Stacks & Queues

Stacks

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

Applications

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

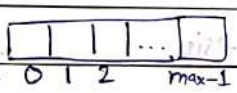
Implementation of Stack using Arrays

Disadvantage:- Predict the size of stack (array)

Advantage:- ① Faster ② 2

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

```
int stack[MAX];
int top = -1 //empty
```




```
void push (int item)
{
    if (top == (MAX-1)
        printf ("Overflow");
    else {
        top = top+1;
        stack [top] = item;
    }
}
```

→ stack [++top] = item.

```
int pop ( )
{   int temp;
    if (top == -1)
        printf ("Underflow");
    return -1;
```

```
    else {
    temp = stack[top--];
    return temp;
}
```

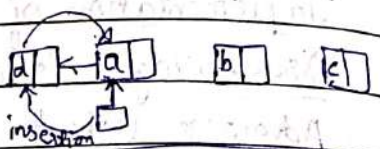
Note:- Here, if we delete an element, top is decremented. Later we push some element, we override the previous element

Takes O(1) time.

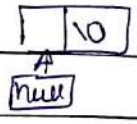
Linked list implementation of stack

Elements are created from heap memory dynamically

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




```
push(int item)
{
    struct node *p = (struct node*) malloc (sizeof(struct node));
    if (p == NULL)
    {
        printf("Malloc error");
        return;
    }
    p->data = item
    p->link = NULL;
    p->link = head;        //new created node linked to old head
    head = p               // old head is updated to new head
}
```



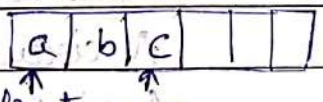
//p → size

```
    int pop()
    {
        int item; struct node *p;
```

```
if(head==NULL)
{
    printf("Underflow");
    return -1;
}
item = head->i;
p = head;
head = head->next;
free(p);
}
```

Implementation of a Queue using circular array

A Queue is a empty hollow cylinder. Both its ends are open we shall put watermelons into it. It will drop due to gravitational force



Front → Return first element in the queue (~~// last element~~)

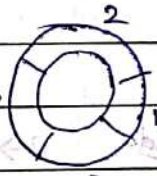
Rear → Return last element in the queue

Deletion is done from front

Insertion is done from rear

so that the insertion order of queue is maintained

```
Enqueue(item)
{
    rear = (rear+1) mod n;
    if(front==rear)
    {
        printf("Q is full");
        if(rear==0)
            rear = n-1;
        else
            rear = rear-1;
        return;
    }
    else {
        q[rear] = item;
        return;
    }
}
```



// Rear & front points to 0 it is overflow

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

GATE 2012

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

Sol:- Queue Full

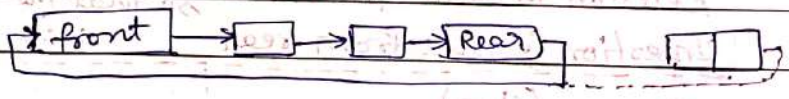
(rear + 1) mod n == front

Queue Empty

rear == full

Gate 2004

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



p →

Sol: p → rear

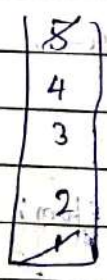
Gate 1994

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

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

Check for a)

3 4 5 2 1



c)

d)

1, 5, 4, 3, 2

5, 4, 3, 1, 2

(b) is

SURYA Gold
Date Page

Gate 1997

A priority queue 'Q' is used to implement a stack 'S' that stores characters PUSH(C) is implemented as Insert(Q,C,K) where K is appropriate integer key chosen by implementation as DEletemin(Q). For a sequence of push operations, the keys chosen are in

a) non-increasing order // decreasing order
b) non-decreasing order
c) strictly increasing order
d) strictly decreasing order

Gate 2003

Let 'S' be stack of size $N \geq 1$ starting with the empty stack. Suppose we push first 'n' natural numbers in sequence, and then n pop operations. Assume that push & pop operations take X seconds each, and Y seconds elapse between the end of one such stack operation and the start of the next operation. For $m \geq 1$, define the stack life of m is the time elapsed from the 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

n :

3 | Push or Pop : X

2 | Bw 2op : Y

1

Push + Stack life − Pop

One element : X + Y − X

Two element : 2X + 2Y − 2X = 2(X+Y) − X

n element : nX + nY − nX = n(X+Y) − X

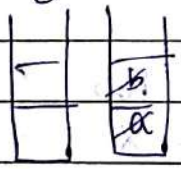
Gate 2006

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

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

Let n insert & m (≤ n) delete operations be performed in arbitrary order on an empty queue Q. Let x & y be the number of push and pop operations performed respectively in the process. Which one is true for all m & n?

Sol



N elements

2 push, 2 pop, 2 push, 1 pop, 1 pop

(x) Push: Best case: $2m + (n-m) = m+n$ // all ele other & insert new

Worst case: $2n$ // Insertion of first ele

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

Worst case = $2n + (m-n)$ // push of ele to er stack & more m ents

$= m+n$

$\therefore m+n \le x \le 2n$

$2m \le y \le 2n+n$

Sol:

(1)

(2)

(3)

(4)

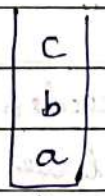
SURYA Gold

Date _____ Page _____

GATE-2000 ↔ GATE 2014

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

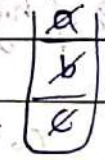
Sol:-



a b c

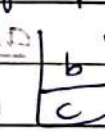
↑ ↑ ↑

Reverse

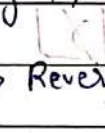


a b c

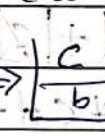
After popping a, if we want to insert then reverse



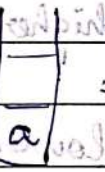
⇒ Reverse ⇒



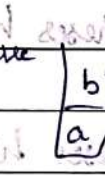
Insert ⇒



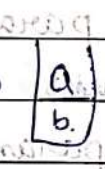
Enqueue → Reverse Stack → 3 operation



Reverse ⇒

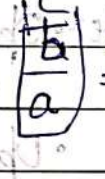


⇒



Reverse & Push ⇒

Push ⇒



Reverse ⇒

Dequeue → 1 operation

↑ bc

Scanned by CamScanner

Applications of Stacks

SURYA Gold
Date _____ Page _____

1) Conversion of infix expression to postfix expression

Eg:-

① a+b*c → a+bc* → abc*+

② a+b-c → ab+-c → ab+c-

③ a+(b-c) → a+bc- → abc-+

④ a*b+c → ab*+c → ab*c+

Using Stack

① a + b * c

Stack:- | + | * |

a b c

Note:-
An operator of high precedence can sit on operator of lower precedence
But an operator of lower precedence can not sit on operator of higher precedence

② a * b + c

| * | + |

a b * c +

③ a * b + (c - d)

| * | + | (| - |

a b * c d - +

a * b + (c - d)
a * b + cd -
ab * + cd -
ab * cd - +
+ < (
(< +

Note:- • Operators within parantheses has higher precedence than paranitheses
• Operators outside paranitheses has lower precedence than paranthese

Space complexity = O(n)

Algorithm

(1) Create a stack
(2) for each character 't' in the input
{
if ('t' is an operand)
output 't';
else if ('t' is a right paranthesis)
{
pop and output tokens until a left paranthesis is popped (but don't output)
}

else // t is an operator or left paranthesis
{
pop and output tokens until one of lower priority than 't' is encountered or a left paranthesis is encountered or stack is empty.
Push t
}

Time complexity :- O(n)
Uses operator stack

2) Postfix evaluation algorithm
Time complexity :- O(n), Uses operand stack

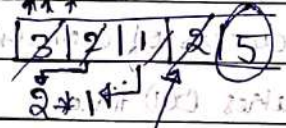
Eg: ① 3 * 2 + 1 ② 3 + 2 * 1

6 + 1 5 * 1

7 5

Postfix :- 3 + 2 1 *

3 2 1 * +



// Push result to stack

3 + 2

5

Note :-
First operand becomes second operand and vice versa

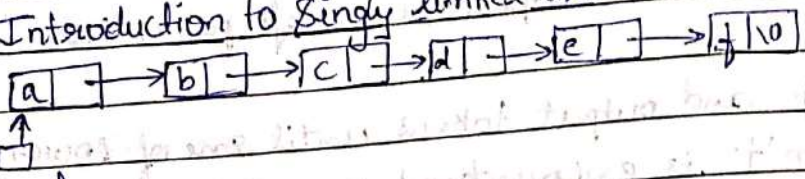
Space complexity :- O(n)

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

Linked List

SURYA Gold
Date _____ Page _____

Introduction to Singly linked list




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

Q. (1)

```
struct node *p;  struct node *head;
p = head->link->link;
p->link->link->link = p;
head->link = p->link;
printf("%c", head->link->link->link->link->data);
```

O/p:-

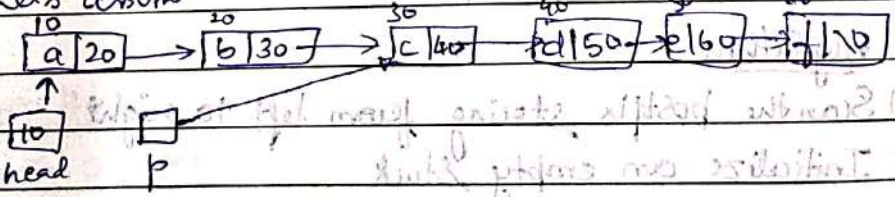
Advantage:-

1> Sequential access of element. Takes O(n) time
In arrays, it takes O(1) time

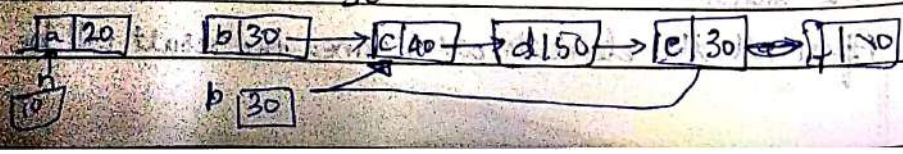
2> Searching takes O(n) time if elements are unordered

Ans for Q

Let's assume

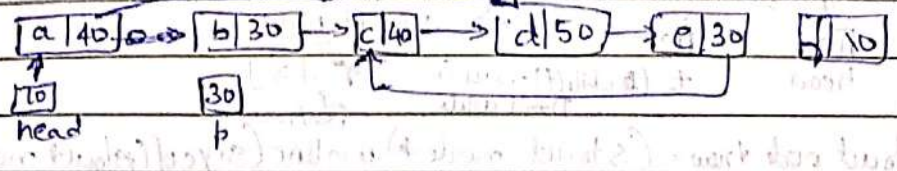



```
p = head->link->link
  = 20->link
p = 30
p->link->link->link = p
30->link->link
40->link->link = 50->link
50 = p
```



head->link = p->link;

10 = 40



print head->link->link->link->link->data

40 -> link -> link -> link -> data

50 -> link -> link -> data

30 -> link -> data

40 -> data

d

Traversing a singly linked list

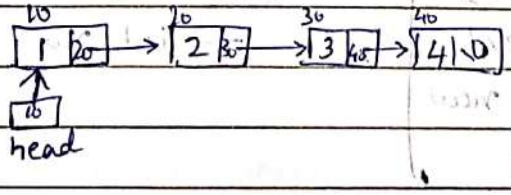
On a list, we can perform

i) Traversing

ii) Insertion

iii) Deletion

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



Moving head may cause loss of nodes created dynamically (since it does not have any name)

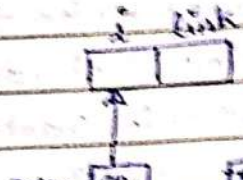
```
struct node *t;
t = head
while(t != NULL) {
printf("%d", t->data);  or  printf("%d", t->i);
t = t->link;
}
```

~~printf("%d", t->link);~~

while(t != NULL) ≡ while(t)

Inserting an element in SLL

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



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

At beginning :- new → link = head

head = new

At end :- struct node *t; t = head

while (t != NULL) // Don't write

t = t → next

while (t → next != NULL) t → temporary variable

t = t → next

t → next = new

new → next = NULL

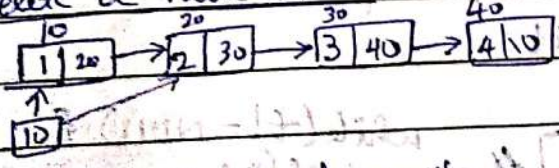
After node :- which we want

Eg : After 2nd node

struct node *t = head

while (t → i != 2)	X	new → link = t → link
t = t → next		t → link = new
t → next = new		
new → next		

Delete a node in SLL



At beginning :-

struct node *t = head; head = head → next

free (t)

if (head == NULL) return;

if (head → next == NULL)

free (head)

head = NULL;

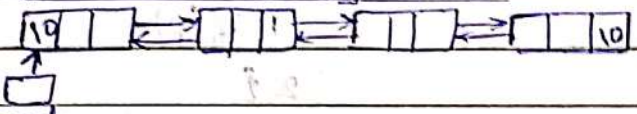
From: tail

```
struct node *t = head
while (t->next->next != NULL)
    t = t->next
free (t->next)
t->next = NULL
```

Applicable for: Eg: 3

```
struct node *t = head
while (t->next->i != 3)
    t = t->next;
struct node *t1 = t->next
t->next = t->next->next
free (t1)
```

Insertion into doubly linked list




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

At Beginning:

```
t->next = head
head = t
t->previous = NULL
head->next->previous = head;
```

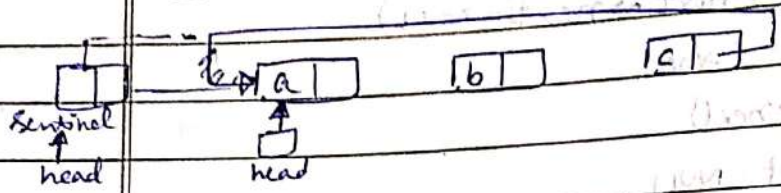
At end.

```
while((p->next) != NULL)
    p = p->next
p->next = t
t->prev = p
t->next = NULL;
```

After Node:- Eg:

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

Circular linked list




```
p = head
while (p->next != head)
```

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

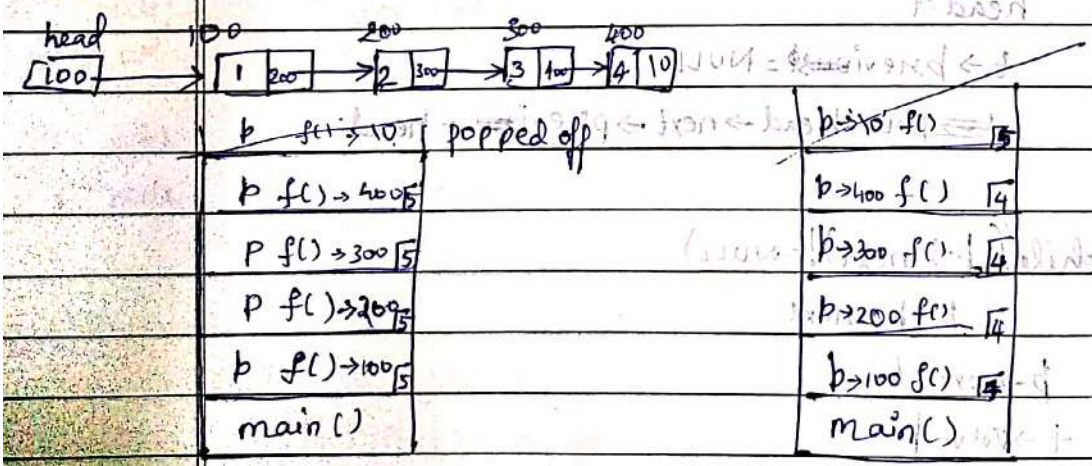
Printing the elements of SLL using recursion

Program 1:-

```
void f(struct node *p)
{
1 if(p)
2 {
3 printf("%d", p->data);
4 f(p->link);
5 }
}
```

Program 2:-

```
void f(struct node *p)
{
1 if(p)
2 {
3 f(p->link);
4 printf("%d", p->data);
5 }
}
```



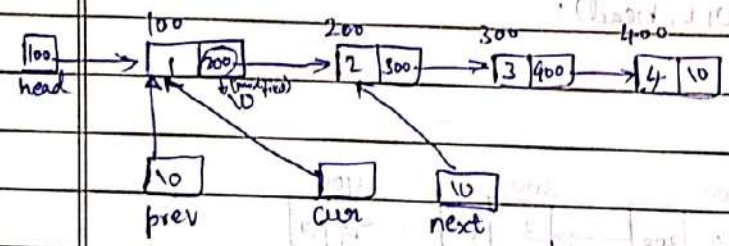
O/P:- 1 2 3 4

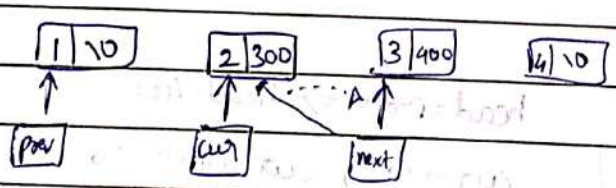
O/P:- 4 3 2 1

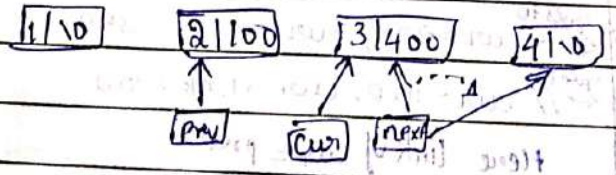
Reversing an SLL using iterative program

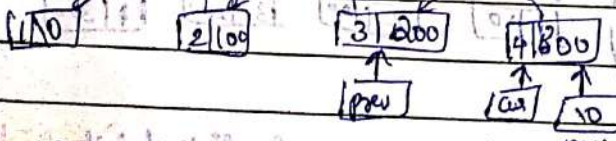
```
struct node{
```

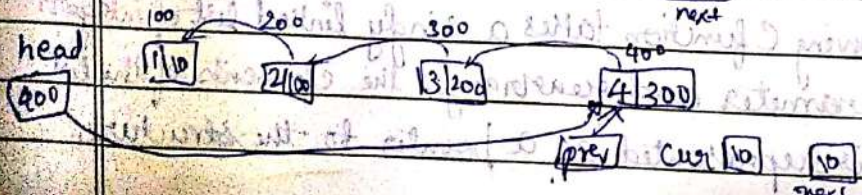
```
int i;
struct node *next;
};
struct node *reverse(struct node *cur)
{
    struct node *prev = NULL, *nextNode = NULL
    while (cur){
        nextNode = cur->next;
        cur->next = prev;
        prev = cur;
        cur = nextNode;
    }
    return prev;
}
```







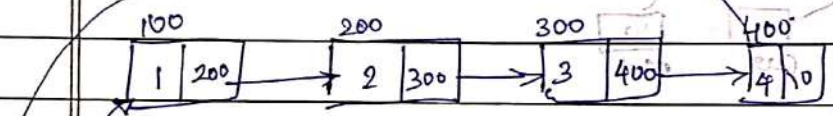




Date Page

Recursive program for reversing an SLL

```
Struct node *head;
void reverse(struct node *prev, struct node * cur)
{
1 if(cur)
  {
2 reverse(cur, cur->link);
3 cur->link = prev;
4 }
  else
      head = prev;
}
void main(){
      :
  reverse( NULL, head);
      :
}
```

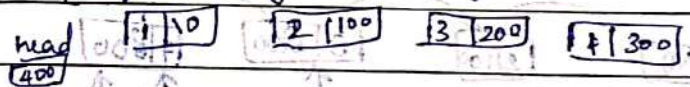


head
100

*head
400

head = prev ⇒ head = 400

rev (400 prev, 10 cur)	cur → 400, cur → link = 10
rev (300 prev, 400 cur, 3)	cur = 300 cur → link = 400
rev (200 prev, 300 cur, 3)	goes to // cur = 200, cur → link = 300.
rev (100 cur, 200 prev, 3)	goes to // cur = 100, cur → link = 200
rev(0 prev, 100 cur, 3)	Here link of cur = prev;
main()	After popping. LL changes to



Gate 2008

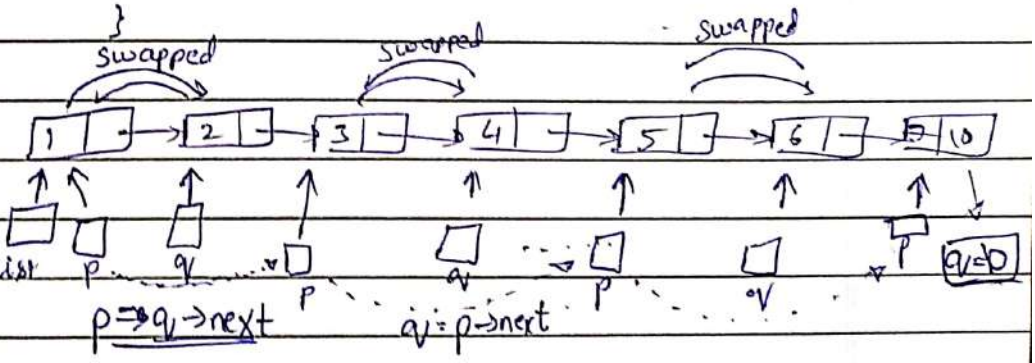
The following C function takes a singly linked list of integers as a parameter and rearranges the elements of the list. The list is represented as a pointer to the structure.

SURYA Gold

Date _____ Page _____

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

```
struct node {
    int value;
    struct node *next;
};
void rearrange (struct node *list)
{
  struct node *p, *q;
  int temp;
  if(!list || !list->next) return;
  p = list;
  q = list->next;
  while(q)
  {
    temp = p->val;
    p->val = q->val;
    q->val = temp;
    p = q->next;
    q = p ? p->next : 0;
  }
}
```



o/p:- 2143657

Gate 2010

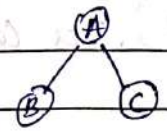
The follow

Trees

SURYA Gold
Date _____ Page _____

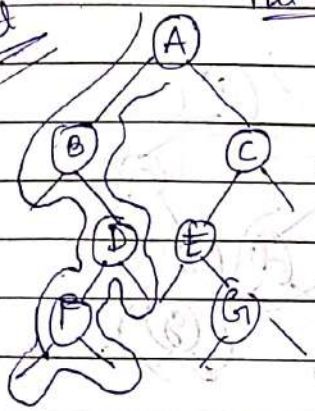
Introduction to tree traversals

1) Inorder traversal – Left Root Right – BAC
2) Preorder traversal – Root Left Right – ABC
3) Postorder traversal – Left Right Root – BCA



Put dummy nodes (links)

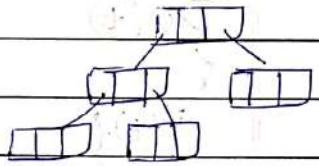
Shortcut



Pre – ABDFCEG – I^st visit
In – BFDAEGC – II^nd
Post – FDBGECA – III^rd

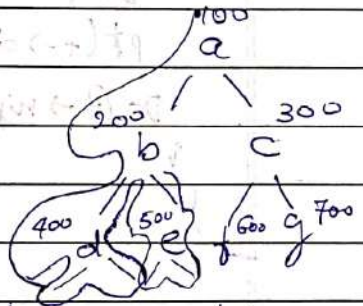
Implementation of traversals and time and space analysis

```
struct node
{
  int data;
  struct node *left, *right;
};
```



```
void Inorder(struct node *t)
{
  if(t!=NULL) {          (or)  if(t)
    Inorder(t->left);
    printf("%d", t->data);
    Inorder(t->right);
  }
}
```

Pre	Post	In
1	2	1 Inorder(t->left);
3	1	2 printf("%d", t->data);
2	3	3 Inorder(t->right);



t	t
~~t=500~~ '10	t='10
t='10	t='10
d ~~t=400~~ 2	t=100 2
~~t=200~~ 2	t='10'
Inorder t=100	t=~~'10~~'
main()	t=500 2

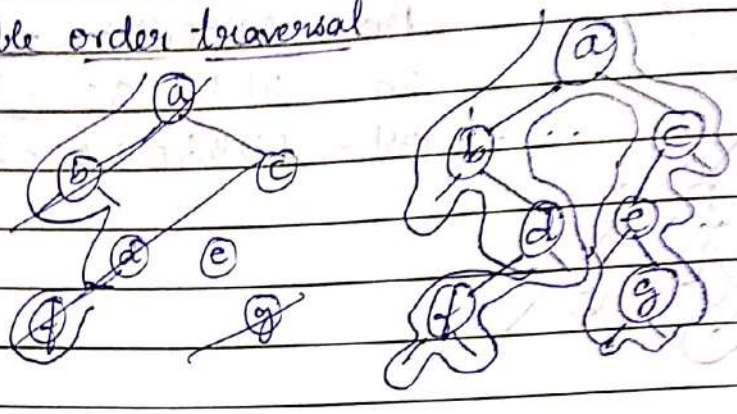
points: node during second visit

O/p: ~~400 200 500~~

d b e a b.

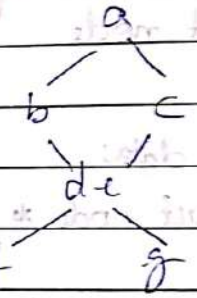
Visiting node takes constant time
Every node visited 3 times
Time complexity :- O(n)
Space complexity :- Depends on levels of tree
n nodes → n levels
All nodes in one stream (all left). Its called
Queue tree O(n)

Double order traversal



b f d a c e g c

```
DO(t)
{
 if(t){
   pf("%d", t->data);
   DO (t->left)
   pf(t->data)
   DO (t->right)
 }
}
```

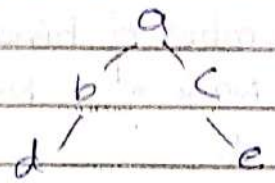


a b d f f d a c e g g e c

Triple order Traversal

```
TO (t)
{
 if(t){
```

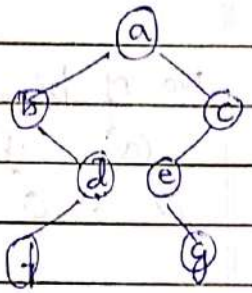
```
{
  pf("%d", t->data)
  TO(t->LC);
  Pf("%d", t->data)
  TO(t->RC)
  Pf("%d", t->data).
}
```



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

Indirect recursion on trees

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



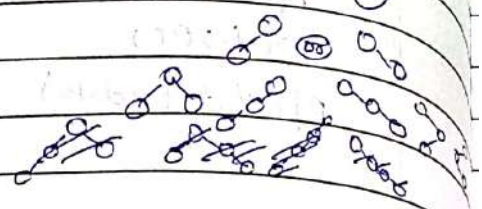

```
void B(struct node *t)
{
  if(t)
  {
  1. ~~printf~~ A(t->left);
  2. printf("%d");
  3. A(t->right);
  }
}
```

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

SURYA Gold
Date Page

Number of binary trees possible

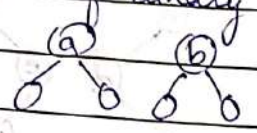
No. of Node	No. of binary trees possible
1	1
2	2
3	3
4	14



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

3 nodes $= \frac{2nC_n}{n+1} = \frac{2\times3C_3}{4} = \frac{6C_3}{4} = 5$

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



$= 2nC_n \quad 6C_3 = \frac{6!}{3!3!}$

$(n+1)n! \quad 4\times3\times2\times1 \quad 4\times3\times2\times1$

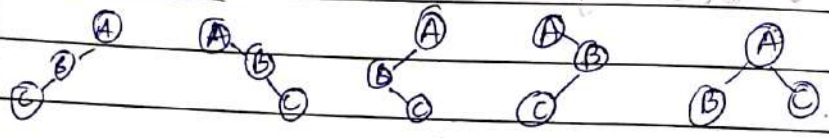
$= \frac{6\times5\times4}{3\times2\times1} \times \frac{1}{4\times3\times2\times1}$

(00) m^{n-2}

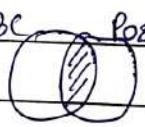
$3^{3-2} \cdot 3$

4

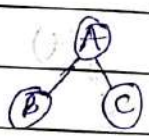
Preorder ABC - 5 trees



Preorder ABC & Postorder CBA



Inorder - B+C & Preorder - A



Inorder BCA & Postorder CBA

n nodes, Preorder → $\frac{2nC_n}{n+1}$

Binary trees

Preorder & Postorder - No unique BT or more than 1

Inorder, Postorder, Preorder - Unique BT & exactly 1

Construction of unique binary tree

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

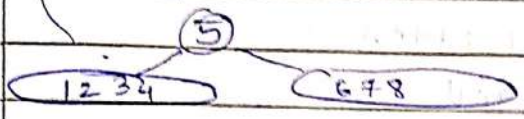
(1)

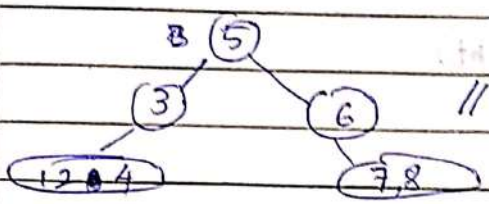
SURYA Gold
Date Page

Preorder :- 5, 3, 1, 2, 4, 6, 8, 7

Postorder = ?

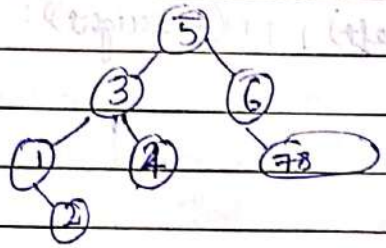
Inorder :-

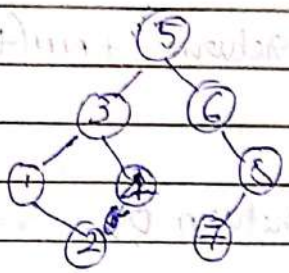




// first is Root from preorder

// left & right subtree from inorder

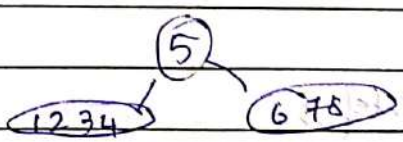


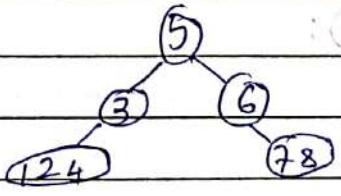


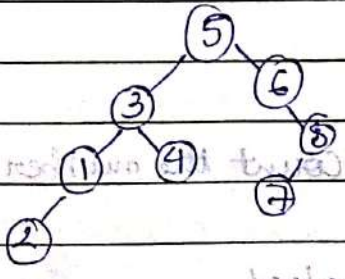
Postorder :- 2 1 4 3 7 8 6 5 // 3rd time

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

Postorder: 2 1 4 3 7 8 6 5



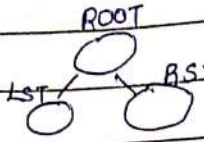




Date Page

Imp question

Recursive program to count the number of nodes



NN(T) = 1 + NN(LST) + NN(RST)

= 0; if T is null

```
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;
    l = NN(t->left);
    r = NN(t->right);
    return (1+l+r);
  }
  else
    return 0;
}
```

Imp question

Recursive program to count the number of leaves and non leaves

NL(T) = 1 ; Tree is a leaf

= NL(LST) + NL(RST

```
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));
}
```

Recursive program to find the full nodes

Node having 2 children - Full node.

Full Node is a subset of Non leaf

Not every non leaf is a full node

FN(T) = 0 , T = NULL

0 , T = leaf

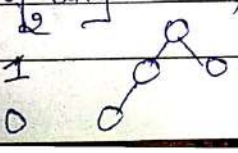
= FN(T->LST) + FN(T->RST) , T has 1 child

= FN(T->LST) + FN(T->RST) + 1

```
int FN(struct node *t)
{
  if (!t) return 0;
  if(!t->left && !t->right)
    return 0;
  return FN(t->left + t->right) + (t->left && t->right
                                   1 : 0);
}
```

Recursive program to find the height of a tree

Height of any node, to leaf its distance



$H(T) = 0$ if T is Null
0 if T is a leaf
$1 + \max(H(LST), H(RST))$

```
int H(struct node *t)
{
  if(!t) return 0;
  if(!t->right && !t->left) return 0;
  return(1+ H(t->left) > H(t->right))
  int l, r;
  l= H(t->left)
  r= H(t->right)
  //return (1 + max(l,r));
  return (1+ (l>r)?l:r;);
```

GATEFORUM

Binary Trees

Properties:-

A binary tree T has n leaf nodes. The no. of nodes

(1) A tree with n nodes has exactly n-1 edges

(2) In a tree, every node except root has exactly one parent

(3) Maximum no. of nodes in a binary tree of height k:
$2^{k+1} - 1, \quad k \geq 0$

(4) Consider a nonempty binary tree with n_i: nodes in number of nodes with i children $(i=0,1,2)$
then, $\boxed{n_2 = n_0 - 1}$

(5) Maximum no. of nodes in a binary tree at level i is
$2^i, \quad i \geq 0$

(6) Maximum depth of a binary tree with n nodes is
n

(7) The Minimum height of a binary tree with n nodes is
$\log_2 n$

(8) Maximum no. of leaves in a binary tree of height h / level l
2^h or 2^l

SURYA Gold
Date _____ Page _____

GATE 2007

(1) How many distinct binary search trees can be created out of 4 distinct keys

$n = 4$

$B_n = \frac{1}{n+1}\binom{2n}{n} = \frac{2nC_n}{n+1}$

$= \frac{1}{4+1} \times \frac{8!}{4!4!}$

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

$= 14$

3

$\frac{1}{4} \cdot \frac{6!}{3!3!}$

$\frac{1}{4} \times \frac{6 \times 5 \times 4}{3 \times 2 \times 1} = 5$

GATEFORUM

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

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

(12) A complete binary tree with n non leaf nodes contain $2n+1$ nodes

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

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

(5B) The average total path length of a randomly built binary search tree with n nodes is $O(\log_2 n)$

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

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

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

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

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

In L - Root - R
Pre Root Left Right
Post Left Right Root

(21) Total number of external nodes in a binary tree are internal nodes + 1

$e = i + 1$

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

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

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

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

(26) In heap property,

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

LEFT(i) $\rightarrow 2i$

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

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

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

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

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

Sol:- 32

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

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

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

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

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

(7) $|V|-1$ iterations of relaxing the entire set of edges are performed in Bellman-Ford algorithm

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

(9) $2n-1$ comparisons are required for merging 2 sorted lists of n element

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

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

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

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

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

(15) Let H be a finite collection of hash functions that map a universe U of keys into {0,1,..., m-1}. H is said to be universal if for each pair of distinct keys k, l ∈ U the no. of hash functions h∈H for which h(k) = h(l) is atmost H/m

Note:- In Preorder, Root is first element
In Postorder, Root is last element

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

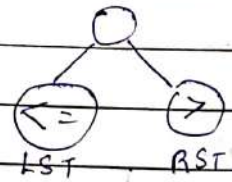
Binary Search Trees

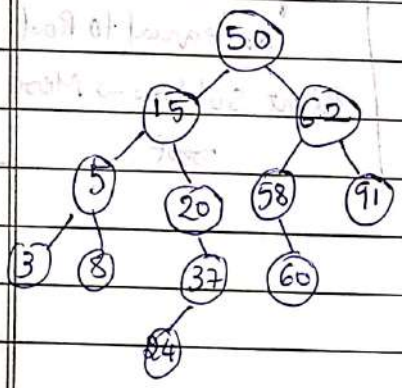
It is a special kind of binary which are helpful in searching of elements

GATE 96

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

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





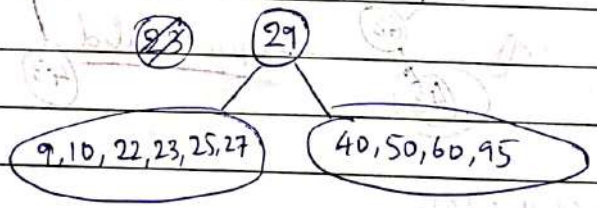
GATE 2005

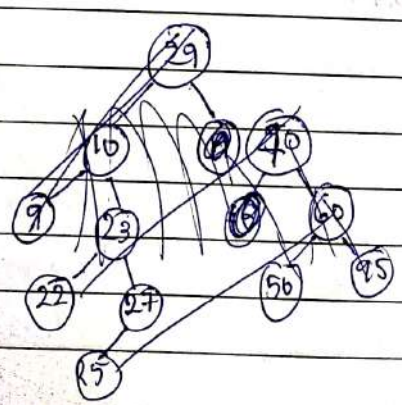
Postorder: 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

Inorder:- Sorted order → 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

Preorder:-

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





GATE 2006 → 55 search

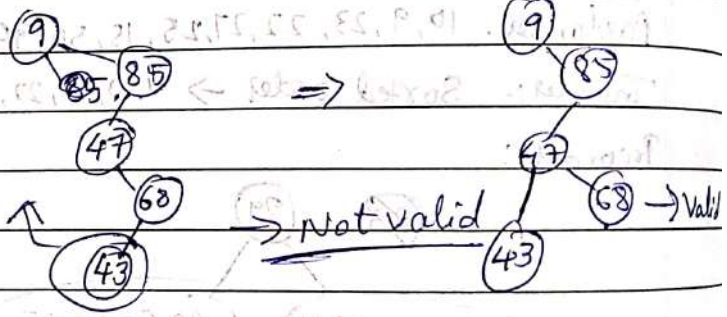
a) 10, 75, 64, 43, 60, 57, 55

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

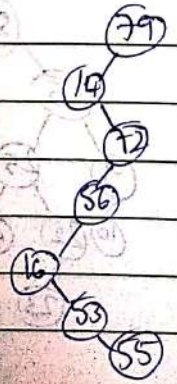
Left Subtree → Less than or equal to Root

Right Subtree → More than root

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



d) 79, 14, 72, 56, 16, 53, 55

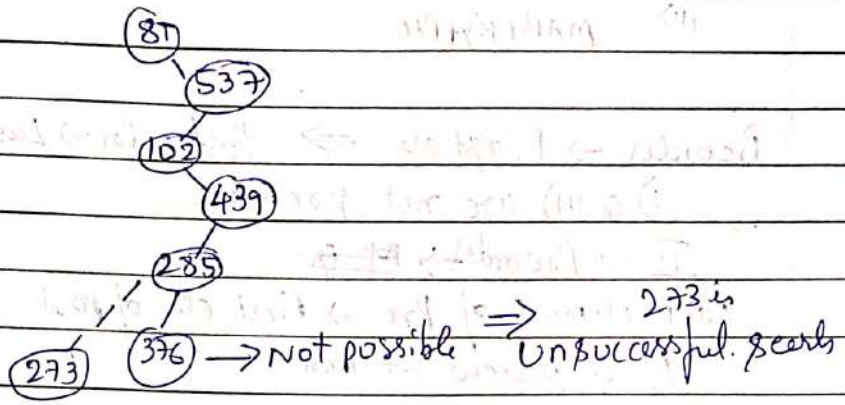


SURYA Gold
Date Page

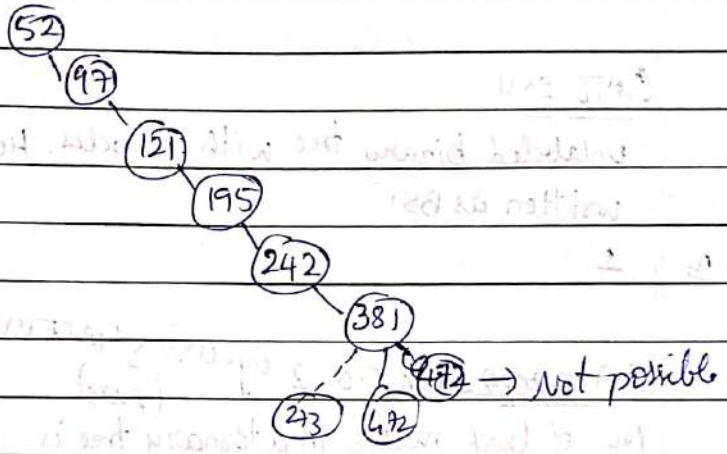
GATE 2008

273 is unsuccessful search

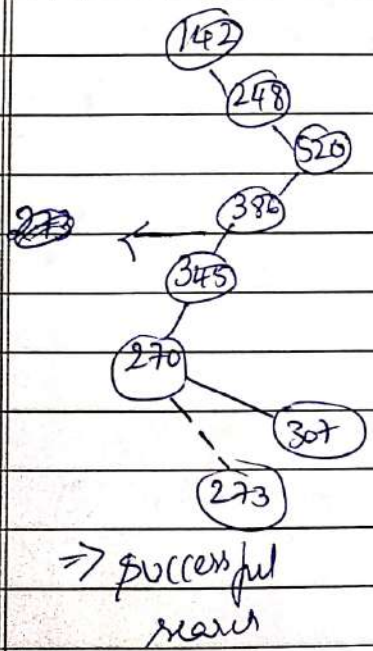
a) 81, 537, 102, 439, 285, 376, 305



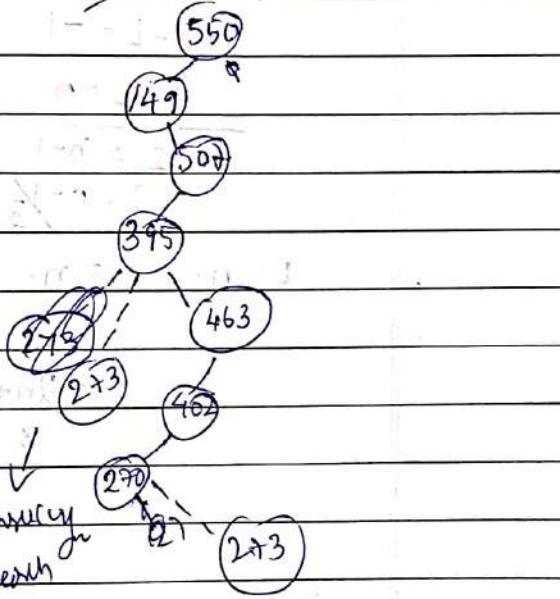
b) 52, 97, 121, 195, 242, 381, 472



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



d) 550, 149, 507, 394, 463, 402, 270



GATE - 2008

(1) i) MBCAFHPYK

ii) KAMBYPFH

iii) MABCKYFPH

Preorder → First ele ⇒ Post order → Last ele

i) & iii) are not preorder

II → Preorder ⇒ ~~At last~~.

Last element of Pre → First ele of Post

K is present at last

I → Postorder

III → Inorder

GATE 2011

Unlabeled binary tree with n nodes. How it can written as BST.

Ans 1

GATE ~~2002~~ ~~1998~~ 2002 by using GATE 1998 formula (pure) IT 2005

No. of leaf nodes in a ternary tree is

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

$$L + I = n$$

$$2I - L = -1$$

$$I + L = n$$

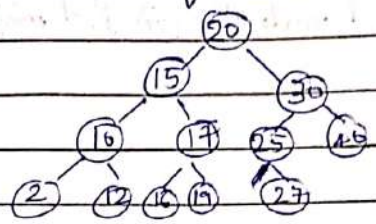
$$3I = n-1$$

$$I = n-1/3$$

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

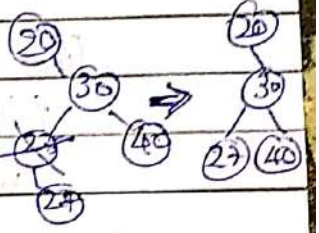
$$= \frac{2n+1}{3}$$

Delete a Node from BST



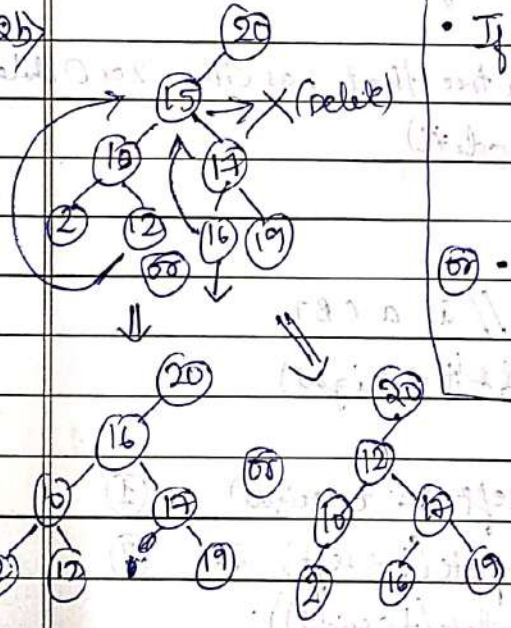
3 types of deletion
1. Leaf
2. Non Leaf [a) one child
   b) two children

2 a)



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

2b)

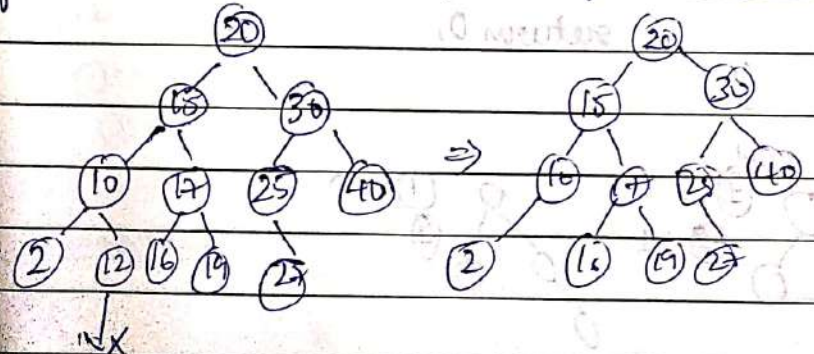


- If a node has 2 children, go to right subtree, take least element & replace it ↓ inorder successor

(or)

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

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



Finding maximum & minimum in a BST

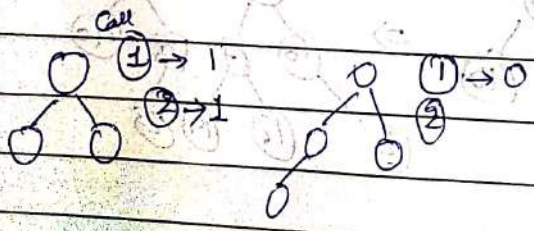
```
find_inorder_pre(struct node *t)   (or) find min(struct node *t)
{
  while (t -> left)
    t = t -> left
}

find max (struct node *t)
{
  while (t -> right)
    t = t -> right
}
```

Recursive program on testing whether a tree is complete binary tree

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

```
isComplete (struct node *t)
{
  if (t == NULL)
    return 1; // is a CBT.
  if (!t -> left && t -> right)
    return 1;
  else if (t -> left && t -> right)        (1)
    return (isComplete (t -> left) &&      (2)
            isComplete (t -> right));
  else
    return 0;
}
```



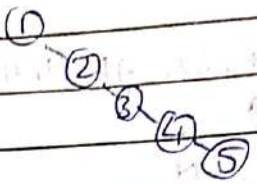
SURYA Gold
Date Page

AVL trees

Linked List Search time O(n)

Tree reduces time But some cases like

1 2 3 4 5



→ Here time – O(n)

Height – O(n)

So comes concept of Balancing

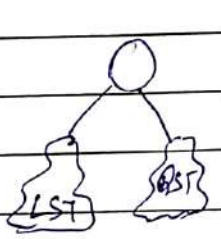
AVL – Balanced tree

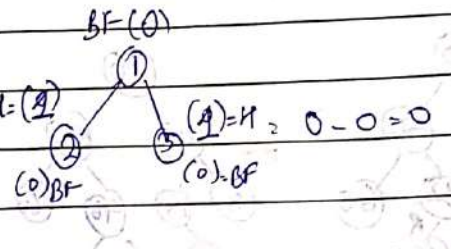
It is a balanced BST

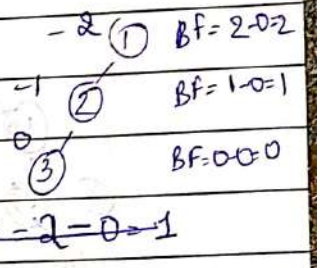
Nodes – n, Height – O(log n)

∴ Search time – O(log n)

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



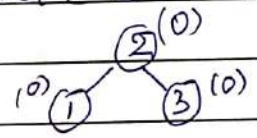




~~BF = 2 = 0 = 1~~

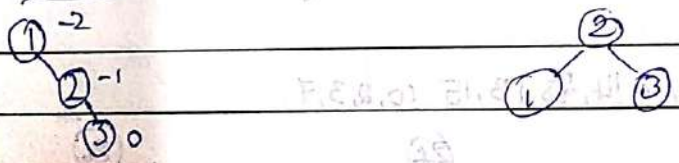
It is called LL imbalance

Whenever L-L imbalance, rotate clockwise



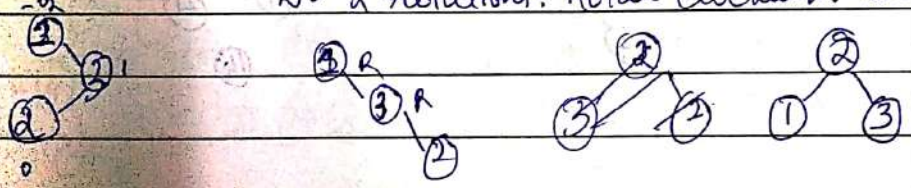
BF = 0

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

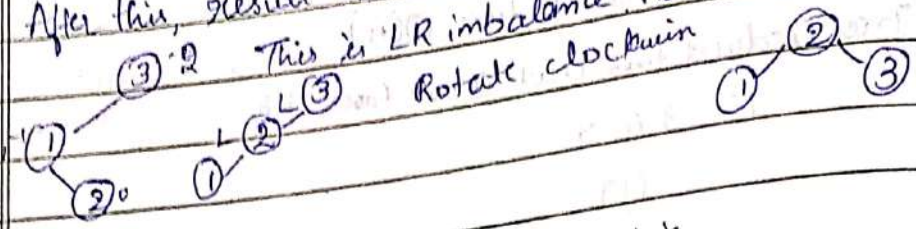


R-L imbalance

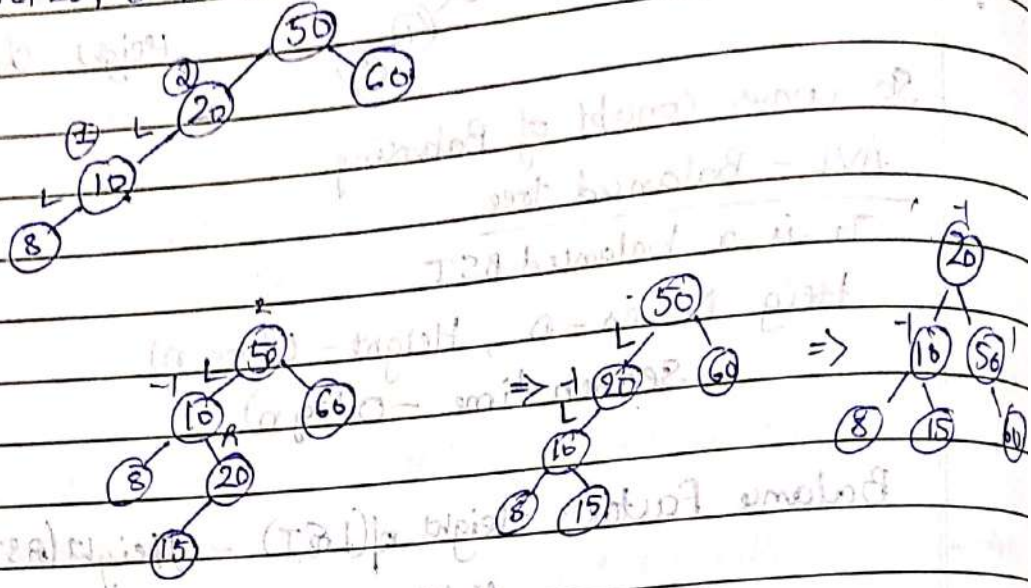
Do 2 rotations. Rotate clockwise. Rote anticlockwise

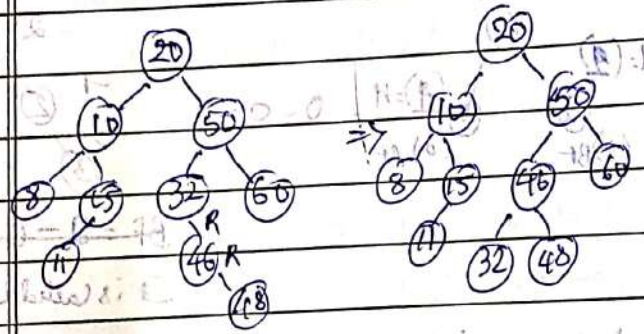


After this, result obtained is Binary Search Tree.



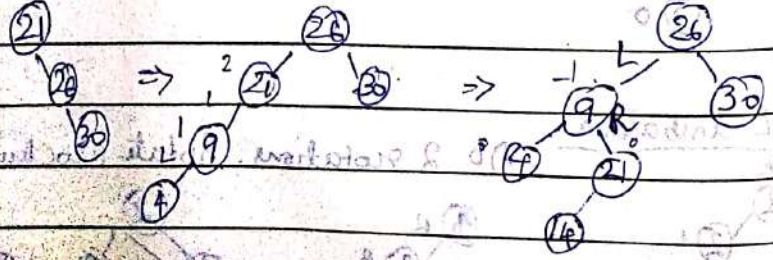
(1) 50, 20, 60, 10, 8, 15, 32, 46, 11, 48

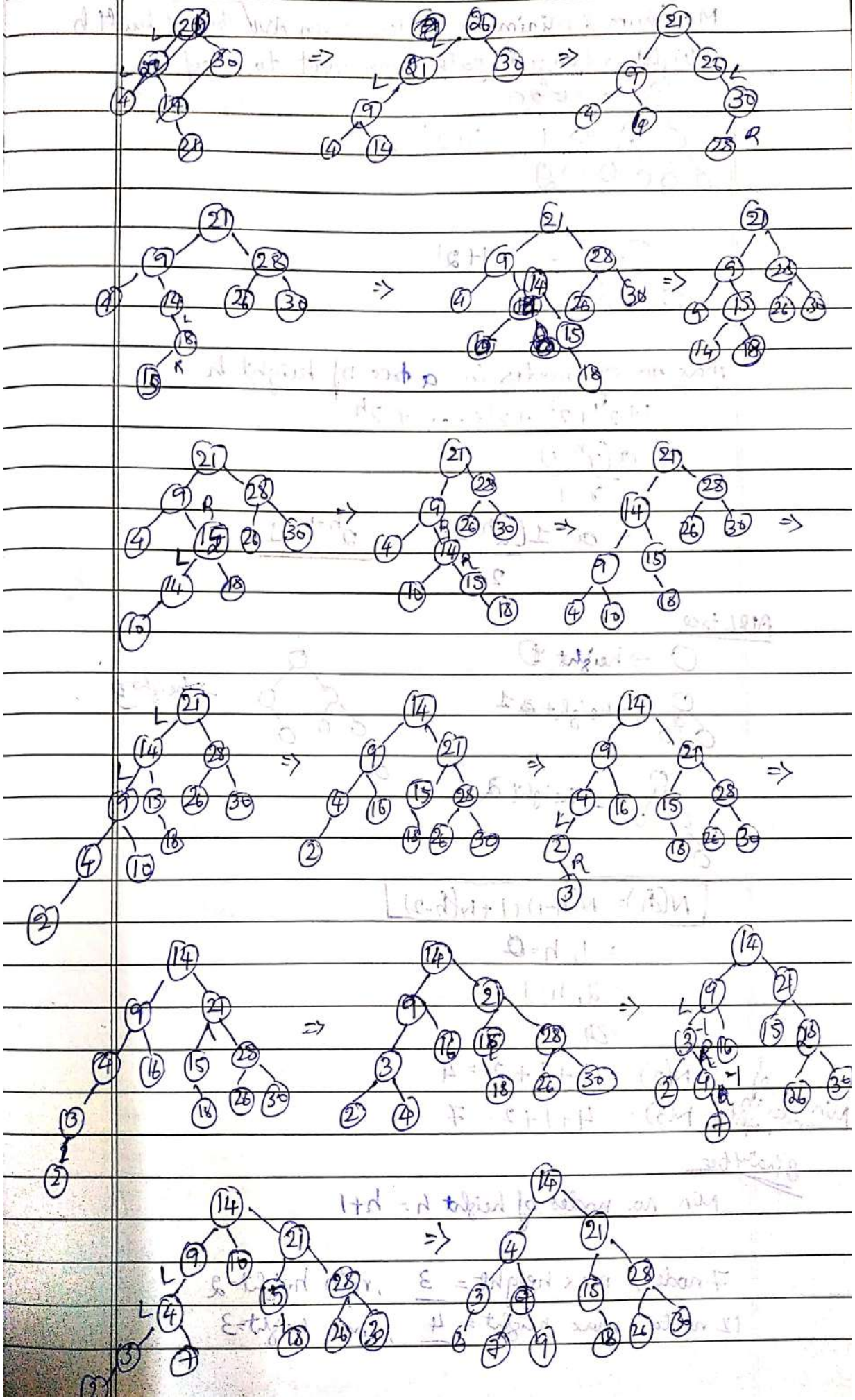




Operations	BST	BBST
Search	O(n)	O(log n)
Height	O(n)	O(log n)
Insertion	O(n)	O(log n) + O(log n) + constant

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

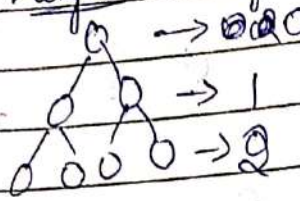




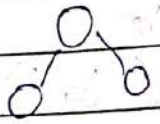
Binary

Maximum & Minimum nodes in an AVL tree of height h

Height → Longest path from root to a leaf



$1+2^1+2^2$



$= 1+2^1$

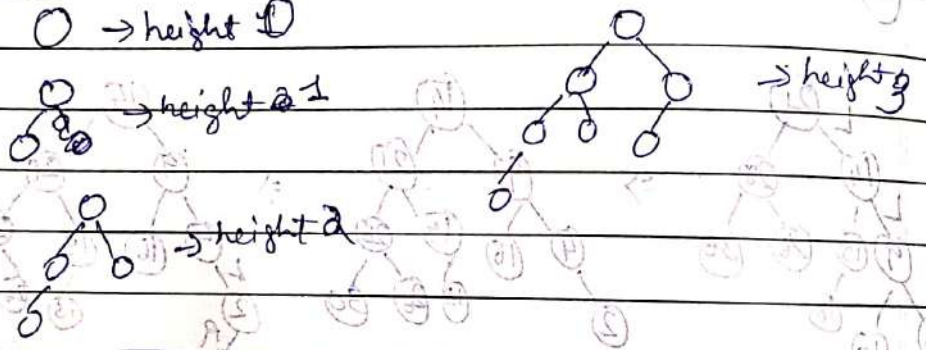
max no. of nodes in a tree of height h

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

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

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

AVL tree



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

$= 1, h=0$

$= 2, h=1$

$= 4$

Min no. of nodes in height

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

$N(3) = 4+1+2 = 7$

Binary tree

Min no. nodes of height h = h+1

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

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

SURYA Gold

Date _____ Page _____

GATE 2006

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

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

$N(1) = 2$

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

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

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

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

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

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

$N(n) = N(n-k) + k$ where $n - k = 1$

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

$N(1) = 2 + 0$

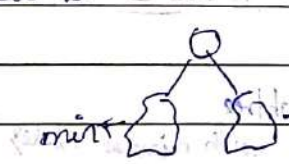
Ref: Degree 2 n = Degree 0 $n+1$

$10 = 10 + 1$

$\therefore$ 11

GATE 2005

In a binary tree, for every node the difference between the number of nodes in the left and right subtree is atmost 2. If the height of tree is $h > 0$, then minimum number of nodes in the trees is



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

$h = 1, n = 3$, Substitute for

a) $2^{h-1} = 2^{1-1} = 2^0 = 1$ ✗

b) $2^{h-1} + 1 = 2^0 + 1 = 2$ ✓, $2^1 + 1 = 3$ ✗

c) $2^h - 1 = 2^1 - 1 = 1$

d) $2^h = 2^1 = 2$ ✓, $2^2 = 4$ ✓

$h = 2$.

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

$= 2 \times 2 - 1 = 3$

Hence 2^h

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

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

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

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

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

$= 2^k N(n-k) 2^{k-2} - 2 - 2^{k-1} = 2^3 N(n-3) - 2^2 - 1$

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

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

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

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

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

$= 2^2(2N(h-3) - 1) - 2 - 1$

$= 2^3 N(h-3) - 2^2 - 2 - 1$

$N(h) = 2^k$:

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

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

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

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

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

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

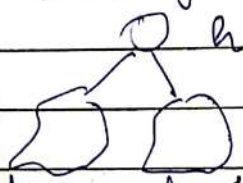
$= 2^{h-1} + 1$

GATE 1997 and GATE 2005

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

(max dist)

Sol:- A binary tree of height h has 2^h nodes



Any one subtree with $n-1$ nodes

$N(h) = N(h-1) + 1 + (N(h-1) - 1)$ // no. of nodes decreased

AVL tree $N(h) = N(h-1) + 1 + N(h-2)$ // no. of height decreased

$N(h) = 2N(h-1)$ — (1)

$N(h-1) = 2N(h-2)$ — (2)

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

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

SURYA Gold

Date _____ Page _____

$N(h) = 2^k N(h-k)$ where $h-k=1$

$k = h-1$

maximum height with 4 nodes

$4 = 2^h$

$\boxed{h=2}$

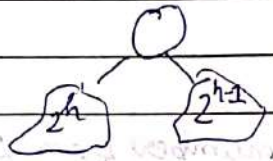
max height with 8 nodes

$8 = 2^h$

$\boxed{h=3}$

$N(h) = 2^h$

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



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

$= 2 \cdot 2^h$

$= 2^{h+1}$

Gate 2014

(1) Suppose we have a balanced binary search tree 'T' holding n numbers. We are given two numbers 'L' and 'H' and wish to sum up all the numbers in 'T' that lie between L and H. Suppose there are m such numbers in T. If the tightest upper bound on time to compute the sum is $O(n^a \log^b n + m^c \log^d n)$, the value of $a+10b+100c+1000d$

Sol

L H

100 200

100-199 → ~~$m(\log n)$~~ $O(m+\log n)$

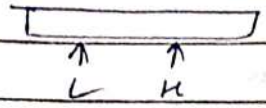
$c=1, d=0, a=0, b=0$

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

$= 100$

Other method

Inorder O(n)



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

Time (traversal algorithm given below)

Total (m+logn)

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

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

$= 110$

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

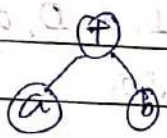
(1) Find L using binary search and keep all nodes encountered in search using stack

(2) After finding L add it to stack as well and initialize sum=0

(3) Now, for all nodes in stack do an inorder traversal starting from their right node & adding the node value to sum. If H is found, stop the algorithm

Expression trees

a+b



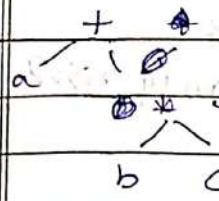
Pre +ab

In a+b

Post ab+

SURYA Gold
Date Page

a + b * c

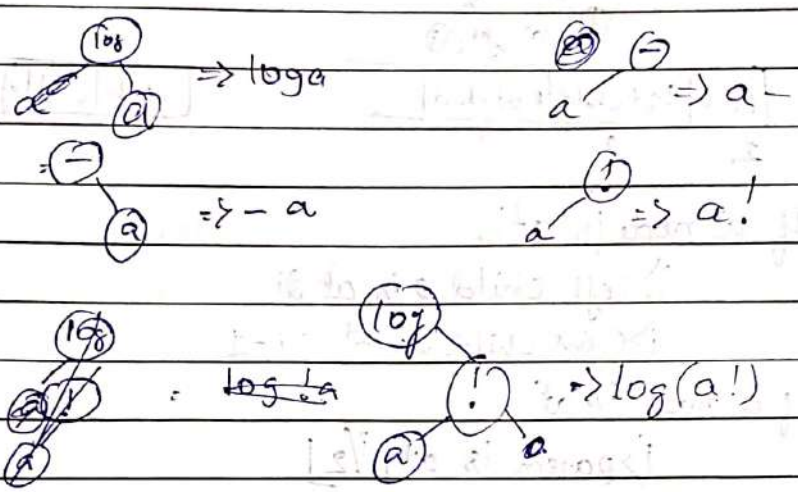


→ high precedence

Pre + a * bc

In a + b * c

Post ~~a b + c *~~ a b c * +



GATE 2005

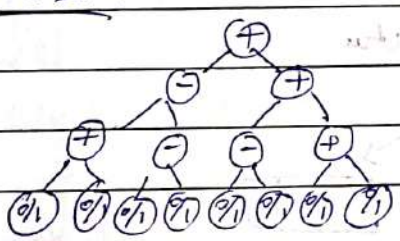
The numbers 1, 2 ... n are inserted into a BST in some order n. In the resulting tree, the right subtree contains p nodes. The first number to be inserted in the tree must be

BST - 1 element // for BBST & BST

LST - n-p+1 = n-p-1

Root - n-p

GATE 2014 max value to be found



⇒ ((a+b) - (c-d)) + ((e-f) + (g+h))

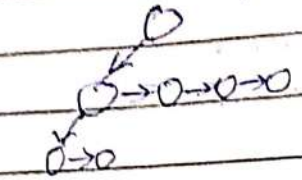
= (1+1) - (0-1) + (1-0) + (1+1)

= 2+1+1+2 = 6

SURYA Gold
Date
Page

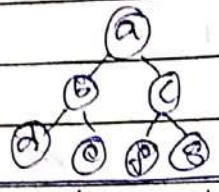
Various representations of a tree

(1) LCRS → Left Child Right Sibling



n array n>>2

(2) Array representation → Mainly for heaps



Root	Left Child	Right child
1	2	3

a	b	c	d	e		

If a node is at i,

i) Left child is at 2i

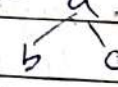
ii) Right child is at 2i+1

If a node is at i

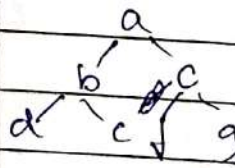
i) parent is at $\lfloor i/2 \rfloor$

n elements, $2^n - 1$ → array size.

(3) Nested representations

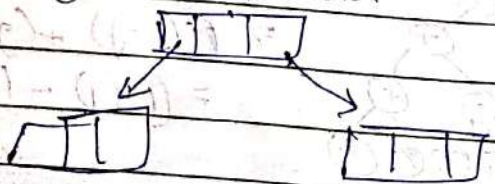


(a b a c) (a) (Root Left Right
LeRi (a b c)



a(b(e)(cfg)

(4) Normal (pointer & structure



SURYA Gold
Date _____ Page _____

GATE 2000

1) Consider the following nested representation of binary tree: (XYZ) indicates Y & Z are the left & right subtrees, respectively, of node X. Note that Y & Z may be NULL, or further nested. Which of the following represents a valid binary tree?

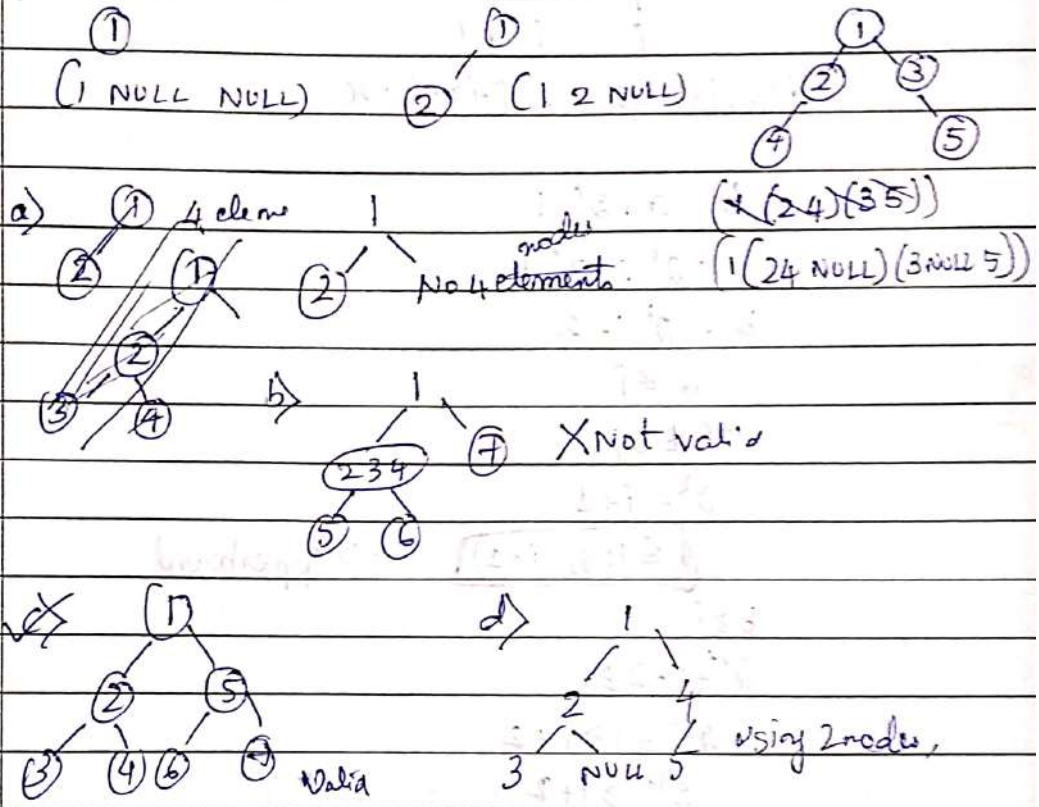
a) (1 2 (4 5 6 7))

b) (1 ((2 3 4) 5 6) 7)

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

d) (1 (2 3 NULL) (4 5))

Sol



GATE 2006

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.

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

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

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

Date Page

3) If the root node is at level 0, the level of element X[i], if 0 ≤ i ≤ n-1

Let i at level 'k'

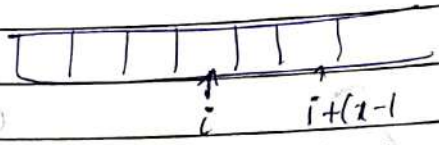
levels
1- o
2- o o
3- o o o o

| 0 | 1 | 2 | 3 | 4 | ≤

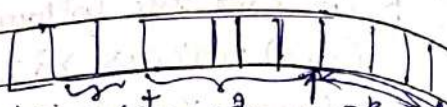
$2^0 + 2^1 + 2^2 + \dots 2^{k-1} = a$

$\frac{1(2^k - 1)}{2-1} = a$

$a = 2^k - 1$



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



$a \le i \le b$

$b = a + 2^k - 1$

$= 2^k - 1 + 2^k - 1$

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

$a \le i$

$2^k \le i$

$2^k = i+1$

$k \le \log_2(i+1)$ → Upperbound

$b \ge i$

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

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

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

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

$k \ge (\log_2(i+2) - 1)$ → Lowerbound

There can be only one integer b/w these bounds

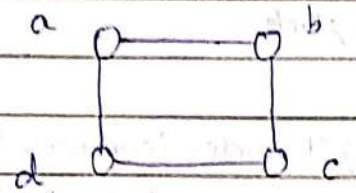
$\lfloor \log_2(i+1) \rfloor$ and $\lceil \log_2(i+2) - 1 \rceil$

Graphs

SURYA Gold
Date Page

1) World wide web is represented as a graph
2) Social Network
3) Person → Nodes, Friend → Edge b/w a person & his friend
   Friends → Nodes
4) Webpage → Node
   Links → Edge
Adjacency list in a graph → friend list in a facebook
Mutual friend suggestion - BFS

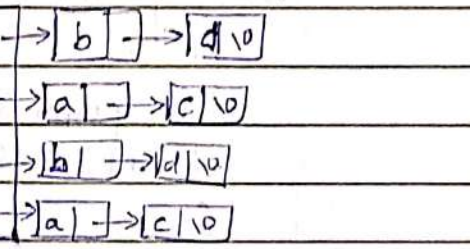
Representation of Graphs



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

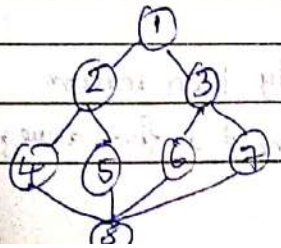
Adjacency List → useful for large/dense graphs



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

No. of elements = $2 \times E$ in undirected graph in Adjacency List
= E in directed graph
= $O(V+E)$ → Adjacency List
$O(V^2)$ → Adjacency matrix

Introduction to BFS and DFS



Visited → I found it
Explored → I found it & its adjacent vertices

Visited
0 → Not visited
1 → Visited

Both BFS & DFS uses array to keep track of visited nodes

DFS algorithm keeps track of unexplored nodes using stack

BFS → using queue

Space complexity → Atleast O(n)
BFS → O(n) → queue
DFS → O(n) → stack

Breadth first search visit nodes levelwise (horizontally)
Depth first search visit nodes depthwise (vertically)

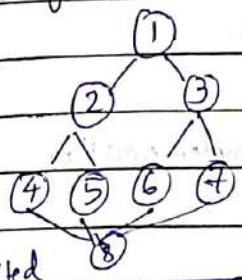
BFS algorithm

```
BFS(v)
// The graph G and array visited[] are global
visited[] is initialized to 0
{
  u = v;
  repeat
  {
    for all vertices w adjacent to u do
    {
      if(visited[w] == 0)
      {
        add w to queue;
        visited[w] = 1;
      }
    }
    If a queue is empty then return
    Delete the next element u from queue;
  }
}
```

SURYA Gold

Date Page

Tracing



Visited

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

→ O(n)

u = 1

w = {2, 3}

Queue: ~~2~~ | 3 |

(n-1) → O(n)

Visited

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

Visited

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

Queue: ~~2~~ | 3 |

u = 2

w = {1, 4, 5}

Queue: 3 | 4 | 5 |

Visited

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

Queue: ~~3~~ | 4 | 5 |

u = 3

w = {1, 6, 7}

Queue: 4 | 5 | 6 | 7 |

Visited

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

Queue: ~~4~~ | 5 | 6 | 7 | 8 |

u = 4

v = {2, 8}

Visited

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

Queue: ~~5~~ | 6 | 7 | 8 |

u = 5

v = {2, 8}

Queue: ~~6~~ | 7 | 8 |

u = 6

v = {3, 8}

Visited

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

Queue: ~~7~~ | 8 |

u = 7

w = {3, 8}

~~Return~~ Visited

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

Queue: ~~8~~ |

u = 8

w = {4, 5, 6, 7}

Orders of visit

① 1 2 6 7 4 5 8

② 1 2 3 4 5 6 7 8

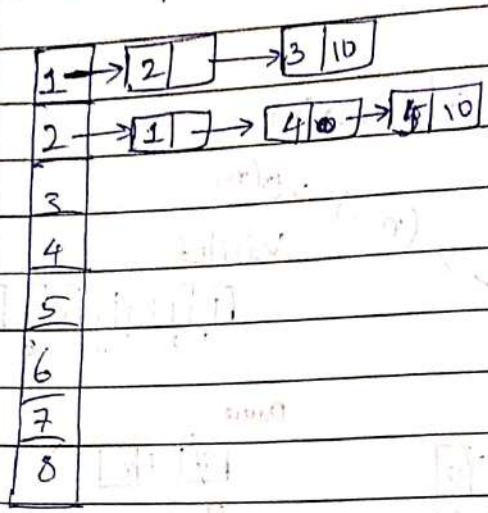
We can get more than 1 orders

SURYA Gold
Date
Page

BFS analysis in case of linked list

Space complexity – O(n)

Time complexity – depends on implementation



O(2E) → undirected graph

O(E) + O(V) → Time complexity

Space complexity → O(V)

BFS analysis in case of Adjacency matrix

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

Space complexity → O(V)

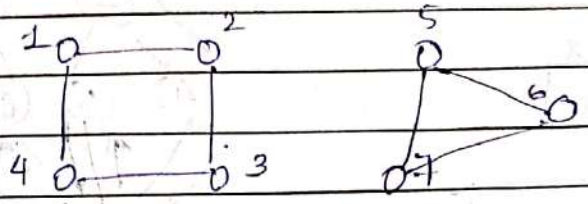
Time complexity → $O(n^2)$

→ $O(V^2)$

Initializing → $O(V^2+V) \rightarrow O(V^2)$

Breadth First Traversal using BFS

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



visited	a	0	0	0	0	0	0
	1	2	3	4	5	6	7

visited	1	1	1	1	0	0	0

call BFS(5)

visited	1	1	1	1	1	1	1

Time complexity – O(E+V)

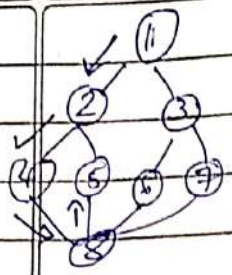
Space complexity – O(V)

DFS algorithm

```
DFS(v)
{
  visited[v]=1;
  for each vertex adj to v do
  {
      if (visited[w]=0) then
          DFS(w)
  }
}
```

SURYA Gold
Date
Page

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



Stack

v=1
w={2,3}

Visited

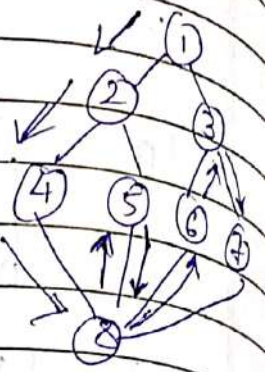
1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0

Stack

v=1	v=2
w={2,3}	w={1,4,5}

Visited

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



Stack

v=1	v=2	v=4
w={2,3}	w={1,4,5}	w={2,8}

Visited

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

Stack

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

Visited

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

Stack

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

Visited

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

Stack

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

Stack

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

SURYA Gold
Date Page

Visited

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

Stack

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

Visited

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

Stack

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

Visited

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

Stack

v = 1	v = 2	v = 4	v = 8	v = 6	v = 3	v = 7
w = {2,3}	w = {1,4,5}	w = {2,8}	w = {4,5,6,7}	w = {3,8}	w = {1,6,7}	w = {3,8}
(7)	(6)	(5)	(4)	(3)	(2)	(1)

Popping order

Space complexity - O(V)

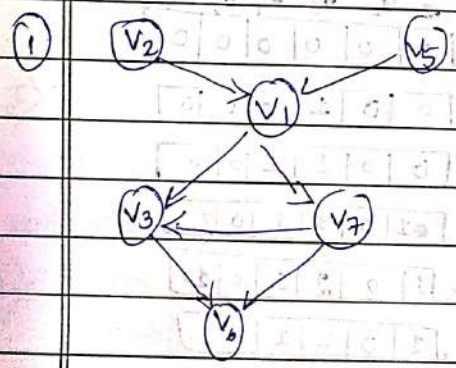
Time complexity - ~~O(E)~~ O(V+E) → Adjacency List

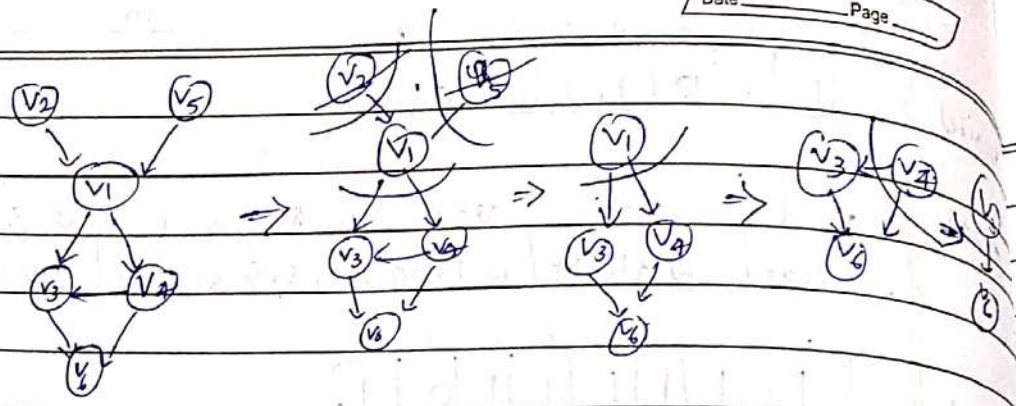
O(V^2) → Adjacency matrix

Topological Sort

Input: Directed Acyclic graph

Output: (i) → (j), i should be first in order i.e., sorted





Orders:- $v_2, v_5, v_1, v_4, v_3, v_6$

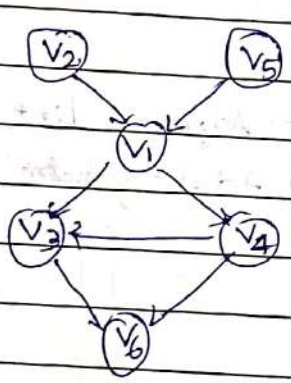
$v_5, v_2, v_1, v_4, v_3, v_6$

Approach-1

Step 1:- Identify vertices that has no incoming edges

Step 2:- Delete this vertex of indegree 0 and all its outgoing edges from the graph. Place it in the output

Step 3:- Ripeat step (1) & (2) until graph is empty



Above method:- Traversal Step 1

Time complexity:- $O(v) \times V = O(v^2)$

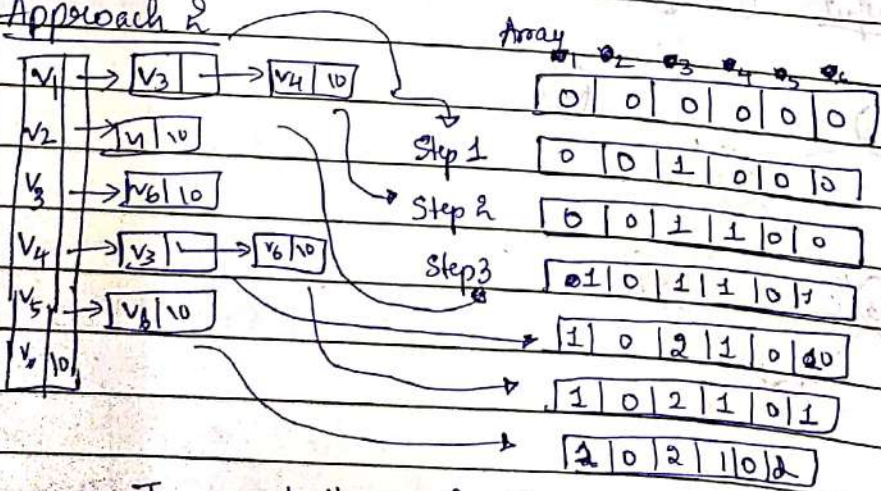
Step 2:- $O(E)$

Overall:- $O(V^2+E)$

It can be reduced to $O(V+E)$

Space complexity:- $O(1)$

Approach 2



Time complexity:- $O(V+E)$ → No. of entries in A.L.

Step 1:- Print highest incoming degree

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

1	2	3	4	5	6	
1	-1	2	1	0	2	V_2
1	2	3	4	5	6	
0	-1	2	1	0	2	V_2, V_5
0	-1	2	1	-1	2	V_2, V_5, V_1
-1	-1	1	0	-1	2	V_2, V_5, V_1, V_4
-1	-1	0	-1	-1	1	V_2, V_5, V_1, V_4, V_3
-1	-1	-1	-1	-1	0	$V_2, V_5, V_1, V_4, V_3, V_6$
-1	-1	-1	-1	-1	-1	

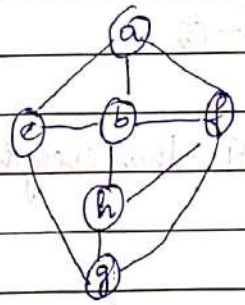
Step 3:- Then the vertice indegree is reduced by 1

Step 4:- Delete the node whose indegree is 0

Time complexity -> $O(V+E)$ -> Topological sort.

(1) GATE 2003

DFS on



I) a b e g h f

II) a b f e g h

III) a b f h g e

IV) a f g h b e

So I, III, IV are DFS

I)

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

II not possible

III)

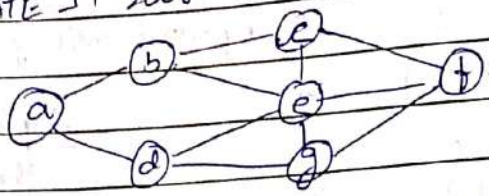
v = a	v = b	v = f	v = g h	v = g	v = e
w = {e, b, f}	w = {a, e, f, h}	w = {a, b, g}	w = {b, f, g}	w = {e, h, f}	w = {a, g, b}

IV)

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

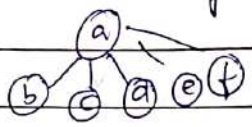
SURYA Gold
Date _____ Page _____

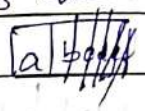
(2) GATE IT 2008



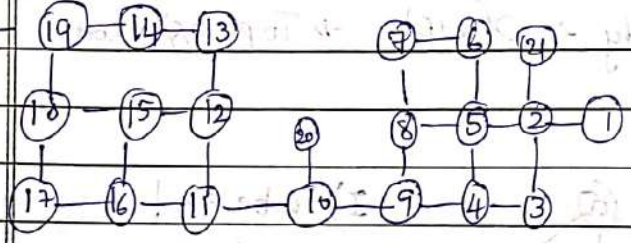
(a) a b e f d g c → Check (direct) connectivity of graph ✗
(b) a b e f c g d ✓
(c) a d g e b c f - ✓
(d) a d b c y e f - ✗

DFS → n-1 of elements BFS queue of 2 elements





(3) GATE 2014 → A graph without number given. We shall number it

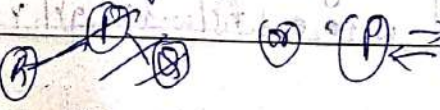


Depth of recursion stack → 19, then everything gets popped off

(4) GATE 2006

Consider a DFS of an undirected graph with 3 vertices P, Q & R. Let 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 vertex 'u' is last visited. Given that d(P) = 5 units, f(P) = 12 units what is true about graphs
d(Q) = 6 units, f(Q) = 10 units
d(R) = 14 units, f(R) = 18 units

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



Two components of graph

∴ P is finished before calling R

Hashing

SURYA Gold
Date _____ Page _____

	Searching (worst case)	
Unsorted Array	O(n)	
Sorted Array	O(log n)	
Linked List	O(n)	
Binary Tree	O(n)	
Binary Search Tree	~~Θ(log n)~~ ⇒ O(n)	
ABST	O(log n)	
Priority Queue	O(n)	O(1) → min & max

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

Direct Address Table

Each record has unique key you have to know key value.
we can store it in array
Separate arrays for record & key to be maintained
Fails when values are large

Introduction to Hashing

1) Chaining → No restriction on no. of elements
Requires more space since we maintain a linked list.

2) Open Addressing
Inserts within table, No. element must be of table size

Chaining → Deletions easier
OA → Insertion, Searching

Insertion, Deletion, Searching – constant time in Hashing

Chaining → Worst case – O(n)
Average case – $\Theta(1 + n/m)$

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

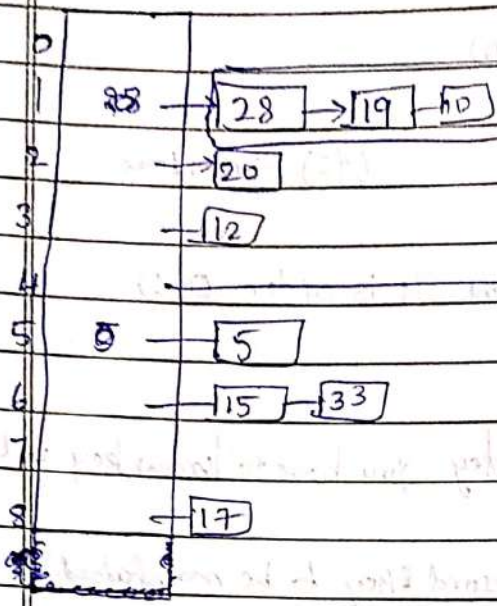
$n = k * m$ k is constant

$\alpha = k$

GATE 2014

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

Sol:



Max chain length:- 3

Min chain length:- 0

Avg chain length $= \alpha = \frac{n}{m} = \frac{9}{9} = 1$

$\alpha = 1$

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

Sol: $100 - 3 = 97$ RBA (88) ME

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

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

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

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

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

SURYA Gold
Date Page

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

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

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

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

Open Addressing

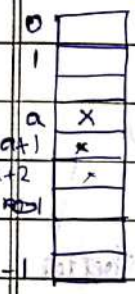
Time complexity - O(n) → Worst case

O(1) → Average case

Deletions is troublesome

$h(k) = U \to \{0, \ldots m-1\}$

Linear Probing



$h(k): U \to \{0 \ldots m-1\}$

$h(k) = a$

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

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

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

$= (a+2) \bmod m$

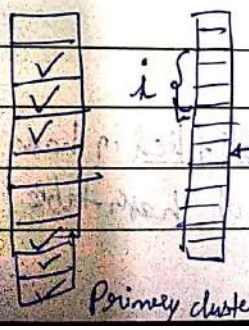
We search till we get free slot

$i = (0, 1, 2 \ldots\ldots, m-1)$

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

Problems:- (1) Secondary clustering → It is a problem in which both elements start from same i & then probing is same for both

(2) Primary clustering → It is a problem in which many elements cluster in a particular part of hash table



filled by probability i/m & but has higher probability the next element after filled slots

Disadvantage:- Search entire cluster until find an empty space

Search time :- O(1) or O(2)

Quadratic Probing

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

Next probe position is quadratic

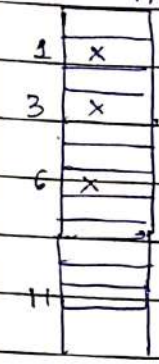
Probe position increases quadratically

Avoids (1) Primary clustering

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

Eg: $m=10$, $(0 \ldots 9)$, $c=1$, $c_2=1$,

$h(k) = \bmod 10$



$h'(k_1,1) = 1+1+1 = 3$

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

$h'(k_1,3) = 1+1+9 = 11$

$h'(k_1,4) = 1+1+16 = 18$

$h'(k_1,5) = 1+1+25 = 27$

$h'(k_1,6) = 1+1+36 = 37$

$h'(k_1,7) = 1+1+49 = 51$

$h'(k_1,8) = 1+1+64 = 66$

$h'(k_1,9) = 1+1+81 = 83$

Problem:- Whole (Entire) table is not examined

Wisely choose c_1, c_2, m

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

Double Hashing

Avoids (1) Primary clustering (2) Secondary clustering

Successive probes depends on keys and does not following same sequence if initial probe position is same

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

Choose m, k are relatively prime

$h_2(k) \rightarrow$ odd $< m$.

$m \rightarrow$ a power of 2

Loadfactor α = Probability of collision = $\frac{\text{No. of slots filled in hash table}}{\text{Size of hash table}}$

SURYA Gold

Date _____ Page _____

(1) Consider a hash table that distributes key 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

Sol: $\frac{i}{20} > 0.5$ $\frac{i}{m} > 0.5$

$\boxed{i = 10}$

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

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

4 elements 42, 23, 34, 46 are in their positions

2 elements 52, 33 are probed

(42, 23, 34) 52 33

46 → 5 ways

42, 23, 34 → 3! ways

$3! \times 5 = 6 \times 5$

$= 30$ ways

Recursion

(1)

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

Activation record is created on calling each function in stack

Stack:

	A(3)	A(2)	A(1)	A(0)
main()	n=3	n=2	n=1	n=0
	5	5	5	

Instruction pointer points to 5 which is next instruction to be executed

n is local variable

main() | A(3) | A(2) | A(1) | A(0)

main() | A(3) | A(2) | A(1)

main() | A(3) | A(2)

main() | A(3)

main()

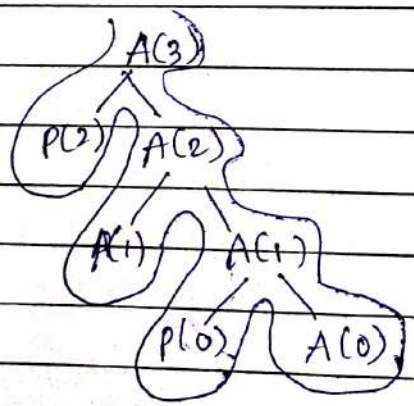
A(0) returns without doing anything. Then ~~A(1) is called. We have to~~ go to line no. 5 & same continues ~~main() is called & gets~~ finished.

2 1 0

Space complexity:
- Stack records = n+1
- Space = O(n)

Time complexity:
$T(n) = C + T(n-1)$
$= O(n)$

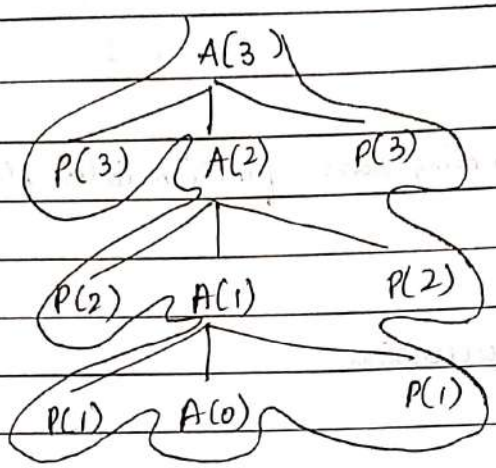
5 Activation records are created



Depth of tree = n+1
= Max length of stack records

(2)

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



O/P:- 3 2 1 1 2 3

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

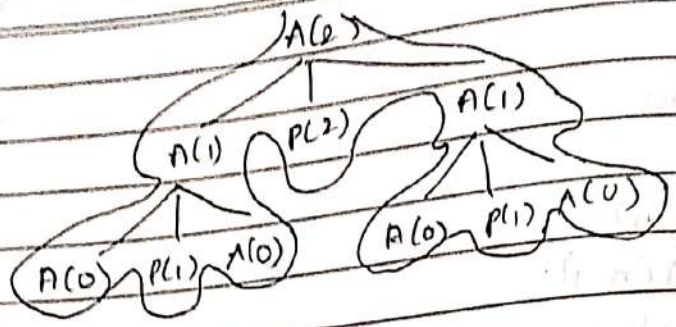
$= O(n)$

Stack records = $n+1$

$= O(n)$

(3)

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



1 2 1

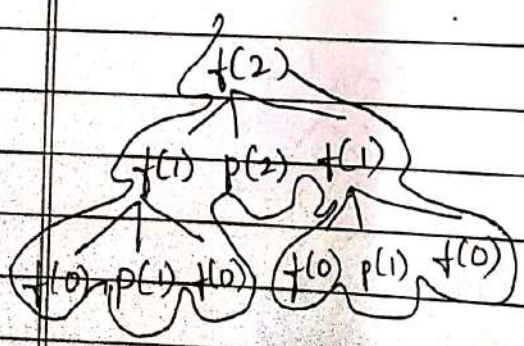
A(0)	A(0) A(0) A(0)
A(1)	A(1)
A(2)	

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

Dynamic programming saves function calls. Hence saving time

Analysing the recursion

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



1 2 1

No. of function calls
f(1) = 3
f(2) = 7
f(3) = 15
$f(n) = 2^{n+1} - 1$

F(n) = No. of times f(n) called
F(n) = 2F(n-1) + 1

1 - already calling

SURYA Gold

Date _____ Page _____

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

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

$2.(2.(2.(F(n-3)+1)+1)+1$

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

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

$= 2^i F(n-i) + 2^{i-1} + 2^{i-2} + 2^{i-3} \ldots$

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

$i = n$

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

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

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

It is in Geometric Progression

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

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

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

~~0~~	~~0~~	~~0~~	~~0~~
1	1		
2			

Time Complexity =

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

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

$= O(2^n)$

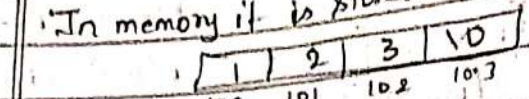
Space complexity = $O(n)$. Since depth of tree is Order of n

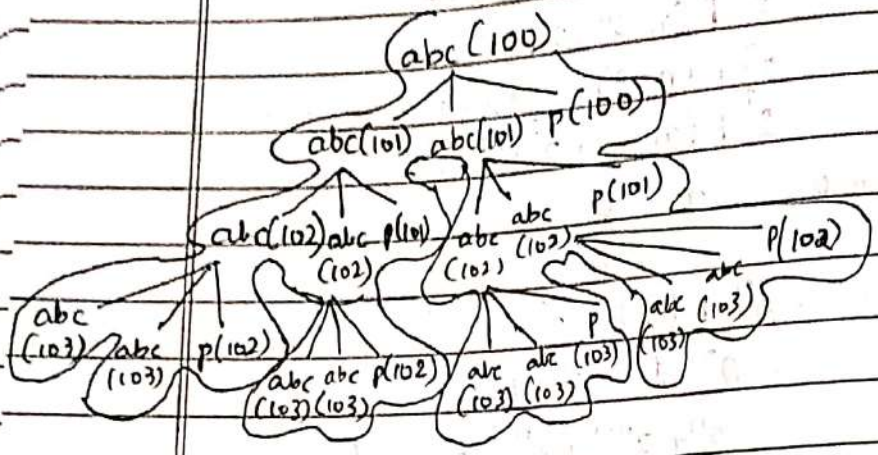
Gate 2001

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

main(){
    abc("123");
}
```

a) In memory it is stored as





O/p:- 3 3 2 3 3 2 1

b) If abc(s) is called with a null terminated string 's' of length 'n' characters (not counting null ('\0') character. How many characters will be printed by abc(s)

C(n) - no. of characters printed printed by giving input of n characters

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

Just above use back substitution method

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

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

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

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

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

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

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

C(1) - 1 char
C(2) - 3 ch
C(3) - 7 ch
(crosscheck with a)

Gate 2004

```
int rec(int n)
{
```

```
if(n==1)
  return
else
  return (rec(n-1)+rec(n-1))
```

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

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

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

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

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

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

$2-1$

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

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

Q9

Gate 2005

```
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=12	sum=12	sum=14	sum=14
	k=8	k=4	k=0	k=2	
	j=204	j=20	j=2	j=0	

2 0 4 8 0

Gate 2010

```
int f(int *a, int n)
{
  if (n<=0)
    return 0;
  else
    if(*a%2==0)
      return *a + f(a+1, n-1);
    else
      return *a - f(a+1, n-1);
}
int main(){
  int a[] = {12,7,13,4,11,6};
  printf("%d", f(a,6));
  return 0;
}
```

100	102	104	106	108	110
12	7	13	4	11	6

f(a,6)

=> f(100,6)

↓

12%2==0,

12 + f(102,5)

↓

7%2==0

↓

7 - f(104,4)

↓

13%2==0

↓

13 - f(106,3)

↓

SURYA Gold

Date _____ Page _____

4%2 == 0

4 + f(108, 2)

11%2 == 0

11 − f(1100, 2)

6%2 == 0

6 + f(112, 0)

6 + 0 = 6

12 11 33 7 10

12+7 − 13 − 4 + 11 − 6

4 + 5

13 − 9

7 − 4

12 + 3

= 15

12+(7 −(13−(4+(11 − (6+(0)

11 − 6

4 + 5

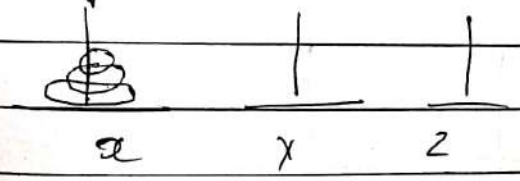
13 − 9

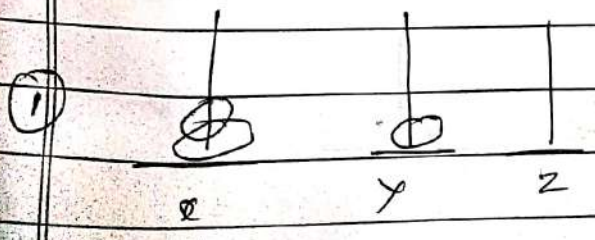
7 − 4

12 + 3

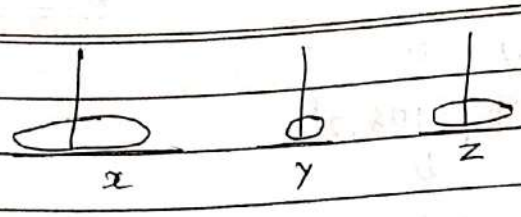
= 15

Tower of Hanoi

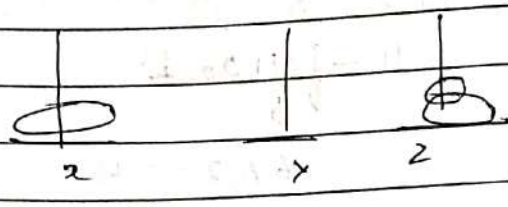




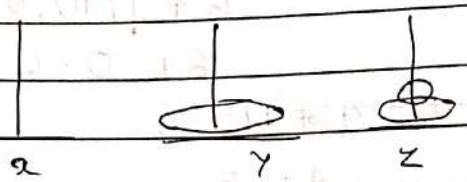
(2)



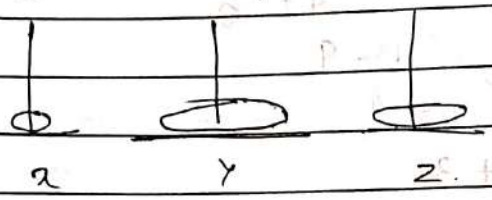
(3)



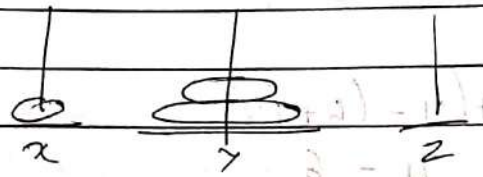
(4)



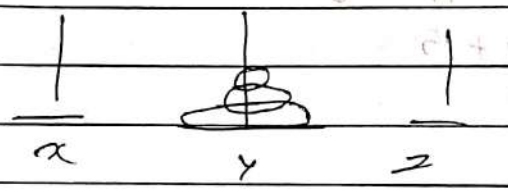
(5)



(6)

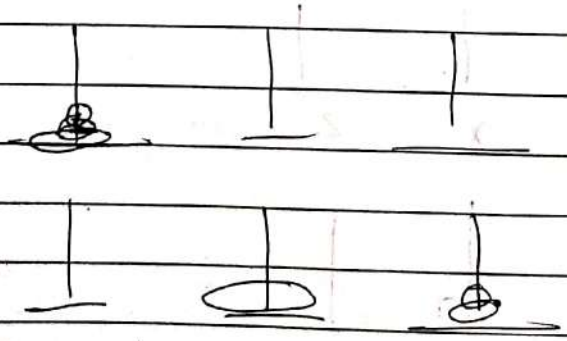


(7)



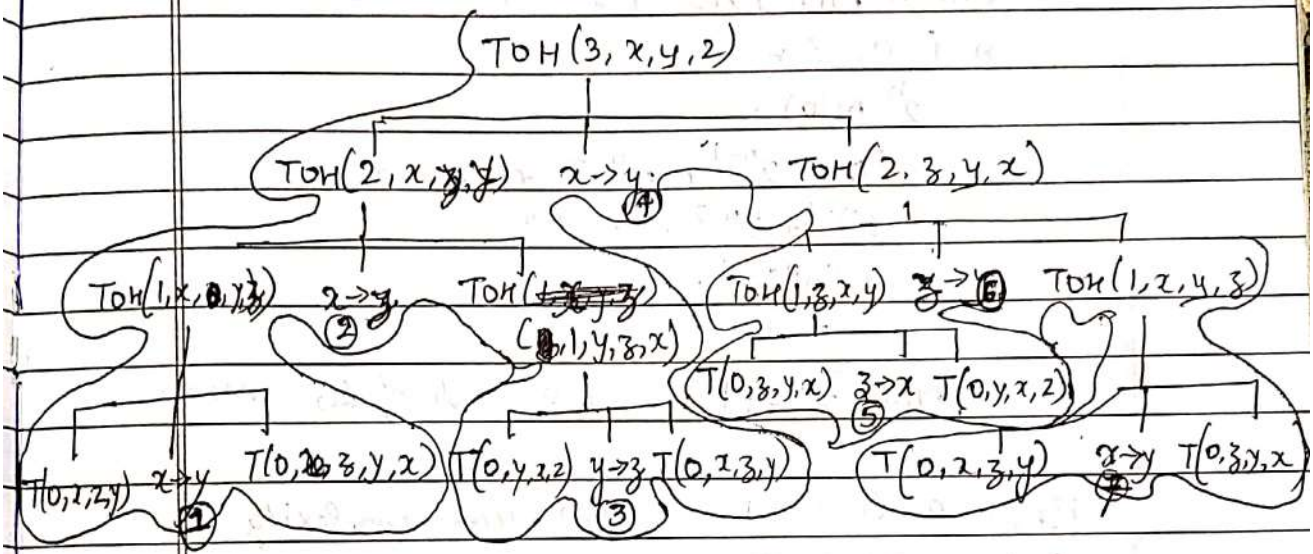
n a x→y z

n-1, x→z, y x→y n-1, z→y, x



Algorithm for Towers of Hanoi

```
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);
  }
}
```



→ 3 disc: No. of function invocations = 15
No. of movements = 7

Analysis of tower of hanoi

```
TOH(n, x, y, z)
{
  if(n >= 1)
  {
```

$I(n) = 1 + 2I(n-1)$ — (1) $I(n)$ – Invocation required for n disc

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

$I(n-1) = 1 + 2I(n-2)$ — (2)

$I(n-2) = 1 + 2I(n-3)$ — (3)

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

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

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

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

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

$n - i = 0, \; i = n$

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

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

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

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

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

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

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

$n - i = 0, \; i = n$

$= 2^n M(0) +$

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

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

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

$\boxed{M(n) = 2^n - 1}$ $\quad$ Movements of disc $M(n)$

$T(n) = 2T(n-1) + 1$ $\quad$ $T(n)$ – Time complexity

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

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

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

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

T(0) T(0) T(0)

T(1)

T(2)

T(3)

Stack records $= n + 1$

$= O(n)$

Improved algo

SURYA Gold

Date _____ Page _____

```
TOH(n, x, y, z)
{
  if (n>1)
  {
    TOH(n-1, x, 3, y)
    move for 'x' to 'y'
    TOH(n-1, 3, y, x)
  else
    if (n==1) move 'x to y
```

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

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

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

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

$n - i = 1$

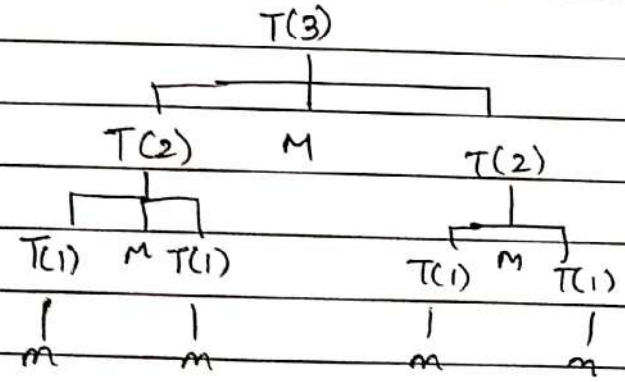
$\Rightarrow i = n - 1$

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

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

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

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



No. of invocations = 7