Module 6

Date :
Page No.

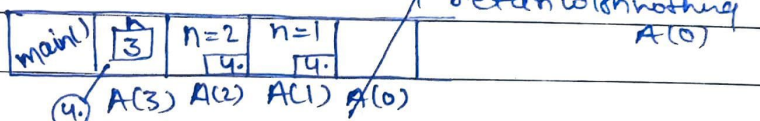
"Recursion"

Tracing of Recursion :-

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

Data structure used- Stack, Tree

Stack



4. → Instruction to execute after return

o/p → 2 1 0

Activation record formed = 5

No. of records required to execute A(3) = 4 [A(0..3)]

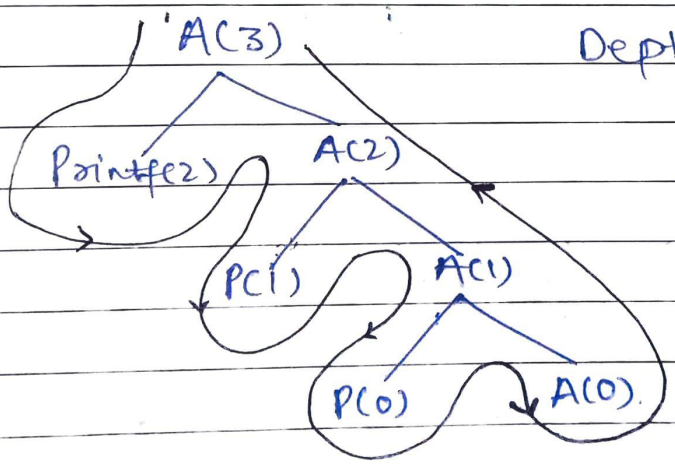
So,

Space complexity of A(n) = n+1 = O(n)

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

↳ for printf ().

Using Tree method



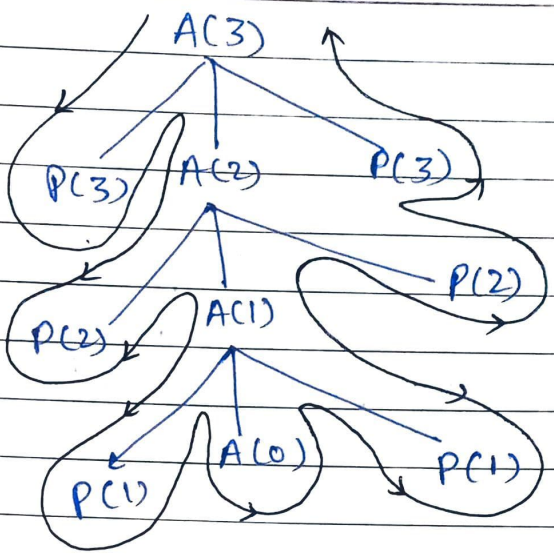
Depth of tree = (n+1)

Note → for short recursion program go with tree method either use stack method for long programs.

Date :
Page No.

P(2):

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



Depth of tree = 4

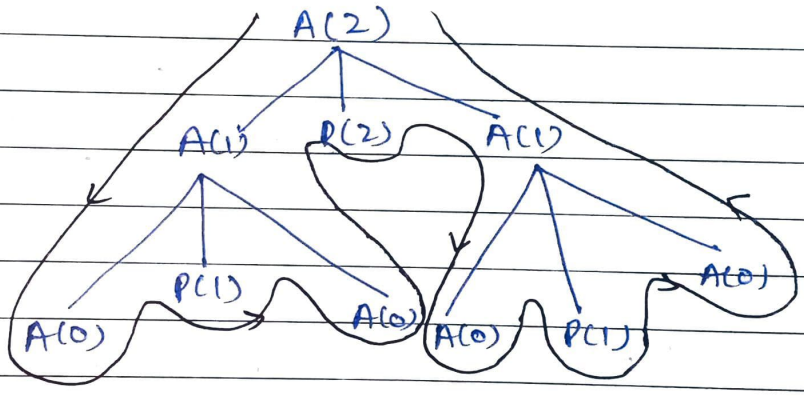
Time complexity
= C + T(n-1)

Space Complexity = n+1 = O(n)

o/p → 321123

P(3)

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



o/p → 1 2 1

Depth of Tree = 3

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

Only 3 stack record requires.

If we use dynamic programming then, we can save some function calls.

Imp. for Gate → Tracing of recursion.

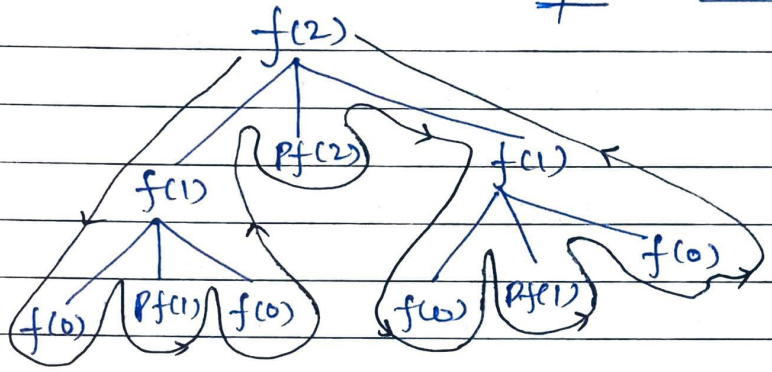
Imp Question → ① How many times a function have been called?
② What is the time complexity?

Date :
Page No.

Analysis Recursion :-

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

o/p → 1 2 1



No. of function calls → 7

f(2) → 7 calls
f(1) → 3 calls
f(3) → 15 calls.
f(10) → $2^{n+1} - 1$ (n=10) calls.

We need to generalise the things.

Recursive equation

Let number of times the fcn is called = F(n)
then, = F(n-1)

So,

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

So,

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

Date :
Page No.

And from program we can say,
if $n=0$, $F(n)=1$ (constant time)

So, if $F(n-i)=0$ it will make easy then assume $i=n$

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

It is sum of $(n+1)$ terms

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

If given

```
if (n == 1)
return;
```

then, put, $F(n)=1$ at $n=1$ and $n-i=1$

similaly for more cases.

Time complexity → same as function call.
So, here time complexity $T(n)$ and $F(n)$ both going to order of 2^n

$$\Rightarrow T(n) = C + 2F(n-1) \quad [\text{If } C=1 \text{ same as } F(n)]$$
$$T(n) = F(n) = \boxed{O(2^n)}$$

Space Complexity

It depends on depth of tree that will always order of n so,
space complexity = $\boxed{O(n)}$

Problem → Gate [2001]

- Two concepts -
  - → Recursion
  - → Pointer

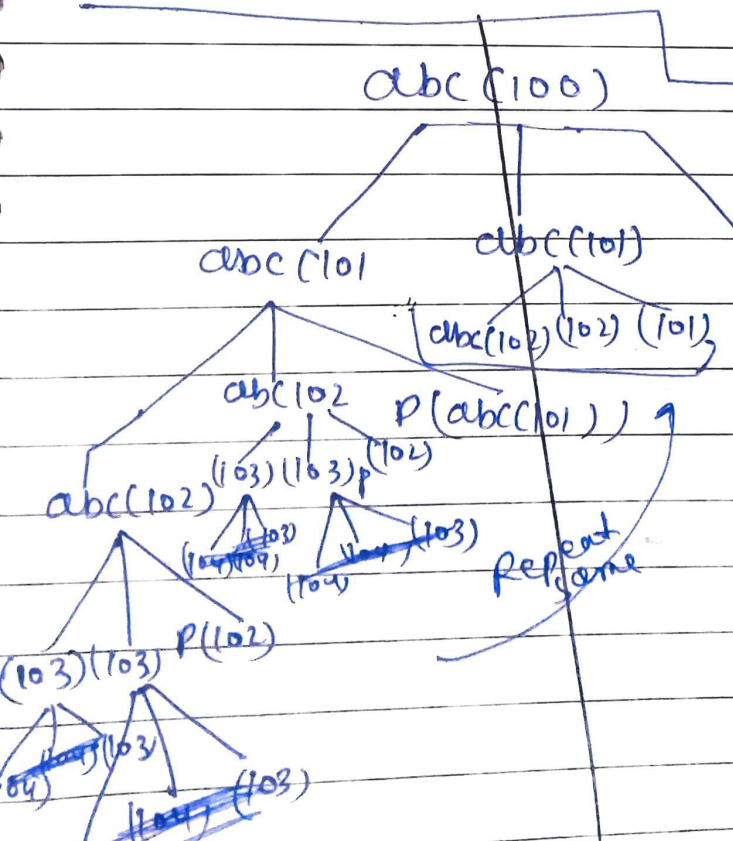
```
void abc (char *s) {
    if(s[0] == '\0')
        return;
    abc (s+1);      # s[0]+1 → 100 changes to 101
    abc(s+1);         because assume size of
    Pf ("%c", s[0]);  character is 1 byte.
}
main(){
    abc ("123");}   #
```

1	2	3	\0
100	101	102	103

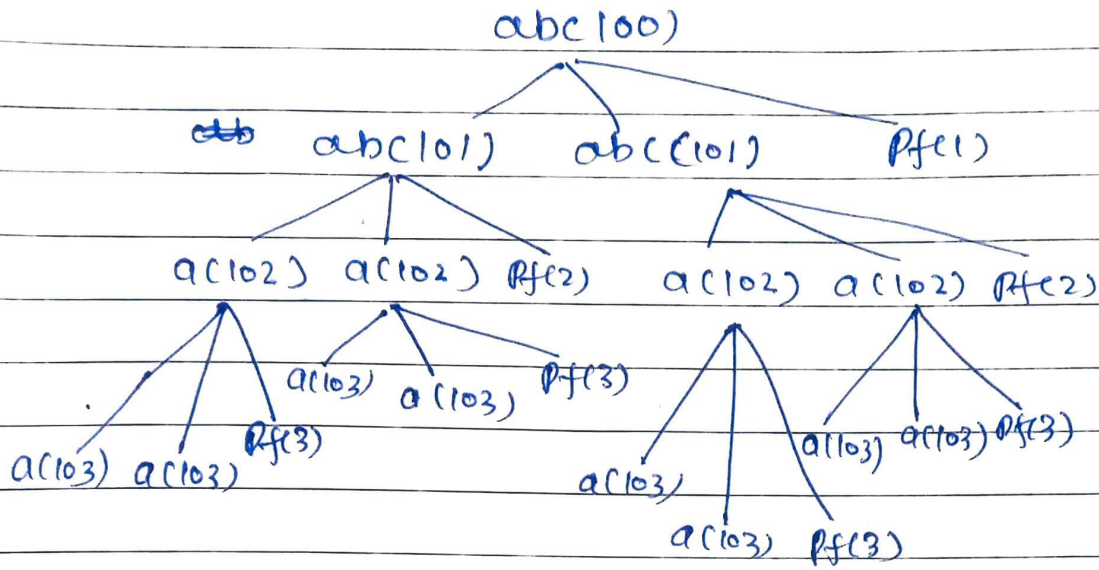
Soln

With declaration abc("123"); we are sending base adress of 1 or index 0.

if (s[0] == '\0') → If nothing inserted in string or empty string.



On next page



o/p → 3 3 2 3 3 2 1

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

Soln If n character in abc(s) then (n-1) ——— in abc(s+1) Logic

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

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

we know C(1) = 1, so, ~~not~~ assume n-i = 1

on substitute i = n-1

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

sum of GP to check $\frac{a(2^r - 1)}{(r-1)}$

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

Problem → Gate [2004].

```
int rec (int n) {
    if (n == 1)
     return 1;
    else
    return (rec(n-1) + rec(n-1));
}
```

Time complexity = ?

(a) $O(n)$ (b) $O(n \log n)$ (c) $O(n^2)$ (d) $O(2^n)$

Soln

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

Base condition $T(1) = 1$ if $n == 1$;

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

So, $n - i = 1 \Rightarrow i = n-1$

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

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

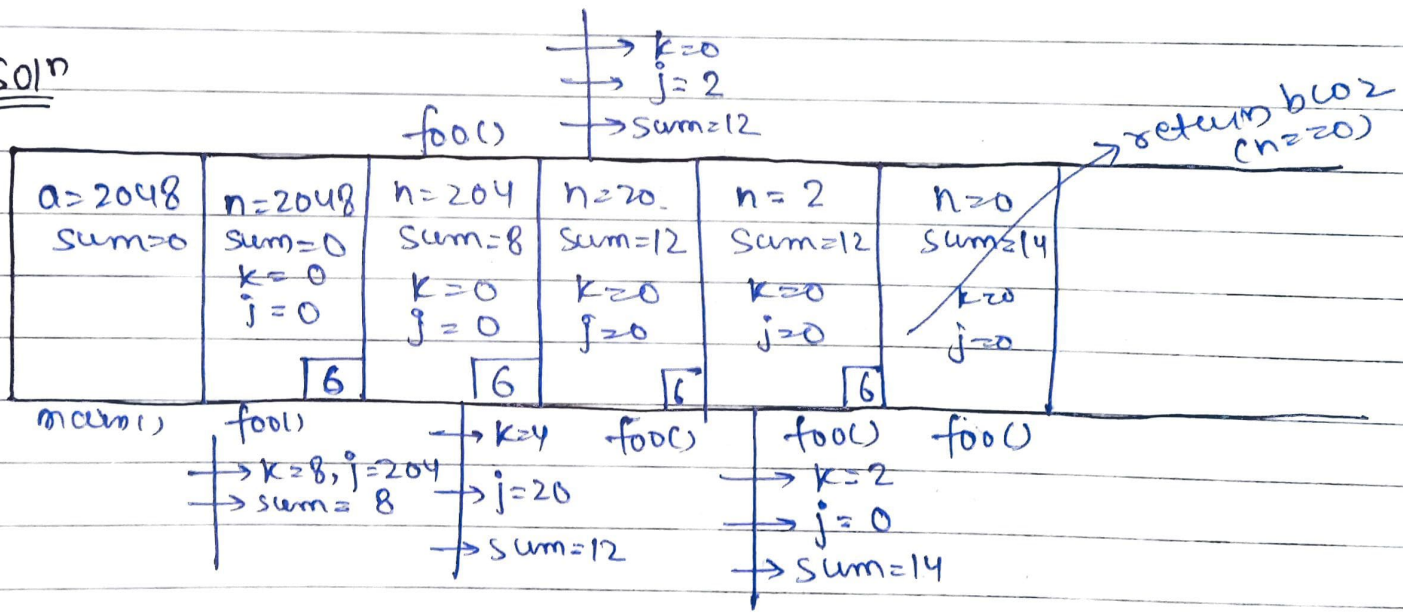
So, $T(n) = O(2^n)$ → (d).

Gate [2005]

```
Void foo (int n, int sum)
{
1. int k = 0; j = 0;
2. if (n==0) return;
3.    k = n%10; j = n/10;
4. sum = sum + k;
5.    foo (j, sum);
6. printf ("%d", k); }

int main(){
 int a = 2048, sum = 0;
  foo (a, sum);
  printf ("%d", sum);
```

Soln



o/p → 2 0 4 8 → for Pf (%d, k)
0 → for Pf (%d, sum)

o/p → 2, 0, 4, 8, 0

sum ≠ 14 bcoz it calculated inside local function so, it will destroy after execution.

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; }
```

Ans = 15

Sol^{n}

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

f(100,6)

12 + f(102,5)

return 12+3 = 15

7 - f(104,4) → return 7-4 = 3

13 - f(106,3) → return (13-9) = 4

4 + f(108,2) → return (4+5) = 9

11 - f(110,1) → return (11-6) = 5

6 + f(112,0) → return 6

0

Trick

According to question
on n is even do n+..
or n is odd do n-..
so,

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

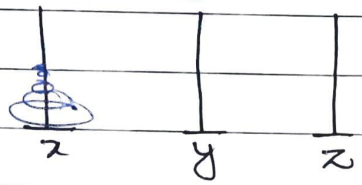
3 4 9 5

(15) → Output to print

Date :
Page No.

Towers of Hanoi :-

3 Diamond Towers and on one tower there are 64 plates or disk placed in order of decreasing size.



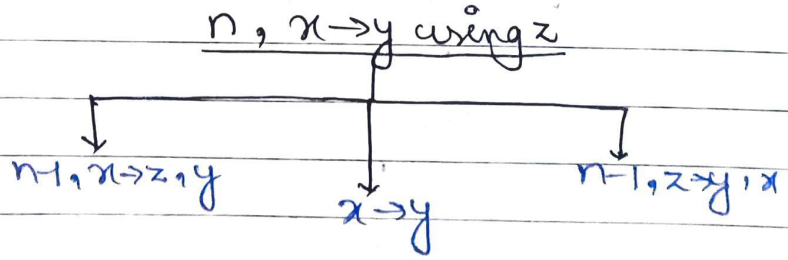
All plates are transfer from $x \to y$ using z and not lighter disk should be placed under heavier one.

We can only move 1 disk at a time.
Assume let 3 disk on x tower so, analyze

$x_1 \to y_1$
$x_2 \to z_1$
$y_1 \to z_2$
$x_3 \to y_1$
$z_2 \to x_1$
$z_1 \to y_2$
$x_1 \to y_3$

Transfering action.

Logic → from x move top (n-1) disk to z using y, then transfer heavier one from $x \to y$ and then transfer (n-1) disk from $(z \to y)$ using x.



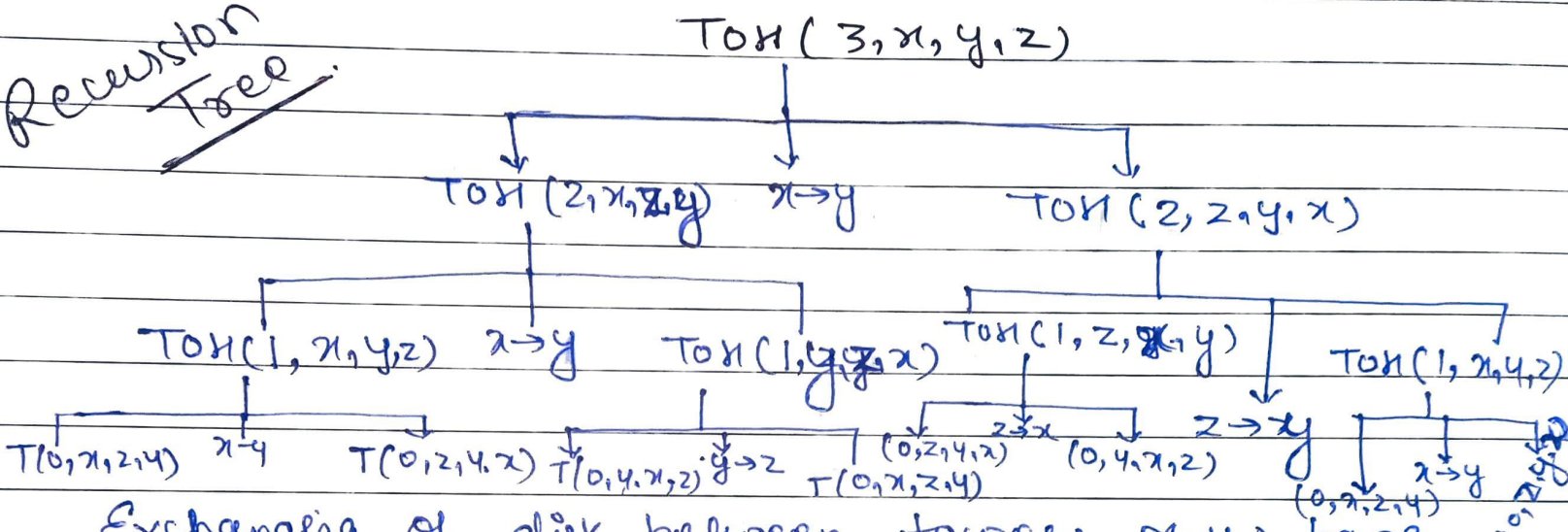
Algorithm TOH(n, x, y, z) # Transfer n disks from $x \to y$ using z.

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

move top 'x' to 'y';
TOH (n-1, z, y, x);
}
}

for n=3

Recursion Tree:



Exchanging of disk between towers x, 4, z have a pattern to notice. Remember it.

Total no. of function invocations —:
for n=3 is 15.
No. of movements = 7

for n disks —:
Invocation → $2^{n+1} - 1$
movements → $2^n - 1$
Time Complexity → $O(2^n)$

Analyzing the recursion program of Towers of Hanoi —:

1 → TOH(n)
I(n-1) → TOH(n-1)

$I(n) = 1 + 2I(n-1)$
$I(n) = 1, \quad n=0$

Sum of GP $\left[\frac{a(r^{n+1}-1)}{r-1}\right]$

Date :
Page No.

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

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

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

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

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

$\Rightarrow n - i = 0 \Rightarrow n = i$

$\Rightarrow$

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

$I(n) = \boxed{2^{n+1} - 1} \rightarrow \boxed{O(2^n)}$

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

1 for mov top 'n' to 'y'

$2M(n-1)$ for TOH$(n-1)$

Base condition → $M(n) = 0$, if $n = 0$

$M(n) = 2^{n-1} + 2^{n-2} + \cdots + 1$ [Using $I(n)$]

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

Time Complexity -

$T(n) = 2T(n-1) + 1$ → 1 is just a constant and may be any other number according to situation here.

Base condn

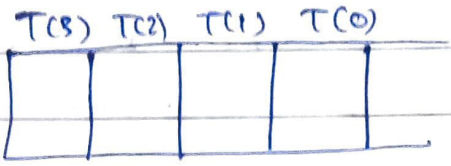
$T(n) = 1, n = 0$

Same as $I(n)$ Invocation so,

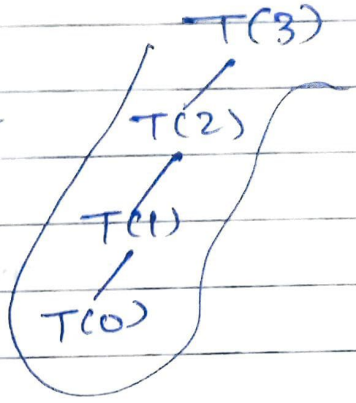
$\boxed{T(n) = 2^{n+1} - 1}$ $\boxed{O(2^n)}$

↳ $2 \cdot (2^n) - 1$

Space Complexity → Depth of recursion tree -



or



Stack records = depth = n+1

So,

Space Complexity = $\boxed{O(n)}$

To decrease number of invokation bcoz we can't reduce movements →

if (n > 1) instead of if (n >= 1)

and

else if (n == 1)

mov 'x' → 'y'

It avoids all the transitions with TOH(0, x, y, z) bcoz they do not transit further. So dont need to call it.

Base condition becomes

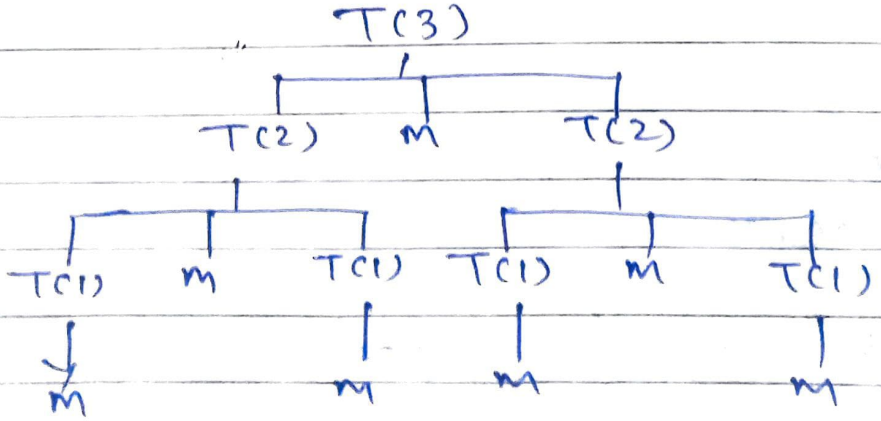
$I(n) = 1$, $n = 1$

So, finally we will get , $n - i = 1$!-

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

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

Sq n=3



M → Movement

Here we avoid all TOH(0) transitions

Linked List

Date :
Page No.

Single linked list

Every structure have only one link, so we can traverse in only one direction.

Linked list have sequential access not dynamic access but array have dynamic access.

You can access any element of array randomly.

In array, to access any particular (nth position) number have complexity O(1), because we can access the nth position directly in 1 instance but,

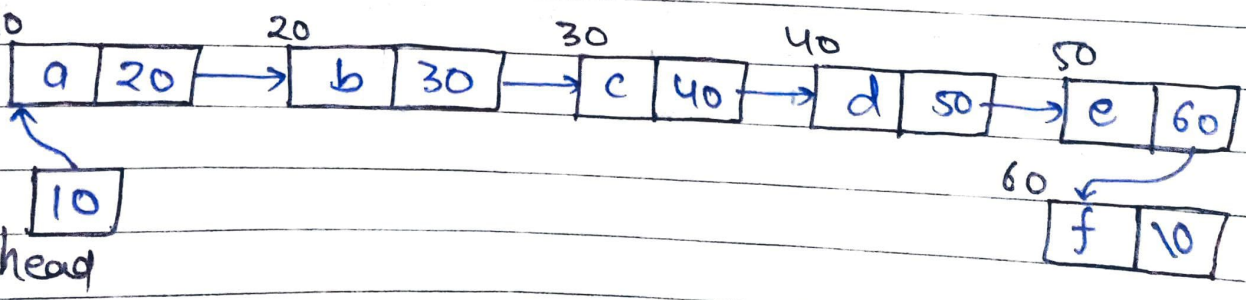
in linked list complexity O(n) bcoz you have need traverse all the elements befor nth elements.

Searching anywhere will take O(n) either in array or linked list.

Create element of linked list as structure:-

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

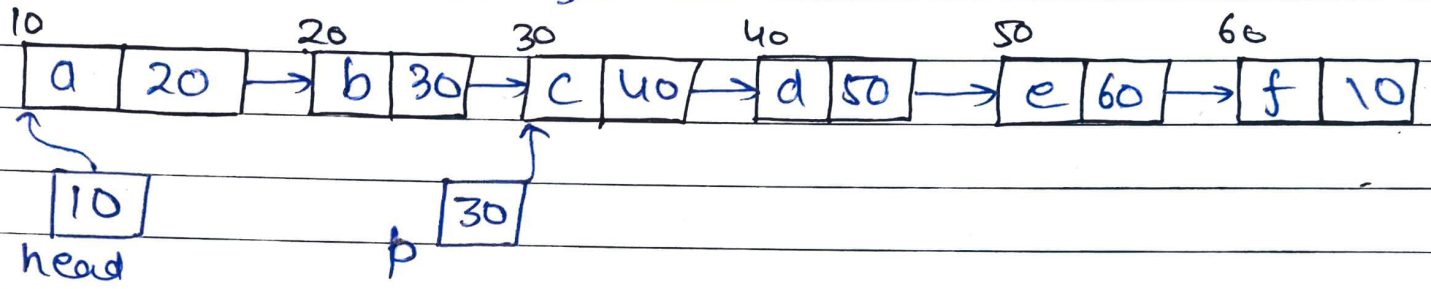
Ex



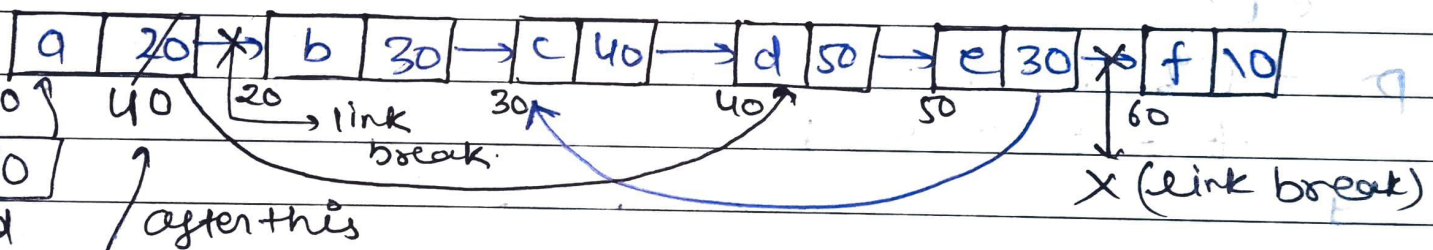
Date :
Page No.

Struct node *p p

p = head → link → link ;



p → link → link → link = p; → 50 e 30

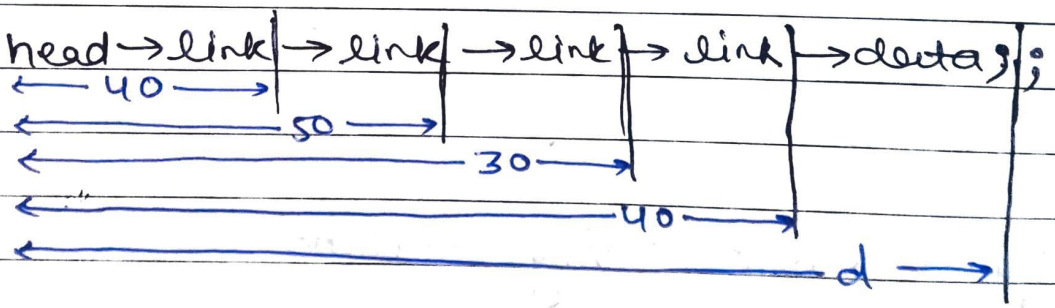


after this

head → link = p → link;

20 40

Pf ("%c", head → link → link → link → link → data);;



o/p → "d"

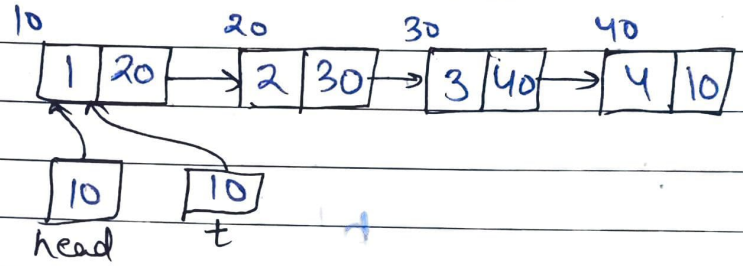
While dealing with pointer atleast check first that the pointer may not have value 'NULL'

Date
Page No.

Traversing a single linked list :-

i) Traversing
ii) inserting
iii) delete

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




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

→ t needs to be created because if you move head from one node to another the previous will lost or delete. So we need to make a copy. Or a temporary variable

t, ~~20~~

while (t != NULL) and while (t) — Both are same.

O/P - 1 2 3 4

~~10~~ ~~20~~ ~~30~~ ~~40~~ (10)

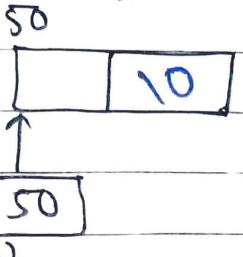
t 20 30 40

→ found NULL so no movement further.

(II) Inserting

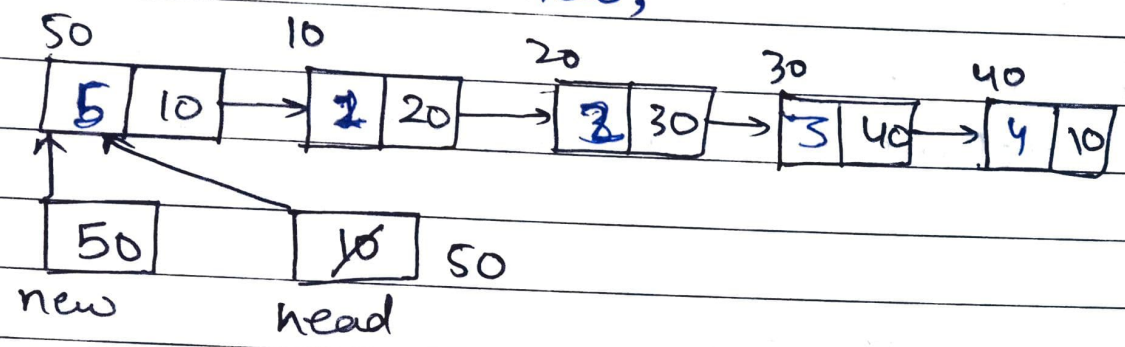
Creating a node using malloc

```
struct node *new;
new = (struct node*) malloc (sizeof (struct node));
new -> link = NULL;
```



a) insert at beggining;

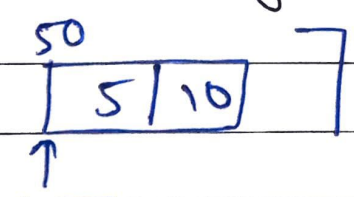
```
new -> link = head;
head = new;
```



b) insert at ending :-

```
Struct node *t
    t = head;
while (t -> link != NULL){
 t => t -> link; }
t -> link = new;
```

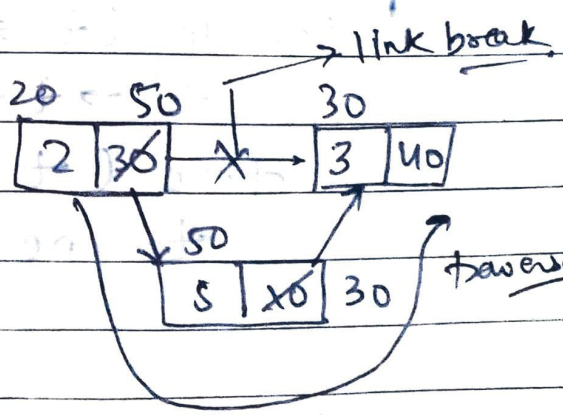
c) At the given position :-



Insert the node as the position pointed by another pointer

Let insert after node that contain value 2.

```
Struct node *t = head;
while (t -> data != 2)
    t -> t -> next;
new -> link = t -> link;
    t -> link = new;
```



(iii) Deletion

i) At beggining.

`head = head → next;`] It loss the node so above codeline is not appropriate.

Correctone →

```
struct node *t = head;
head = head->next;
free(t);
```

↳ It also make the m/m available for next node

lower code is then move

```
if (head == NULL)
    return;
if (head → link == NULL)
    free (head);
head = null;
```

ii) Delete at last → (from tail)

for atleast two element

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

iii) Delete at position;

```
Let position at value 3
while (t → link → data != 3)
    t = t → link;
```

```
Struct node *t1 = t -> link;
t -> link = t1 -> link;
free(t1);
```

Gate Question [2008] | [2005]

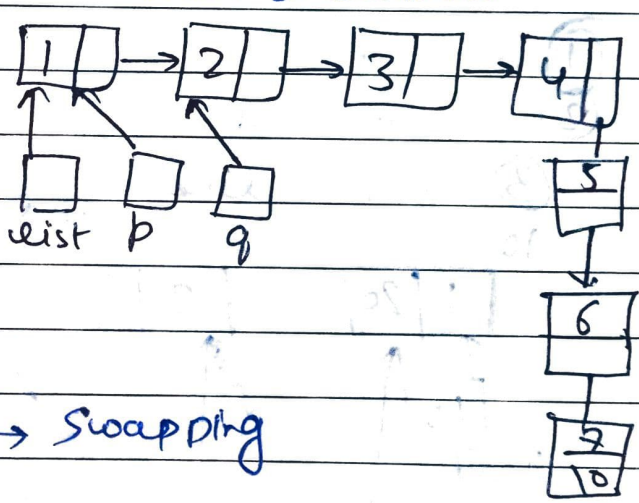
```
Struct node {
    int val;
    struct node *next;
};
void rearrange (struct node *list)
{
struct node *p, *q;
  int temp;
if (! list || !(list -> next))
    return;
  p = list ; q = list -> next;
while (q) {
    temp = p -> val;
    p -> val = q -> value;
    q -> val = temp;
    p = q -> next;
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

Sol^n

element [list == NULL and ! list are same]

[!(list -> next)] 1 element

So, program work on more than 1 element



→ Swapping

→ p will point the node next after q

→ If p = NULL then, take 0 else q = p -> next.

1 2 3 4 5 6 7

i) 1234567

ii) 2143657

iii) 1325476

iv) 2345671

O/p 2 1 4 3 6 5 7

(b is the answer;)

Gate [2010]

Moving a last node to front of a list.

Gate previous yr solved Pg no. 346

Q. no — 4.8

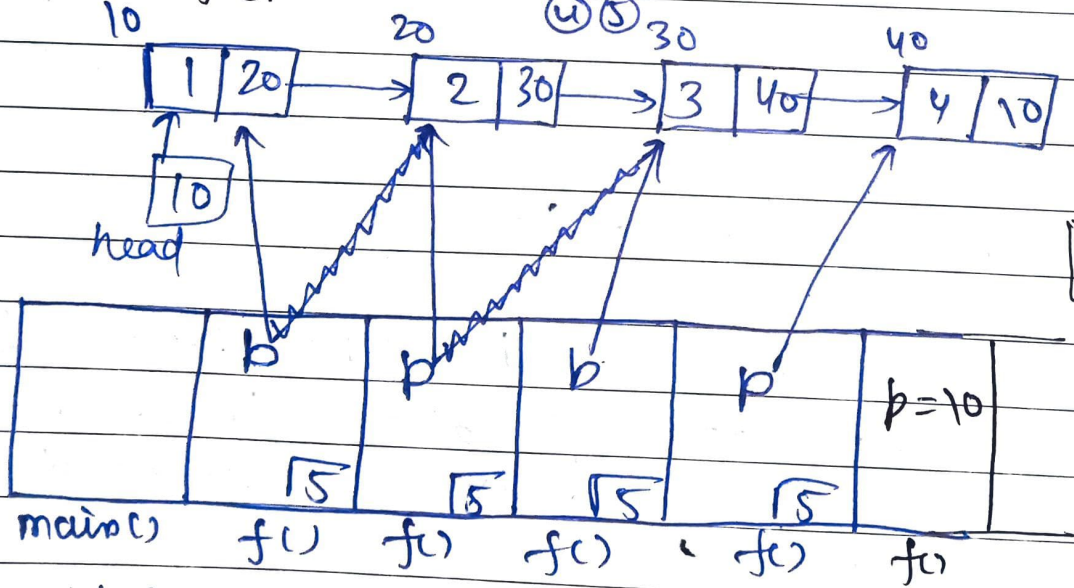
Done()

Ans — (d)

Print the elements of SLL using recursion :-

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

```
void main(){
    f(head);
}
```



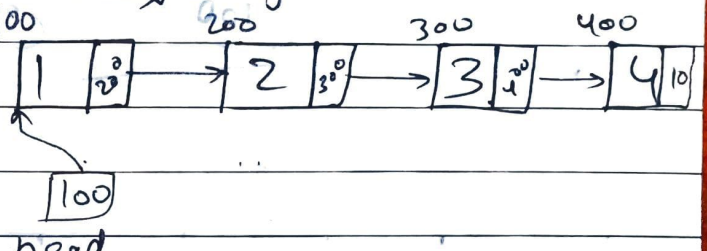
o/p → 1 2 3 4

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

(reversely,

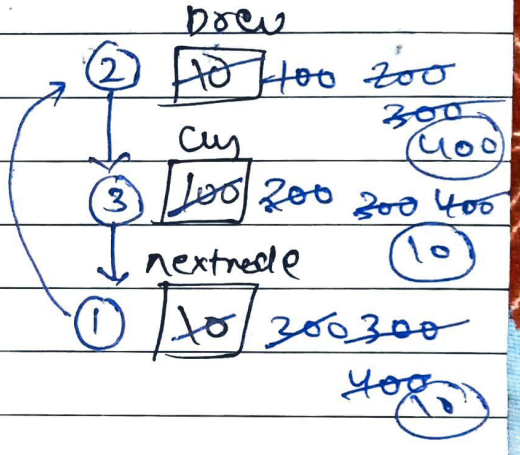
o/p → 4 3 2 1

Reversing an SLL using Iterative program:-



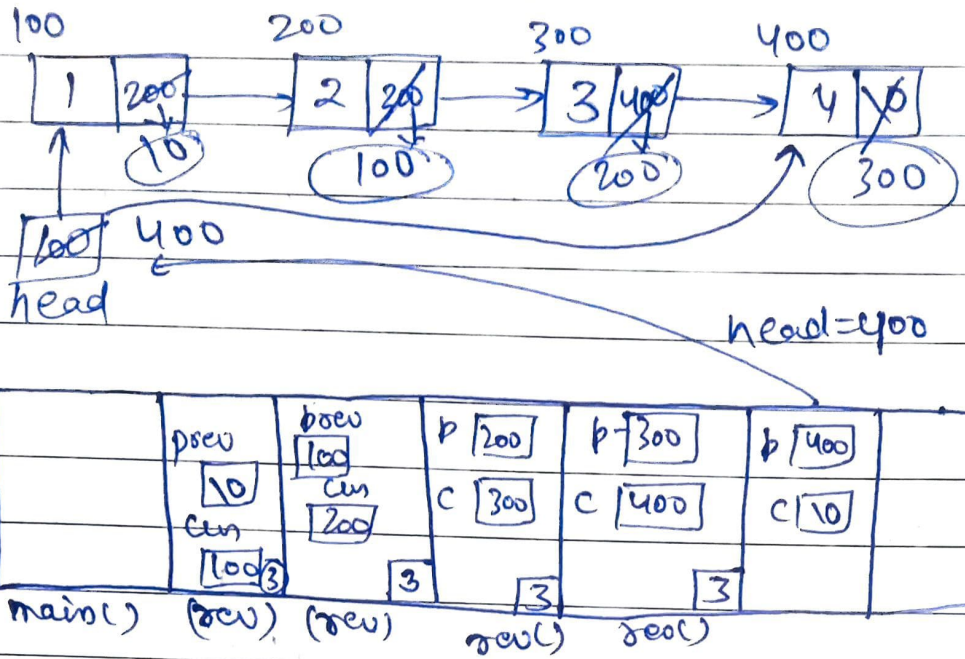

```
main() {
head = reverse (head); }

Struct node *reverse (struct node *cur) {
    struct node *prev = NULL, *nextnode = NULL;
    by: while(cur) {
        nextnode = cur->next;
        cur->next = prev;
        prev = cur;
        cur = nextnode;
    }
    return prev;
}
```

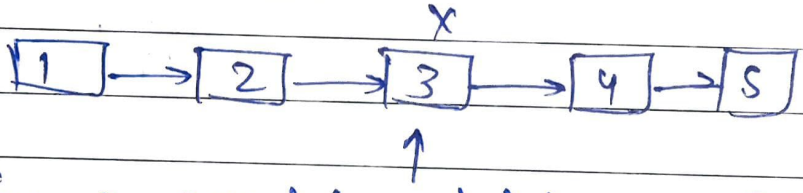


Recursive program for reversing SLL

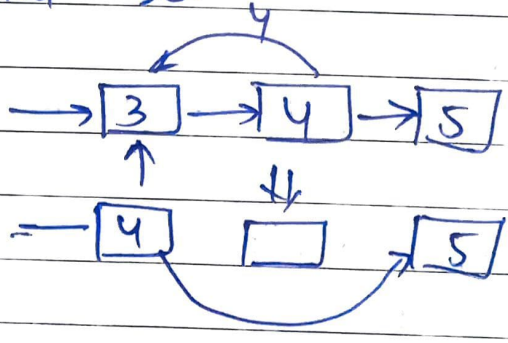
```
Struct node *head;
void reverse (Struct node * prev, *cur)
{ (1) if ((cur) {
(2) reverse (cur, cur->link);
(3) cur->link = prev;
(4) } else
        head = prev;
} void main() {
    |
    |
reverse (NULL, head)
}
```



Q Pg-345 Q-4.5 (GPV)



for a need to delete we need information about a node just previous to it but here we have information about node to be deleted so.

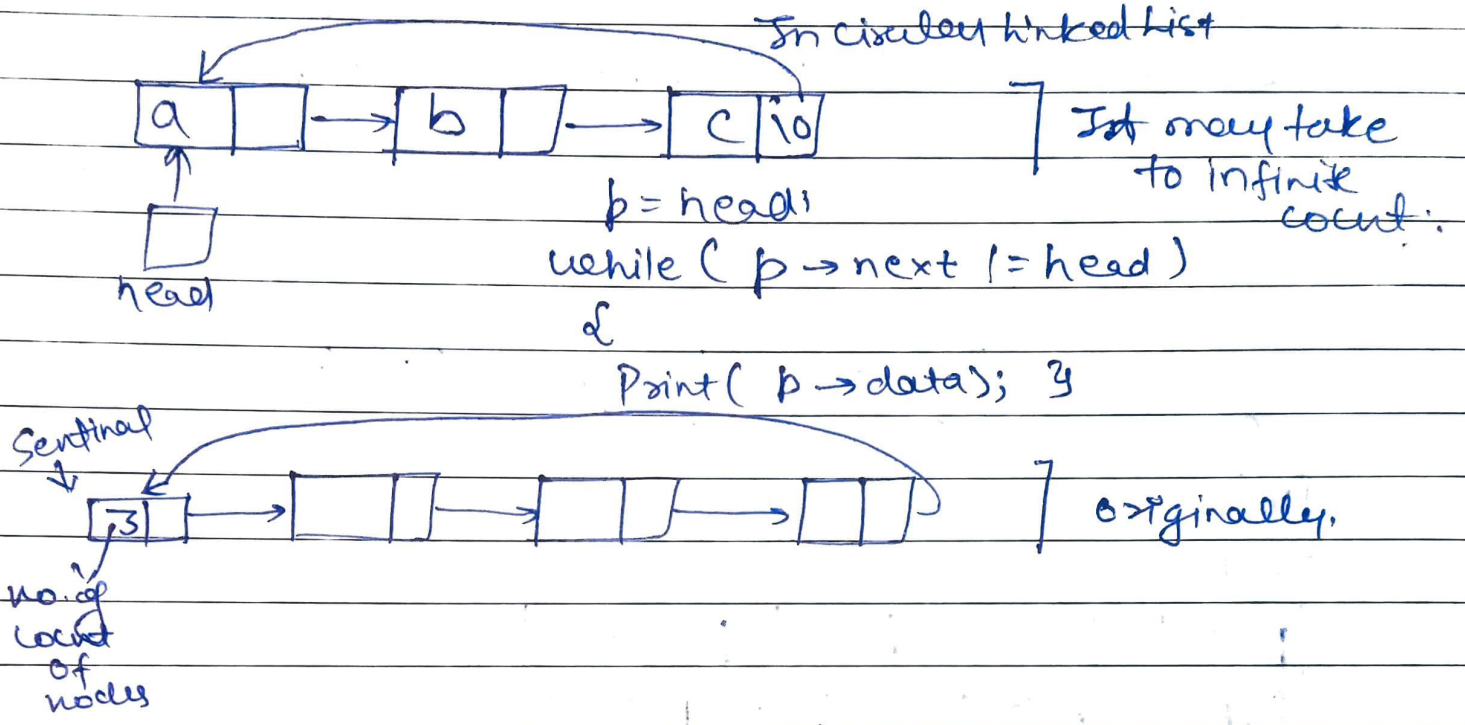


We do that copy the information of next node to particular node and delete the next node. This will take a constant (1) time. So

Ans - O(n).

In worst case to search any element in linked list have time complexity O(n).

Circular Linked List



Gate [2003] Checking if the list is in non-decreasing order.

```
struct item {
  int data;
  struct item *next; };
int f(struct item *p)
return ((p==NULL) || (p->next==NULL) || ((p->data <= p->next->data) && f(p->next)));
```

a → ((p==NULL)) , b → (p->next==NULL) , c → (p->data <= p->next->data) , d → f(p->next)

a || b || (c && d)

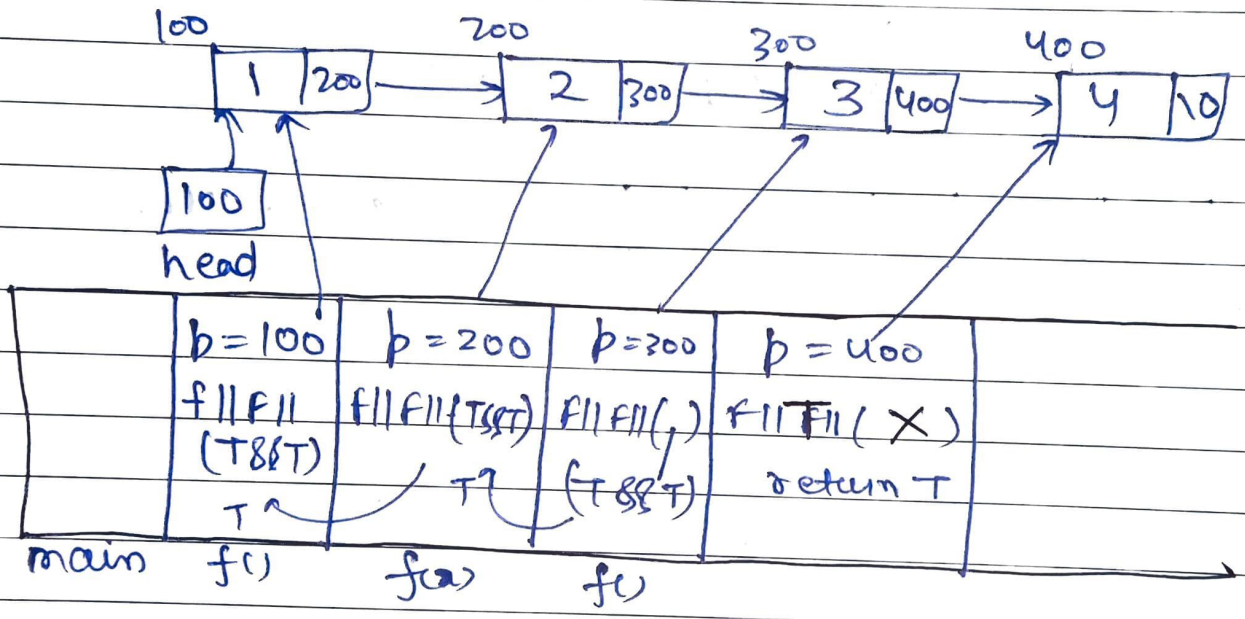
F	T	X	X	→ T
T	X	X	X	→ T
F	F	T	T	→ T
F	F	F	F	→ F
F	F	F	T	→ F

i) a will true if no node in list.
ii) p→next will true if atleast ~~one~~ or only one node

function f will return true if 0 element or 1 element or if more than 1 element then value at 2nd element must be greater than first element.

Detailing

Ex →



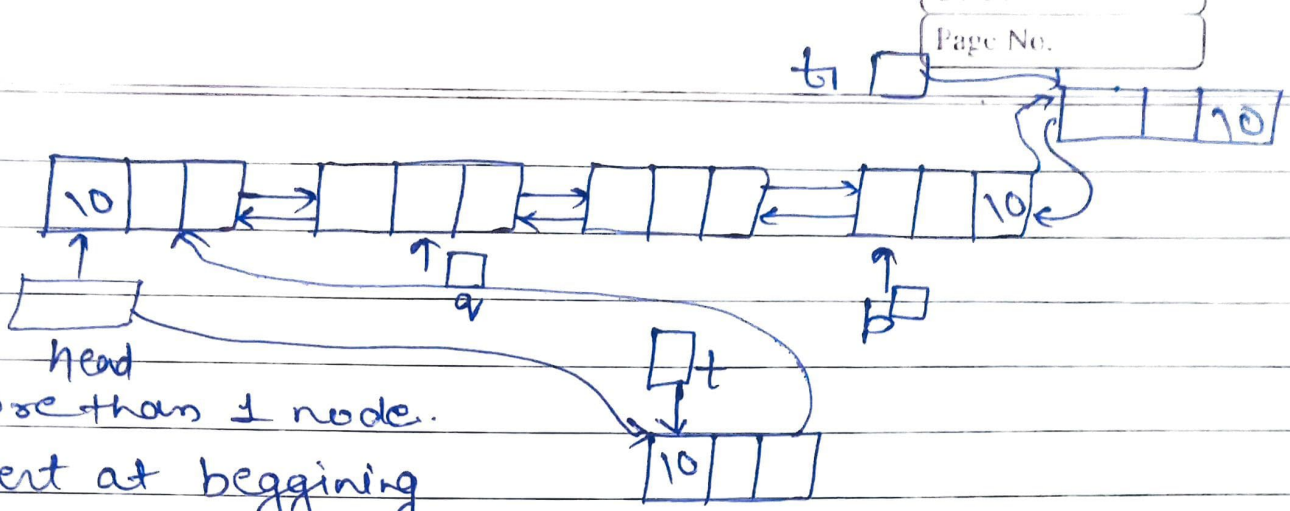
return True as o/p

Ascending

Not decreasing

Insertion into Doubly Linked List -:

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



for more than 1 node.

i) Insert at beggining

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

ii) Insert at end :- `p = head;`

```
while (p -> next != NULL){
    p = p -> next; }
    p -> next = t1;
    t1 -> prev = p;
    t1 -> next = '\0';
```

iii) In intermediate

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