

Stacks & Queues

SURYA Gold

Date _____ Page _____

Stacks

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

Applications

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

Implementation of Stack using arrays

Disadvantage:- Predict the size of stack (array)

Advantage:- ① Faster ② 2nd disadvantage

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

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

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

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

```
{
```

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

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

```
else {
```

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

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

```
    stack [top] = item; // if n = 1, n < 4
```

```
int pop ()
```

```
{ int temp;
```

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

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

```
return -1;
```

```
else {
```

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

Note:-

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

Later we push some element, we override the previous element

Takes $O(1)$ time.

Linked list implementation of stack

Elements are created from heap memory dynamically

Struct node

```
{
```

```
    int data;
    struct node *link;
}
```

$\therefore \text{head} \rightarrow \text{top}$

push(item int item)

```
{
```

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

```
{
```

printf(" Malloc error");

return;

```
}
```

p->data = item

p->link = NULL; // old node

p->link = head; // new created node linked to old head

head = p // old head is updated to new head

```
}
```

int pop()

```
{
```

int item; struct node *p;

Scanned by CamScanner

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

STOP JAP

return; // Condition for singly linked list is empty

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

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

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

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

ptr = head; // If item is present then "head" is

head = head->next;

Not found

free(ptr);

front = item->data

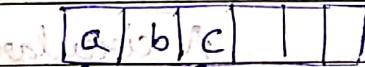
}

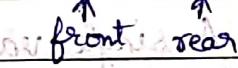
Overflow

Not found

Implementation of a Queue using circular array

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

We shall put watermelons into it. 

It will drop due to gravitational force 

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

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

Deletion is done from front so that the insertion order of

Insertion is done from rear because it is maintained

de));

enqueue(item)

{

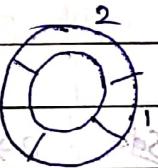
rear = (rear + 1) mod n;

if (front == rear)

{

front

3



// Rear & front points to 0

APPY IT IS overflow

if (rear == front) pointf ("Q is full");

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

else front = rear - 1; return if (front == 0) pointf ("Q is empty")

else { q[front] = item; return -1; }

{ rear = rear - 1; }

return;

{

else { q[rear] = item; }

return;

{

else

front = (front + 1) mod n;

item = q[front];

return item;

{ }

GATE 2012

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

Sol:-

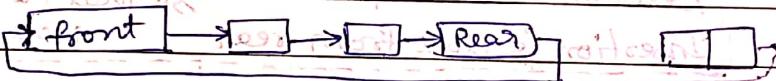
Queue Full

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

Queue Empty

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

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



Ans:-

 $p \rightarrow ?$

Sol:-

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

Question

Gate 1994

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

(option)

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

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

Check
for a)

5

4

3

2

1

3 4 5 2 1
(b) is

8

4

3

2

1

1

1000 = 1000

insertion

4

8

1

1

1000 = 1000

insertion

5

8

2

2

1000 = 1000

insertion

5

8

2

2

1000 = 1000

insertion

1

5

4

3

2

Gate 1997

(T.O.H. 2010)

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 DElemin(Q). For a sequence of push operations, the keys chosen are in

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

Gate 2003

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

Sol.

Fix t = 1: Suppose stack has no pending push/pops at t = 0

Time for 1 push or pop = $2X$ (to read/write + time)Time for 2 push or pop = $2X + 2Y$ (with 2 op. every Y sec. after previous op.)

1 transition

Push: $+2X$ Stack life - Pop: $-2X$ One element in S: X ready $-X$ Two elements in S: $2X + 2Y$ $-2X = 2(X+Y) - X$ Three elements in S: $3X + 3Y$ $-3X = 3(X+Y) - X$ Four elements in S: $4X + 4Y$ $-4X = 4(X+Y) - X$ Five elements in S: $5X + 5Y$ $-5X = 5(X+Y) - X$ Six elements in S: $6X + 6Y$ $-6X = 6(X+Y) - X$ Seven elements in S: $7X + 7Y$ $-7X = 7(X+Y) - X$ Eight elements in S: $8X + 8Y$ $-8X = 8(X+Y) - X$ Nine elements in S: $9X + 9Y$ $-9X = 9(X+Y) - X$ Ten elements in S: $10X + 10Y$ $-10X = 10(X+Y) - X$ Eleven elements in S: $11X + 11Y$ $-11X = 11(X+Y) - X$ Twelve elements in S: $12X + 12Y$ $-12X = 12(X+Y) - X$ Thirteen elements in S: $13X + 13Y$ $-13X = 13(X+Y) - X$ Fourteen elements in S: $14X + 14Y$ $-14X = 14(X+Y) - X$ Fifteen elements in S: $15X + 15Y$ $-15X = 15(X+Y) - X$ Sixteen elements in S: $16X + 16Y$ $-16X = 16(X+Y) - X$ Seventeen elements in S: $17X + 17Y$ $-17X = 17(X+Y) - X$ Eighteen elements in S: $18X + 18Y$ $-18X = 18(X+Y) - X$ Nineteen elements in S: $19X + 19Y$ $-19X = 19(X+Y) - X$ Twenty elements in S: $20X + 20Y$ $-20X = 20(X+Y) - X$ Twenty-one elements in S: $21X + 21Y$ $-21X = 21(X+Y) - X$ Twenty-two elements in S: $22X + 22Y$ $-22X = 22(X+Y) - X$ Twenty-three elements in S: $23X + 23Y$ $-23X = 23(X+Y) - X$ Twenty-four elements in S: $24X + 24Y$ $-24X = 24(X+Y) - X$ Twenty-five elements in S: $25X + 25Y$ $-25X = 25(X+Y) - X$ Twenty-six elements in S: $26X + 26Y$ $-26X = 26(X+Y) - X$ Twenty-seven elements in S: $27X + 27Y$ $-27X = 27(X+Y) - X$ Twenty-eight elements in S: $28X + 28Y$ $-28X = 28(X+Y) - X$ Twenty-nine elements in S: $29X + 29Y$ $-29X = 29(X+Y) - X$ Thirty elements in S: $30X + 30Y$ $-30X = 30(X+Y) - X$ Thirty-one elements in S: $31X + 31Y$ $-31X = 31(X+Y) - X$ Thirty-two elements in S: $32X + 32Y$ $-32X = 32(X+Y) - X$ Thirty-three elements in S: $33X + 33Y$ $-33X = 33(X+Y) - X$ Thirty-four elements in S: $34X + 34Y$ $-34X = 34(X+Y) - X$ Thirty-five elements in S: $35X + 35Y$ $-35X = 35(X+Y) - X$ Thirty-six elements in S: $36X + 36Y$ $-36X = 36(X+Y) - X$ Thirty-seven elements in S: $37X + 37Y$ $-37X = 37(X+Y) - X$ Thirty-eight elements in S: $38X + 38Y$ $-38X = 38(X+Y) - X$ Thirty-nine elements in S: $39X + 39Y$ $-39X = 39(X+Y) - X$ Forty elements in S: $40X + 40Y$ $-40X = 40(X+Y) - X$ Forty-one elements in S: $41X + 41Y$ $-41X = 41(X+Y) - X$ Forty-two elements in S: $42X + 42Y$ $-42X = 42(X+Y) - X$ Forty-three elements in S: $43X + 43Y$ $-43X = 43(X+Y) - X$ Forty-four elements in S: $44X + 44Y$ $-44X = 44(X+Y) - X$ Forty-five elements in S: $45X + 45Y$ $-45X = 45(X+Y) - X$ Forty-six elements in S: $46X + 46Y$ $-46X = 46(X+Y) - X$ Forty-seven elements in S: $47X + 47Y$ $-47X = 47(X+Y) - X$ Forty-eight elements in S: $48X + 48Y$ $-48X = 48(X+Y) - X$ Forty-nine elements in S: $49X + 49Y$ $-49X = 49(X+Y) - X$ Fifty elements in S: $50X + 50Y$ $-50X = 50(X+Y) - X$ Fifty-one elements in S: $51X + 51Y$ $-51X = 51(X+Y) - X$ Fifty-two elements in S: $52X + 52Y$ $-52X = 52(X+Y) - X$ Fifty-three elements in S: $53X + 53Y$ $-53X = 53(X+Y) - X$ Fifty-four elements in S: $54X + 54Y$ $-54X = 54(X+Y) - X$ Fifty-five elements in S: $55X + 55Y$ $-55X = 55(X+Y) - X$ Fifty-six elements in S: $56X + 56Y$ $-56X = 56(X+Y) - X$ Fifty-seven elements in S: $57X + 57Y$ $-57X = 57(X+Y) - X$ Fifty-eight elements in S: $58X + 58Y$ $-58X = 58(X+Y) - X$ Fifty-nine elements in S: $59X + 59Y$ $-59X = 59(X+Y) - X$ Fifty elements in S: $60X + 60Y$ $-60X = 60(X+Y) - X$ Fifty-one elements in S: $61X + 61Y$ $-61X = 61(X+Y) - X$ Fifty-two elements in S: $62X + 62Y$ $-62X = 62(X+Y) - X$ Fifty-three elements in S: $63X + 63Y$ $-63X = 63(X+Y) - X$ Fifty-four elements in S: $64X + 64Y$ $-64X = 64(X+Y) - X$ Fifty-five elements in S: $65X + 65Y$ $-65X = 65(X+Y) - X$ Fifty-six elements in S: $66X + 66Y$ $-66X = 66(X+Y) - X$ Fifty-seven elements in S: $67X + 67Y$ $-67X = 67(X+Y) - X$ Fifty-eight elements in S: $68X + 68Y$ $-68X = 68(X+Y) - X$ Fifty-nine elements in S: $69X + 69Y$ $-69X = 69(X+Y) - X$ Fifty elements in S: $70X + 70Y$ $-70X = 70(X+Y) - X$ Fifty-one elements in S: $71X + 71Y$ $-71X = 71(X+Y) - X$ Fifty-two elements in S: $72X + 72Y$ $-72X = 72(X+Y) - X$ Fifty-three elements in S: $73X + 73Y$ $-73X = 73(X+Y) - X$ Fifty-four elements in S: $74X + 74Y$ $-74X = 74(X+Y) - X$ Fifty-five elements in S: $75X + 75Y$ $-75X = 75(X+Y) - X$ Fifty-six elements in S: $76X + 76Y$ $-76X = 76(X+Y) - X$ Fifty-seven elements in S: $77X + 77Y$ $-77X = 77(X+Y) - X$ Fifty-eight elements in S: $78X + 78Y$ $-78X = 78(X+Y) - X$ Fifty-nine elements in S: $79X + 79Y$ $-79X = 79(X+Y) - X$ Fifty elements in S: $80X + 80Y$ $-80X = 80(X+Y) - X$ Fifty-one elements in S: $81X + 81Y$ $-81X = 81(X+Y) - X$ Fifty-two elements in S: $82X + 82Y$ $-82X = 82(X+Y) - X$ Fifty-three elements in S: $83X + 83Y$ $-83X = 83(X+Y) - X$ Fifty-four elements in S: $84X + 84Y$ $-84X = 84(X+Y) - X$ Fifty-five elements in S: $85X + 85Y$ $-85X = 85(X+Y) - X$ Fifty-six elements in S: $86X + 86Y$ $-86X = 86(X+Y) - X$ Fifty-seven elements in S: $87X + 87Y$ $-87X = 87(X+Y) - X$ Fifty-eight elements in S: $88X + 88Y$ $-88X = 88(X+Y) - X$ Fifty-nine elements in S: $89X + 89Y$ $-89X = 89(X+Y) - X$ Fifty elements in S: $90X + 90Y$ $-90X = 90(X+Y) - X$ Fifty-one elements in S: $91X + 91Y$ $-91X = 91(X+Y) - X$ Fifty-two elements in S: $92X + 92Y$ $-92X = 92(X+Y) - X$ Fifty-three elements in S: $93X + 93Y$ $-93X = 93(X+Y) - X$ Fifty-four elements in S: $94X + 94Y$ $-94X = 94(X+Y) - X$ Fifty-five elements in S: $95X + 95Y$ $-95X = 95(X+Y) - X$ Fifty-six elements in S: $96X + 96Y$ $-96X = 96(X+Y) - X$ Fifty-seven elements in S: $97X + 97Y$ $-97X = 97(X+Y) - X$ Fifty-eight elements in S: $98X + 98Y$ $-98X = 98(X+Y) - X$ Fifty-nine elements in S: $99X + 99Y$ $-99X = 99(X+Y) - X$ Fifty elements in S: $100X + 100Y$ $-100X = 100(X+Y) - X$ Fifty-one elements in S: $101X + 101Y$ $-101X = 101(X+Y) - X$ Fifty-two elements in S: $102X + 102Y$ $-102X = 102(X+Y) - X$ Fifty-three elements in S: $103X + 103Y$ $-103X = 103(X+Y) - X$ Fifty-four elements in S: $104X + 104Y$ $-104X = 104(X+Y) - X$ Fifty-five elements in S: $105X + 105Y$ $-105X = 105(X+Y) - X$ Fifty-six elements in S: $106X + 106Y$ $-106X = 106(X+Y) - X$ Fifty-seven elements in S: $107X + 107Y$ $-107X = 107(X+Y) - X$ Fifty-eight elements in S: $108X + 108Y$ $-108X = 108(X+Y) - X$ Fifty-nine elements in S: $109X + 109Y$ $-109X = 109(X+Y) - X$ Fifty elements in S: $110X + 110Y$ $-110X = 110(X+Y) - X$ Fifty-one elements in S: $111X + 111Y$ $-111X = 111(X+Y) - X$ Fifty-two elements in S: $112X + 112Y$ $-112X = 112(X+Y) - X$ Fifty-three elements in S: $113X + 113Y$ $-113X = 113(X+Y) - X$ Fifty-four elements in S: $114X + 114Y$ $-114X = 114(X+Y) - X$ Fifty-five elements in S: $115X + 115Y$ $-115X = 115(X+Y) - X$ Fifty-six elements in S: $116X + 116Y$ $-116X = 116(X+Y) - X$ Fifty-seven elements in S: $117X + 117Y$ $-117X = 117(X+Y) - X$ Fifty-eight elements in S: $118X + 118Y$ $-118X = 118(X+Y) - X$ Fifty-nine elements in S: $119X + 119Y$ $-119X = 119(X+Y) - X$ Fifty elements in S: $120X + 120Y$ $-120X = 120(X+Y) - X$ Fifty-one elements in S: $121X + 121Y$ $-121X = 121(X+Y) - X$ Fifty-two elements in S: $122X + 122Y$ $-122X = 122(X+Y) - X$ Fifty-three elements in S: $123X + 123Y$ $-123X = 123(X+Y) - X$ Fifty-four elements in S: $124X + 124Y$ $-124X = 124(X+Y) - X$ Fifty-five elements in S: $125X + 125Y$ $-125X = 125(X+Y) - X$ Fifty-six elements in S: $126X + 126Y$ $-126X = 126(X+Y) - X$ Fifty-seven elements in S: $127X + 127Y$ $-127X = 127(X+Y) - X$ Fifty-eight elements in S: $128X + 128Y$ $-128X = 128(X+Y) - X$ Fifty-nine elements in S: $129X + 129Y$ $-129X = 129(X+Y) - X$ Fifty elements in S: $130X + 130Y$ $-130X = 130(X+Y) - X$ Fifty-one elements in S: $131X + 131Y$ $-131X = 131(X+Y) - X$ Fifty-two elements in S: $132X + 132Y$ $-132X = 132(X+Y) - X$ Fifty-three elements in S: $133X + 133Y$ $-133X = 133(X+Y) - X$ Fifty-four elements in S: $134X + 134Y$ $-134X = 134(X+Y) - X$ Fifty-five elements in S: $135X + 135Y$ $-135X = 135(X+Y) - X$ Fifty-six elements in S: $136X + 136Y$ $-136X = 136(X+Y) - X$ Fifty-seven elements in S: $137X + 137Y$ $-137X = 137(X+Y) - X$ Fifty-eight elements in S: $138X + 138Y$ $-138X = 138(X+Y) - X$ Fifty-nine elements in S: $139X + 139Y$ $-139X = 139(X+Y) - X$ Fifty elements in S: $140X + 140Y$ $-140X = 140(X+Y) - X$ Fifty-one elements in S: $141X + 141Y$ $-141X = 141(X+Y) - X$ Fifty-two elements in S: $142X + 142Y$ $-142X = 142(X+Y) - X$ Fifty-three elements in S: $143X + 143Y$ $-143X = 143(X+Y) - X$ Fifty-four elements in S: $144X + 144Y$ $-144X = 144(X+Y) - X$ Fifty-five elements in S: $145X + 145Y$ $-145X = 145(X+Y) - X$ Fifty-six elements in S: $146X + 146Y$ $-146X = 146(X+Y) - X$ Fifty-seven elements in S: $147X + 147Y$ $-147X = 147(X+Y) - X$

void insert(Q, x)

if stack not empty then push(s1, x) else push(s2, x)

initially s1 contains all elements and s2 contains none

void delete(Q, x) if stack not empty then pop(s1, x)

{

if stack_empty(s2) then pop(s1, x) else

if stack_empty(s1) then return x

else swap(s1, s2)

pop(s2, x); print("x = ", x); return x

return;

}

Sol:

code

numbers

words

subroutine

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

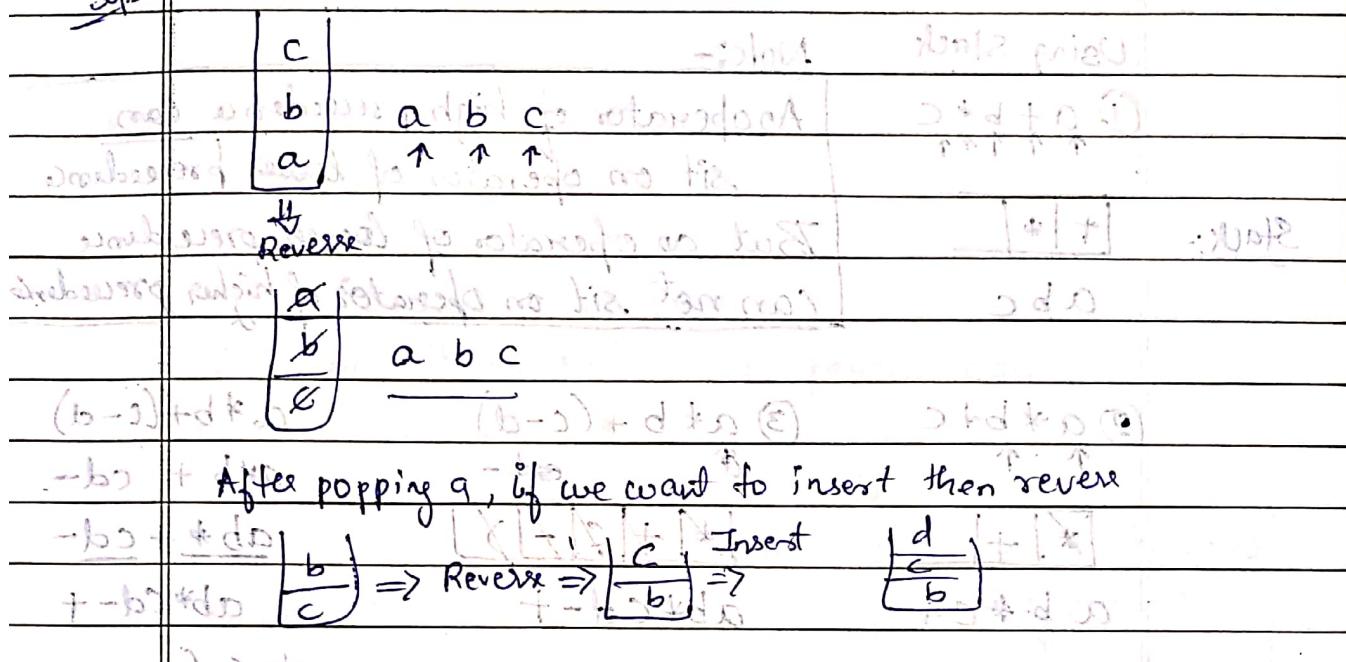
-

-

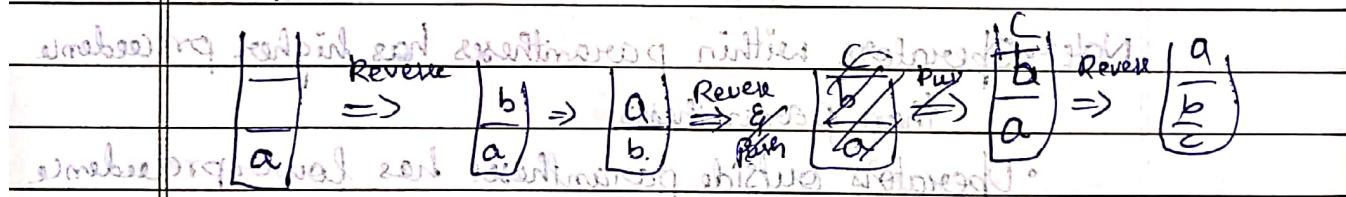
-

GATE-2000 ↔ GATE 2014

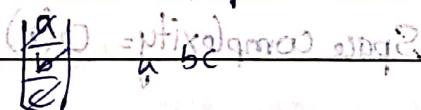
- (1) Suppose a stack implementation supports, in addition to PUSH and POP, an operation "reverse" the order of elements on the stack.
- (2) Implement a queue using the above stack implementation. Show how to implement "ENQUEUE" using a single operation and "DEQUEUE" using a sequence of 3 operations.

Sol:

Enqueue → Reverse Stack → 3 operation



Dequeue → 1 operation (stack, pop, mont)



all ele
then
insert
new
in
of
le to
er
ick &
n

Scanned by CamScanner

Applications of Stacks

SURYA Gold

Date _____ Page _____

Conversion of infix expression to postfix expression

Eg:- ~~operator precedence and associativity in infix expression~~ (1)

④ $a * b + c$ ① $a + b * c$ ② $a + b - c$ ③ $a + (b - c)$

~~ab * + c~~ ~~a + bc *~~ ~~ab + - c~~ ~~a + bc -~~

~~ab * c +~~ ~~abc * +~~ ~~ab + c -~~ ~~abc - +~~

Using Stack

Note:-

① $a + b * c$
↑ ↑ ↑ ↑ ↑ ↑

Stack:-

[+ | * |]

a b c .

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 $a * b + c$
↑ ↓ ↑

[* | + | b |]

a b . * c +

③ $a * b + (c - d)$

[* | + | b | (| c | - | d |) |]

ab * cd - +

$a * b + (c - d)$

$a * b + cd -$

$ab * + cd -$

+ < (

Note:- Operators within parentheses has higher precedence than parentheses

• Operators outside parentheses has lower precedence than parentheses

Space complexity = $O(n)$

Algorithm

① Create a stack

② for each character 't' in the input

 if ('t' is an operand)

 output 't';

 else if ('t' is a right parenthesis)

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

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

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

Push t

}

~~above function~~

→ Time complexity :- $O(n)$

~~(stack push)~~

Uses operator stack ~~(different kinda)~~

b) Postfix evaluation algorithm ~~(different kinda)~~

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

Ex: ① $3 * 2 + 1$ ② $3 + 2 * 1$ ~~39 - 8013 - this is diff. of 1st & 2nd~~

$6 + 1$ $5 * 1$ ~~divide Noki's board~~

(stacks & diff. of diff. of diff. of diff. of board)	First operand becomes second operand and vice versa
Postfix:- \cdot $3 + 2 \rightarrow$ $3 2 * +$	

~~using (i) O cannot be 3 2 1 1 1 2 (5) is it wrong?
 5 1 1 1 1 1 1 if this is wrong, then
 we move 2 to 1 1 1 1 1 1 1 // Push result to stack~~

$3 + 2$

Space complexity:- $O(n)$

5

~~ways of doing~~

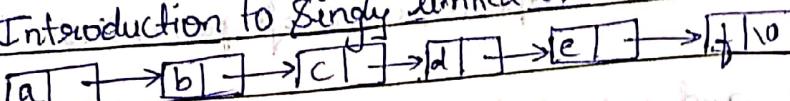
~~diff.~~

~~Algorithm for Postfix Evaluation~~

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

Linked List

Introduction to Singly linked list



head → first node in linked list
→ pointer to first node in list
→ address of first node in memory

Struct node
of

char data;

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

3.

① struct node *p; struct node *head;

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

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

~~head->link != b->link;~~ 1+2 1+2

pointf("%c", head->link->link->link->link->link->data);

Advantage :-

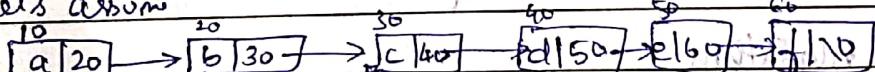
i) Sequential access of element. Takes $O(n)$ time

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

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

Au.Pm.Q

Let's assume



head  **list** contains objects with type information. ①
linked pointers are significant. ②

$p = \text{head} \rightarrow \text{link}$ links all nodes. Insert at i

$20 \rightarrow \text{link}$ ~~is already connected with 17~~

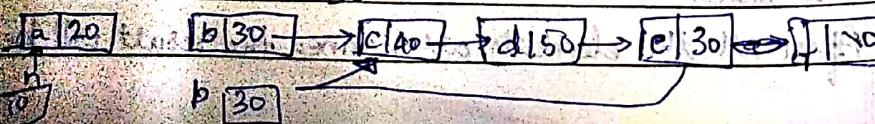
p-30

$p \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link}$ at branch elements or leaf nodes

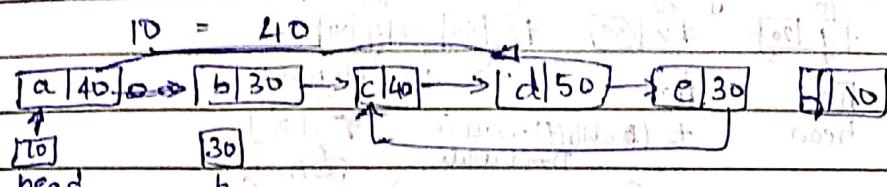
$30 \rightarrow 11 \text{nd} \rightarrow \text{link}$ with several diagonals and bad marks

$40 \rightarrow \text{link} \rightarrow \text{link} = 50 \rightarrow \text{link}$

$$50 - b = p$$



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



($a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$) initial state (3 free blocks) - work done by user.

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

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

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

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

total of 4 free blocks

$d_{\text{node}} \rightarrow \text{data} \rightarrow \text{more} \rightarrow \text{initial ID}$
more = 3000

Traversing a singly linked list : for loop

On a list, we can perform (from i=0 to n-1) traverse along EA

i) Traversing $\text{while } t \neq \text{NULL}$

II) Inserting ($\text{new} = (\text{t} \rightarrow \text{link} \leftarrow \text{t})$ linked)

iii) Deleting $\text{t} = \text{t} \rightarrow \text{link}$

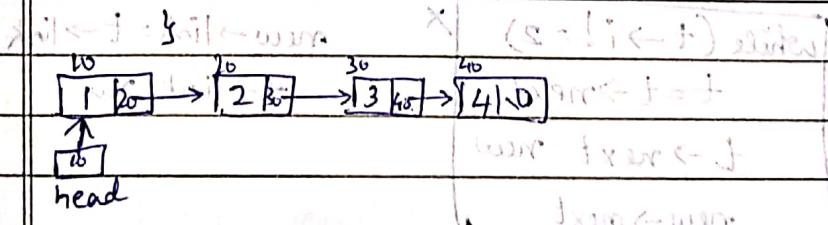
struct node

{

int i;

struct node *link;

head: first node created



new blocks
were created

old blocks
were freed

old blocks
were freed

Moving head may cause loss of nodes created dynamically

(since it does not have any name) are lost

struct node *t; $\text{t} = \text{head}$

$t = \text{head}$

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

~~printf("%d", t->data);~~ $\text{printf("%d", t->i);}$ + share memory

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

nodes (1000-1000) if

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

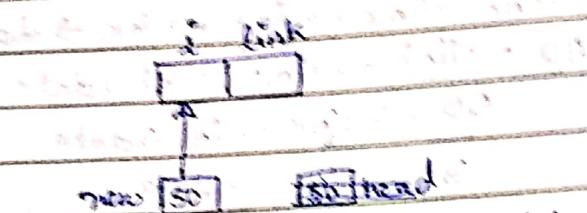
(1000) work

Inserting an element in SLL



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

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



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

At beginning: new \rightarrow link = head

head = new

At end:

Struct node *t; t = head // What is parameter?

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

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

while ($t \rightarrow \text{next} \neq \text{NULL}$) // t → next → t → temp variable

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

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

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

After node:

which we want

Eg: after
2nd node

Struct node *t = head // Main idea: Update

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

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

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

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

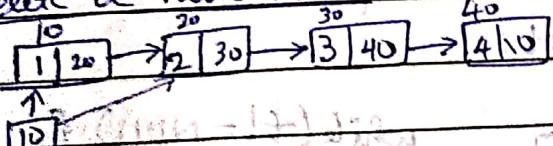
$\text{new} \rightarrow \text{link} = t \rightarrow \text{link}$

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

head

Diagram into below code for 2nd node from head given

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



At beginning:

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

free(t); // Free the deleted node

if (head == NULL) return;

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

free(head)

node = NULL;

From:
tail

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

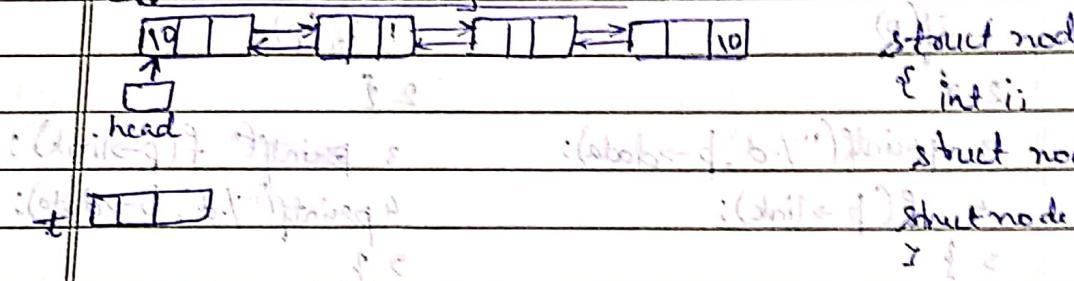
A practical
note:
Eg: 3

struct node *t = head
while ($t \rightarrow \text{next} \rightarrow i \neq 3$)
 $t = t \rightarrow \text{next}$, $i = i + 1$
struct node *t1 = $t \rightarrow \text{next}$
 $t \rightarrow \text{next} = t \rightarrow \text{next} \rightarrow \text{next}$
free ($t1$)

(deletion from left)

(deletion from right)

Insertion into doubly linked list



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

head = t

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

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

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

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

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

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

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

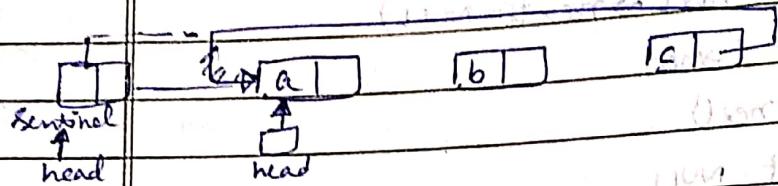
After
Node:
Eg:

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

$t \rightarrow \text{next} = p \rightarrow \text{next}$ (i.e. previous node's next node)

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

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

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

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

Printing the elements of SLL using recursion

Program 1:-

void f(struct node *p)

{

1 if(p)

{

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

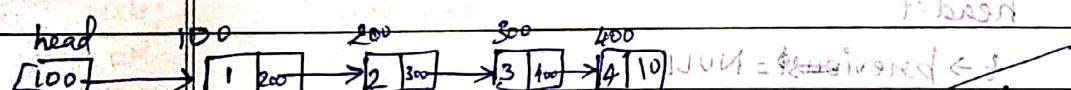
4 f(p->link);

5 }

{

Board - function f

F - board

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

main()

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

main()

O/p: 1 2 3 4

O/p: 4 3 2 1

Reversing an SLL using iterative program +
sentinel node

int t3() {
 return sum of all even numbers
}

struct node *next;

head & done function

(f) reverse linked list (using previous node)

struct node *reverse (struct node *cur)

{

 struct node *prev = NULL, *nextnode=NULL

 while (cur){

 nextnode = cur->next;

 cur->next = prev;

 prev = cur;

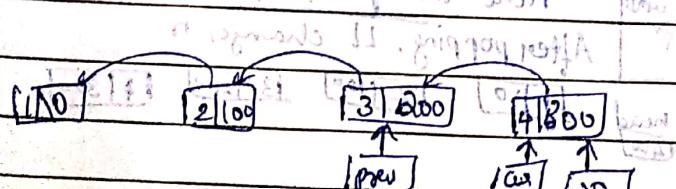
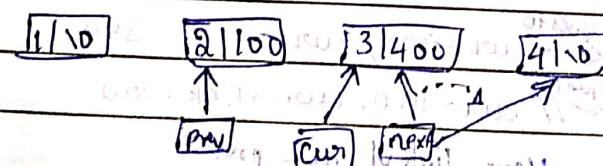
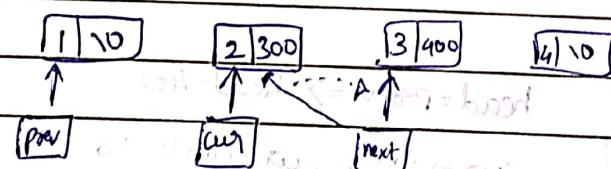
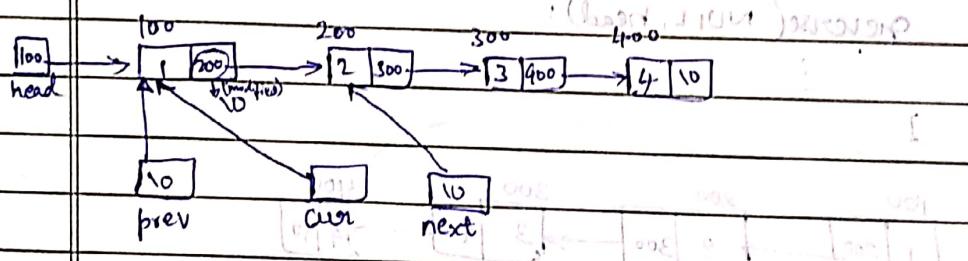
 cur = nextnode;

}

 return prev;

}

beginning



head

400

100 200 300 400 a list of pointers to individual diff. nodes
returning a list of nodes

Recursive program for reversing a Singly Linked List

Struct node *head;

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

1 if(cur)

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

2 reverse(prev, cur->link);

3 cur->link = prev;

4 }

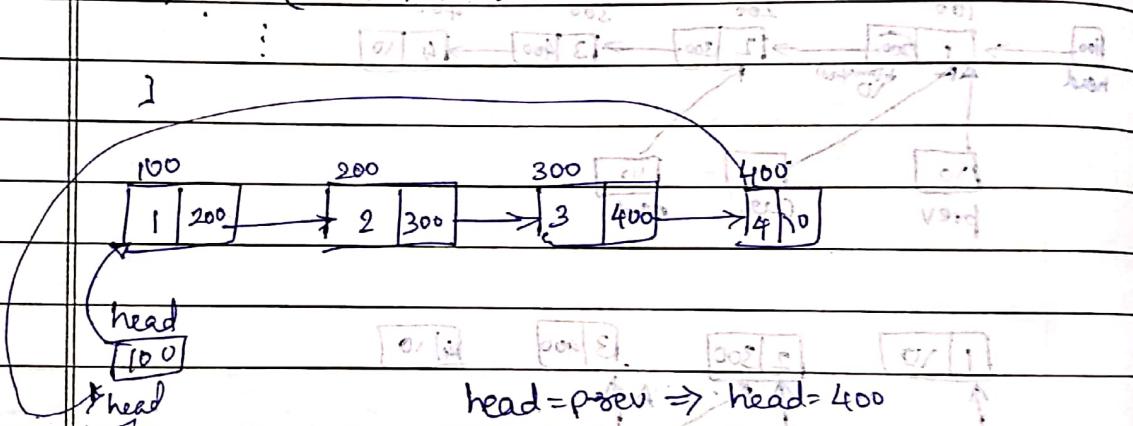
else

head = prev;

void main()

:

reverse(NULL, head);



head = prev \Rightarrow head = 400

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

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

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

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

400	rev	0 prev cur	reverse() break here
-----	-----	---------------	----------------------

main()	After popping, LL changes to
--------	------------------------------

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

Gate 2008

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

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

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

`struct node {`

`int value;`

`struct node *next;`

`};`

`void rearrange(struct node *list)`

`{`

`struct node *p, *q;`

`int temp;`

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

`p = list;`

`q = list->next;`

`while (q)`

`{`

`temp = p->val;`

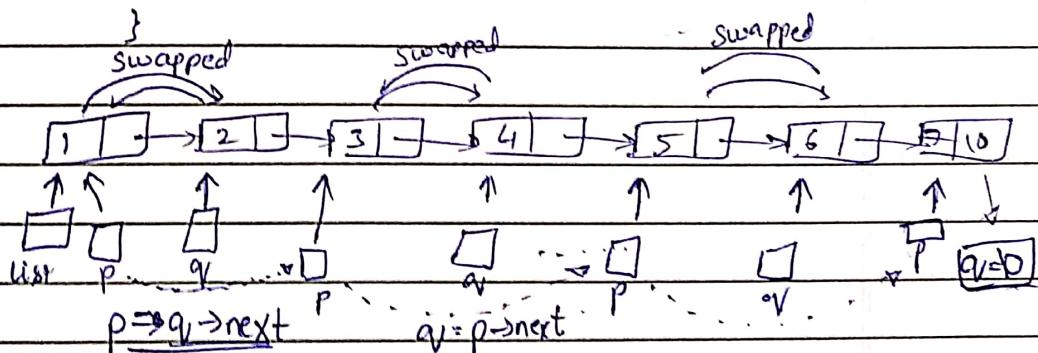
`p->val = q->val;`

`q->val = temp;`

`p = q->next;`

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

`}`



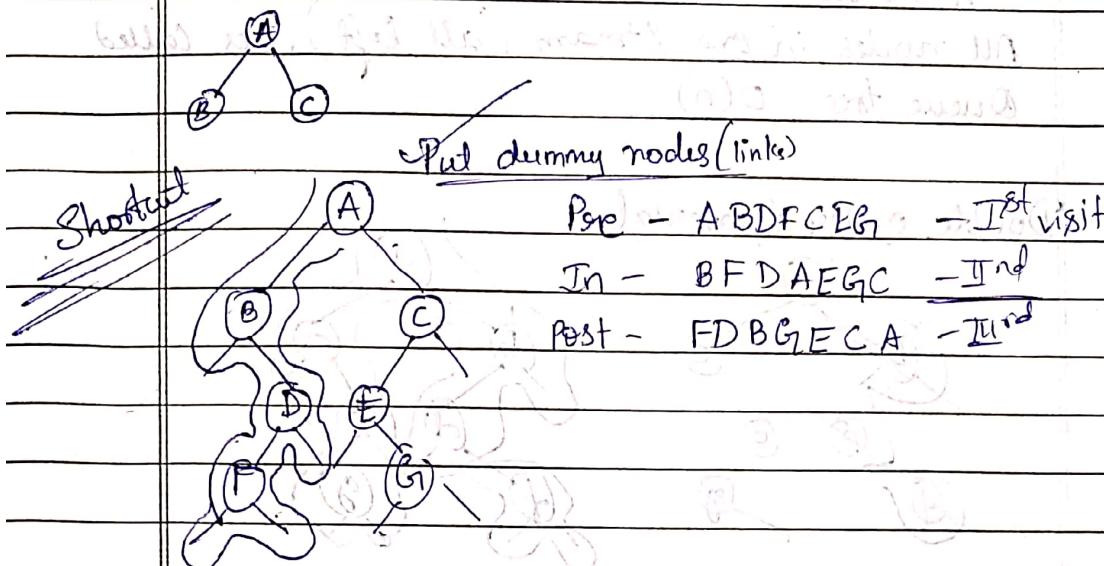
O/P: 2 1 4 3 6 5 7

Gate 2010

The follow

Introduction to tree traversals

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

Implementation of traversals and time and space analysis

struct node

{

int data;

struct node *left, *right;

}



void Inorder(struct node *t)

{

if(t!=NULL) {

if(t)

Inorder(t->left);

printf("%d", t->data);

Inorder(t->right);

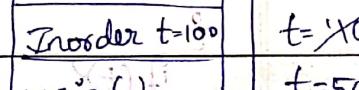
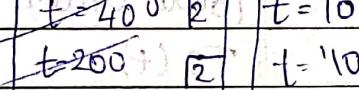
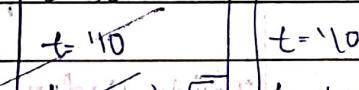
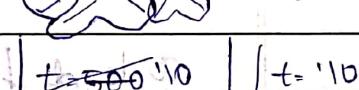
}

points

node during second visit

Op: 400 200 500

d b e a b.



visiting node takes constant time

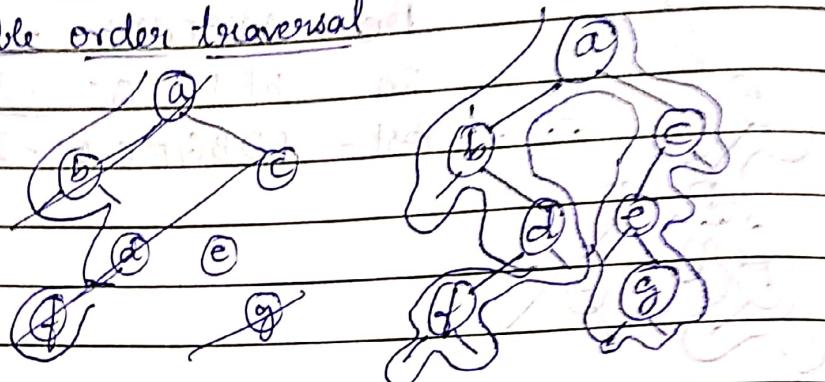
Every node visited 3 times

Time complexity: $O(n)$

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

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

Double order traversal



b f d a e c g o

explains along two unit how traversal is maintained

DO(t)

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

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

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

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

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

(Info in node t is printed)

Triple Order Traversal

TO (t)

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

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

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

traversing prints like this

and so on

and so on

and so on

and so on

$\{$
 Pf("/.d", t->data) \rightarrow a
 TO(t->RC); b c
 Pf("/.d", t->data) d e
 TO(-t->RC)
 Pf("/.d", t->data).
 $\}$

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

✓ Indirect succession on trees

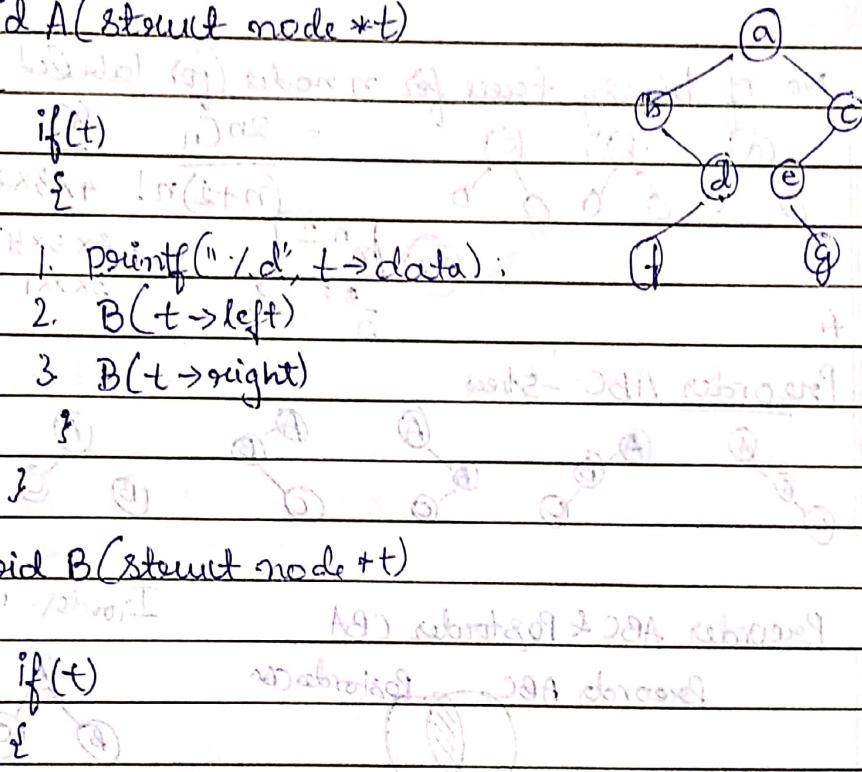
void A(Struct node *t)

if (t)

1. printf("%d", t->data);

2. B($t \rightarrow \text{left}$)

3 B($t \rightarrow$ right)



void B(steuert node + t)

if (+)

1. ~~positive~~: $A(t \rightarrow \text{left})$;

2. `parent("1.d")` about 3 sec

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

三

1750 land grants

1416 part 2

Lamia pionaris TS supine st. - subrotundish stiff

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 and vice versa.

Number of binary trees possible

① Node \Rightarrow No. of binary trees possible

1	1	
2	2	
3	3	
4	4	

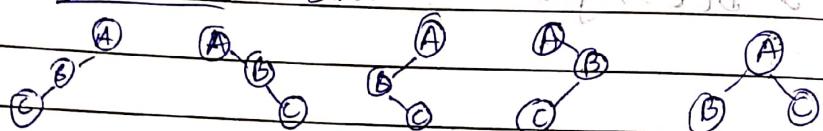
No. of binary trees for n nodes (for unlabeled nodes)

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

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

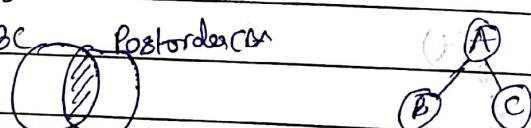
$$\begin{array}{c} \text{Diagram of a binary tree with 5 nodes labeled A, B, C, D, E.} \\ \text{Left side: } = 2n C_n = \frac{6!}{3!3!} = \frac{6!}{3!3!} \\ \text{Right side: } (n+1)n! = 4 \times 3 \times 2 \times 1 = 4! \end{array}$$

Preorder ABC - 5 trees

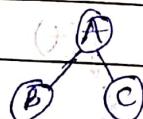


Preorder ABC & Postorder CBA

Preorder ABC Postorder CBA



Inorder - Back Preorder



Inorder BCA & Postorder CBA

(Inorder C) A B

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

Binary tree $\frac{n!}{(n+1)!}$

Preorder & Postorder - NO unique BT or more than

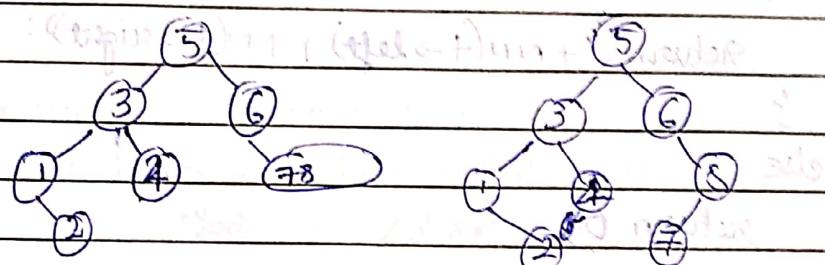
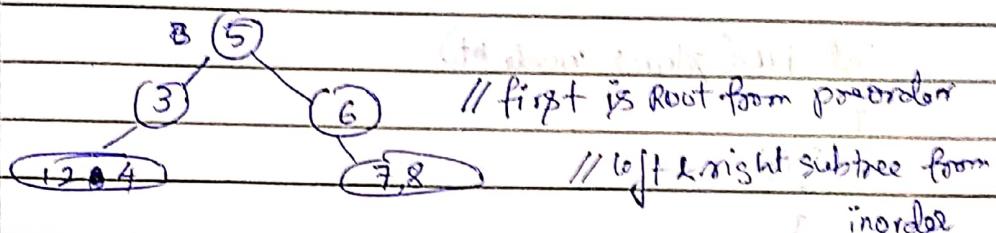
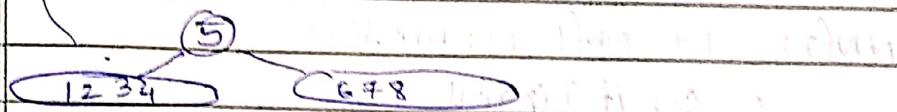
Inorder, Postorder, Preorder \rightarrow Unique BT & exactly one

Construction of unique binary tree

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

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

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



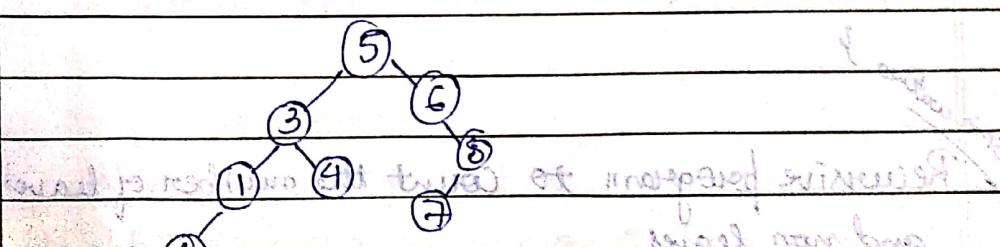
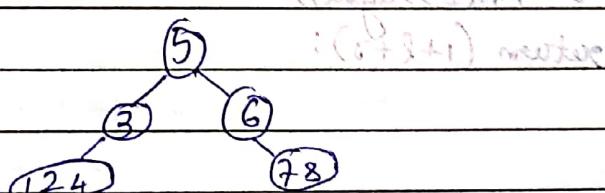
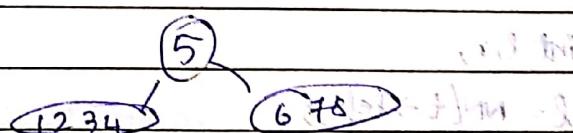
Postorder: 2, 4, 3, 7, 8, 6, 5 // 3rd time

(+ down arrow) last this

② Inorder: 1, 2, 3, 4, 5, 6, 7, 8

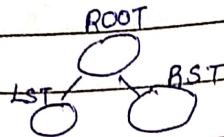
Postorder: 2, 4, 3, 7, 8, 6, 5

(+) ↓



post is not i.e. (1).in

Time watched
Recursive program to count the number of nodes



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

= 0; if T is null

int NN(struct node *t)

{ if (t) { } else { } }

return {

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

}

else { }

 return 0;

}

OR

int NN(struct node *t)

{

if (t)

{ }

int l, r;

l = NN(t->left);

r = NN(t->right);

return (1 + l + r);

}

else

 return 0;

}

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

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

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

(if t is not NULL)

NL (struct node *t); // function declaration

{

```

if (t == NULL)
    return 0; // if t is NULL then
else if (t->left == NULL & t->right == NULL)
    return 1; // if left & right are NULL
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

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

if T has 2 children, then $FN(T) = FN(T \rightarrow LST) + FN(T \rightarrow RST)$

$= FN(T \rightarrow LST) + FN(T \rightarrow RST)$ as, T has 1 child

if T has 1 child then $FN(T \rightarrow LST) + FN(T \rightarrow RST) + 1$

$$0 \leq x \leq 1 - \frac{1}{h}$$

int FN(struct node *t); // main is recursive (A)

(C, O = 1) methods in A is 1 for reduction of work

if ($\neg t$) return 0;

$[1 - \frac{1}{h}] = 0$, next

if ($t \rightarrow \text{left} \neq \text{right}$) then min of C

return 0;

$0 \leq x \leq 1 - \frac{1}{h}$

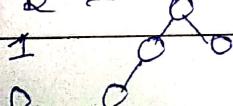
as above if return $FN(t \rightarrow \text{left}) + FN(t \rightarrow \text{right}) + (t \rightarrow \text{left} \neq t \rightarrow \text{right})$

$1 : 0;$

more realistic cost per node is to take minimum of C

Recursive program to find the height of a tree.

Height of any node, to leaf its distance



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

int H(struct node *t)

{

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

int l, r;

l = H(t->left)

r = H(t->right)

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

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

GATEFORM

Binary ~~Search~~ Trees = binary

Properties:-

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

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

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

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

$$\log_2 n$$
- 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) is

$$2^h$$
 or 2^l

GATE 2017

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

$$n = 4$$

$$\text{B}_n = \frac{1}{n+1} \binom{2n}{n} = \frac{2^n C_n}{n+1} = \frac{1}{1, 6!} \times \frac{1}{4+1} \times \frac{8!}{4! \cdot 4!} = \frac{1}{5} \times \frac{8 \times 7 \times 6 \times 5}{4 \times 3 \times 2 \times 1} = 14.$$

GATE FORMUM

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

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

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

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

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

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

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

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

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

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

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

To Root
Root Left Right
Root Left Right Post

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

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

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

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

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

(26) In heap property,

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

LEFT(i) $\rightarrow 2i$

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

(27) Number of comparisons in searching an element in BST is $O(n)$ in worst case in a skewed binary tree.

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

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

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

Sol:-

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

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

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

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

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

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

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

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

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

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

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

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

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

Note: In Preorder, Root is first element

In Postorder, Root is last element

Steps:

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

Ans: Implement min-heap using max-heap
using max-heap will be difficult since max-heap is implemented using min-heap.

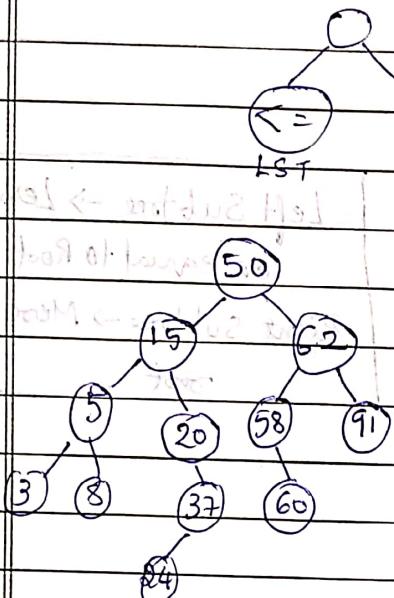
Binary Search Tree

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

GATE 96

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

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



GATE 2005

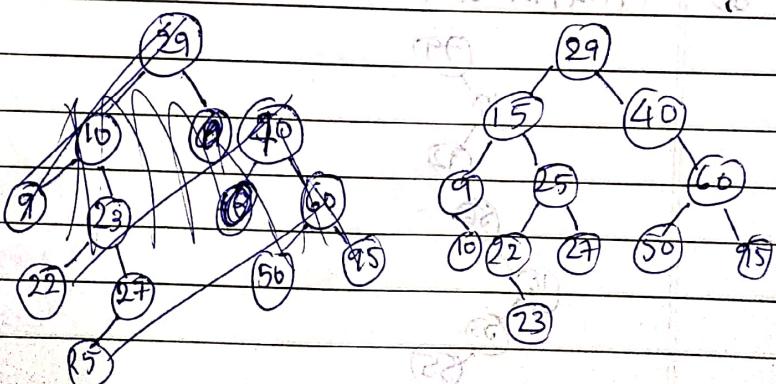
Postorder: 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

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

Preorder:-

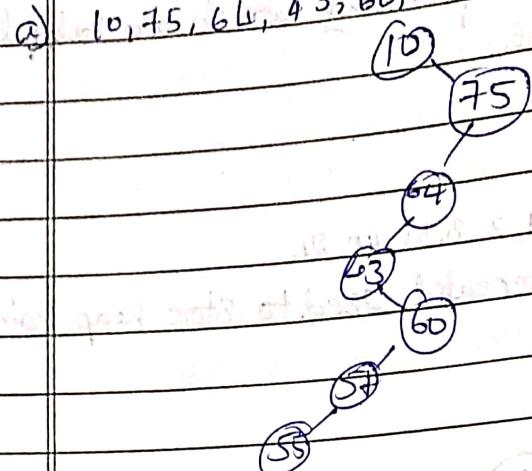
Preorder: 29, 15, 9,
10, 25, 22, 23, 27,
40, 60, 50, 95

9, 10, 22, 23, 25, 27 40, 50, 60, 95

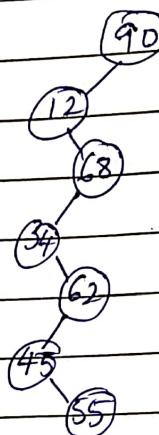


GATE 2006 \rightarrow 55 search

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



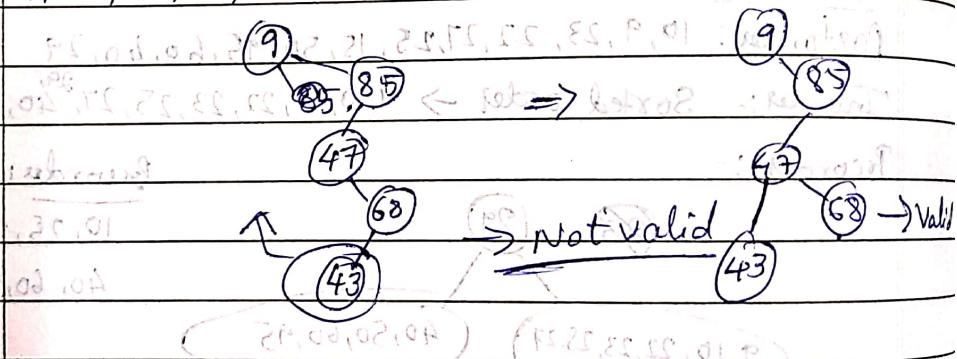
b) 90, 12, 68, 34, 62, 45, 55



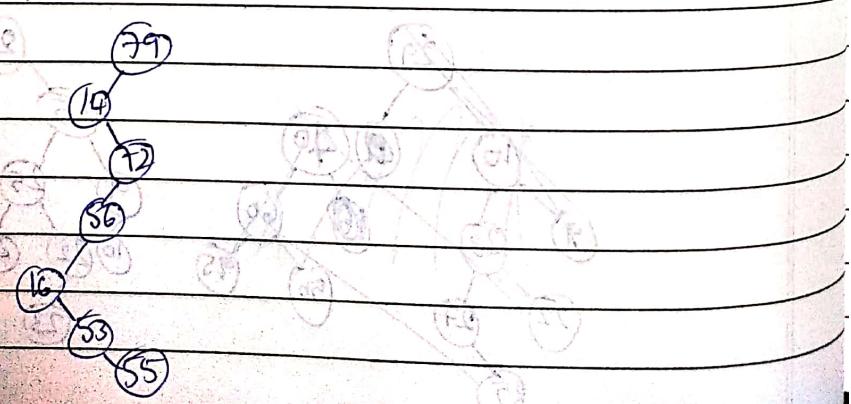
Left Subtree \rightarrow Less than
or equal to Root

Right Subtree \rightarrow More than
Root

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



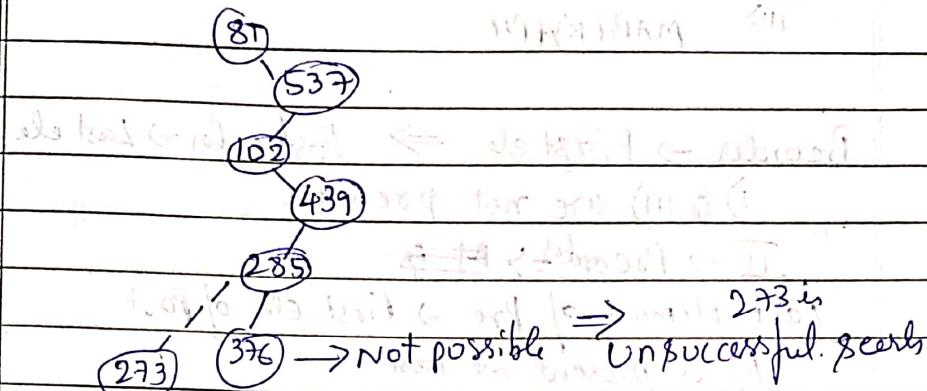
d) 79, 14, 77, 56, 16, 53, 55



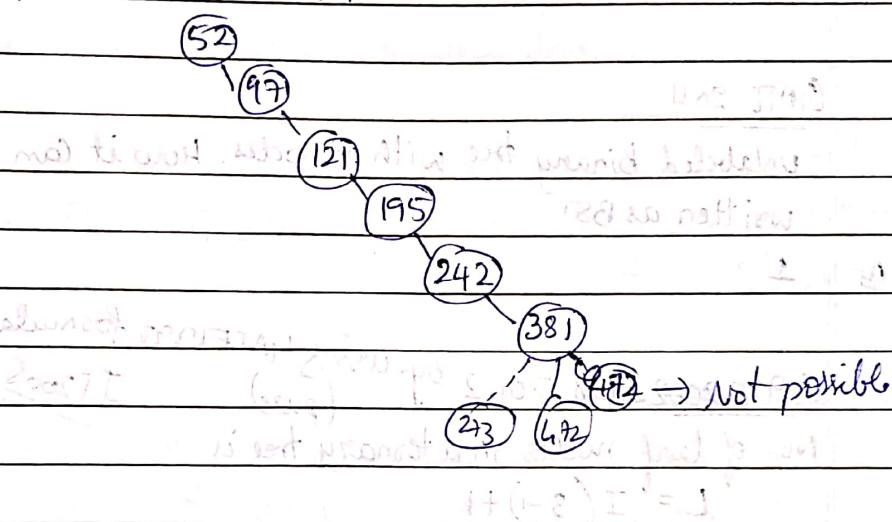
GATE 2008

273 is unsuccessful search

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

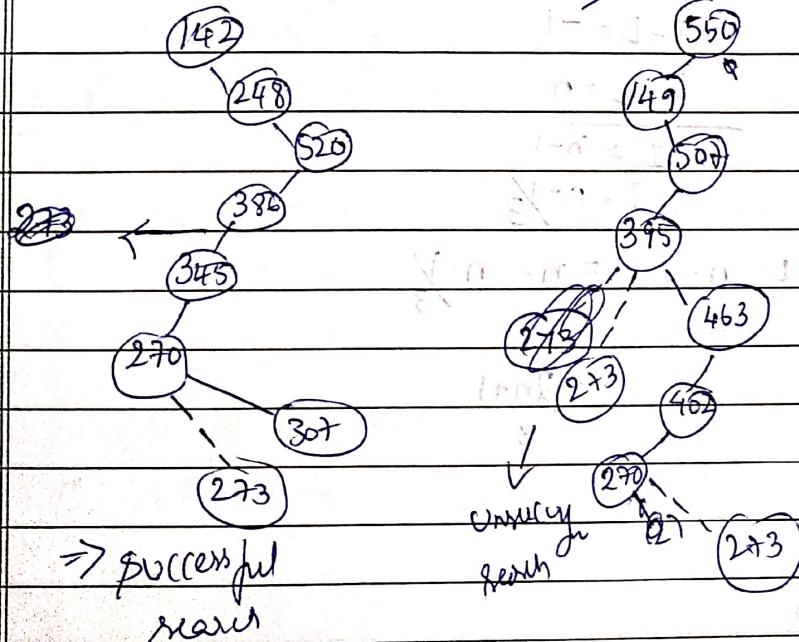


- b) 52, 97, 121, 195, 242, 381, 472



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

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



GATE - 2008

- ① I) MBCAFH PYK
II) KAM BYPFH
III) MABCKYfPH

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

I) & III) are not preorder

II \rightarrow Preorder \Rightarrow ~~M~~ is.

Last element of Pre \rightarrow First ele of Post

K is present at last

I \rightarrow Postorder

III \rightarrow Inorder

GATE 2011

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

n 1

(188) Using GATE 1998 formula

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

No. of leaf nodes in a ternary tree is

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

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

$$2I - L = -1$$

$$I + L = n$$

$$3I = n-1$$

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

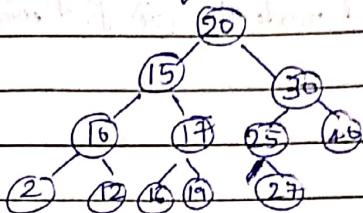
$$L = n - I = n - \frac{n-1}{3}$$

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

(185)

(ERS)

What is \leq
Answer

Delete a Node from BST

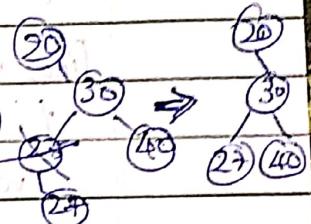
3 types of deletion

① Leaf

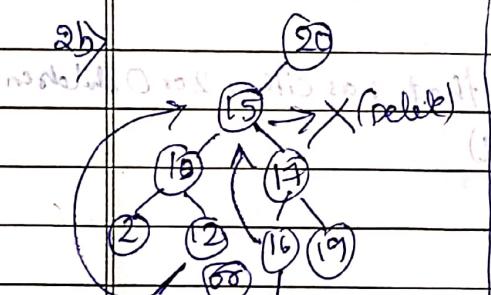
② Non Leaf [a) one child

b) two children

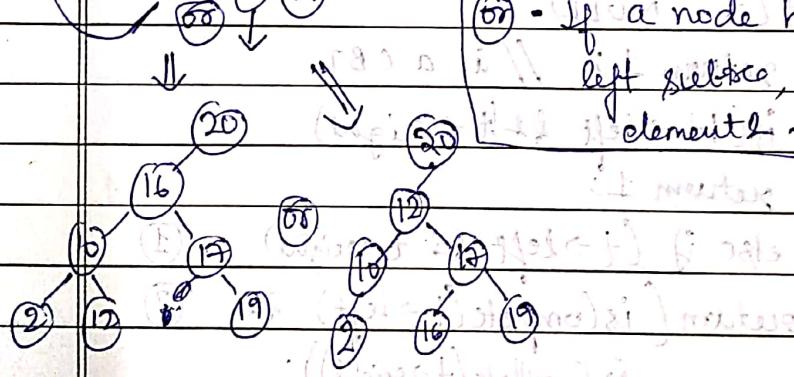
2 a)



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



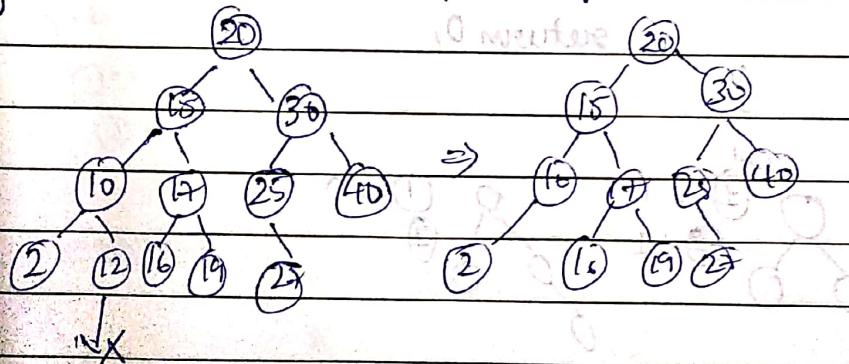
If a node has 2 children, go to right subtree, take least element & replace it in inorder succeed



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

inorder predecessor

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



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

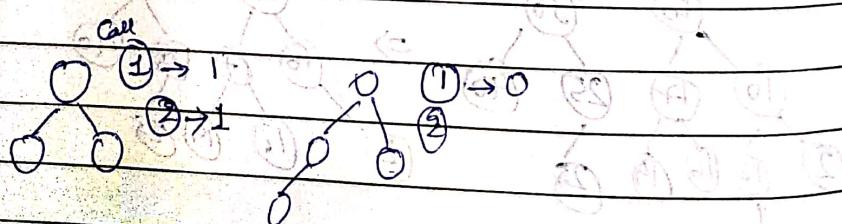
find max (struct node *t)
{
 while ($t \rightarrow \text{right}$)
 $t = t \rightarrow \text{right}$
 }

Recursive program on testing whether a tree is
complete binary tree

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

isComplete (struct node *t)

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



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

1 2 3 4 5

(1)

(2)

(3)

(4)

(5)

→ Here time - $O(n)$ Height - $O(n)$

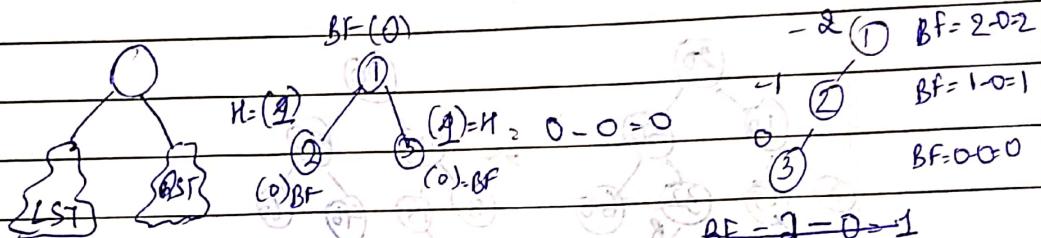
So comes concept of Balancing

AVL - Balanced tree

It is a balanced BST

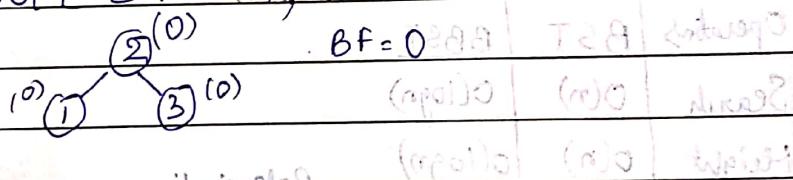
Height Nodes - n , Height - $O(\log n)$ Search time - $O(\log n)$

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

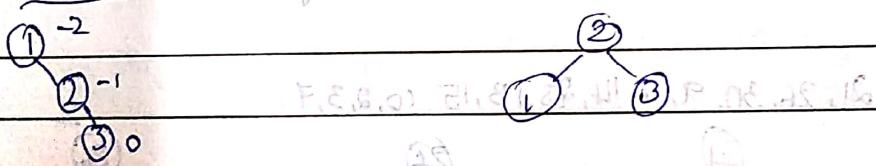


It is called LL imbalance

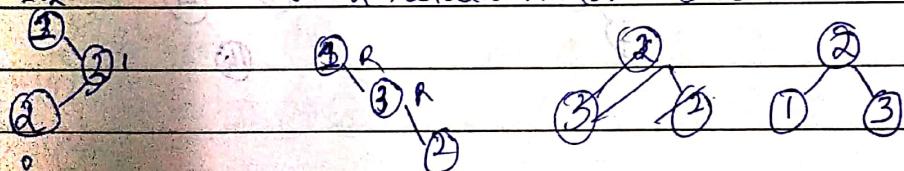
Whenever L-L imbalance, rotate clockwise



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

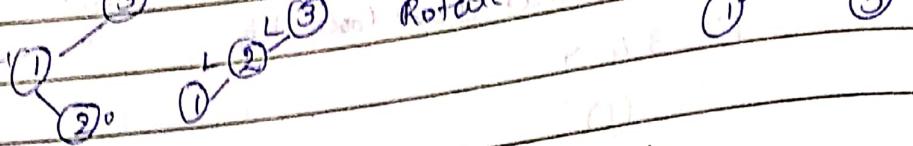
R-L imbalance

Do 2 rotations. Rotate clockwise. Rotate anticlockwise

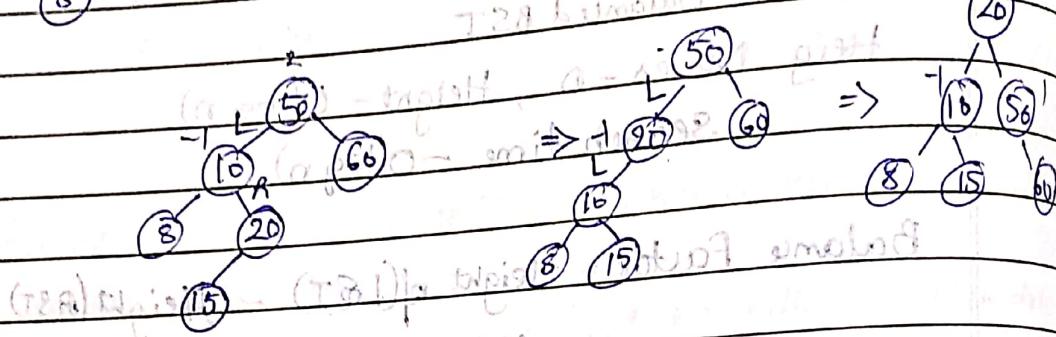
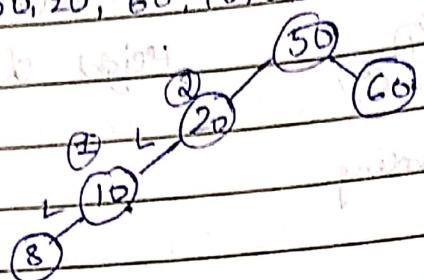


After this, result obtained is Binary Search Tree.

③ 2 This is LR imbalance. Rotate anticlockwise.



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



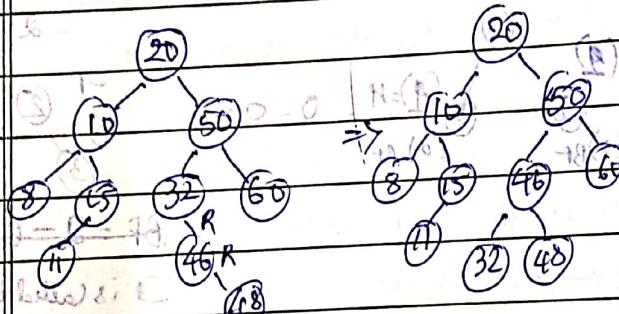
50, 8, 78, 0

100, 114

000, 78

114

78



initially status imbalance 1-1 respectively

Operations BST BBST = 78

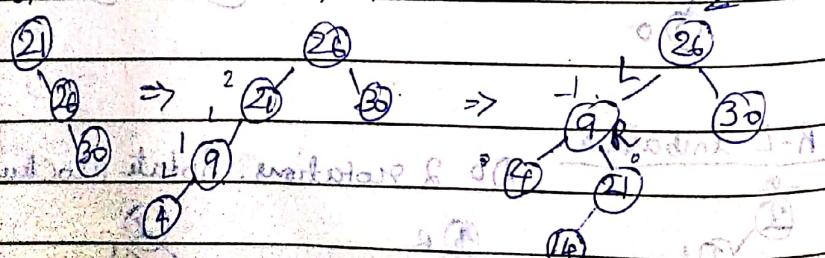
Search $O(n)$ $O(\log n)$

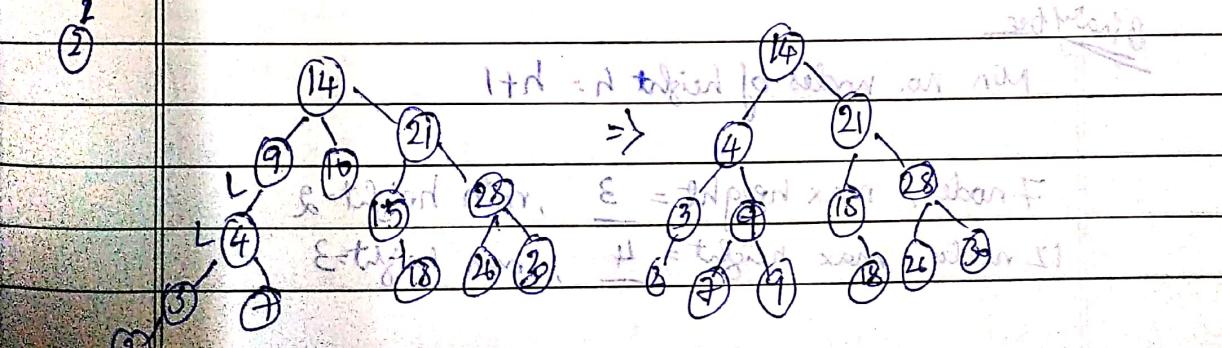
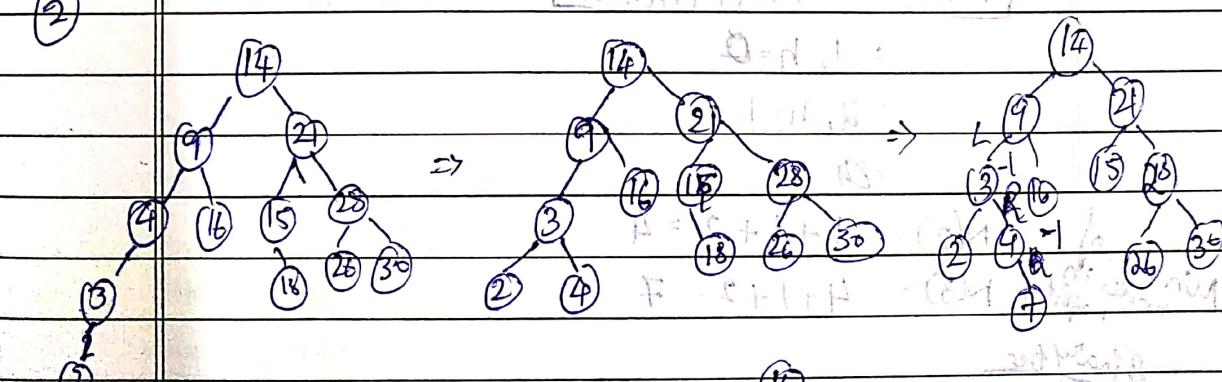
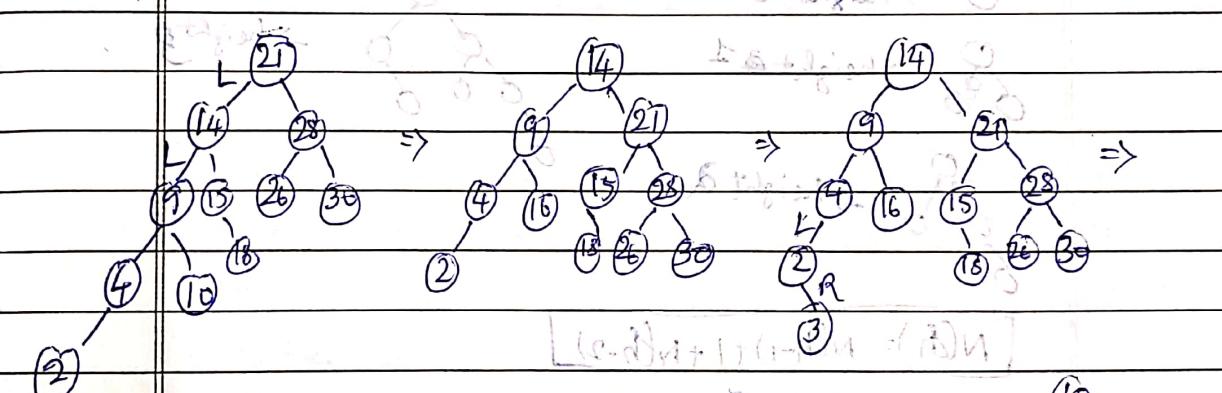
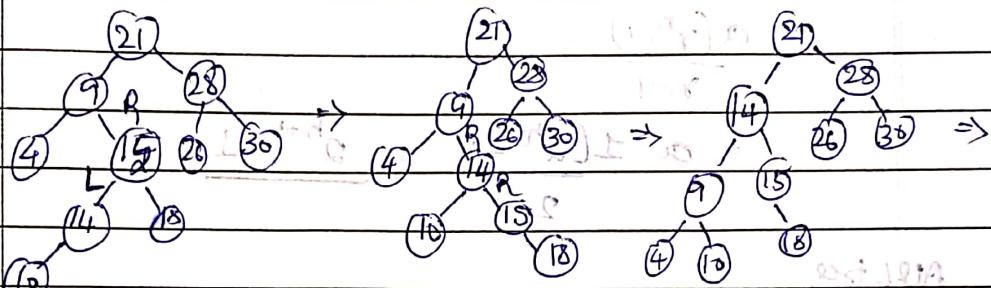
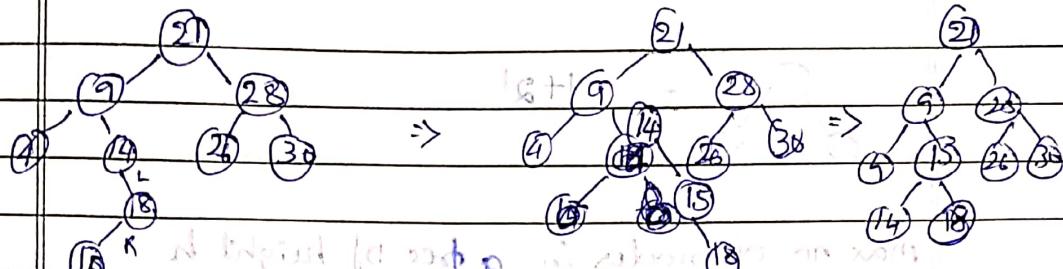
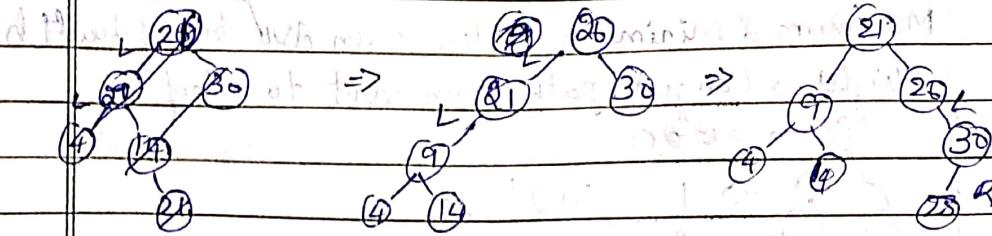
Height $O(n)$ $O(\log n)$

Insertion $O(n)$, $O(\log n)$ + Balancing time

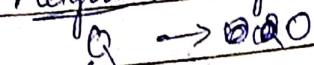
constant $O(1)$

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

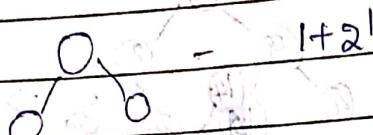




Maximum & Minimum nodes in an AVL tree of height h
Height \rightarrow Longest path from root to a leaf



$$1 \rightarrow 1+2^1+2^2$$
$$0 \quad 0 \quad 0 \rightarrow 2$$



Max no. of nodes in a tree of height h

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

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

$$r-1$$

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

AVL tree

0 \rightarrow height 0

0 \rightarrow height 1

0 \rightarrow height 2

0 \rightarrow height 3

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

$$= 1, h=0$$

$$= 2, h=1$$

$$= 4$$

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

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

Binary tree

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

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

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

GATE 2006

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

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

$$N(1) = 2$$

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

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

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

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

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

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

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

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

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

Ans: Degree of n = Degree of $n+1$

$$10 = 10 + 1$$

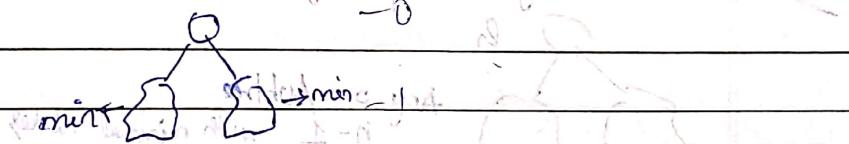
$$\therefore \underline{11}$$

GATE 2005 question paper 2nd year question no. 10. A

GATE 2005 question paper 2nd year question no. 10. A

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

If the height of tree is $h > 0$, then minimum number of nodes in the tree is



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

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

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

$$a) 2^{h-1} = 2^0 = 1 \quad \text{X} \quad \text{B.C. } (1-2) + 1 + (1-2) + 1 - N(2) = 2N(1) - 1 = 2 \times 2 - 1 = 3$$

$$b) 2^{h-1} + 1 = 2^0 + 1 = 2 \quad \checkmark, \quad 2^0 + 1 = 3 \quad \text{X} \quad \text{B.C. } (1-2) + 1 + (1-2) + 1 - N(2) = 2N(1) - 1$$

$$c) 2^{h-1} - 2^0 - 1 = 1 \quad \text{X} \quad \text{B.C. } (1-2) + 1 + (1-2) + 1 - N(2) = 2N(1) - 1$$

$$d) 2^h = 2^1 = 2 \quad \text{V} \quad \text{B.C. } 2^1 = 2 \quad \text{X} \quad \text{B.C. } (1-2) + 1 + (1-2) + 1 - N(2) = 2N(1) - 1$$

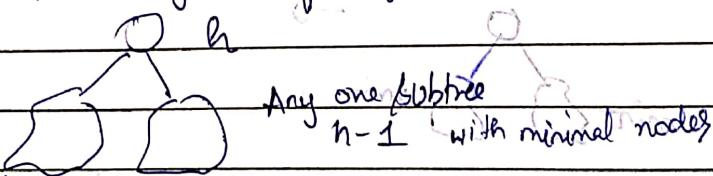
$$= 2^h (n-2) 2^{h-2} = 2^h (n-2) 2^{h-2} \geq 2^3 (n-3) 2^{3-2} = 2^2 (2N(2)-2-1) = 2^2 (2N(2)-2-1)$$

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 at most k . If the height of the tree is $h > 0$, then the minimum number of nodes in the tree is $\frac{1}{2}(k+1)2^h - k + 1$ and the maximum height is $h = \lceil \log_2(k+1) \rceil$.

max dist

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



$$1 - (1-\alpha) \left[\log(n) + \frac{1}{\alpha} \log(n^{-1}) + (1-\alpha)\bar{n} \right] + 1 + (1-\alpha)n = (n)\bar{n}$$

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

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

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

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

$$N(n) = 2^{\lfloor \frac{n}{2} \rfloor} (2N(n-2))$$

$$MC(E) = \frac{1}{2} g^2 m N(h-3) = \frac{g^2}{E} N(h-3)$$

Page 10 of 10

$$= -1.854(t+M)e^{(t+M)^2/2} - 1.854(t-M)e^{-(t-M)^2/2}$$

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

maximum height with 4 nodes

$$A = 2^m$$

(april) 1. Solving $n=2$ (and a quick $n=3$ and $n=4$) tomorrow

and became permanent throughout the world and major oil companies

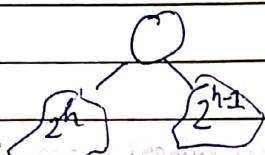
max height with 8 nodes

$g = 2^h$ kg what will the mass be when it reaches the surface?

[$\text{R}=3$] und einige weiteren Ergebnisse, S. 12

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

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



Ans: Abnormal Host's responses will be brief as

do we have $N(h+1) \leq 2^{h+1} + 2^h j + 1$ now let's see what we get

mit Jungs mit 2. 274 mit-well. 17 reppa bzw. 2 reppa

Lebih mudah lagi = 2^{n+1} menterjemah ke dalam bentuk

soil, & so dense amidst other vegetation.

Gate No 14 of bus stand near Amta

① Suppose we have a balanced binary search tree 'T' holding

In numbers, we are given two numbers 'L' and 'H', and

wish to sum up all the numbers in 'T' that lie between 1

Lemma 11. Suppose there are m such numbers in T . If the tightest upper bound time to compute the sum is $O(n^a \log^b n + m^c \log^d m)$, the

Value of $a + 100 + 1000c + 10000d$

100 200

$$100 - 199 \rightarrow \cancel{O(m\log n)} O(m + \log n)$$

$$c=1, d=10, a=0, b=0.1$$

$$= 0 + 10(0) + 1000(1) + 10000(0)$$

$$= 110^{\circ}$$

Other methodInorder $O(n)$

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

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

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

Time (traversal algorithm given below)

Total $(m + \log n)$

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

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

$$= 110$$

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

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

Expression tree

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

Left = 0, Right = 1 = Pre tab

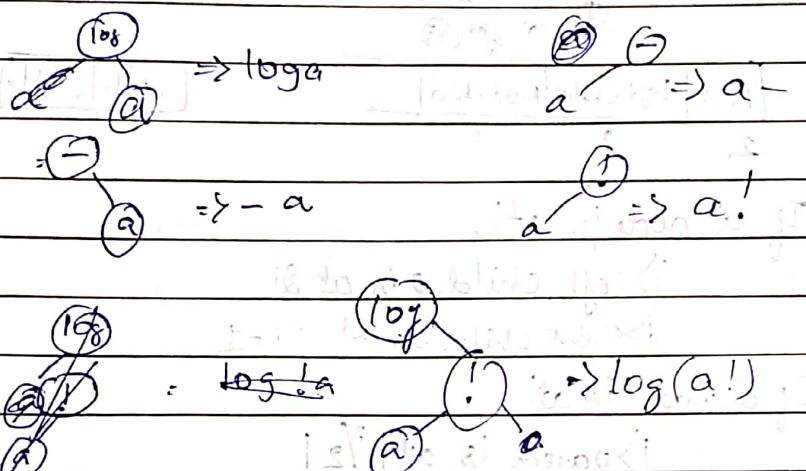
(a) (b) In (left + right) = Post ab+1

$a + b * c$

$$Poe + a^*bc$$

In $a + b \neq c$

Post ~~ab+c*~~ abc++



GATE 2005 question paper + 1 - 70, download free

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

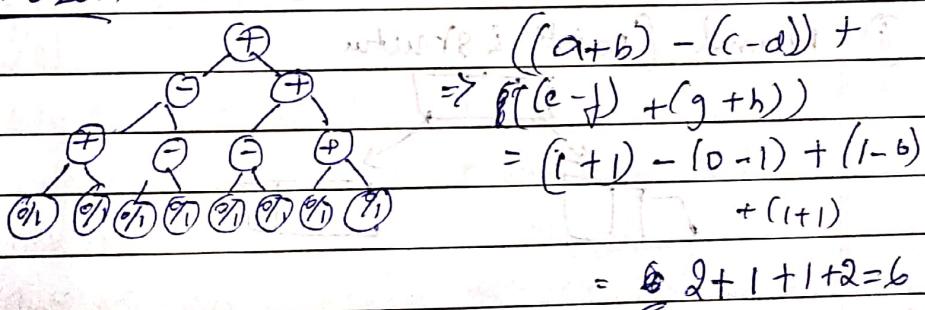
BST - β -element (β & ω) // for BBST & BST

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

$$\text{Root} = n - p(p_1)(p_2)(p_3) \dots (p_k)$$

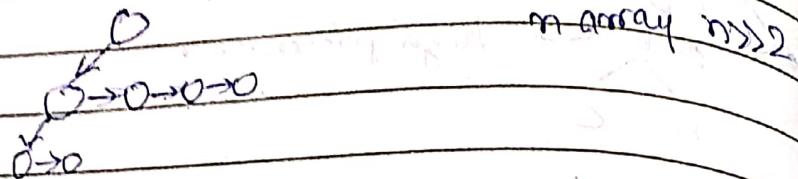
GATE 2014

max value to be found



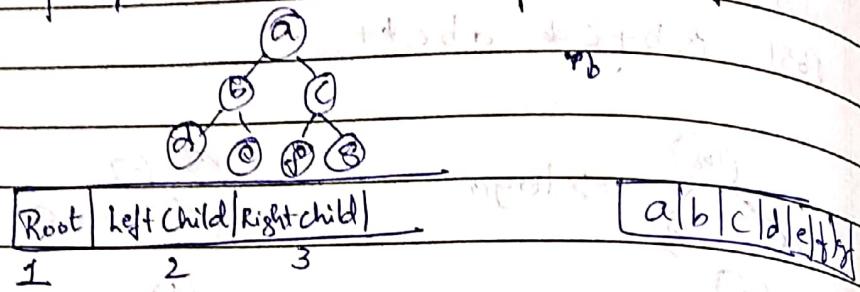
Various representations of a tree

① LCBS \rightarrow Left child Right Sibling



in array $n \gg 2$

② Array representation \rightarrow Mainly for Heaps



If a node is at i ,

i) left child is at $2i$

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

If a node is at i ,

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

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

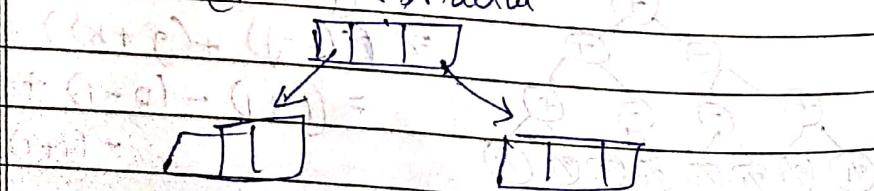
③ Nested representation

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

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

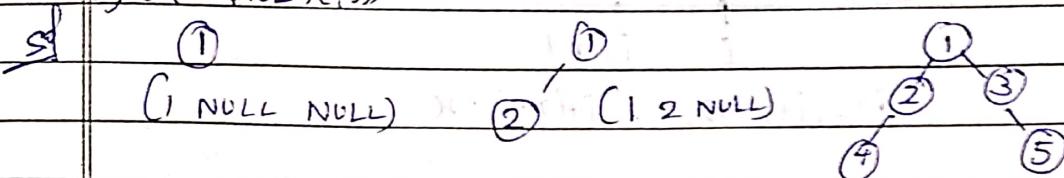
a
b' c
d' c' g
a(b(c))((d)e)(f(g))

④ Normal - Pointer & structure

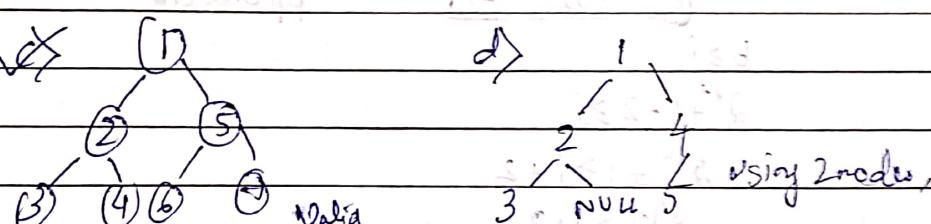
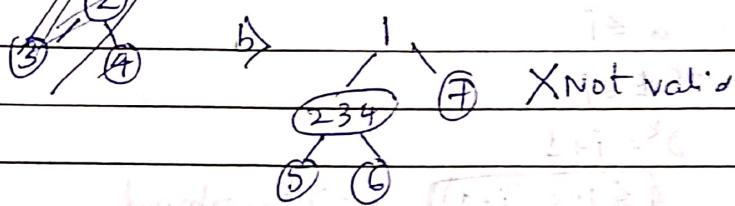


GATE 2000

- Consider the following nested representation of binary tree:
 (xyz) indicates y & z are the left & right subtrees, respectively,
of node x . Note that y & z may be NULL, or further nested.
which of the following represents a valid binary tree?
- $(1_2(4567))$
 - $4((234)56)7$
 - $((1234)(567))$
 - $(1(23\text{NULL})(45))$



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

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.

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

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 $O(n \log n)$.

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

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

Let i at level ' k '

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

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

$$a = 2^k - 1$$


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

$$(a + 2^k - 1) = a + 2^k - 1$$

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

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

$$a \leq i$$

$$2^k \leq i$$

$$2^k = i+1$$

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

$$b \geq i$$

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

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

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

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

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

There can be only one integer b/w these bounds

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

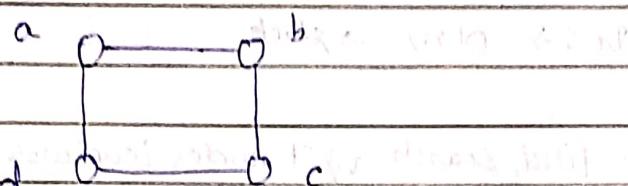
Level

Geaphys

Date _____ Page _____

- World Wide Web is represented as a graph
 - Social Network
 - Person \rightarrow Node, Friend \rightarrow Edge b/w a person & his friend
 - Friends \rightarrow Nodes
 - Webpage \rightarrow Node
 - Links \rightarrow Edge
 - Adjacency List in a graph \rightarrow Friend list in a facebook
 - Mutual Friend suggestion \rightarrow BFS

Representation of Graphs



Adjacency matrix

Adjacency List → useful for large/dense graphs

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

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

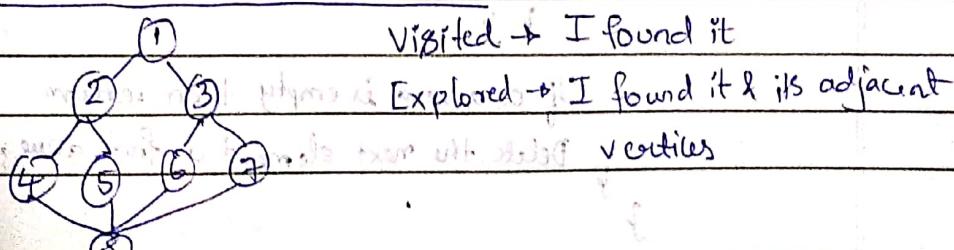
No. of elements = $2 \times E$ in undirected graph in Adjacency List

$\Rightarrow E$ in directed graph

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

$O(V^2) \rightarrow$ Adjacency matrix

Introduction to BFS and DFS



Visited Boolean array Both BFS & DFS uses

0 → Not visited array to keep track of

1 → Visited visited nodes

DFS algorithm keeps track of unexplored nodes
using stack

BFS algorithm

using queue

Space complexity → at least $O(n)$

BFS → $O(n)$ → queue

DFS → $O(n)$ → stack

Breadth first search visit nodes levelwise (horizontally)

Depth first search visit nodes depthwise (vertically)

BFS algorithm

// BFS(v)

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

visited[] is initialized to 0

graph G;

u = v;

repeat

{

 for all vertices w adjacent to u do

 if (visited[w] == 0) {

 start printing (v, w)

 add w to queue;

 visited[w] = 1;

 }

 }

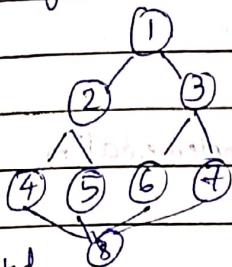
 if (queue is empty) {

 return;

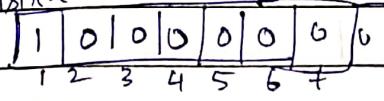
 Delete the next element, u from queue;

}

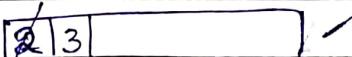
Tracing



Visited

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

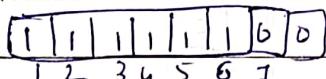
Queue



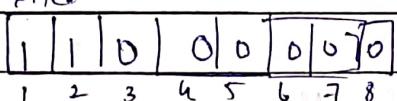
(u) <= t [8] = 1

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

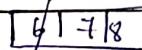
Visited



Visited



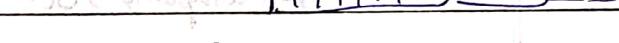
Queue

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

Queue

 $u=2$

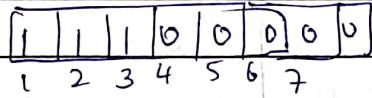
(v) <= t [8] = 1

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

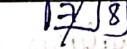
Queue



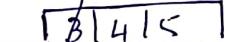
Visited



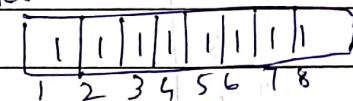
Queue

 $u=7$ $w = \{3, 8\}$

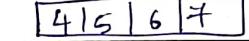
Queue

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

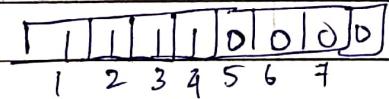
Return Visited



Queue



Visited

 $w = \{4, 5, 6, 7\}$

Orders of visit

(1) 9, 2, 6, 7, 4, 5, 8

(2) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

we can get more than 1 orders

Queue

 $u=4$ $v = \{2, 8\}$

Visited



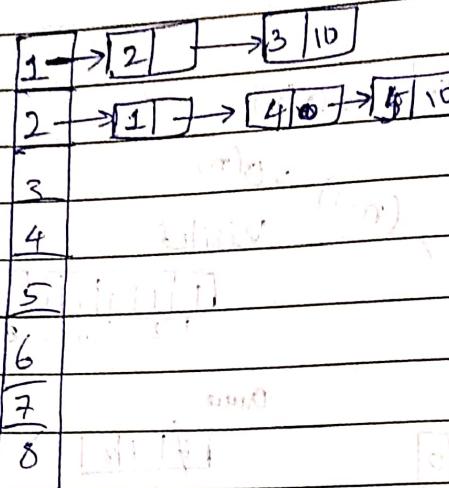
Queue

 $u=5$ $v = \{2, 8\}$

BFS analysis in case of linked list

Space complexity = $O(n)$

Time complexity - depends on implementation



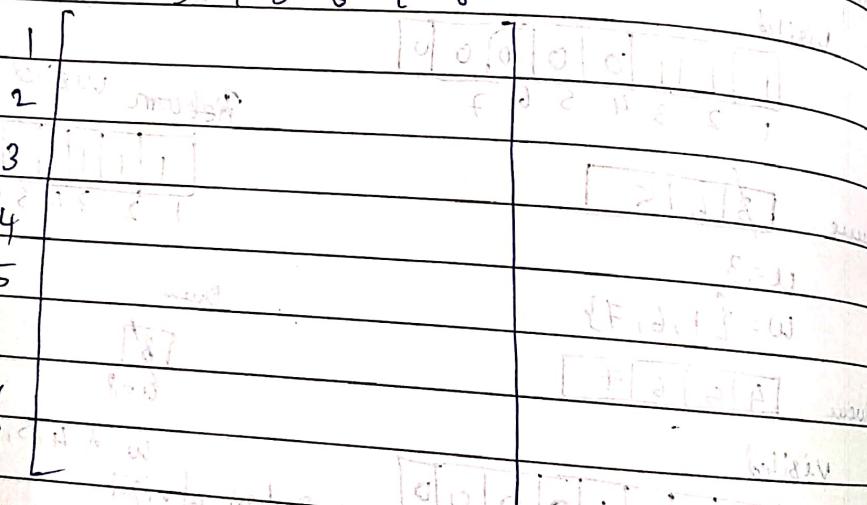
$O(2E) \rightarrow$ undirected graph

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

Space complexity $\rightarrow O(v)$

BFS analysis in case of Adjacency matrix

1 2 3 4 5 6 7 8



Space complexity $\rightarrow O(v)$

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

$\rightarrow O(v^2)$

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

Breadth First Traversal using BFS

BFT(G, n)

{

for $i=1$ to n do

visited[i] = 0;

for $i=1$ to n do

if (visited[i] = 0)

then BFS(i);

}

1

2

0

5

0

3

4

0

6

7

visited [] 0 0 0 0 0 0 0

1 2 3 4 5 6 7

visited [] 1 1 1 1 0 0 0

call BFS(5)

visited [] 1 1 1 1 1 1 1

Time complexity - $O(EV)$

Space complexity - $O(V)$

DPS algorithm

DFS(v)

{

visited[v] = 1;

for each vertex adj to v do

{

if (visited[w] = 0) then

DFS(w)

}

}

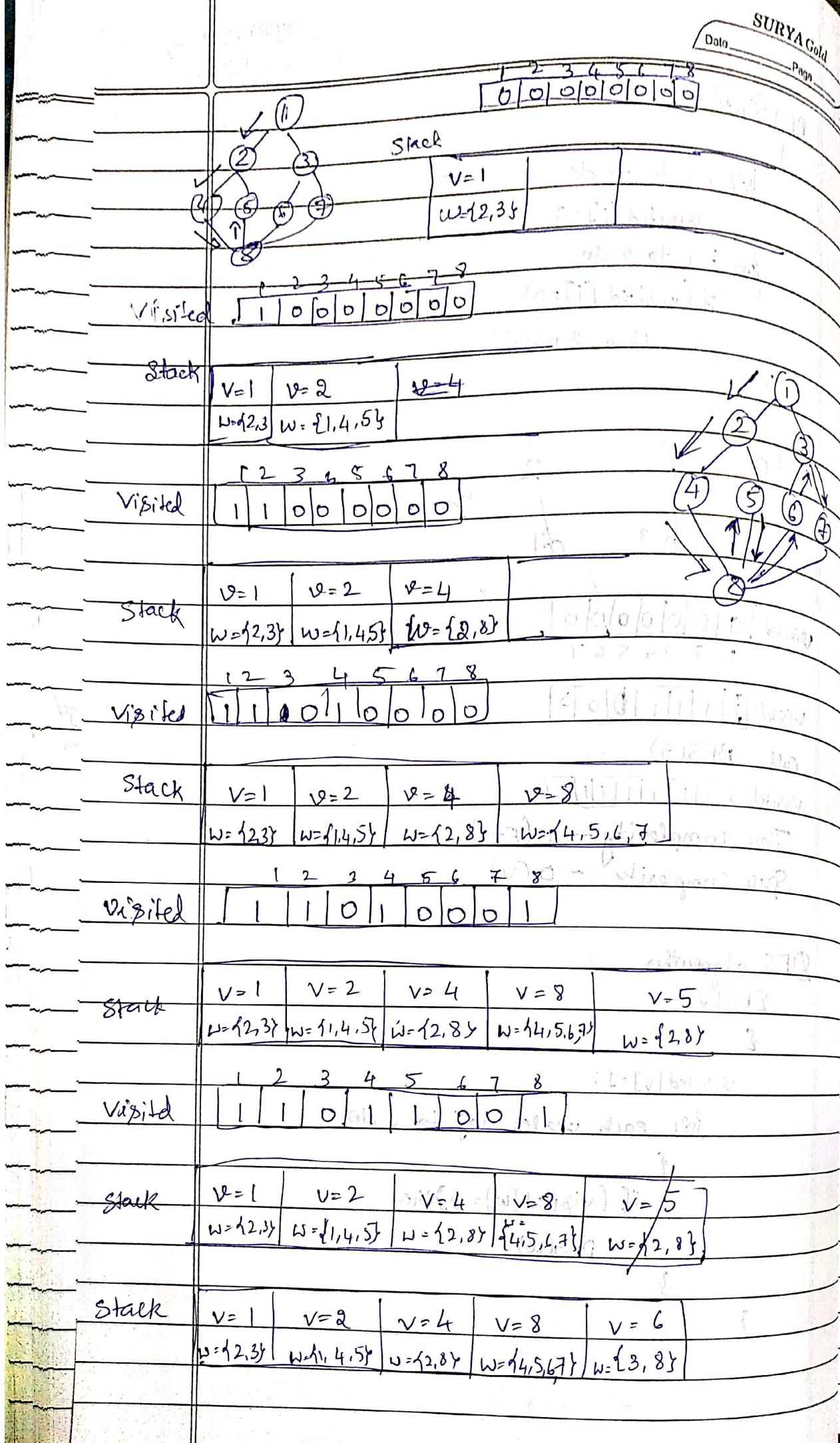
}

}

}

}

}



Visited

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

Stack

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

Visited

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

Stack

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

Visited

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

Stack

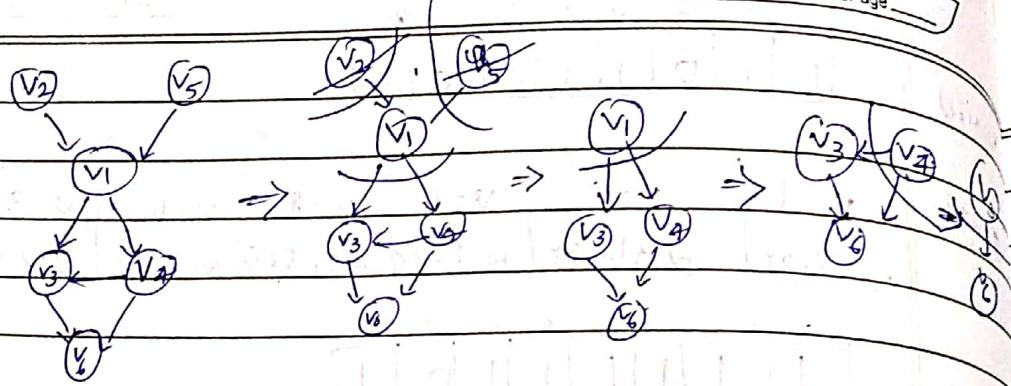
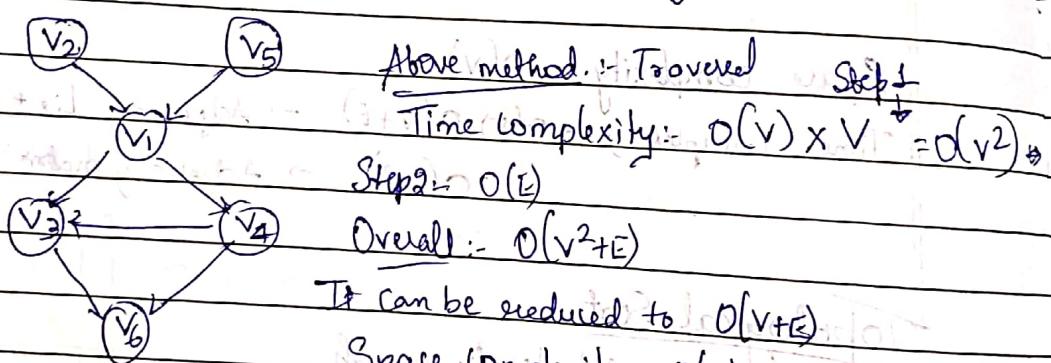
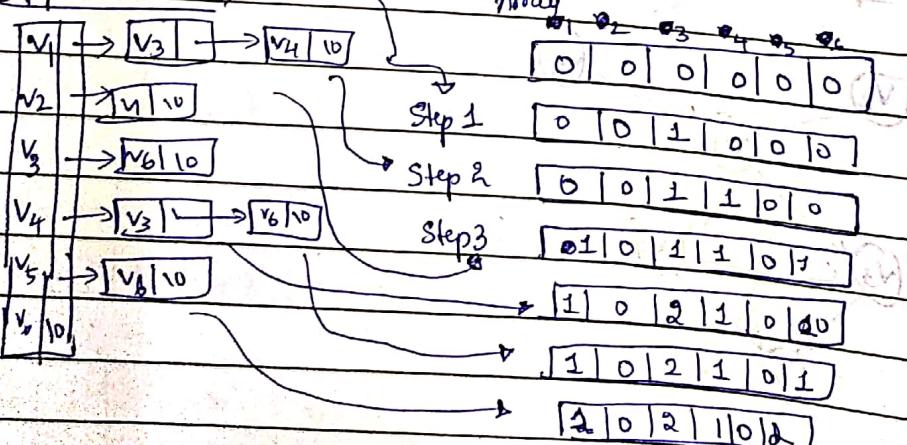
$v=1$	$v=2$	$v=4$	$v=8$	$v=6$	$v=3$	$v=7$
$w=\{2, 3\}$	$w=\{1, 4, 5\}$	$w=\{2, 8\}$	$w=\{4, 5, 6, 7\}$	$w=\{3, 8\}$	$w=\{1, 6, 7\}$	$w=\{3, 8\}$

① ⑥ ④ ③ ⑤ ② ⑦ ① Popping order

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

Input: Directed Acyclic graph

Output: $i \rightarrow j$, j should be first in order i.e., sorted① V_2 V_1 V_3 V_7 V_6 V_5 V_4 V_8 V_9 V_{10} V_{11} V_{12} V_{13} V_{14} V_{15} V_{16} V_{17} V_{18} V_{19} V_{20} V_{21} V_{22} V_{23} V_{24} V_{25} V_{26} V_{27} V_{28} V_{29} V_{30} V_{31} V_{32} V_{33} V_{34} V_{35} V_{36} V_{37} V_{38} V_{39} V_{40} V_{41} V_{42} V_{43} V_{44} V_{45} V_{46} V_{47} V_{48} V_{49} V_{50} V_{51} V_{52} V_{53} V_{54} V_{55} V_{56} V_{57} V_{58} V_{59} V_{60} V_{61} V_{62} V_{63} V_{64} V_{65} V_{66} V_{67} V_{68} V_{69} V_{70} V_{71} V_{72} V_{73} V_{74} V_{75} V_{76} V_{77} V_{78} V_{79} V_{80} V_{81} V_{82} V_{83} V_{84} V_{85} V_{86} V_{87} V_{88} V_{89} V_{90} V_{91} V_{92} V_{93} V_{94} V_{95} V_{96} V_{97} V_{98} V_{99} V_{100} V_{101} V_{102} V_{103} V_{104} V_{105} V_{106} V_{107} V_{108} V_{109} V_{110} V_{111} V_{112} V_{113} V_{114} V_{115} V_{116} V_{117} V_{118} V_{119} V_{120} V_{121} V_{122} V_{123} V_{124} V_{125} V_{126} V_{127} V_{128} V_{129} V_{130} V_{131} V_{132} V_{133} V_{134} V_{135} V_{136} V_{137} V_{138} V_{139} V_{140} V_{141} V_{142} V_{143} V_{144} V_{145} V_{146} V_{147} V_{148} V_{149} V_{150} V_{151} V_{152} V_{153} V_{154} V_{155} V_{156} V_{157} V_{158} V_{159} V_{160} V_{161} V_{162} V_{163} V_{164} V_{165} V_{166} V_{167} V_{168} V_{169} V_{170} V_{171} V_{172} V_{173} V_{174} V_{175} V_{176} V_{177} V_{178} V_{179} V_{180} V_{181} V_{182} V_{183} V_{184} V_{185} V_{186} V_{187} V_{188} V_{189} V_{190} V_{191} V_{192} V_{193} V_{194} V_{195} V_{196} V_{197} V_{198} V_{199} V_{200} V_{201} V_{202} V_{203} V_{204} V_{205} V_{206} V_{207} V_{208} V_{209} V_{210} V_{211} V_{212} V_{213} V_{214} V_{215} V_{216} V_{217} V_{218} V_{219} V_{2

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

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

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

 v_2

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

 v_2, v_3

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

 v_2, v_5, v_1

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

 v_2, v_5, v_1, v_4

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

 v_2, v_5, v_1, v_4, v_3

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

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

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

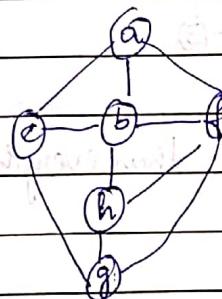
Time complexity $\rightarrow O(V+E) \rightarrow$ Topological sort.

$$= O(V^2)$$

(1)

GATE 2003

DFS on



I) abeghf

II) abfegh

III) abfghed

IV) afghabe

So I, III, IV

are DFS

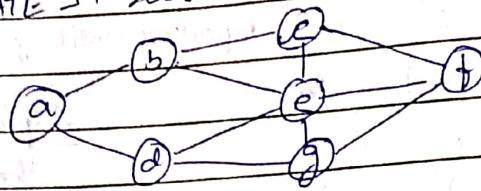
I)	$v=a$ $w=f, b, h, g$	$v=b$ $w=a, e, f, h$	$v=c$ $w=a, g, b$	$v=d$ $w=e, h, f$	$v=e$ $w=d, b, f, g$	$v=f$ $w=d, b, f, g$
----	-------------------------	-------------------------	----------------------	----------------------	-------------------------	-------------------------

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

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

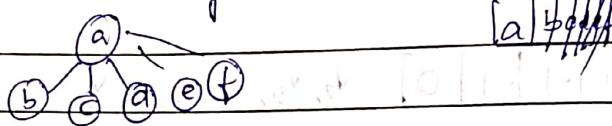
(2)

GATE IT 2008



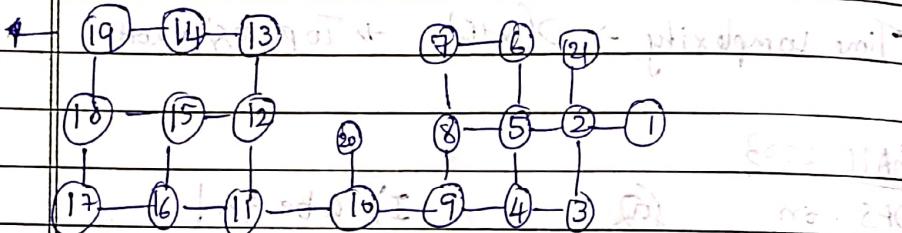
(direct)

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

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

(3)

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



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

(4)

GATE 2006

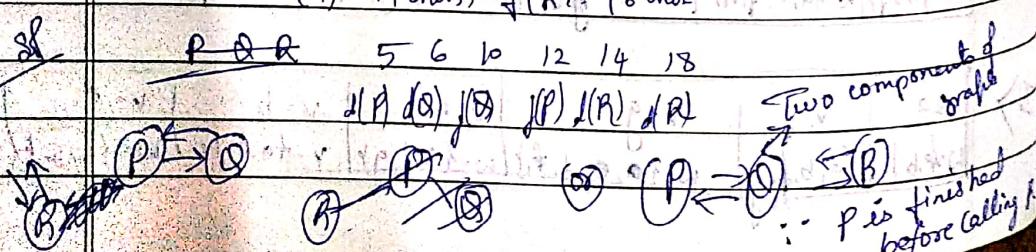
Consider a DFS of an undirected graph with 3 vertices P, Q, R. Let discover time $d(u)$ represent the time instant when the vertex 'u' is first visited and finish time $f(u)$ represent the time instant when vertex 'u' is last visited. Given that $d(P)=5$ units, $f(P)=12$ units.

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

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

P Q R 5 6 10 12 14 18

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



∴ P is finished before calling R

Hashing

SURYA Gold

Date _____

Page _____

Searching (Worst case)

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

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

Direct Address Table

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

We can store it in array

Separate arrays for record & key to be maintained

Fails when values are large

Introduction to Hashing

→ No restriction on no. of elements

Requires more space since we maintain a linked list

Open Addressing

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

Chaining → Deletions easier

OA → Insertion, Searching

In insertion, deletion, searching - constant time in hashing

Chaining → Worst Case - $O(n)$

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

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

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

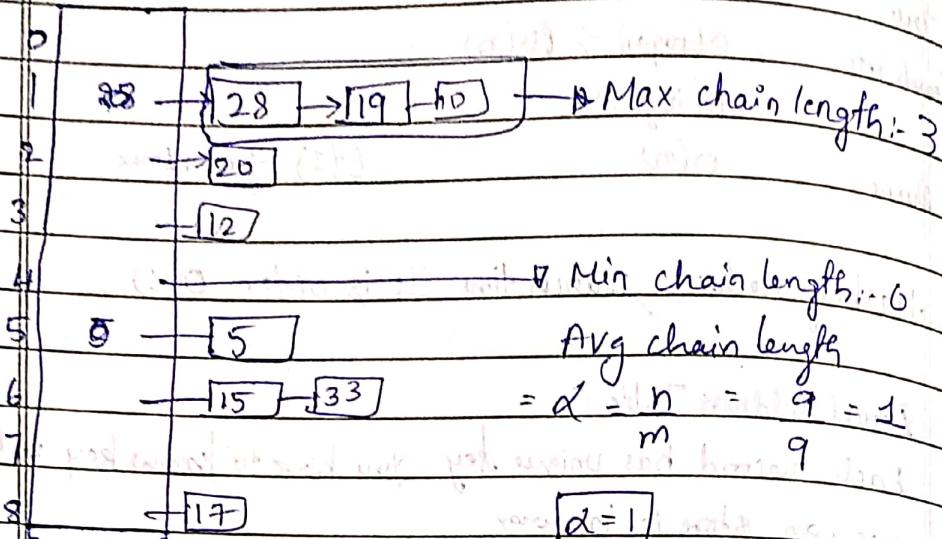
$$\alpha = K_1$$

GATE 2014

(1)

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

Sol:



(2)

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

Sol:

$$100 - 3 = 97$$

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

(3)

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

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

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

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

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

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

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

Open Addressing

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

$O(1) \rightarrow$ Average case

Deletions is troublesome

$$h(k) = (a + i \cdot m + j) \mod m \quad (i, j \in \{0, 1, 2, \dots, m-1\})$$

Linear Probing

0	$h(k) : U \rightarrow \{0, \dots, m-1\}$
1	$h(k) = a_2 + i_1 \cdot m + j_1 \mod m$
$a+1$	$h'(k, 1) = (h(k) + 1) \mod m$
$a+2$	$h'(k, 2) = (h(k) + 2) \mod m$
$m-1$	$h'(k, m-1) = (h(k) + m-1) \mod m$

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

$$h(k) = a_2 + i_1 \cdot m + j_1 \mod m$$

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

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

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

$$= (a + 2) \mod m$$

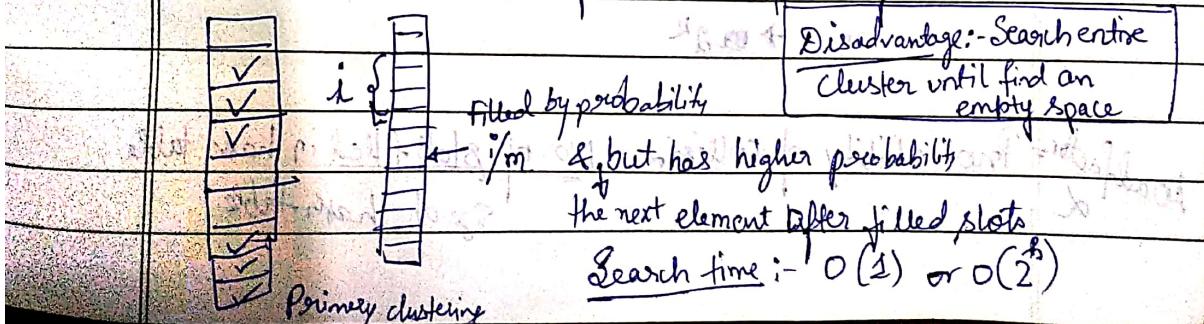
We search till we get free slot

$i = \{0, 1, 2, \dots, m-1\}$ provided $i < m$

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

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

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



Quadratic Probing

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

Next probe position is quadratic

Probe position increases quadratically

Avoids ① Primary clustering

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

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

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

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

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

$$= 3 \quad = 27$$

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

$$= 6 \quad = 37$$

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

$$= 11 \quad = 51$$

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

$$= 18 \quad = 66$$

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

Problem :- Whole (Entire) table is not examined

Widely choose c_1, c_2, m

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

in form $(s + t)$.

Double Hashing

Avoids ① Primary clustering ② Secondary clustering

Successive probes depends on keys and does not

follow same sequence if initial probe position is same

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

Choose m, n are relatively prime

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

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

Load factor
 α

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

Size of hash table

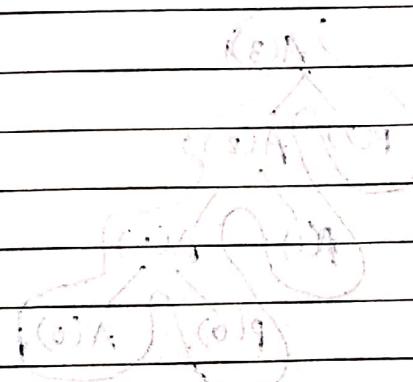
- ① Consider a hash table that distributes keys uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5

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

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

0		52, 23, 34, 46
1		42, 23, 34, 52, 33, 46
2	42	4 elements 42, 23, 34, 46 are in their position
3	23	2 elements 52, 33 are probed
4	34	(42, 23, 34) 52 33, 46 → 5 ways
5	52	(42, 23, 34) 52 33, 46 → 3! ways
6	46	46 → 5 ways
7	33	42, 23, 34 → 3! ways
8		$3! \times 5 = 6 \times 5 = 30$ ways
9		

$$\text{Total no. of ways} = (1 + 5 + 7 + 3!) \times 3! = (1 + 5 + 7 + 6) \times 6 = 29 \times 6 = 174$$



174 ways exist to insert

174 ways exist to insert

Recursion

```

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

```

Activation record is created on calling each function in stack

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

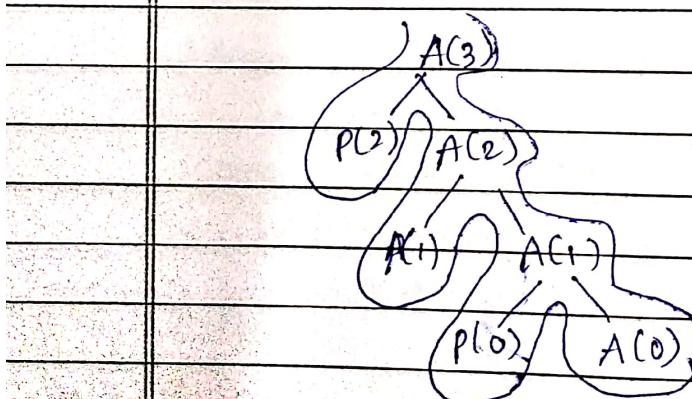
A(0) returns without doing anything. Then A(1) is called. We have to go to line no. 5 & same continues.

main() is called & gets finished.

Space complexity - Stack records = $n+1$

Space = $O(n)$

T(n) = C + T(n-1) Activation records are created
 $= O(n)$



Depth of tree = $n+1$

= Max length of stack records

(2)

 $A(n)$

{

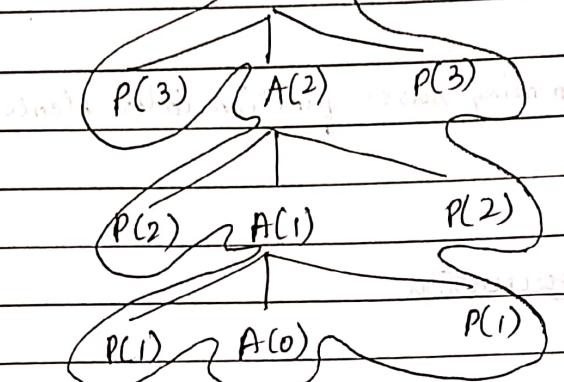
if ($n > 0$)

{

 $P_f(n);$ $A(n-1);$ $P_d(n);$

}

}

 $A(3)$ 

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

$$= O(n)$$

O/p :- 3 2 1 1 2 3

$$\text{Stack records} = n+1$$

$$= O(n)$$

(3) $A(n)$

{

if ($n > 0$)

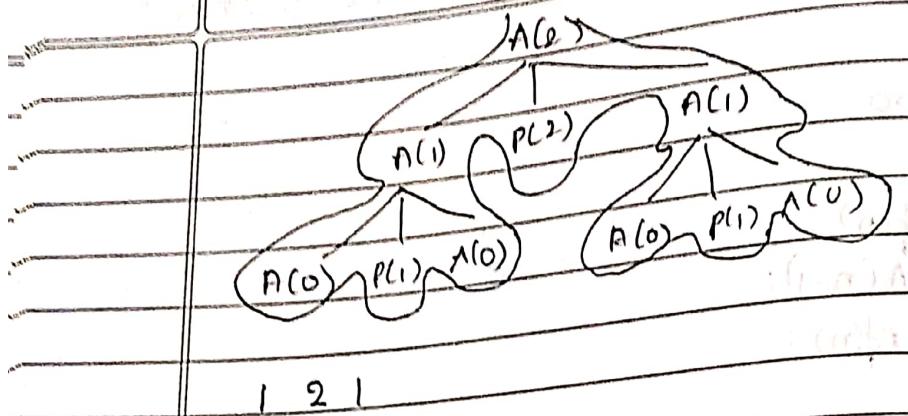
{

 $A(n-1);$ $P_f(n);$ 2 + $A(n-1)(a);$

}

}

2 + $(n-1) \cdot 2 + (n-1)^2;$ 2 + $(n-1) \cdot 2 + (n-1)^2;$



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

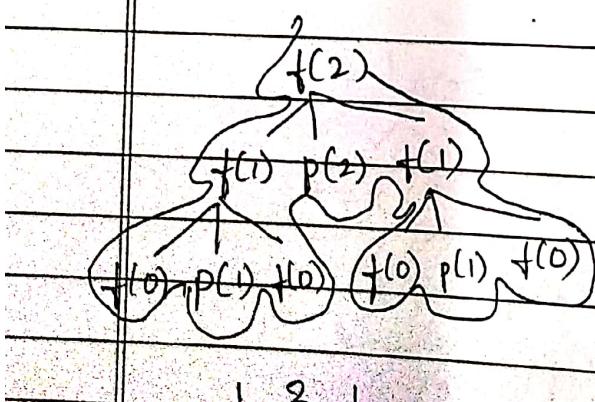
No. of stack records = $n+1$
 $= 2+1$
 $= 3$

Dynamic programming saves function calls! Hence saving time

Analysing the recursion

```

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



No. of function call

$$f(1) = 3$$

$$f(2) = 7$$

$$f(3) = 27$$

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

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

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

1 - already calling

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

It is in Geometric Progression

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

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

Time Complexity =

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

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

$$\Rightarrow O(2^n)$$

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

Gate 2001

void abc(char *s)

{

if (*s == '\0') return;

abc(s+1);

abc(s+1);

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

}

main()

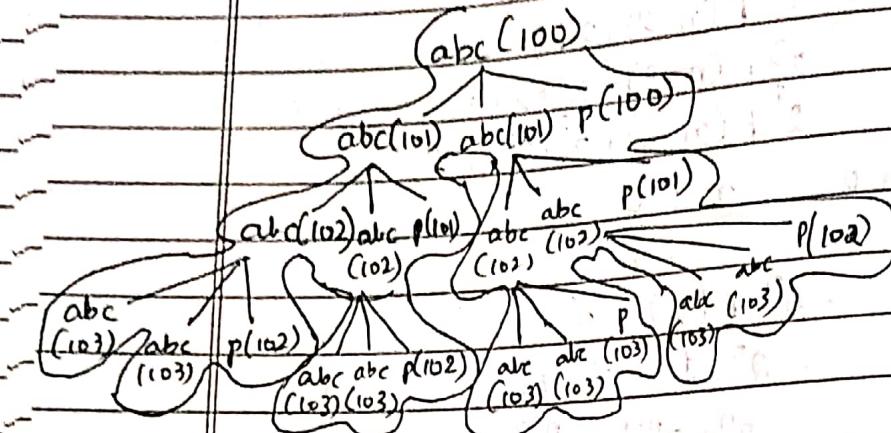
abc("123");

}

a)

'In memory it is stored as

	1	1	2	3	10
	100	101	102	103	

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

$$c(n) = \text{no. of characters printed by } abc(s) \text{ giving input of } n \text{ characters}$$

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

Just above use back substitution method

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

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

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

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

$$= \frac{2^{n-1} (2^n - 1)}{2-1} \quad (\text{using formula for sum of GP})$$

$$\frac{1}{2-1} (2^{n-1} (2^n - 1)) \quad (\text{canceling common factor})$$

$$c(n) = 2^n - 1 \quad (\text{canceling common factor})$$

(cross multiply with 2)

GATE 2004

int suc(int n)

{

if ($n == 1$)

return 1;

else

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

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

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

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

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

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

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

Ques 2

gate 2005

void foo(int n, int sum)

{

 int k=0, j=0;

 if (n==0) return

 k=n%10, j=n/10;

 sum = sum+k;

 foo(j, sum)

 printf("%d", k);

}

int main()

{

 int a = 2048, sum=0;

 foo(a, sum);

 printf("%d", sum);

}

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

2048 1024 512 128 2 14

Gate 2010

```

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

```

```

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

```

100 102 104 106 108 110

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

f(a, 6)

$\Rightarrow f(100, 6)$

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

$12 + f(102, 5)$

\downarrow
 $7 \% 2 == 0$

$7 - f(104, 4)$

\downarrow
 $13 \% 2 == 0$

$13 - f(106, 3)$

$$4 \cdot 2 = 0$$

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

↓

$$11 \cdot 2 = 0$$

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

$$6 \cdot 2 = 0$$

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

$$6 + 0 = 6$$

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

$$4 + 5$$

$$13-9$$

$$7-4$$

$$12+3$$

$$= 15$$

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

$$11 - 6$$

$$4 + 5$$

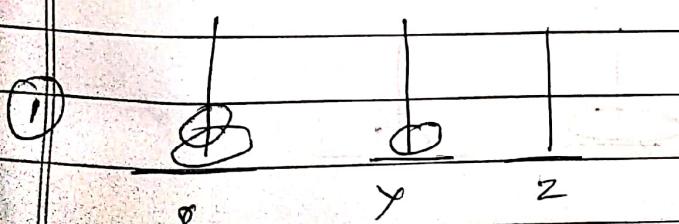
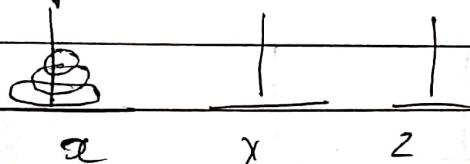
$$13-9$$

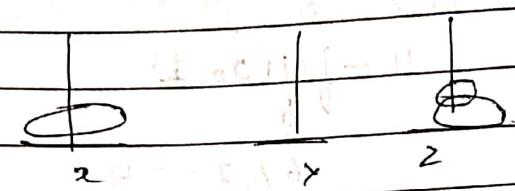
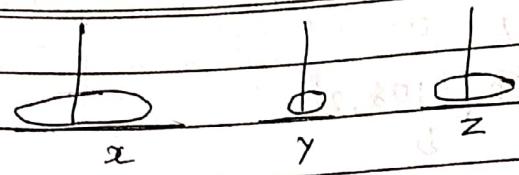
$$7-4$$

$$12+3$$

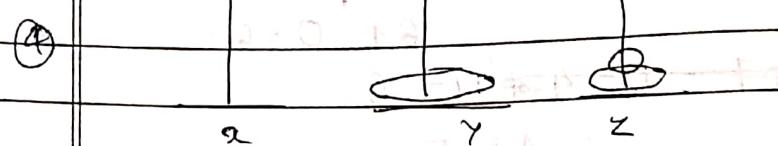
$$= 15$$

Tower of Hanoi



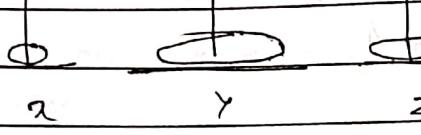


(a, b), + 2



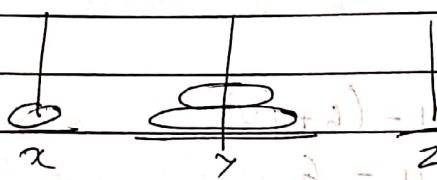
a + b + 2

P = 1



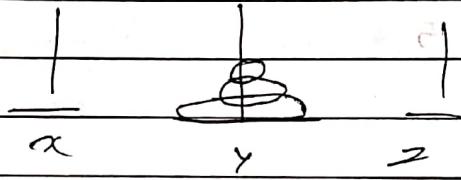
a + b + 2

P = 1



a + b + 2

P = 1



a + b + 2

P = 1



a + b + 2

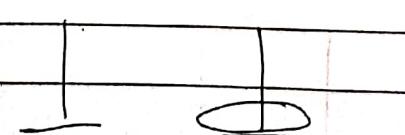
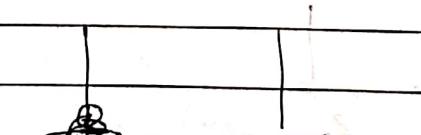
P = 1

$\max_{x \rightarrow y} z$

$n-1, x \rightarrow y$

$x \rightarrow y$

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



Algorithm for Towers of Hanoi $\text{TOH}(n, x, y, z)$

{

if ($n \geq 1$)

{

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

move top 'x' to y;

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

}

}

 $\text{TOH}(3, x, y, z)$ $\text{TOH}(2, x, y, z)$ $\text{TOH}(2, z, y, x)$ $\text{TOH}(1, x, y, z)$ $x \rightarrow y$ $y \rightarrow z$ $z \rightarrow x$ $\text{TOH}(1, x, z, y)$ $(1, y, z, x)$ $z \rightarrow x$ $\text{TOH}(1, z, y, x)$ $(1, y, x, z)$ $T(0, x, y, z)$ $x \rightarrow y$ $T(0, y, z, x)$ $y \rightarrow z$ $T(0, z, x, y)$ $z \rightarrow x$ $T(0, x, y, z)$ $x \rightarrow y$ $T(0, y, z, x)$ $y \rightarrow z$ $T(0, z, x, y)$ $\rightarrow 3 \text{ disc; No. of function invocations} = 15$ $\text{No. of movements} = 7$ Analysis of tower of hanoi $\text{TOH}(n, x, y, z) \quad I(n) = 1 + 2I(n-1) \quad \text{①} \quad \begin{matrix} \text{Invocation} \\ \text{required for } n \text{ disc} \end{matrix}$

{

 $I(n) = 1, n=0$ if ($n > 1$) $I(n-1) = 1 + 2I(n-2) \quad \text{②}$

{

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

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

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

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

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

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

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

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

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

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

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

$$= 2^n - 1$$

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

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

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

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

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

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

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

based on result of algorithm

T(0)

T(1)

T(2)

T(3)

Stack records = $n+1$

= $O(n)$

$+ (n+1) \cdot O(1) = O(n)$

Improved algo

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

{

if ($n > 1$)

{

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

more tor 'x' to 'y'

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

else

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

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

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

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

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

$$n-i = 1$$

$$\therefore i = n - 1$$

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

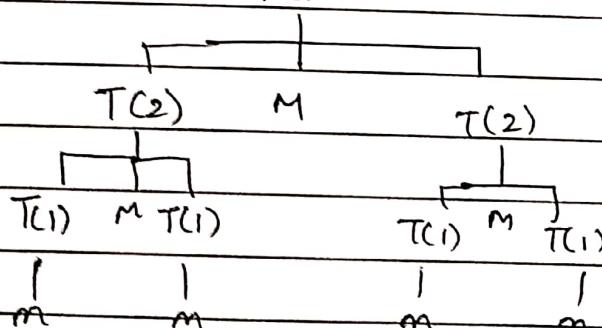
$$= 2^{n-1} + 2^{n-2} + \dots + 1$$

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

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

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

$T(3)$



No. of invocations = 7