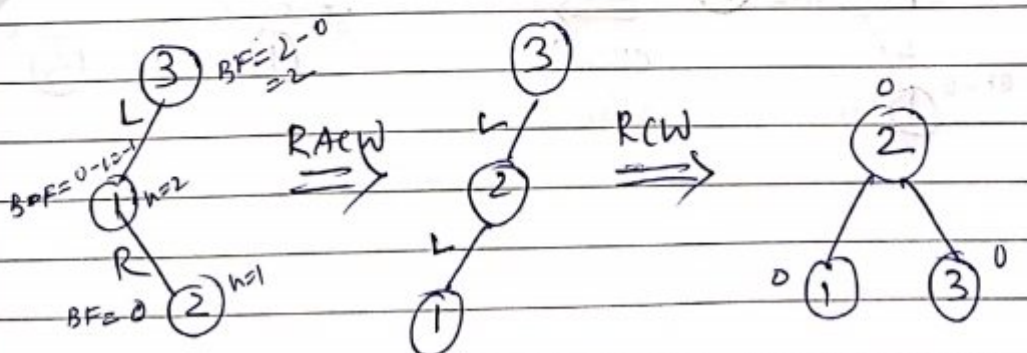
(3) Right-left (RL) Imbalanced :

BF = Balanced factor
Should (-1, 0, 1)



→ result will be BST otherwise something wrong.

(4) LR Imbalanced :



www.GateNotes.in

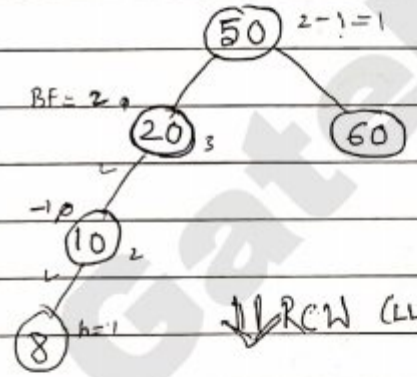
classmate
Date
Page

→ After Inserting a newnode you should travel from newly inserted node toward root to see on the path, if there is any Imbaleneed.

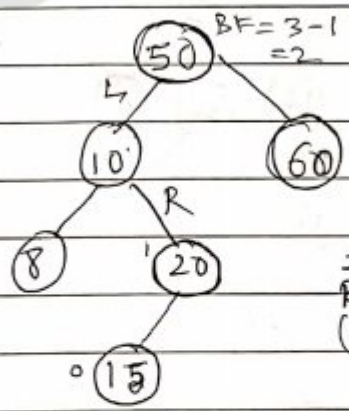
if there is any Imbaleneed node then start from Imbalanced node, and start going toward added newnode to then you understand what type of Imbaleneed (LL, RR, LR, RL). then rotate them accordingly.

- Constructing AVL trees and time complexity analysis =
- example - (1)

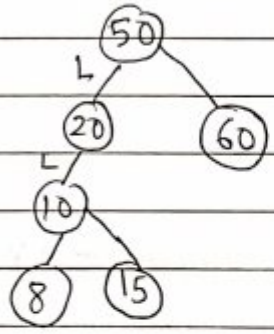
50, 20, 60, 10, 8, 15, 32, 46, 11, 48.



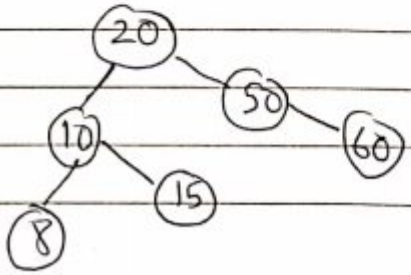
RCW (LL)



RACW (LR)

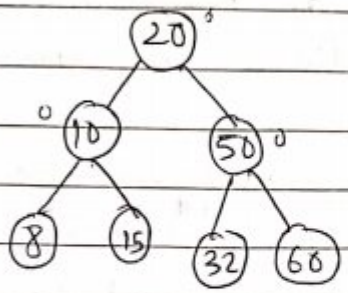


LL (RCW)

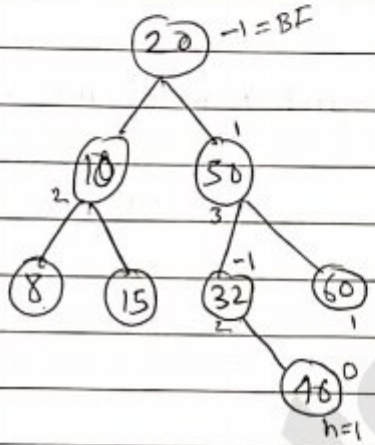


www.gatenotes.in

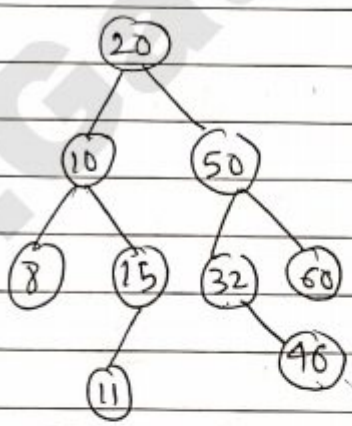
then insert 32.



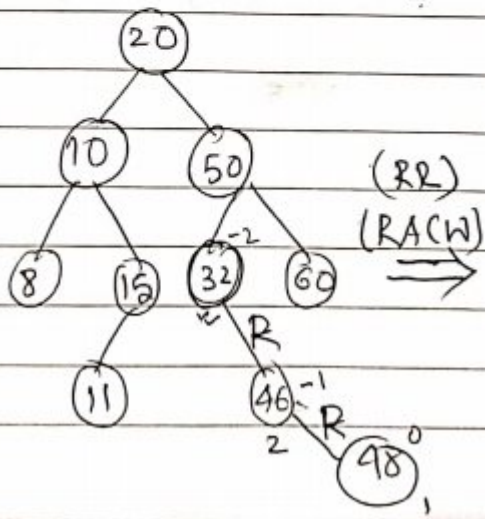
Insert -46



insert - 11

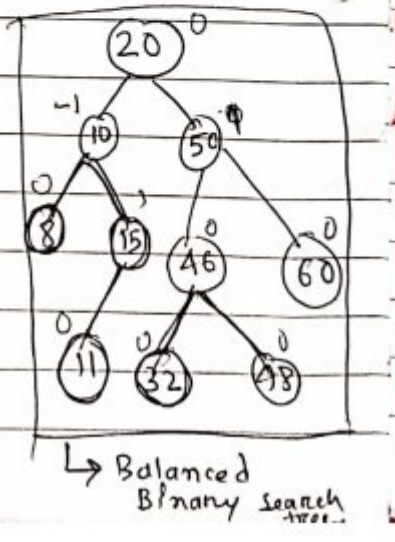


Insert - 48



(RR)
(RACW)

(-1, 0, 1) = BF



↳ Balanced Binary Search tree

www.GateNotes.in

classmate
Date
Page

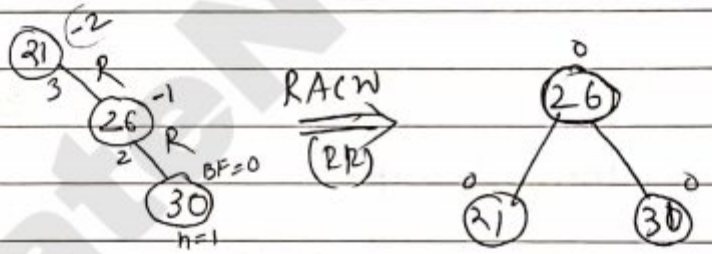
- Complexity analysis =

	BST	(balanced) BBST (AVL)
search -	O(n)	O(log n)
hight -	O(n)	O(log n)
Insertion -	O(n)	O(log n) + O(log n) + c => O(log n)

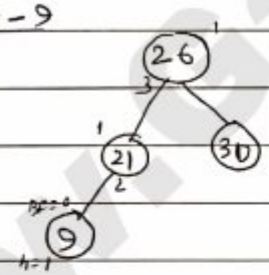
- Construction of AVL tree =
  - Example - (2)

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

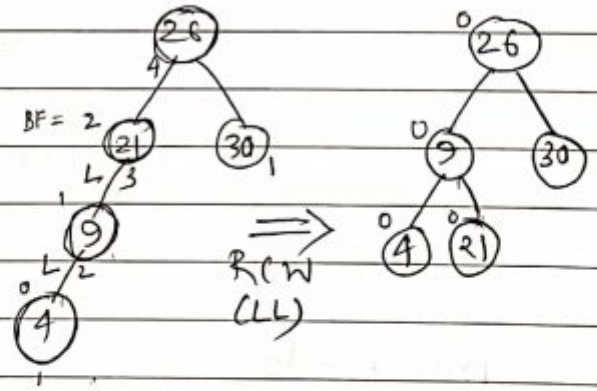
→



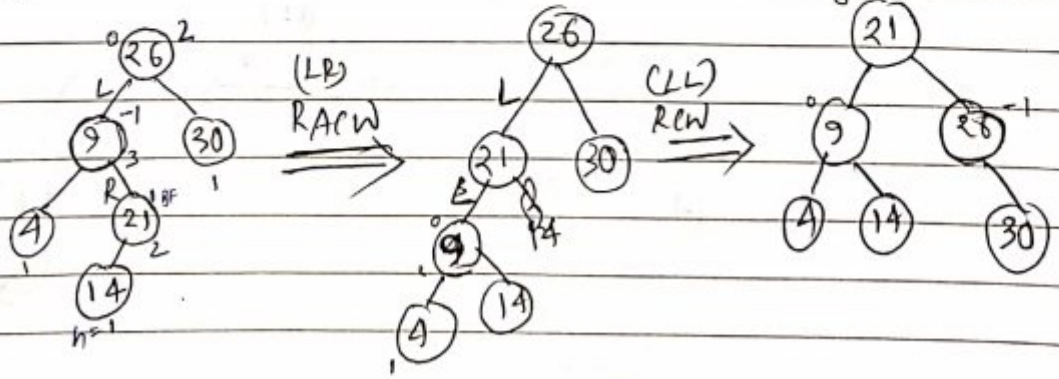
Insert - 9



Insert - 4

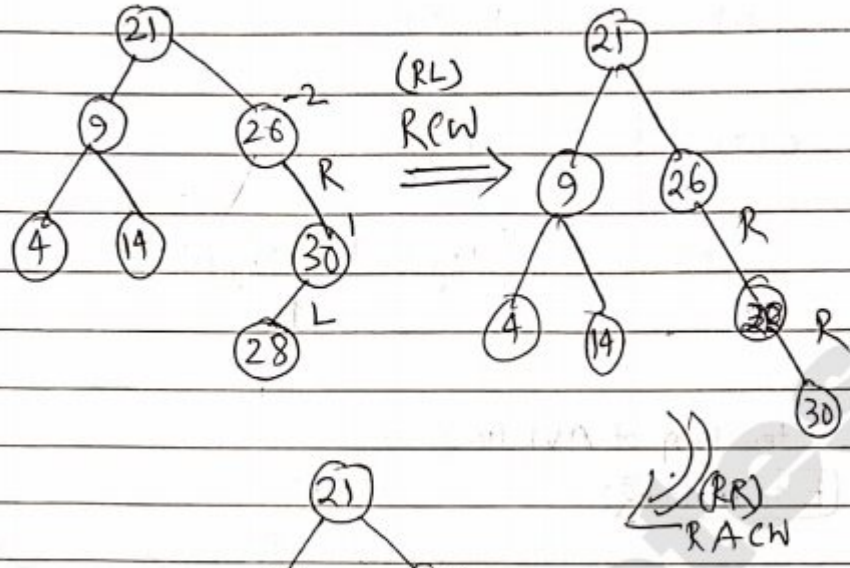


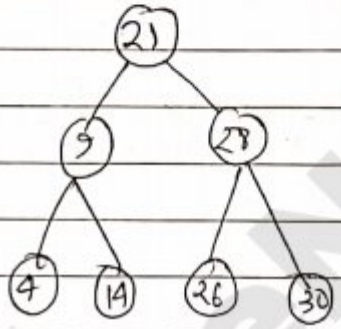
Insert - 14



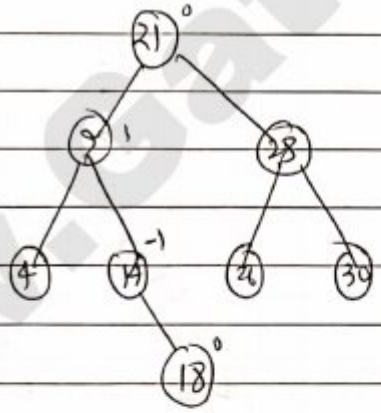
www.gatenotes.in

Insert - 28

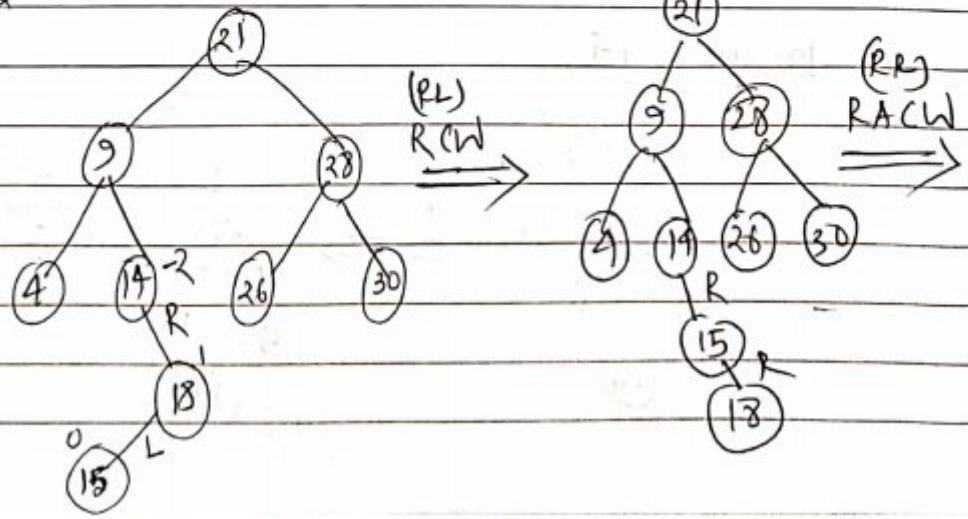




then, Insert - 18

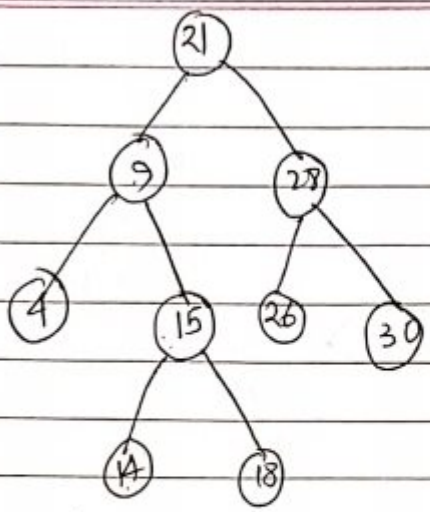


Insert - 15

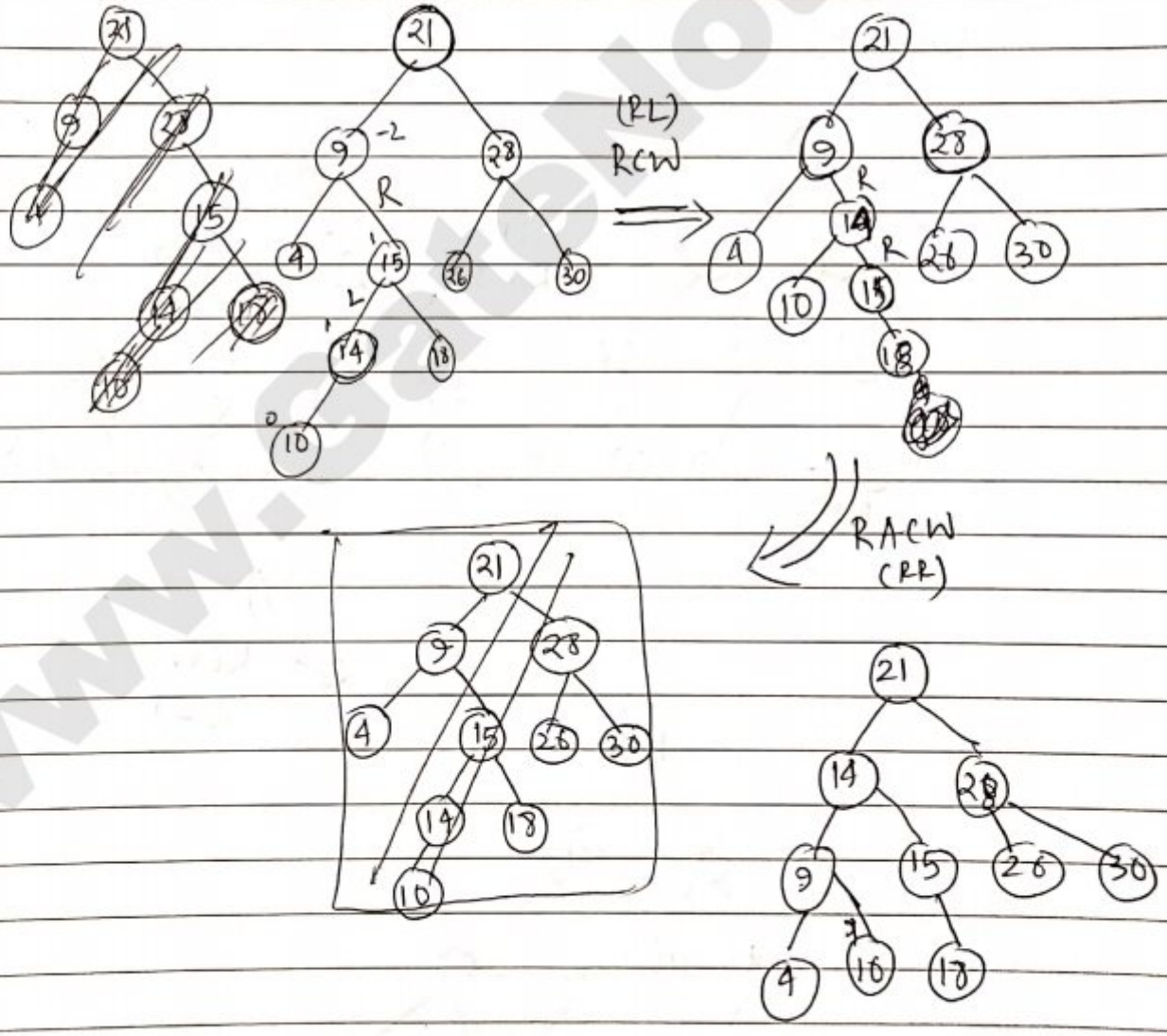


www.GateNotes.in

classmate
Date
Page



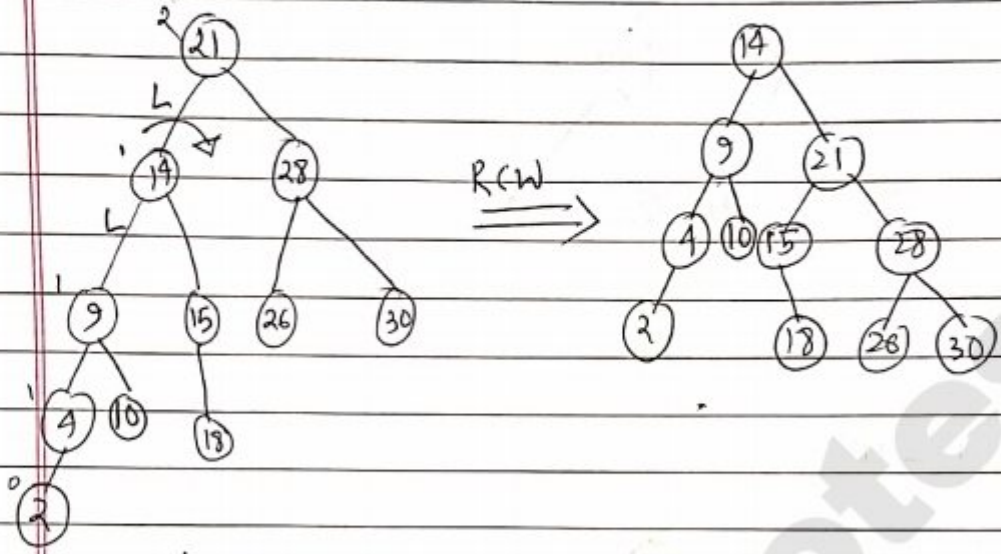
Insert - 10



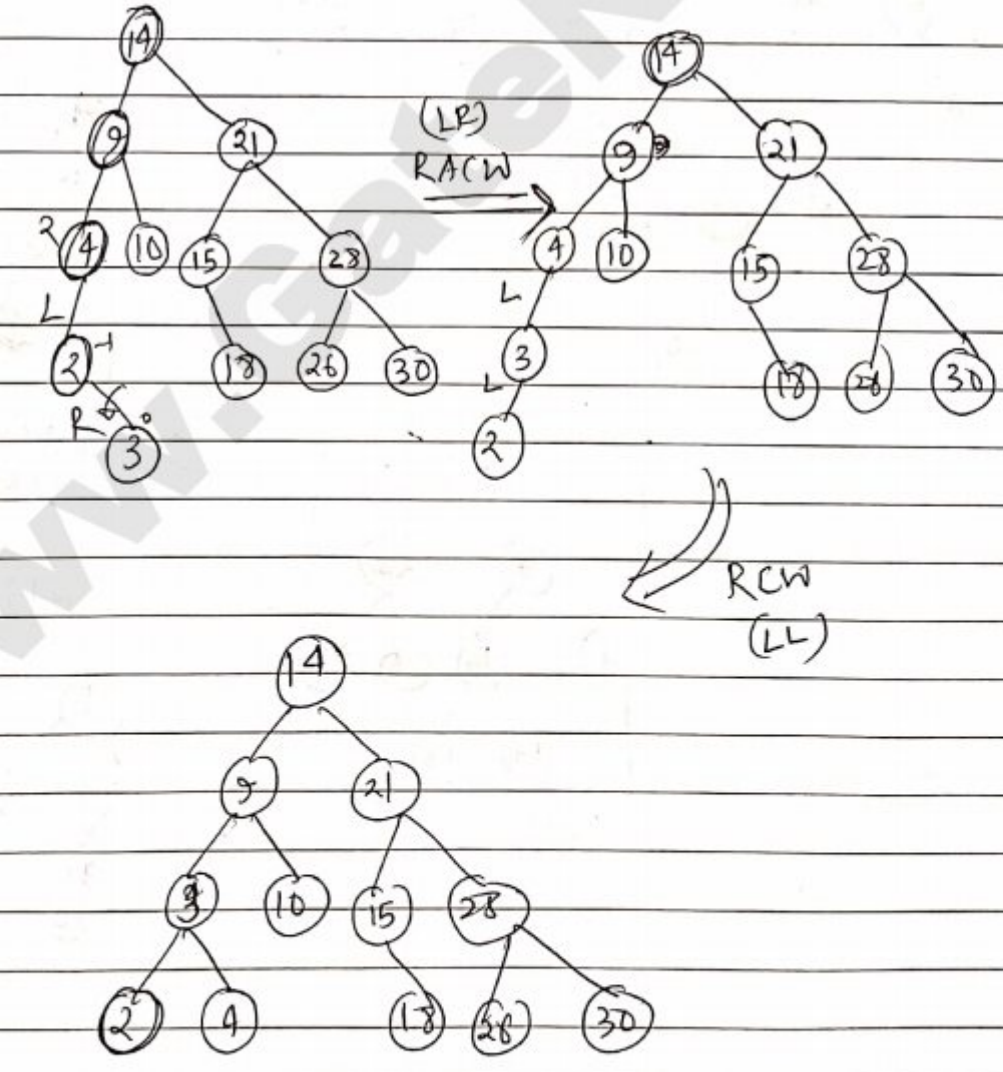
www.gatenotes.in

classmate
Date
Page

Insert-2

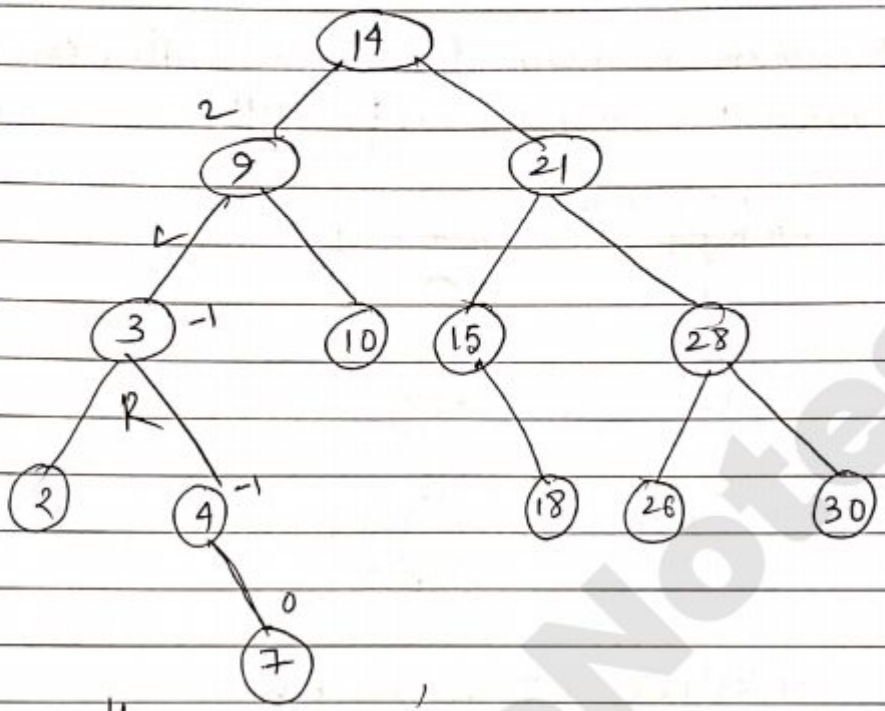


Insert-3

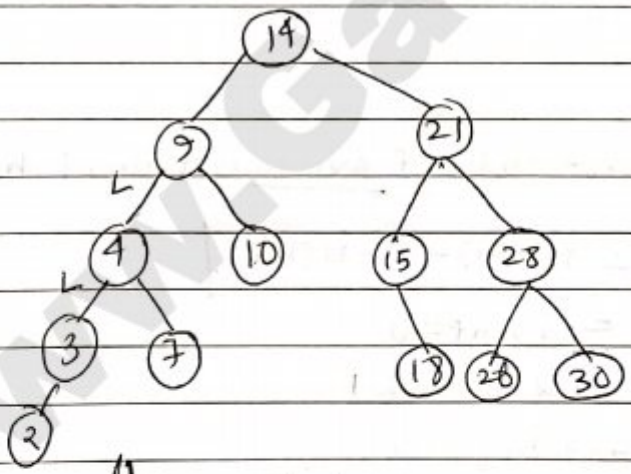


www.GateNotes.in

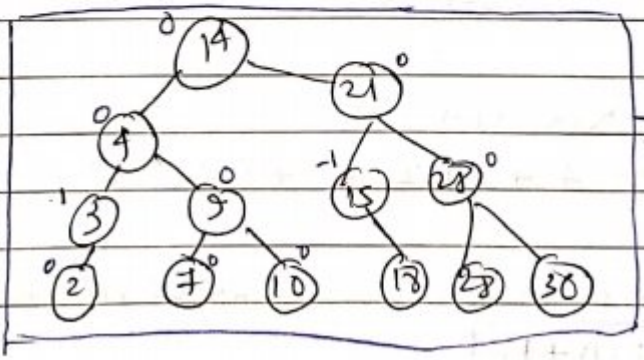
Insert ~~next~~ last ~~element~~ Input - 7



RACW (LR)



RCW (LL)

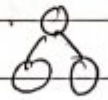
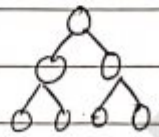


→ Final balanced tree

www.GateNotes.in

• Minimum and Maximum nodes in an AVL tree of hight h —

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

no. of node for hight	node hight
1	
$1+2^1$	
$1+2^1+2^2$	
⋮	
$1+2^1+2^2+2^3+\cdots+2^h$	for hight h

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

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

→ Minimum no. of node of AVL tree of hight-h is,

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

$$= 1 \; ; \; h = 0$$

$$= 2 \; ; \; h = 1$$

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

$$= 2 + 1 + 1$$

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

(With hight 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 hight-h $= (h+1)$

www.GateNotes.in

classmate
Date
Page

g-1995

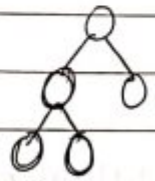
Question - (1)

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

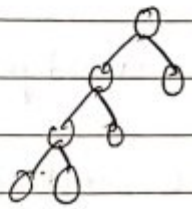
a) $\log_2 n$ b) $n-1$ c) n d) 2^n

→

1 node of degree 2 = to 2 leaf.



2 " " = 3 "



3 " " " = 4 leaf.

n node with degree 2 = n+1 leaf.

node leaf

$n = (n+1)$

~~When 'n' leaf = (n-1) node of degree 2.~~

(n no. -1) node with deg 2 = n leaf.

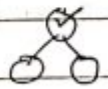
g-2004

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 =

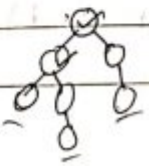
a) 10 b) 11 c) 12 d) 15

→



→ 2 leaf

10 node with deg 2 = 10+1 = 11



→ 3 leaf

www.gatenotes.in

gate-2008

Question (3)

Which of the following is true about search times —

	AVL	BST
a)	$\theta(\log n)$	$O(n)$
b)	$\theta(\log n)$	$\theta(n \log n)$
c)	$\theta(n)$	$\theta(\log n)$
d)	$\theta(n \log n)$	$O(n)$

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 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 ~~node~~ no. of nodes in a binary tree of hight 'h' is $(2^{h+1}-1)$.

gate-~~1995~~ 2005

Question (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.

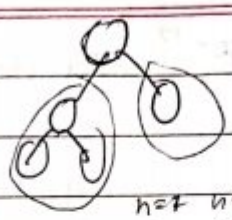
www.gatenotes.in

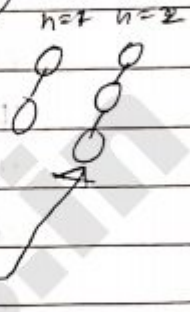
$N(h)$ → no. of node of hight h.

Recurrence equation –

→ $N(h) = N(h-1) + 1 + (N(h-1)-2)$

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





$$N(2) = 2N(2-1) - 1$$
$$= 2N(1) - 1$$
$$= 2(2) - = (3), \quad h = 2$$

$$N(h) = 2N(h-1) - 1 \quad \text{—— (I)}$$
$$N(h-1) = 2N(h-2) - 1 \quad \text{—— (II)}$$
$$N(h-2) = 2N(h-3) - 1 \quad \text{—— (III)}$$

(II) and (III) put into (I)

$$N(h) = 2^3 N(h-3) - 2^2 - 2 - 2^0$$
$$\vdots$$
$$= 2^k N(h-k) - 2^{k-1} - 2^{k-2} \pm \cdots + 1$$
$$= 2^k N(h-k) - (2^{k-1} + 2^{k+1} + \cdots + 1)$$
$$= 2^k N(h-k) - (2^k - 1)$$

$h - k = 2$
$k = h - 2$

$$= 2^{h-2} N(2) - (2^{h-2} - 1)$$
$$= 2^{h-2} * 3 - (2^{h-2} - 1)$$
$$= 3 \cdot 2^{h-2} - 2^{h-2} + 1$$
$$= 2 \cdot 2^{h-2} + 1$$
$$= (2^{h-1} + 1)$$

→ first we substitution method. By drawing the tree, according to question.

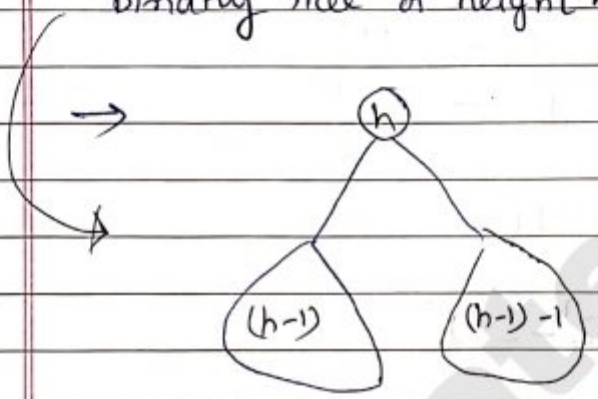
www.gatenotes.in

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 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.

Prove by induction (without) on h, that a size-balanced binary tree of height 'h' contains at least 2^h nodes.



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

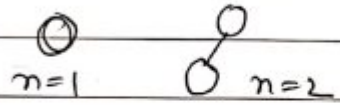
$N(h) = \underbrace{N(h-1)}_{LST} + \underbrace{1}_{root} + \underbrace{N(h-1)-1}_{RST}$

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

$\boxed{N(h) = 2N(h-1)}$ —(1)

($h=0$, min node $=1$)

$h=0$ $h=1$



$N(h-1) = 2N(h-2)$ —(2)

$N(h-2) = 2N(h-3)$ —(3)

Back substitution -

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

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

$= 2^h N(0)$.

$= 2^h \cdot 1$.

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

$h-K=0$

$K=h$

www.gatenotes.in

• Can ask Question on this topic —

(1) What is the min no. of node required ~~to make~~ in order to make hight ( ).

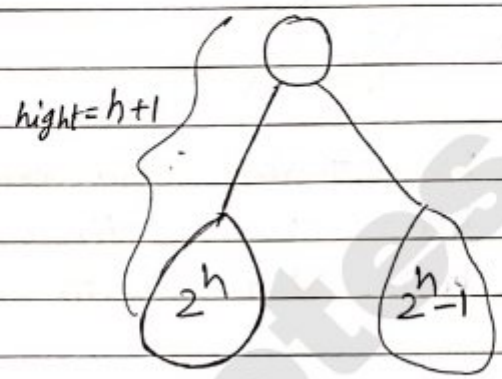
(2) given node then how much hight build.

~~2. N(h) = 2^h~~

By induction -

$N(n) = 2^h$

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



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

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

g-2014

Question - ⑦ (Balanced binary search tree)

Suppose we have a balanced binary search tree 'T' holding 'n' numbers. We are ~~giving~~ 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 T. If the highest upper bound on the time to complete the sum is $O(n^a \log^b n + m^c \log^d m)$, the value of $a + 10b + 100c + 1000d$.

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

$a = 0, \quad b = 0$

$c = 1, \quad d = 1$

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

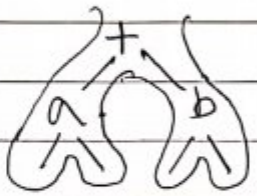
$\Rightarrow 0 + 10*0 + 100*1 + 1000*1$

$\Rightarrow$ 1100.

www.gatenotes.in

• Expression trees =

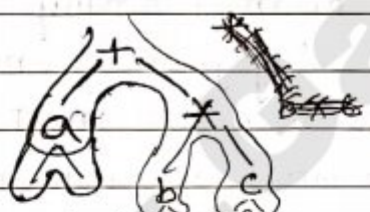
① a+b



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

IN : a-

IN : -a

④ a!

IN : a!

⑤ log a

IN : log a

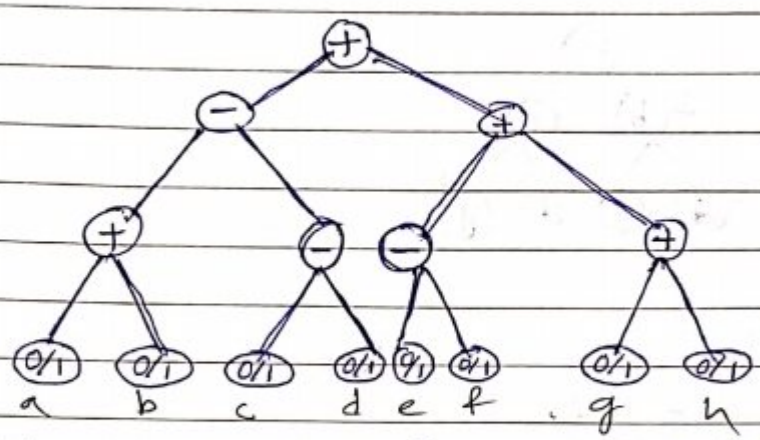
⑥ log(a!)

IN order : log(a!).

→ By traversing In order, find out the tree.
→ When combination of operators, then build tree step by step.

www.gatenotes.in

g-2014 Question-①

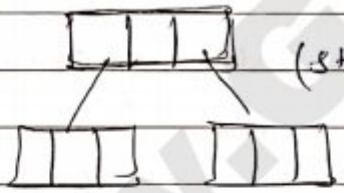


When every operand can take ~~0/1~~ '0' or '1' then what will be the max value of expression.

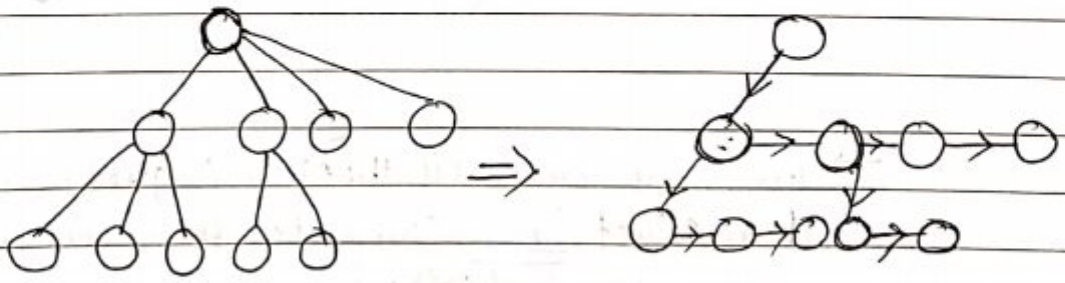
→ $\underset{1\ 1}{(a+b)} - \underset{0\ 1}{(c-d)} + \underset{1\ 0}{(e-f)} + \underset{1\ 1}{(g+h)}$

→ 3 + 3 → (6)

- Various tree representations ==

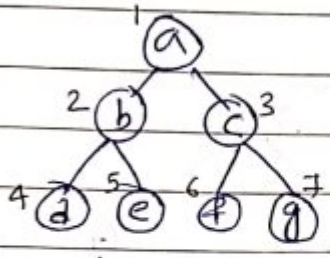

(structure representation)

- Left child Right sibling representation == (LCRS)



www.gatenotes.in

- array representation of Binary tree —



i = index

LC = $2i$

RC = $2i+1$

parent(x) = $x/2$.

A

1	2	3	4	5	6	7	
a	b	c	d	e	f	g	

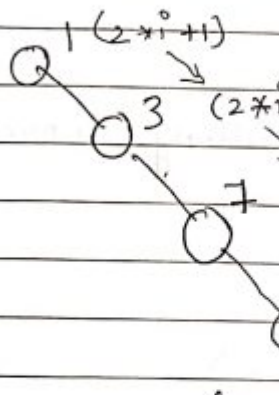
adv:

→ 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:



entire

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

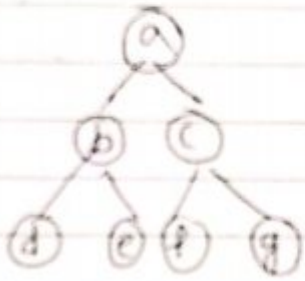
www.GateNotes.in

classmate
Date
Page

- Nested representation =>

(b a c) (a b c)
L Int R Int L R

Ex:



= a(bde)(cfg)
Int L R

4 - representation =>

(i) → Normal (pointer, struct)
(ii) → LCRS (n-ary, n>>2) ——— 

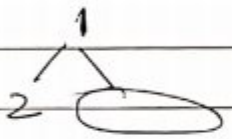
(iii) → Array (used in Heaps, all must complete BT)
(iv) → Nested.

gate-2000 **Question** - (2) Root L R
(x y z), y and z can be NULL,
which of the following does represent binary tree -
a) (12(4567))
b) (1((239)56)7)
✓ c) (1(239)(567))
d) (1(23 NULL)(45))

www.gatenotes.in

classmate
Date
Page

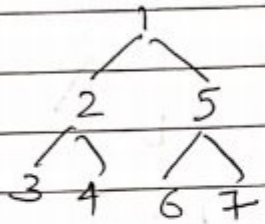
(a) (12(4567)) ✗



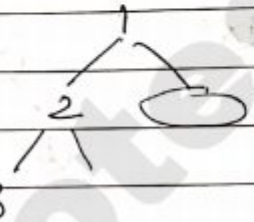
(b) (1 ((234) 56) 7) ✗

(c)

(c) (1 (234) (567)) ✓



(d) (1(23NULL)(45)) ✗

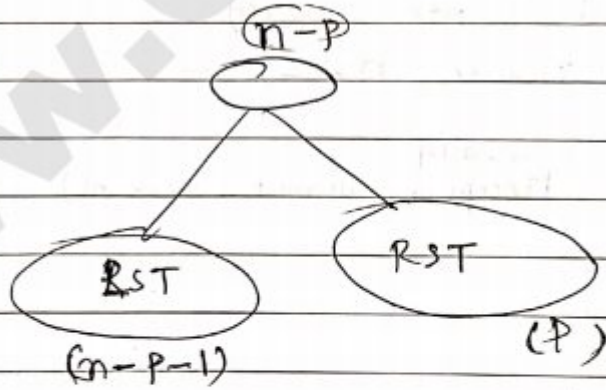


gate 2005 **Question - (3)**

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

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

→



(1 2 3 ... (n-p-1)) (n-p+1) ... n
(p)

www.GateNotes.in

gate-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'.

(8-1) The ~~dir~~ index of the parent of element X[i], i≠0, is

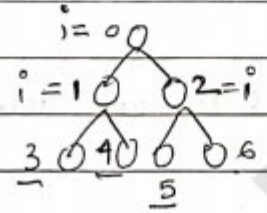
(a) $\lfloor i/2 \rfloor$ ✓ (both correct)

(b) $\lceil i/2 \rceil$

(c) $\lceil (i-2)/2 \rceil$ ✓ (both correct)

(d) $\lceil i/2 \rceil - 1$

→



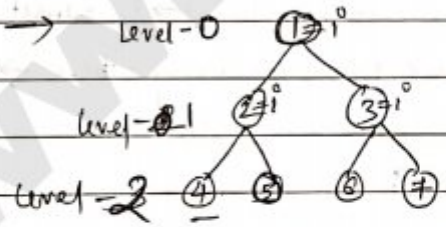
Value of i by substitute ~~them~~ in option find current option.

(8-2)

If the root node is at level 0, the level of ~~elem~~ element X[i], i≠0, is

(a) $\lfloor \log_2 i \rfloor$ (b) $\lceil \log_2 (i+1) \rceil$ (c) $\lfloor \log_2 (i+1) \rfloor$ (d) $\lceil \log_2 i \rceil$.

→



Round

$\lfloor 2.58 \rfloor = 2$

$\lceil 2.58 \rceil = 3$

by using subtitution method ~~find~~ founded correct option.

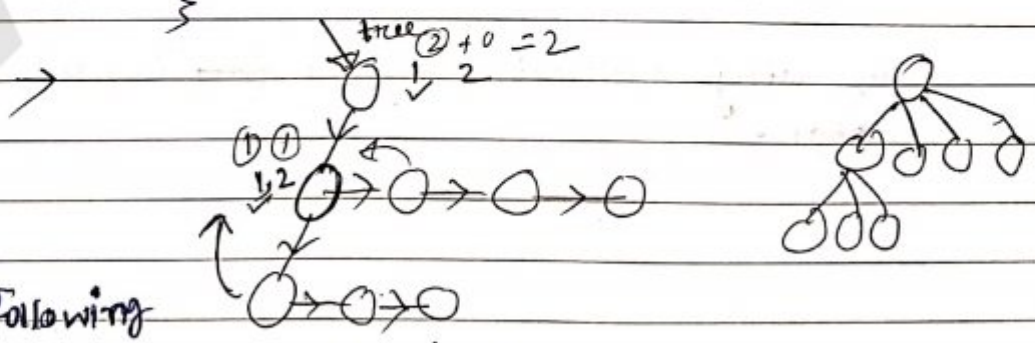
www.gatenotes.in

q-2014

Question-⑤ (LCRS)

leftmostchild – rightsibing 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);
}
```



Following ∧ Which of option is false –

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

www.gatenotes.in

classmate
Date
Page

Ø no. of leaf node are counted.

www.GateNotes.in

classmate
Date
Page

www.GateNotes.in

RECURSION

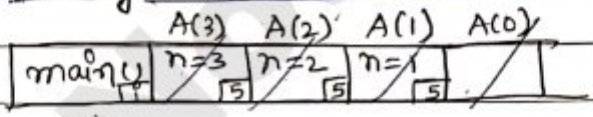
- Tracing the recursion =

ex: 1

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

```
main()
{ :
  :
  A(3);
  :
}
```

- (using stack method)



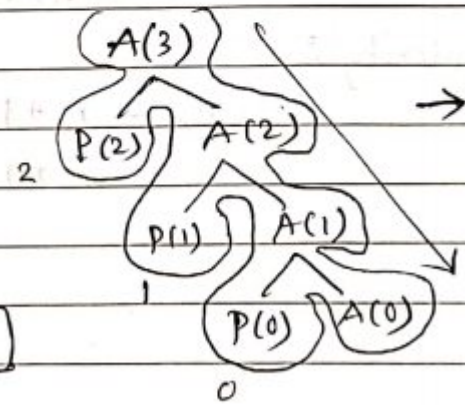
O/P: 2 1 0

→ (5 activation record are pushed)

- Space complixity, $A(n) = n+1 = O(n)$ | $A(3) = 3+1 = 4 = O(3)$.

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

- (using tree method)



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

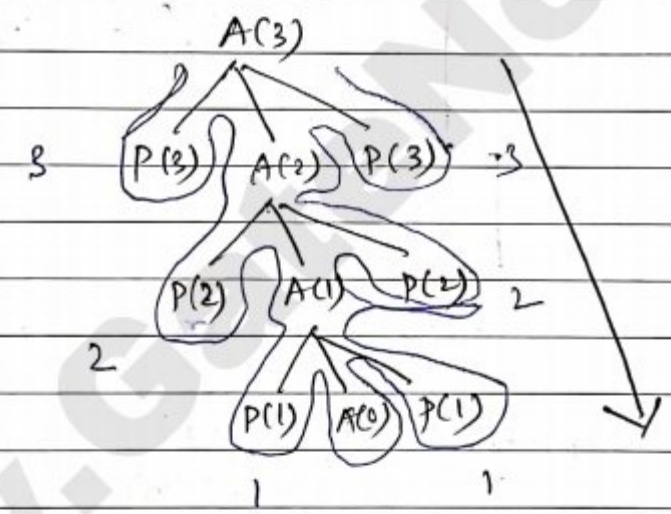
O/P = 2 1 0

www.gatenotes.in

ex:2

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

using tree method find output when n=3.



o/p: 3 2 1 1 2 3

Time complexity: (constant time required.)

$$T(n) = c + T(n-1)$$
$$= O(n).$$

Space complexity:

$$A(n) = n + 1$$
$$= O(n)$$

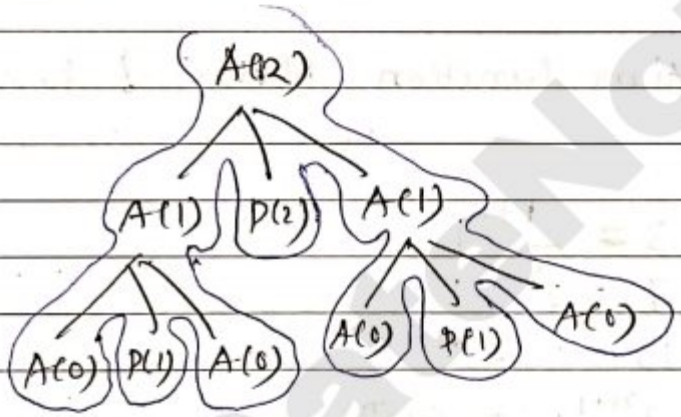
$A(3) = 4$
$= O(3)$.

www.gatenotes.in

ex: 3

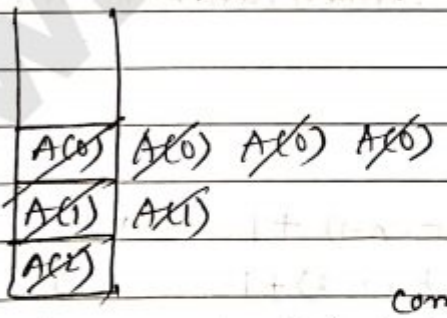
```
A(n)
{
 if(n>0)
  {
   A(n-1);
   Pf(n);
   A(n-1);
  }
}
```

What will be ~~Col~~ output ?



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

(3)



when first time visited push and last time visited poped.

complete

→ '3' record are created to execute ~~this~~.

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

→ " " " more " " " stack method.

www.gatenotes.in

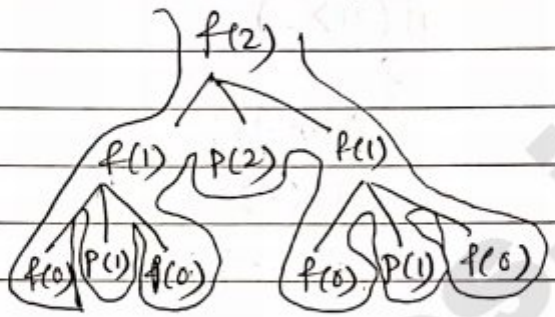
Important
Question asked on recursion.
→ Trace and find out output.

classmate
Date
Page

- **Analysing recursion =**

```
f(n)                    n=2
{
  if(n==0)
    return;
  f(n-1);
  pf(n);
  f(n-1);
}
```

tree method -



output : 1 2 1 f(2) = 7 (times function invocated)

- how many time function f(10) = ? invocated when f(10) ?

→

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

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

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

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

C(1) = 1
C(2) = 2
C(10) = 10

$f(10) = 2^{10+1} - 1$

= 2047 (no. of times functions are invocated).

other way -

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

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

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

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

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

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

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

$F(n) = 2^i F(n-i) + 2^{i-1} + 2^{i-2} + \cdots 1.$

www.gatenotes.in

classmate
Date
Page

$F(n) = 1, \; n = 0$ | $n - i = 0, \; i = n$

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

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

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

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

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

(Total no. of fun call required).

time complexity,

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

space complexity,

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

Gate-2001

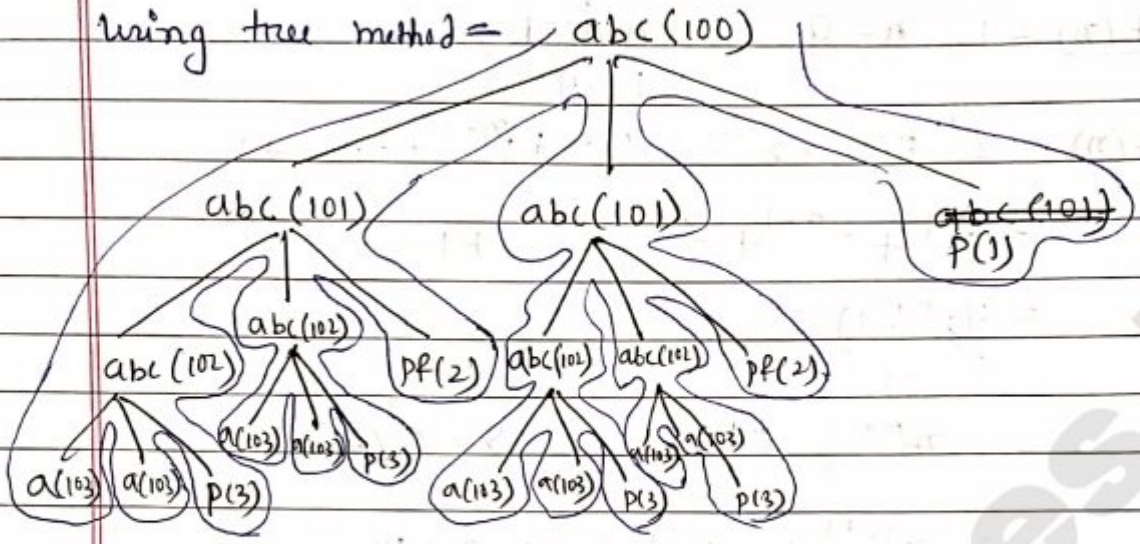
(81)

```
void abc (char *s)
{
 if (s[0] == '\0')
  return;
 abc (s+1);
 abc (s+1);
 pf("%c", s[0]);
}
main()
{ abc ("123");
}
```

1	2	3	\0
100	101	102	103

www.GateNotes.in

using tree method = abc(100)



(3323321) - output.
(7-characters printed)

(82)

```
void abc(char *s)
{
 if (s[0] == '\0') return 0;
 abc(s+1);
 abc(s+1);
 printf("%c", s[0]);
}
main()
{
 abc("123");
}
```

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

→ C(1) = 1

no. of characters printed.

www.gatenotes.in

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

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

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

$C(1) = 1$

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

$n - i = 1$

$i = n-1$

$C(n) = 2^i C(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 1$

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

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

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

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

$= 2^3 - 1$

$= 7$

Gate-2004

(B3)

```
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) $O(n^2)$ d) $O(2^n)$.

→

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

$T(n) = 1$ if $n == 1$

$(n - i = 1)$

$T(1) = 1$

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

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

www.gatenotes.in

classmate
Date
Page

Gate-2005

(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()
a=2048	n=2048	n=204	n=20	n=2	n=0
sum=0	sum=8	sum=8+4	sum=12	sum=12+2	sum=14
	K=8	K=4	K=0	K=2	K=0
	j=204	j=20	j=2	j=0	j=0
	6	6	6	6	pop

bottom stack

Ans 2, 0, 4, 8, 0 – output

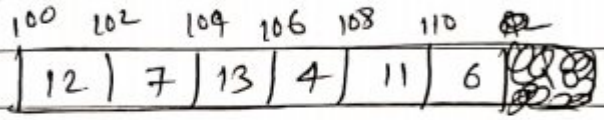
www.GateNotes.in

Gate-2005 2010

(Q5)

```
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



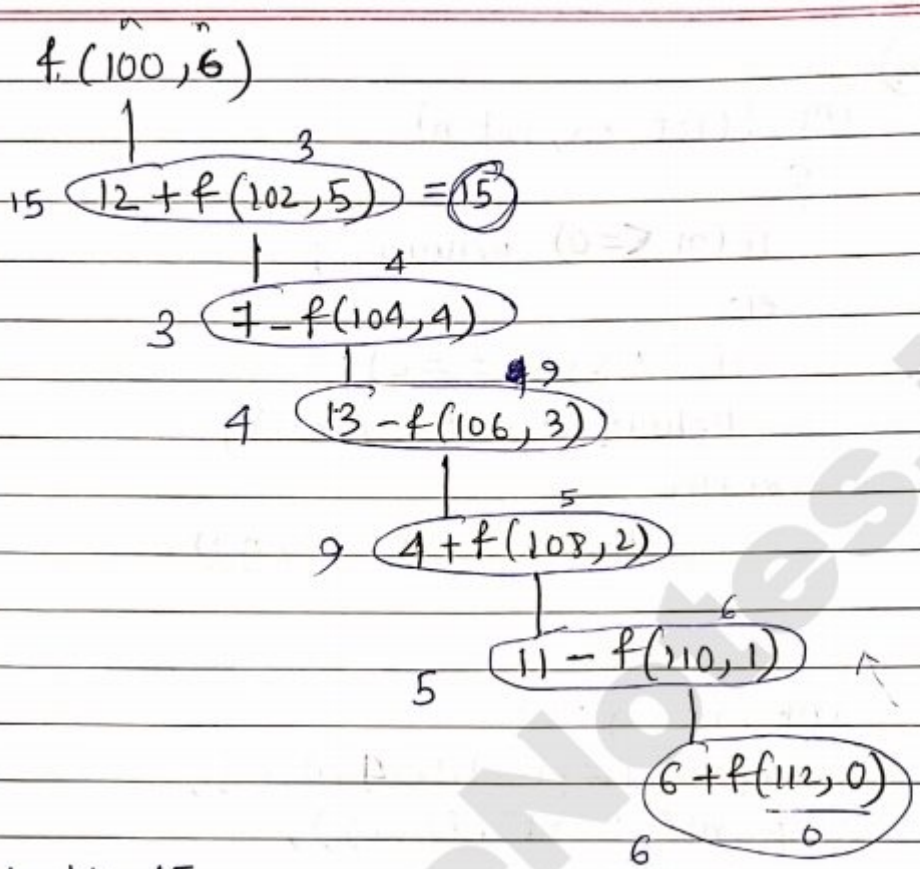
f(100, 6)

↓

next page.

www.GateNotes.in

classmate
Date
Page



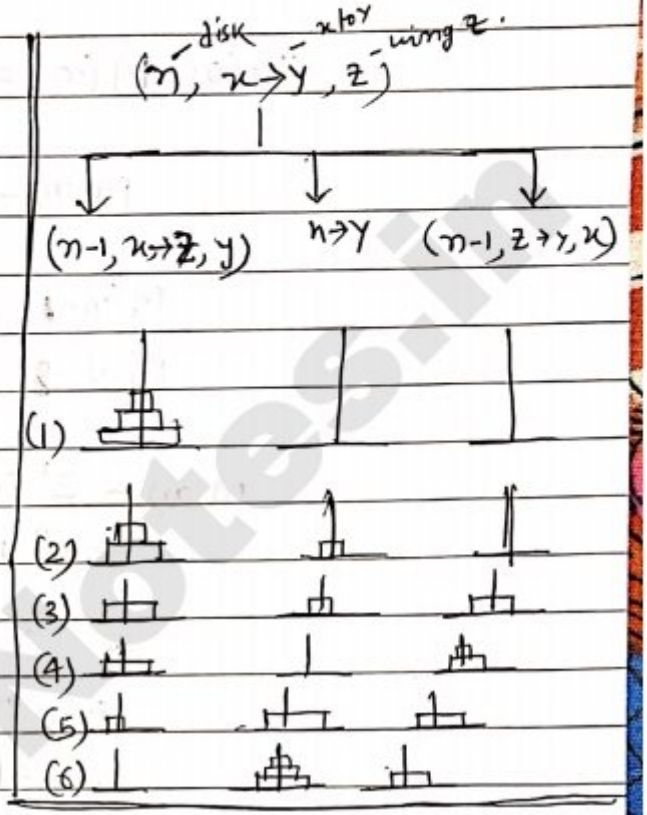
output: 15

www.GateNotes.in

classmate
Date
Page

Analysing the recursion program of Towers of Hanoi =

① TOH(n, x, y, z)
{
if(n>=1)
{
TOH(n-1, x, z, y);
Move top 'x' to 'y';
TOH(n-1, z, y, x);
}
}



ii) No. of function invocation required for 'n' disk =

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

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

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

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

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

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

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

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

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

$= 2^n + 2^{n-1} + 2^{n-2} + \cdots + 1.$

$O(p) = 1\,\frac{(2^{n+1}-1)}{2-1} = \boxed{2^{n+1}-1 = O(2^n)}$

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

For 3 disk $= 2^{3+1} - 1$

$= 15$

www.gatenotes.in

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

for 3 disk = $2^3 - 1$ = 7 Movement.

~~M(0)~~ $M(n) = 2M(n-1) + 1$

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

$M(n-1) = 2^1 M(n-2) + 1$ $\quad n - i = 0$

$M(n-2) = 2M(n-3) + 1$ $\quad n = i$

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

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

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

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

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

$= O(2^n)$.

$T(n) = 2T(n-1) + 1$ → constant time

$T(n) = 1$, $n = 0$ $\quad n - i = 0$, $i = n$

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

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

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

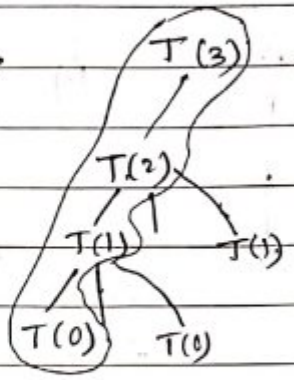
$= 2^n \cdot 2^1 - 1$

$= O(2^n)$.

www.gatenotes.in

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

$\Rightarrow$ depth of the tree.



$S(3) = 3+1 = 4$

(2) TOH(n, x, y, z) $\rightarrow$ modification of previous Algo. to minimize no. of function invocation.

```
{
 if(n > 1)
  { TOH(n-1, x, z, y);
    move top 'x' to 'y';
    TOH(n-1, z, y, x);
  }
 else if (n == 1)
   move "x → y";
}
```

no. of function
now ~~Invocat~~ Invocation required – $I(n) = 2^n - 1 = O(2^n)$

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

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

$I(n) = 1, \quad n = 1$ $\qquad n - i = 1$

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

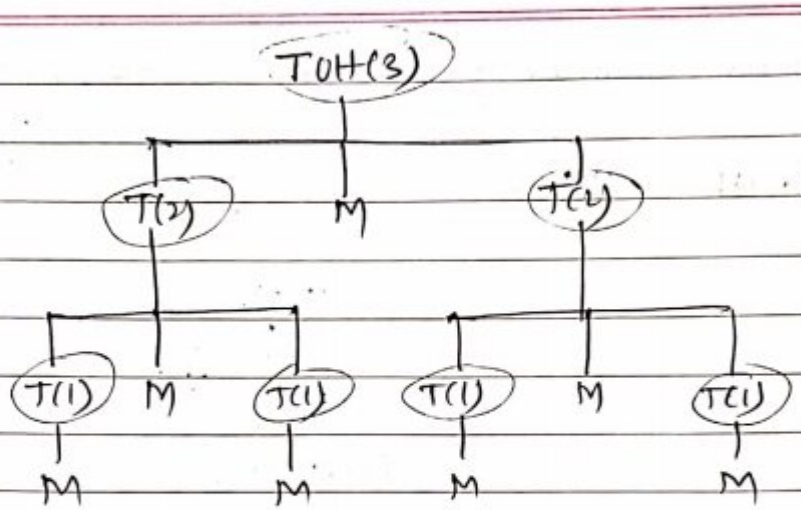
$= 2^{n-1} I(1) + 2^{n-2} + 2^{n-3} + \cdots + 1$

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

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

www.gatenotes.in

classmate
Date
Page



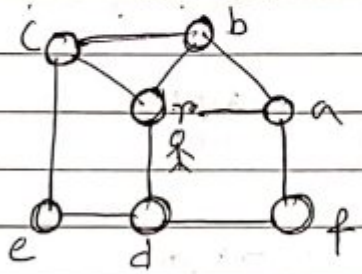
Note
total function
invocation is 7.

www.GateNotes.in

GRAPHS

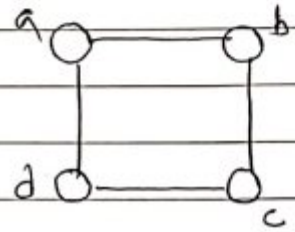
- Introduction of Graphs =

→ set of vertices and edges is called graph.



adjacency of x - (d c b a)
(friend)
(friend list)

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

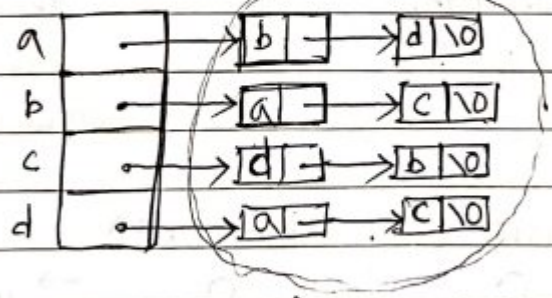


- Adjacency matrix =

	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 ~~dese~~ Dense (if the no. of edge very very high) then go with matrix representation.

$E = O(V^2)$

www.GateNotes.in

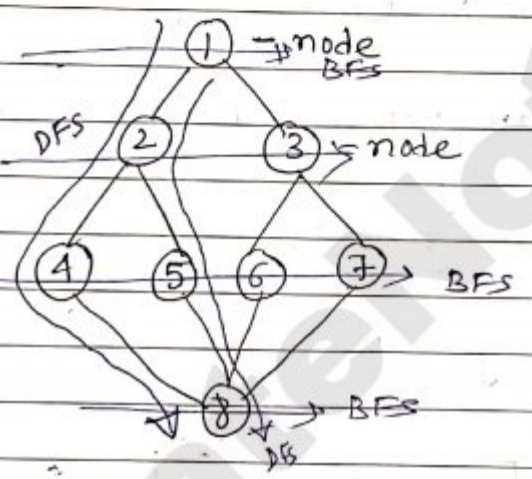
→ When the graph is sparse (few edges), then go with ~~ede~~ Adjacency list representation.

$$E = O(V).$$

- Introduction of BFS and DFS =

↓ breadth first search

↓ depth first search



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

vertices 3-types —

1→ visited, not-visited, Explored.

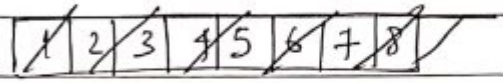
↓ seen that vertex

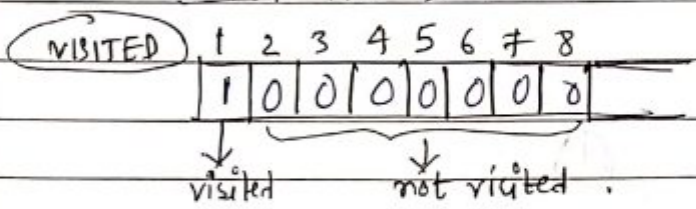
↓ seen it and as well seen all vertex adj with that vertex.

	VISITED	EXPLORED	
Case-1	0	0	→ not visited, not explored.
c-2	1	0	→ visited, but not explored.
c-3	1	1	→ visited and explored.

www.gatenotes.in

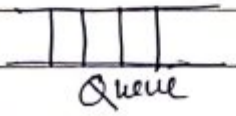
→ Keep track of this two things (visited, Explored) both the algorithm maintain an array - visited or not





✓ space com = O(n).

→ The Algo which use queue to keep track of all unexplored vertice is called - BFS.



— BFS ✓ space complexity = O(n)

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

— DFS

✓ space complexity = O(n).

- BFS algorithm = BFS(v) → address of 1st node

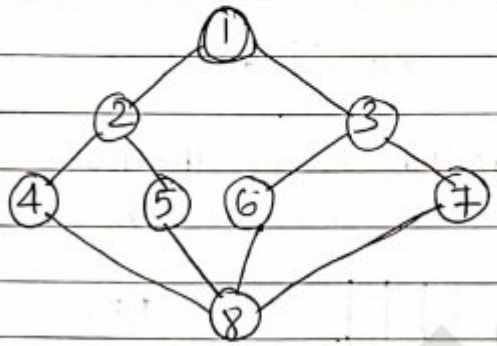
```
// The graph 'G' and array visited[] are global;
   visited[] is initialized to '0'.
{
   u = v;  visited[v] = 1;
   repeat
   {
      for all vertices w adj to u
      { if (visited[w] == 0)
        { add w to queue;
          visited[w] = 1; }
      }
```

→ pf('w').

www.gatenotes.in

if queue is empty then return;
Delete the next element, u from queue, and add to u;
}
}

Example =



array | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |

u = ~~1~~ 2 3 4 5 6 7 8 → NOW all vertices visited

| 2 | 3 | 4 | 5 | 6 | 7 | 8 |

queue

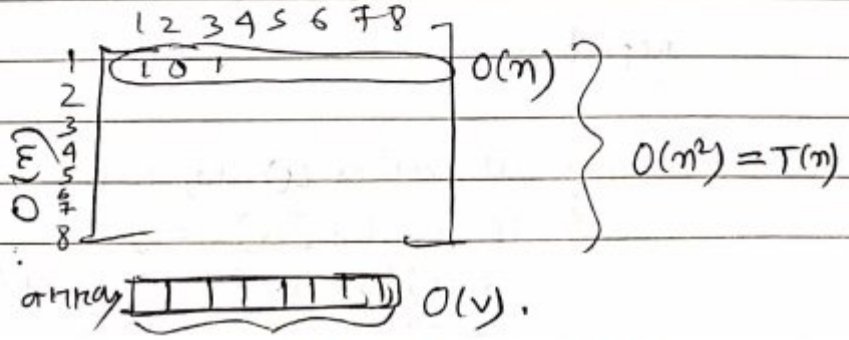
v = 1
w = 2, 3 ✓
v = 2
w = 1, 4, 5 ✓
v = 3
w = 1, 6, 7 ✓
v = 4
w = 2, 8

• BFS analysis on adjacency matrix implementation =

Time complexity = $T(n) = O(n^2)$
$= O(v^2)$

v → vertices

Space complexity = $O(v)$

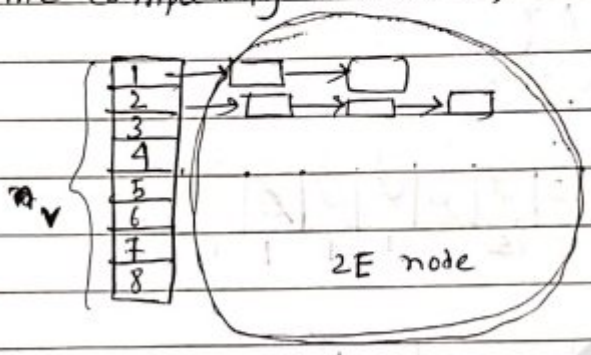


www.gatenotes.in

• BFS analysis In case of linked list implementation of Graph=

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

✓ time complexity = $O(V+E)$



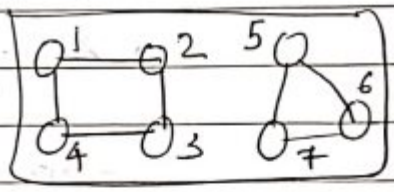
E → edges.

• Breadth First Traversal using BFS =

```
BFT(G, n)
{
  for i=1 to n do
      visited[i] = 0
  for i=1 to n do
      if (visited[i]==0) then
                    BFS(i);
}
```

i = 1 2 3 4 5 6 7 8

visited | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



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

Space complexity =

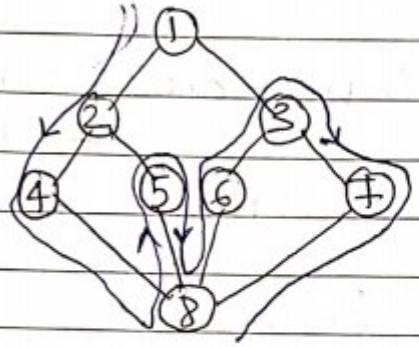
→ Time and ~~Com~~space complexity of BFT are same as BFS.

• DFS algorithm =

```
DFS(v)
{
  visited[v] = 1;

  for each vertex w adj to v do
    { if (visited[w] == 0) then
        DFS(w);
    }
}
```

www.gatenotes.in

Example –



DFS.

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

stack	v = 1	v = 2	v = 4	v = 8	v = 5	v = 6	v = 3	v = 7
	W = {2, 3}	W = {1, 4, 5}	W = {2, 8}	W = {4, 5, 6, 7, 3}	W = {2, 8}	W = {8}	W = {6, 7}	W = {3, 8}

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

- Analysis of DFS and DFT =

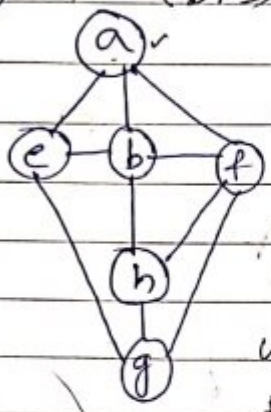
→ In case of Adj° Matrix –
Time complexity = $O(v^2)$
Space complexity = $O(V)$

→ In case of Adj list –
Time complexity = $O(V+E)$
Space complexity = $O(V)$.

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

www.gatenotes.in

q-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 ~~Pass~~ 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,h,f}	W={b,f,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,h,g}	W={b,f,g}	W={e,h,f}	W={a,b,g}

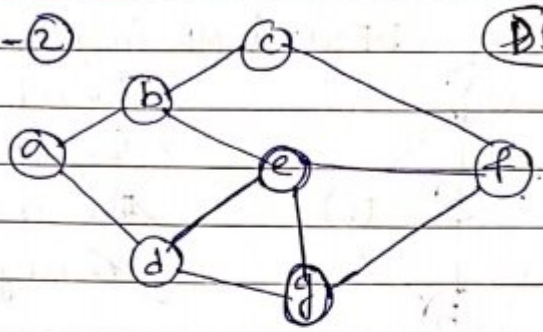
IV.

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,f,h}	W={a,b,g}

www.gatenotes.in

gate-2008

Question - (2) DFS



Which of the following sequence are possible —

✗1) a b e f d g c . (not possible)

✓2) a b e f c g d .

✓3) a d g e b c f .

✗4) a d b c g e f . (not possible)

→

1), 2)

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={d, f, g}

3)

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

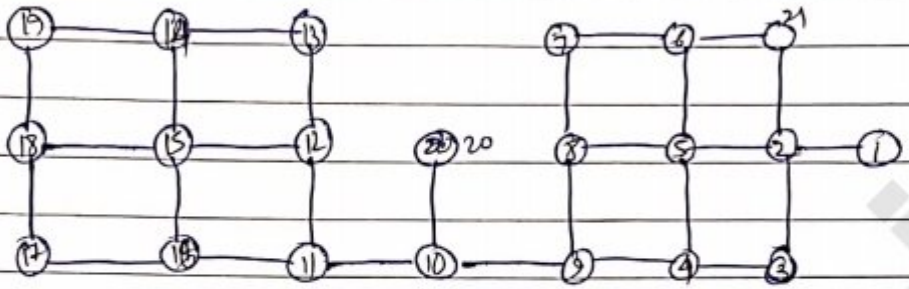
4)

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

www.GateNotes.in

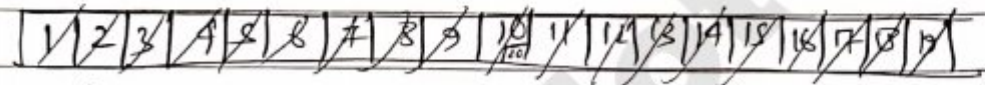
g-2014

Question - (3) (DFS)



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

→



stack

(19 node can present 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 units f(P) = 12 units
d(Q) = 6 units f(Q) = 10 units
d(R) = 14 units f(R) = 18 units

What is true about the graph?

→

→ P and Q are in same module.

→ R in other module.

5 6 10 12 14 18
d(P) d(Q) f(Q) f(P) d(R) f(R)

→ So, P and Q are adj

www.gatenotes.in

classmate
Date
Page

www.GateNotes.in

Hashing

- Direct Address Table =
  (DAT)

Searching time of some DS –

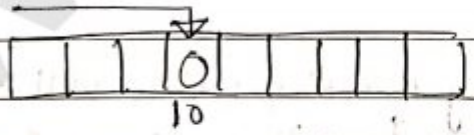
(i) Unsorted array – $O(n)$
(ii) sorted array – $O(\log n)$
(iii) Linked list – $O(n)$.
(iv) Binary tree(BT) – $O(n)$
(v) Binary search tree – $O(n)$
(vi) Balanced BST – $O(\log n)$.
(vii) priority queue (min & max heap) – $O(n)$.

→ 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 usefull only when you know the no. of keys which are going to in small range and

www.gatenotes.in

In case if the Key's used are very larger numbers, but the total no. of Key's are less in number.

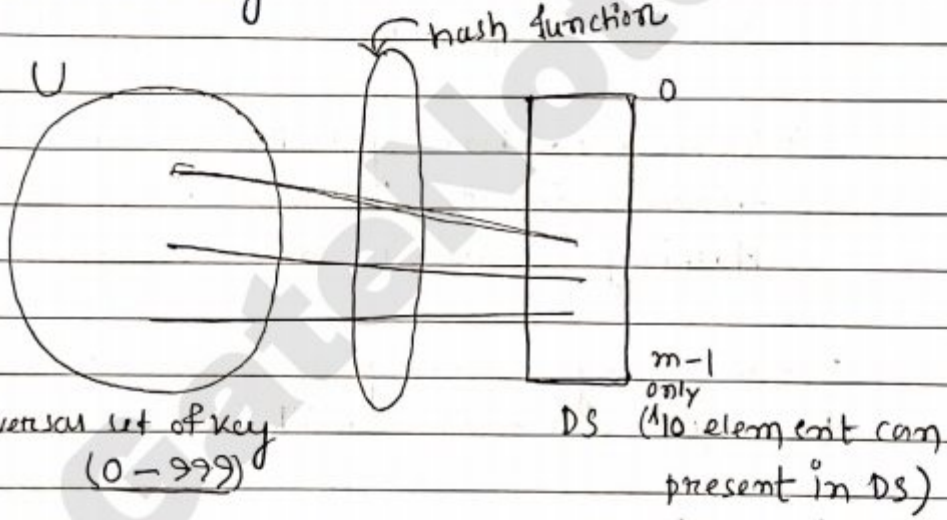
Key - {100000, 100001, 100002, ..}

	100000	100001	...

so, that the array size will be 1M, just to store 10 value. 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 - (mod 10)

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

(size-10)

www.gatenotes.in

When searching for an element - 150, 145

150 % 10
= 0
element is not present.

145 % 10
= 5 element is present.

→ Insertion, deletion and searching is going to take constant time by using the hashing technique.

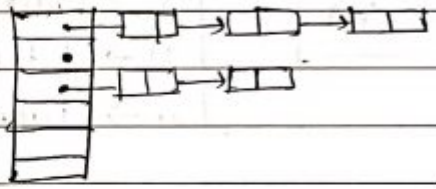
→ ~~In hashing technique, there is a problem called~~
Or
→ Collision problem =

↳ 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. (I, D, S)



(For deletion chaining is better)

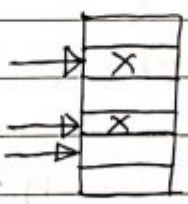
(iii) Open addressing. (I, S)

(probing – Searching for a empty space to insert an element)

(i) linear probing

(ii) quad probing

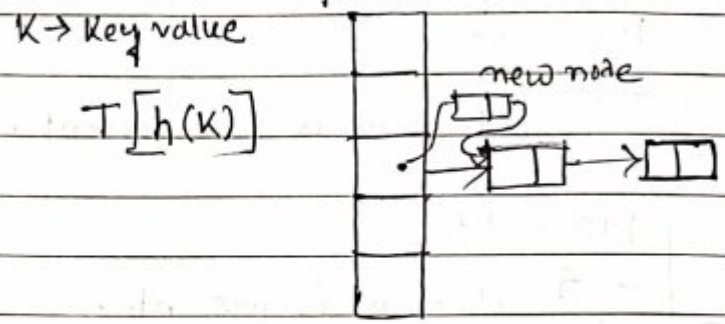
(iii) Double hashing



(For deletion open addressing is not better)

www.gatenotes.in

• Collision solving technique : chaining.



hash table

~~Time taken~~

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

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

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

→ In worst case Deletion time = O(n).

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

α → load factor.
n → total element present in table.
m → total size of table

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

$n = Km$ (constant) $= \theta(1+\alpha)$

$\frac{n}{m} = K(\alpha)$ $= \theta(1)$

average deletion time = θ(1).

~~Insertion~~

www.gatenotes.in

~~Adv:~~ ~~Adv:~~

→ deletion is easy in chaining.

disadv:

→ pointers (space wasted).

1996
g-2014

Question-(1)

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

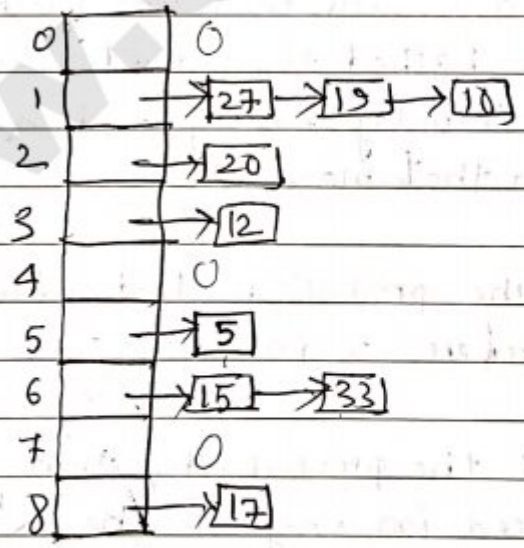
a) Worst case complexity of search is less.
b) space used is less.
c) deletion is easier.
d) none of the above.

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.

→



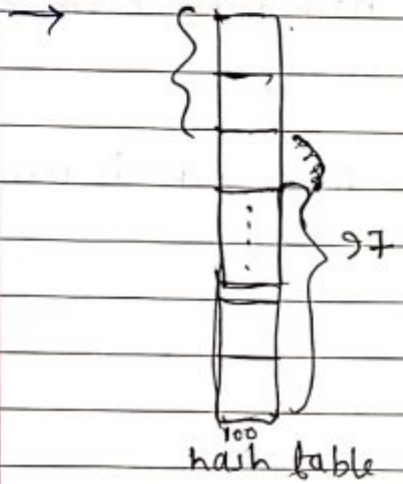
✓ max chain length → 3.
✓ min " " → 0.
✓ avg " " → $\frac{0+3+1+1+0+1+2+0+1}{9} \rightarrow 1.$

www.gatenotes.in

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?



$\left(\frac{97}{100}\right)\left(\frac{97}{100}\right)\left(\frac{97}{100}\right)$

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 $1/n$. 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 occured in any of the 'K' insertions?

c) What is the prov that first collision occurs at the K^{th} insertion.

www.GateNotes.in

(a) → $1/n$

$(1 - 1/n)$

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

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

(b) $\left(\frac{n}{n}\right)\left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right)\cdots\left(\frac{n-(K-1)}{n}\right)$

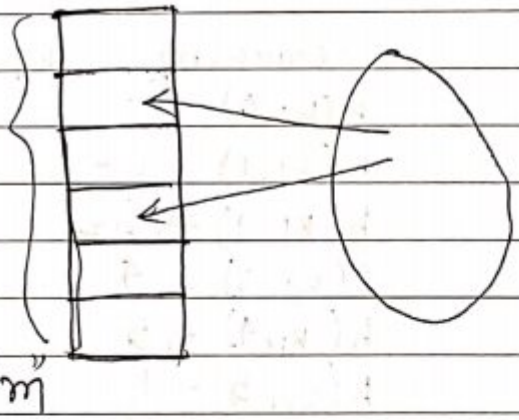
1^{st} ele 2^{nd} ele 3^{rd} ele ---- K^{th} element.

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

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

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

$0 \leq \alpha \leq 1$



$h: U \to \{0, 1, \cdots, m-1\}$

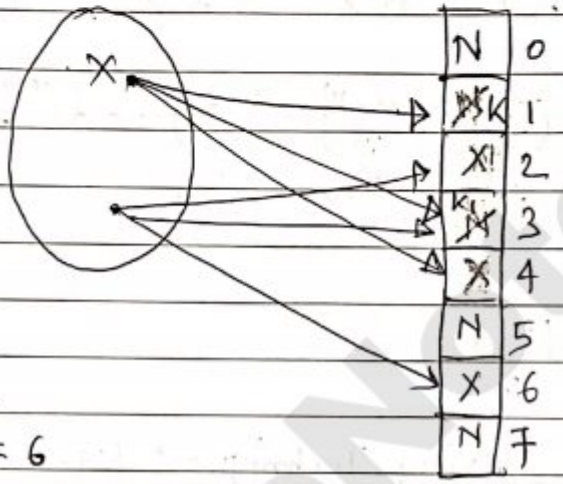
$h: (U \times \{0, 1, 2, \cdots m-1\}) \to \{0, 1, \cdots, m-1\}$.

probe sequence: Whenever I try to Insert a key then follow a sequence of examinations, in case if you find any empty cell in this ~~et~~ sequence you are going to insert the element there, if you don't find any ~~sequence~~ empty cell in that sequence; then declared

www.gatenotes.in

that entire table is full, you are not able to inserted.

Worst time take to searching - O(m)



$h(K) = 1$
$h(K_1) = 2$
$h(K_1,1) = 6$
$h(K_1,2) = 3$ ← no. of collision

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 unsuccessfull search the time complexity for a search is $- O(n)$.

In Average case time complexity $- O(1)$.

www.gatenotes.in

Insert an new element K_1,

$h(K_1, 0) = 2$ ✓

$h(K_1, 1) = 6$ ✓

$h(K_1, 2) = 3$ ✓

$h(K_1, 3) = 4$ ✓

$h(K_1, 4) = 5$ ✓

$h(K_1, 5) = 1$

$h(K_1, 6) = 7$

$h(K_1, 7) = 0$

0	N
1	N
2	a
3	c
4	d
5	K_1
6	b
7	N

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

Then Delete an element 'c' from table.
after deleting c search for K_1.

0	N	
1	N	
2	a	
3	N	→ stop searching here.
4	d	and declare
5	K_1	that K_1 is not
6	b	present even
7	N	when K_1 is present.

→ In case of deletion open addressing creat 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 earlyer there but now it is deleted.

www.gatenotes.in

classmate
Date
Page

0	N
1	N
2	a
3	D → D indicate that, earlier there was
4	d an element but
5	(K1) now it deleted.
6	b
7	N

- Various technique which used in Open addressing =

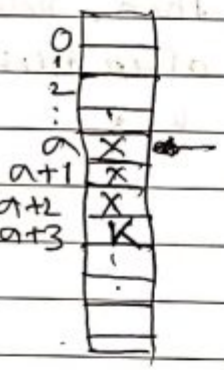
(i) Linear probing :

hash function =

$h : U \to \{0, \cdots, m-1\}$

$h(K) = a$

$\boxed{h'(K,i) = (h(K)+i) \bmod m}$



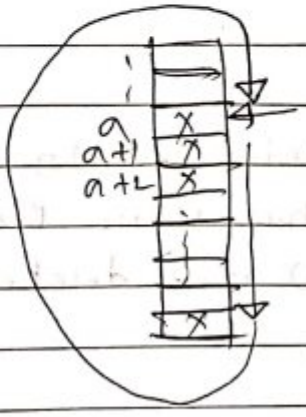
$h'(K,1) = (h(K)+1) \bmod m$

$= (a+1) \bmod m.$

$h'(K,2) = (h(K)+2) \bmod m$

$= (a+2) \bmod m.$

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



Probe sequence :-

(0, 1, 2, 3, ..., m-1) always.

(1, 2, 3, ..., m-1, 0) search for 'm' element

(2, 3, 4, ..., m-1, 0, 1) to insert

(3, 4, 5, ..., m-1, 0, 1, 2) element.

so, probe sequence is possible is "m".

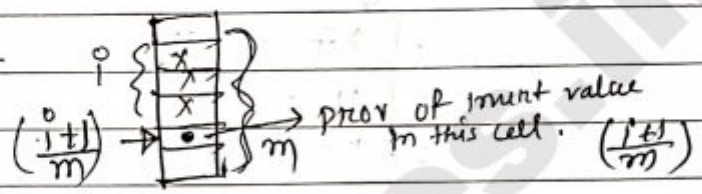
www.gatenotes.in

classmate
Date
Page

Secondary Clustering =

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

Primary clustering =



~~Average~~
dis adv: linear probing suffer two things (secondary and primary clustering)

[Average ~~search~~ time taken to search an element = O(2.5).
Worst case — O(n).

gate 2008 **Question — ①**

Key: 12, 18, 13, 2, 3, 23, 5 and 15.

HST = 10 (Hash table size)
h(K) = K mod 10 and linear probing. What is resultant hash table?

$h'(k,i) = (h(k) + i) \bmod 10$

$h'(2,0) = (h(2) + 0) \bmod 10$
$= (2+0) \bmod 10$
$= 2$

$h'(2,1) = (2+1) \bmod 10 = 3$
$h'(2,2) = (2+2) \bmod 10 = 4$

www.gatenotes.in

gate-2008

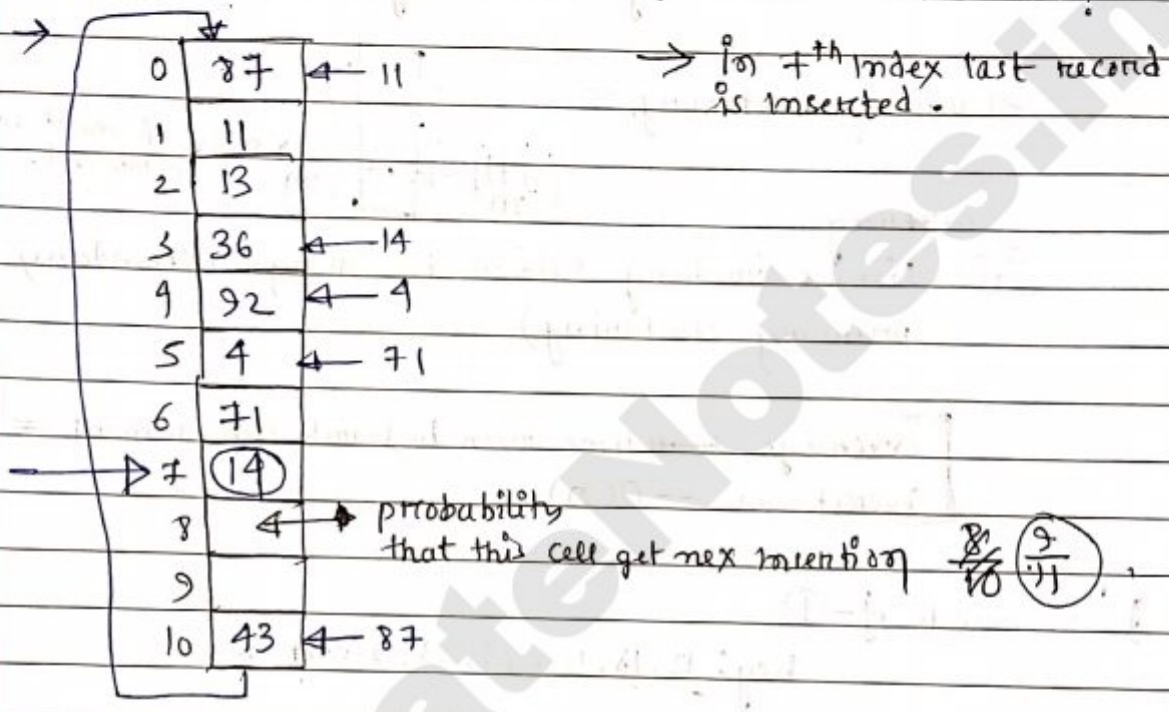
Question-②

HTS = 11 (0-10) using

$h(K) = K \bmod 11$; linear probing method.

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

What is the index into the last record is inserted?

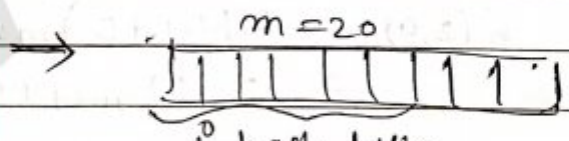


gate-2007

Question-③

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

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



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

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

www.gatenotes.in

gate 2010

Question - (4)

$h(K) = K \bmod 10$, linear probing.

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

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

✗a) 46, 42, 34, 52, 23, 33

✗b) 34, 42, 23, 52, 33, 46

✓c) 46, 34, 42, 23, 52, 33

✗d) 42, 46, 33, 23, 34, 52.

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

→ (42 – 23 – 34) – 52 – (33)

3!

46 can take place any of the blank space.

$3! \times 5 = 30$ (insertion) sequence possible.

www.GateNotes.in

(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)
$c_1 = 1, c_2 = 1$
$h(K) = K \bmod 10$.

$h(K_1) = 1$

$h'(K_1, 1) = 1 + 1 \cdot 1 + 1 \cdot 1^2$
$= 3$

$h(K_1, 2) = 1 + 2 + 4$
$= 7$

$h(K_1, 3) = 1 + 3 + 9$
$= (13) \bmod 10 = 3$

$h(K_1, 9) = 1 + 4 + 16 = 21 \bmod 10$
$= 1$

~~after 10 probs~~

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

"m"

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

You should ~~exam~~ choose c_1, c_2 and m value ~~very well~~. Carefully. Other wise it ~~will~~ is going to lead problems.

www.gatenotes.in

atlantis
Date
Page

probing sequence -

K_1 → (1, 3, 7, 3, ...)

K_2 → (1, 3, 7, 3, ...)

} → secondary Clustering (problem).

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

two keys

→ 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 technique. (iii) **Double hashing**

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 problem.

(i) $(h_2(K), m)$
- $h_2(K)$: $< m$, odd
- m: prime, 2^K

→ Hashing is better compare to other DS. When the most populere operation is only searching and Data is static).

www.gatenotes.in

atlantis
Date
Page

www.GateNotes.in