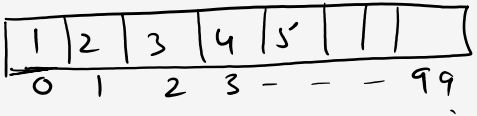
C-Programming and DS

Today Class Topics

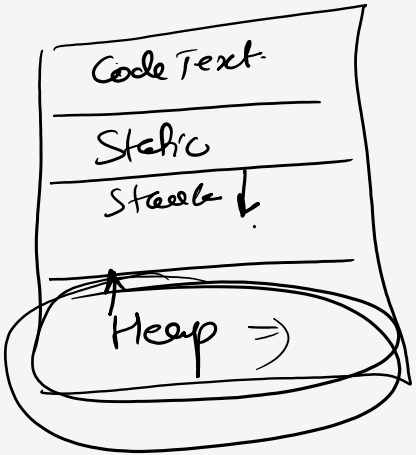
- ✓ Dynamic Memory allocation
- ✓ Singly linked list
- ✓ Singly circular Linked list
- ✓ Doubly Linked List
- ✓ Doubly Circular Linked List

Dynamic memory Allocation: → Heapsegment

int a[100];



99×2 byte



2 00

Dynamic memory Allocation :- The following functions are used in the dynamic memory allocation

- malloc()
- calloc()
- realloc()
- free()

<stdlib.h> latest Turbo C

<malloc.h> old Turbo C

malloc():

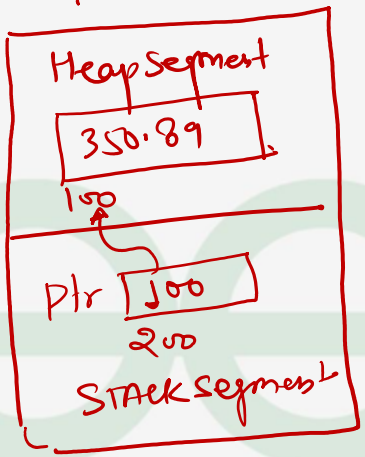
```
Pointer var = (Typecast *) malloc (sizeof (size));
```

To the required pointer

Operator

Ex:-

```
float *ptr;
ptr = (float*) malloc (sizeof(float));
*ptr = 350.89;
printf("*ptr"); // 350.89
```

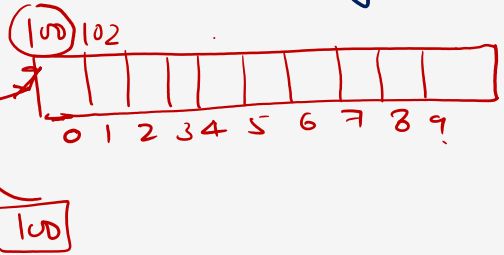


Remarks:

1) malloc() allocates the memory in Bytes and it take one argument that is in Bytes
2) Sizeof() is an operator and it calculates the size in Bytes
3) malloc() allocates the memory in the heap segment and return it Base Address
4) This Base Address is void pointer
5) Because it is returning B.A, The receiver should be Pointer Variable
6) Because its returning void pointer then respective type casting is required to dereference allocated memory
7) malloc() doesn't know to whom its allocating memory
8) whatever the memory allocated in the stack segment can be accessed using variable names and also through Pointers, but whatever memory allocated in heap segment can be accessed by using Pointers only.

9) malloc() return NULL, if it's not able to allocate the memory, when the memory is full

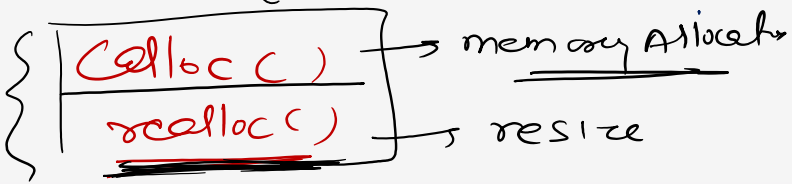
```
int *ptr = (int*) malloc(10 * sizeof(int));
if (ptr == NULL)
{ printf("m/m error");
  exit(1);
}
```

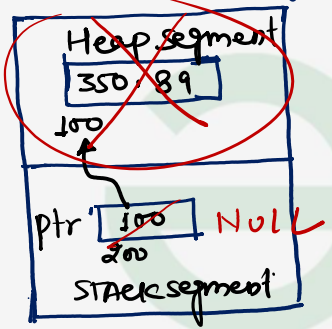


Free() :- Deallocates the memory in Heap segment and it takes one argument i.e Base Address

Ex:- free(ptr);

ptr = NULL;





Array using malloc():-

1D Array:

```
#define Col 10
int *ptr = (int *)malloc(Col * sizeof(int));
if (ptr == NULL)
{
    printf("m/merror");
    exit(1);
}
for(i=0; i<Col; i++)
{
    ptr[i] = i*10;
}
```

10 × 2 = 20

0	1	2	3	4	5	6	7	8	9
0	10	20	30	40	50	-	-	-	90
100									

ptr 100

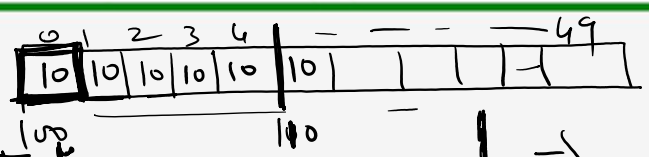
*(ptr+i)

*(100 + 0) = *100

*(ptr + 1)

⇒ *(100 + 1) = *102

2D Array :-



Method-1

```
#define Col 5
#define Row 10
int *ptr = (int*)malloc(Row*Col*sizeof(int));   // 5 x 10 x 2
for(i=0; i<Row; i++)
{ for(J=0; J<Col; J++)
  { *(ptr + i*Col + J) = 10;
  }
}
```

ptr [i][J] ≠ *(*(ptr+i)+J) ✗

*(100 + 0*5 + 0) = 100

*(100 + 0*5 + 1) =

*(100 + 1*5 + 0) =

Method-2

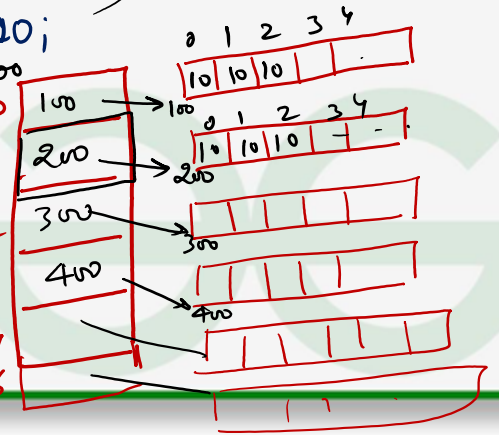
a[i][J] = 10

```
#define Row 6
#define Col 5
int *arr[Row];
for(i=0; i<Row; i++)
  arr[i] = (int*)malloc(Col*sizeof(int));
for(i=0; i<Row; i++)
{ for(J=0; J<Col; J++)
  { a[i][J] = 10;
  }
}
```

((arr+i)+J)

((600+0)+I) = 10

*(100+0) =



Data structure

→ DS is the way of oragnizing the data elements by considering its relationship

→ DS mainly Deals with

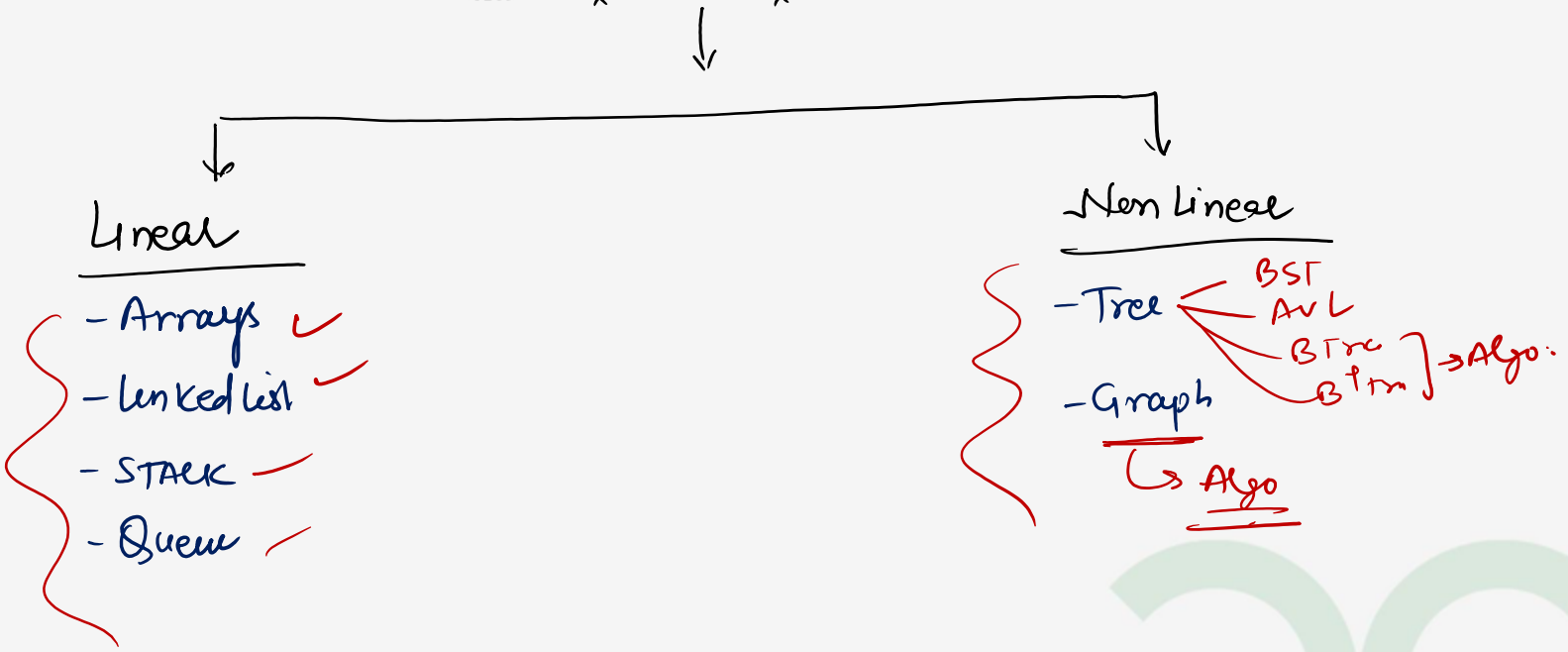
a) How efficiently data can be orgazided and stored in the computer memory (RAM, cache, Secondary memory)

b) How efficiently data can be retrieved and manipulated

Program (Project) = Algorithm + Data structure

Data structure = Organized data + operation
- Insertion
- Deletion
- Searching

DS-classified into 2-types



Abstract datatypes (ADT) :-

In order to simplify the Process of solving the Problems, the DS Combined along with their operations is called A.D.T

A.D.T mainly deals with the

1) Declaration of data
2) Declaration of operation

Linked list:-

1) It is a DS, used to store the data elements and successive element are connected by Pointers, and last element contains NULL

2) We can increase the size of Linked list Dynamically until the memory is full

3) If the requirement is frequent insertion and deletion in various Positions then the Linked list is best suitable DS

4) main draw back of linked list, it support only sequential access

Array	Linked list
1) Adv:- simple - faster Access O(1)	1) Adv:- Increase or decrease The size of memory at runtime
2) DisAdv:- Static memory - one block memory	2) DisAdv:- more Access time - Complex Implementation - Extra space Wastage.

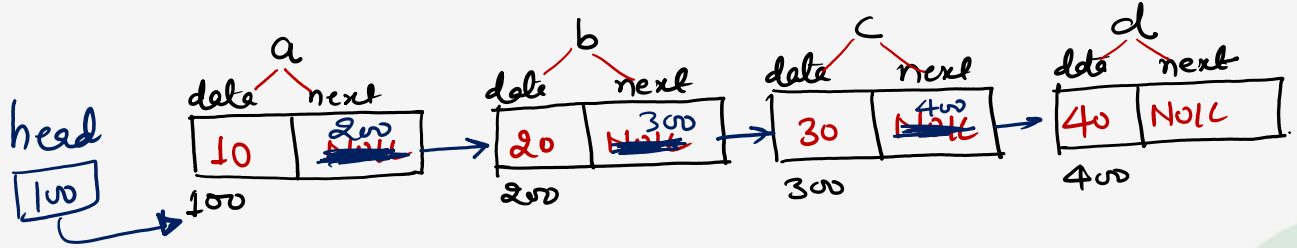
	Linked list	Array
Access time	O(n)	O(1)
Insertion/Deletion at beg	O(1)	O(n)
Insertion/Deletion at last	O(n)	O(1)
Insertion/Deletion at middle	O(n)	O(n)

Singly linked list:-

① Struct node → 2B
{ int data; → 4B
Struct node *next;
};
Typedef Struct node node;

② node a = {10, NOLL};
node b = {20, NOLL};
node c = {30, NOLL};
node d = {40, NULL};

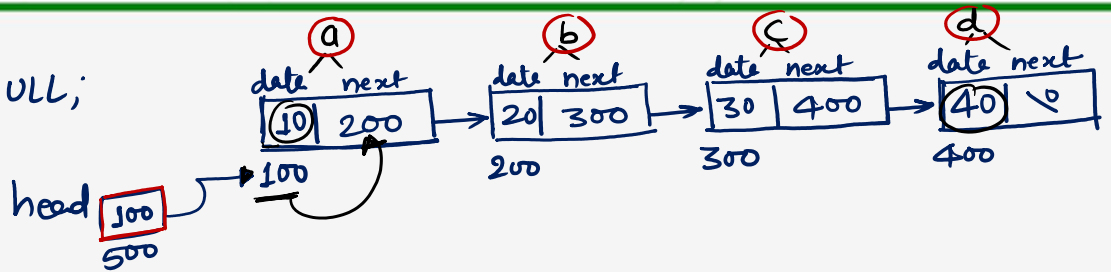
③ a.next = &b; // 200
b.next = &c; // 300
c.next = &d; // 400



④ node *head = NOLL;
head = &a

4) node *head = NULL;

head = &a;



i) Pf(head) = ? // 100

ii) Pf(head->data) = ? // 10

iii) Pf(head->next) = ? // 200

iv) if(head == NULL) ? // Linked list empty

v) if(head->next == NULL) ? // Linked list is having one node

vi) Pf(head->next->next->next->data); = ? // 40

200 → 300 → 400

vii) Pf(head->next->next->next->next->data); = // Error

200 → 300 → 400 → NULL

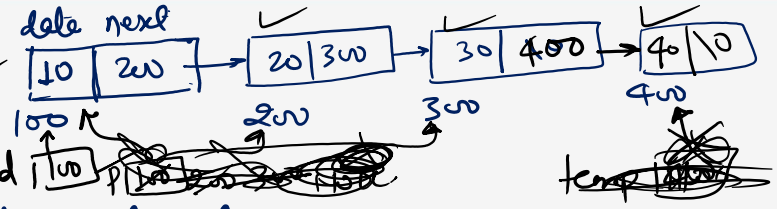
NULL pointer dereferencing also called Segmentation error or Runtime error

Create Singly Linked list:-

$Tc = O(n^2)$

```
#include<stdlib.h>
#include<stdio.h>
void createnode();
struct node
{ int data;
  struct node *next;
};
typedef struct node node;
node *head = NULL;
void main()
{ int i,n;
  printf("Enter no.of nodes");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  createnode();
}
```

→ O(n)

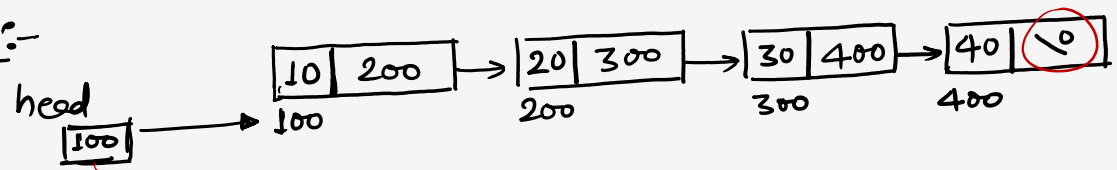



```
void createnode()
{ node *temp = (node*) malloc(sizeof(node));
  if(temp==NULL) exit(1);
  printf("enter data element");
  scanf("%d",&temp->data);
  temp->next = NULL;
  if(head==NULL)
      head = temp;
  else { node *p = head;
         while(p->next!=NULL)
             p = p->next;
         p->next = temp;
       }
  p = NULL
  temp = NULL }
```

$1+2+3+\cdots n$

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

Count No. of nodes:-




```
int ListLength(node *head)
{ if (head = NULL)
      return 0;
  else
  { int Count = 1;
    node *P = head.
    while( P->next != NULL)
    { P = P->next;
      Count++;
    }
    P=NULL
  } return(Count);
}
```

Count = 1 2 3 (4)

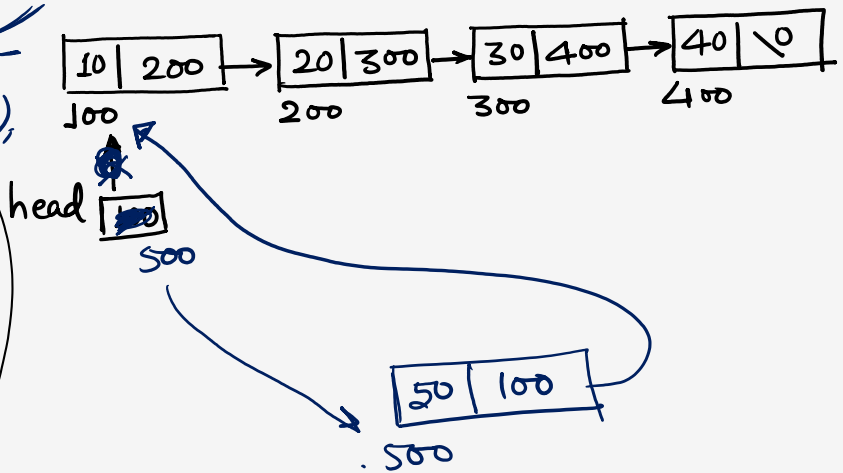
⇒ (4) Ans

TC = O(n)

BC
WC
AC

Insert new node at beg:-

```
node * insertatbeg(node *head)
{
  node *temp = (node*) malloc(sizeof(node));
  if(temp==NULL) exit(1);
  pf("enter data element");
  sf("%d", &temp->data);
  temp->next = NULL;
  if (head==NULL)
    { head = temp;
      return head;
    }
  else { temp->next = head;
         head = temp;
         temp = NULL;
       return head;
       }
}
```

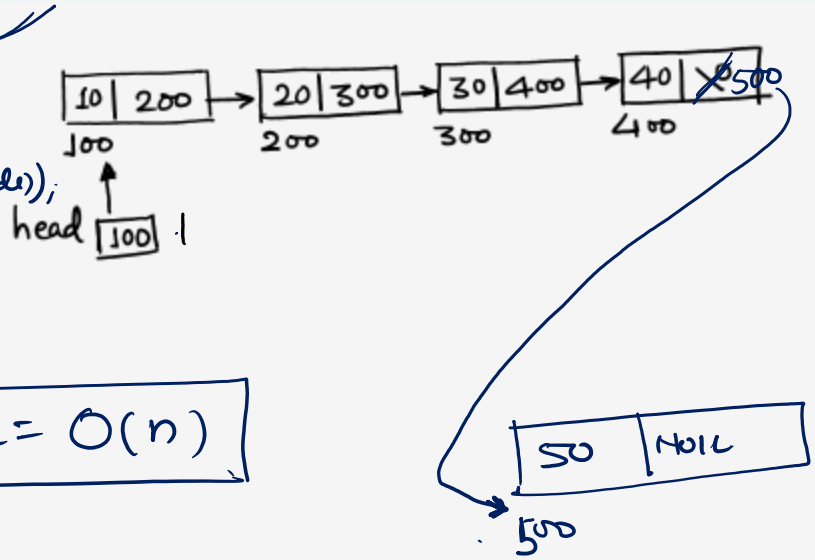


TC = O(1)

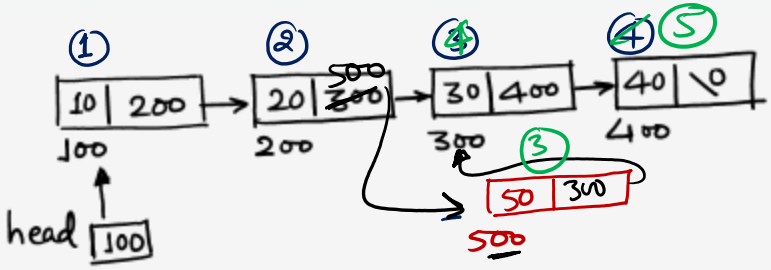
Insert at end of linked list:-

```
Void insertatend(node *head)
{
    node *temp = (node*)malloc(sizeof(node));
    if(temp==NULL) exit(1); //
    pf("enter data element");
    sfd("%d", &temp->data)
    temp->next = null;
    if(head == NULL)
        head = temp;
    else {
        node *P = head;
        while(P->next != NULL)
            P = P->next;
        P->next
        P=null;
    }
    temp= NULL;
}
```

$\Rightarrow$ TC = O(n)



Insert new node at the given Position:- Pos = 3



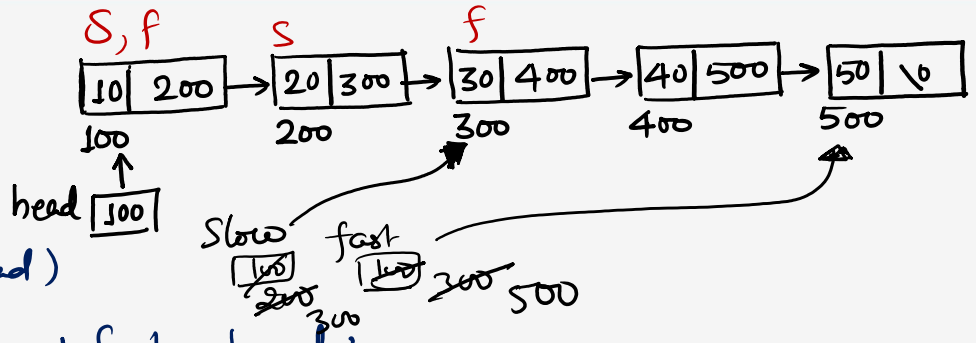

```
insertatPos(node *head, int Pos)
{ int Length = listlength(head);
  if (Pos<=0 || pos > length+1) return ;
  if (Pos==1)
    { head = Insertatbeg(head);
      return;
    }
  if (Pos == Length+1)
    { Insertatend(head);
      return;
    }
  else { int k=1; node *p = head;
         node *temp = (node*)malloc(sizeof(node));
         if(temp==NULL)exit(1);
         pf("Enter data element:");
         sf("%d",&temp->data);
         while(k < pos-1)
           { p=p->next
             k++;
           }
         temp->next = p->next;
          p->next = temp;
           p=NULL
           temp=NULL;
  }
}
```

1 < 2 : TRUE

2 < 2 FALSE

⇒ O(n)

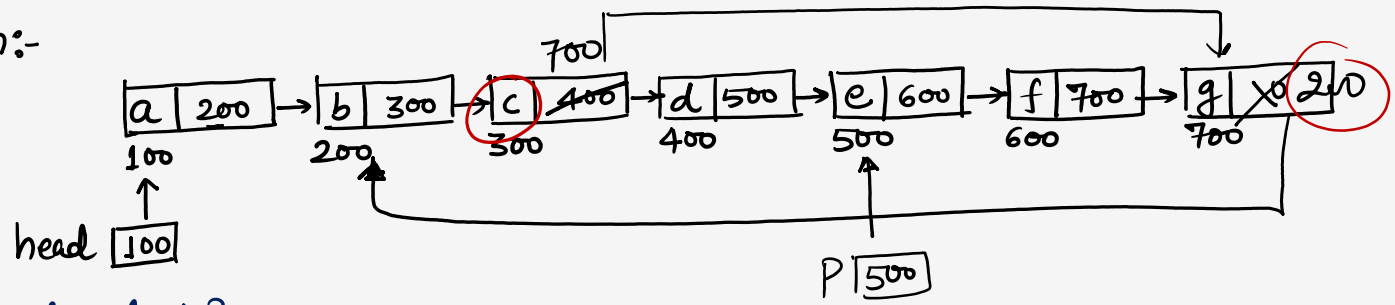
Find middle of linked list :-




```
node * middle(node * head)
{
    node * slow = head, * fast = head;
    while(fast != NULL && fast -> next != NULL && fast -> next -> next != NULL)
    {
        slow = slow -> next;
        fast = fast -> next -> next;
    }
    return slow;
}
```

TC = O(n)

Question:-



Struct node *P;

P = head → next → next → next → next;
200 → 300 → 400 → 500

head → next → next → next = P → next → next;
200 → 300 → 600 → 700

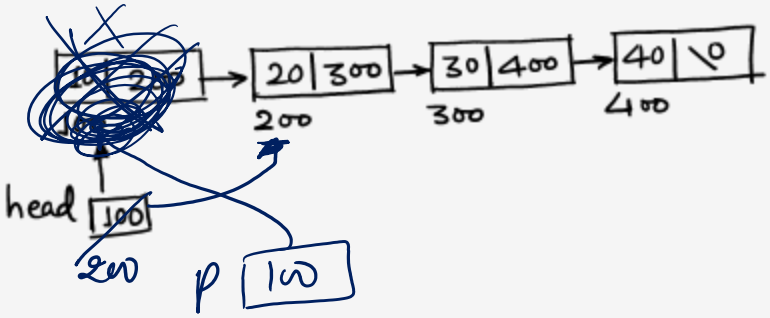
P → next → next → next = head → next;
600 → 700 → 200

Pf("%c", P → next → next → next → next → next → next → next → data); // (c)
600 → 700 → 200 → 300 → 700 → 200 → 300

O/p = ?

Deletenode at beg :-

```
node * deleteatbeg(node *head)
{
  if(head == NULL) return NULL;
  if(head->next == NULL)
      { free(head);
        head = NULL;
        return head;
      }
  else { node *p = head;
         head = head->next;
         free(p)
         p = NULL;
         return head;
       }
}
```

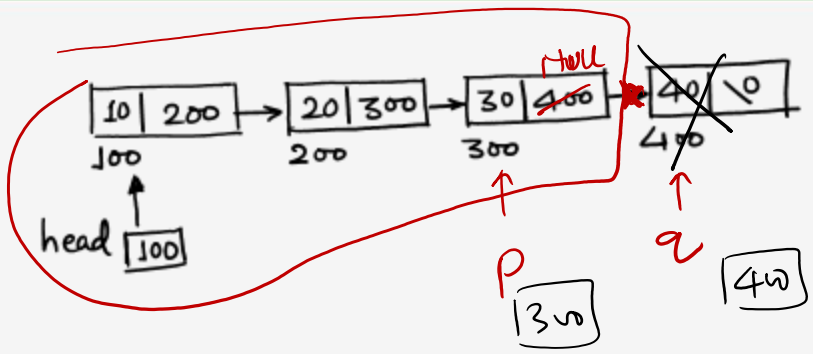


TC

⇒ O(1)

Delete at last :-

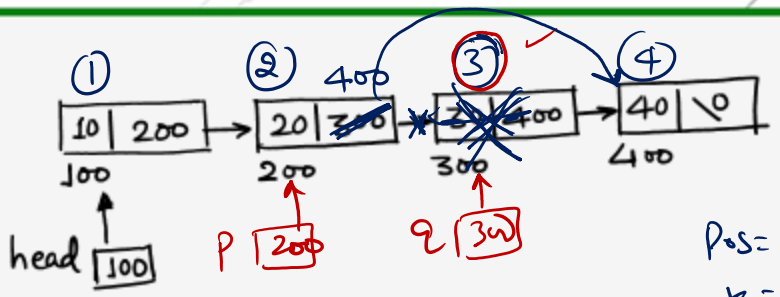
```
delete at end (node *head)
{ if(head == NULL) return;
  if(head -> next == NULL)
    { free(head);
      head = NULL;
    }
  else { node *p, *q = head
         while(q -> next != NULL)
           { p = q;
             q = q -> next;
           }
         p -> next = NULL;
         free(q);
         q = NULL;
         p = NULL;
       }
}
```



TC = O(n)

Delete at Pos :-

```
deleteatPos(node *head, int Pos)
{ int length = listlength(head);
 if(Pos<=0 || pos>length)
   { pf("Invalid position");
     return;
   }
 if(Pos==1)
   {deleteatbeg(head);
    return;
   }
 if(pos==length)
   { deleteatend(head);
     return;
   }
 else {
   int i<=1;
   node *p, *q= head
   while(k<pos)
     { p=q;
       q=q->next;
       k++;
     }
   p->next = q->next
   free(q)
   q=NULL
   p=NULL;
 }
}
```



Pos = 3

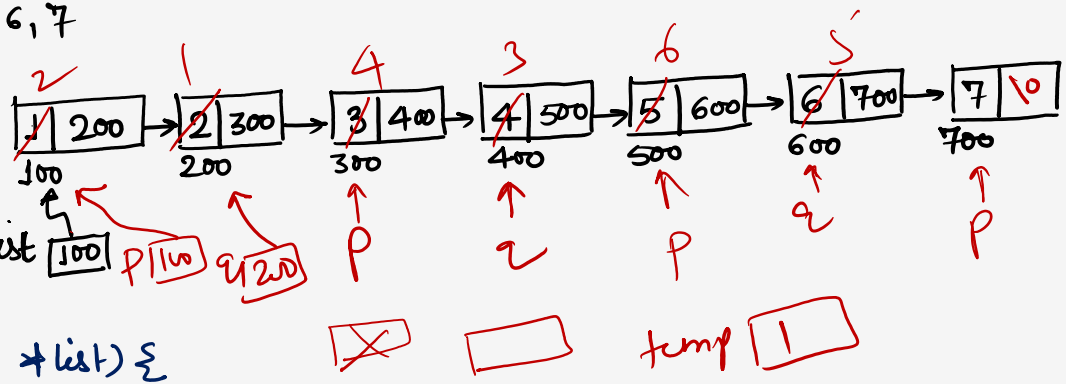
k = 1

i < 3

2 < 3

3 < 3 false

Question:- I/p : 1, 2, 3, 4, 5, 6, 7




```
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->value;
        P->value = q->value;
        q->value = temp;
        P = q->next;
        q = p ? p->next : 0;
   }
}
```

q = p ? p->next : 0

a) 1, 2, 3, 4, 5, 6, 7

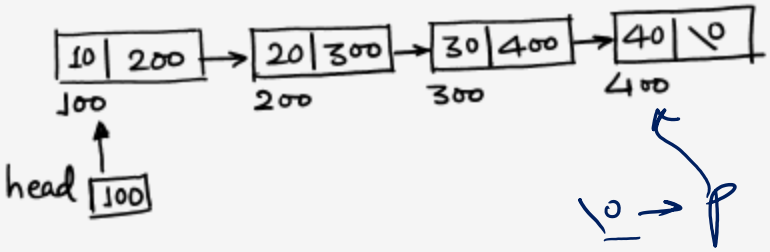
b) 2, 1, 4, 3, 6, 5, 7

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

d) 2, 3, 4, 5, 6, 7, 1

Reverse linked list :-

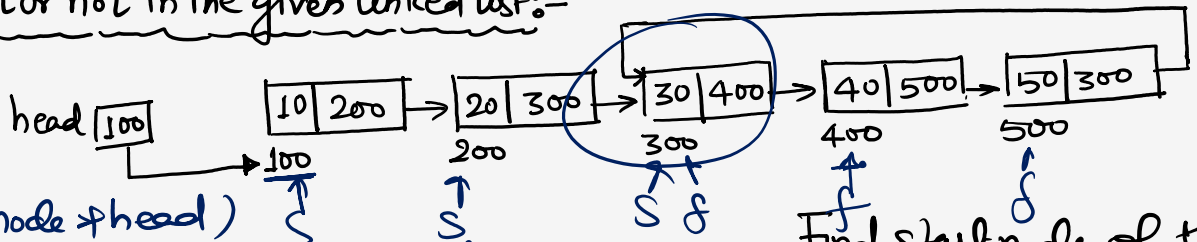
```
node * reverse(node *head)
{
  node *p, *q, *r;
  p=NULL;
  q= head;
  r= q->next
  while(q!=NULL)
  {
    q->next = p;
    p= q;
    q=r
    if(r!=NULL)
    r= r->next
  }
  return(p);
}
```



p=NULL

TC = O(n)

Detect cycle exist or not in the given linked list:-




```
Void cycle(node *head)
{ node * slow = head, *fast = head;
  while(fast && fast -> next && fast -> next -> next)
  { slow = slow -> next;
    fast = fast -> next -> next;
    if(slow == fast)
    { Pf("cycle exists");
      exit();
    }
  }
  Pf("No cycle exists");
}
```

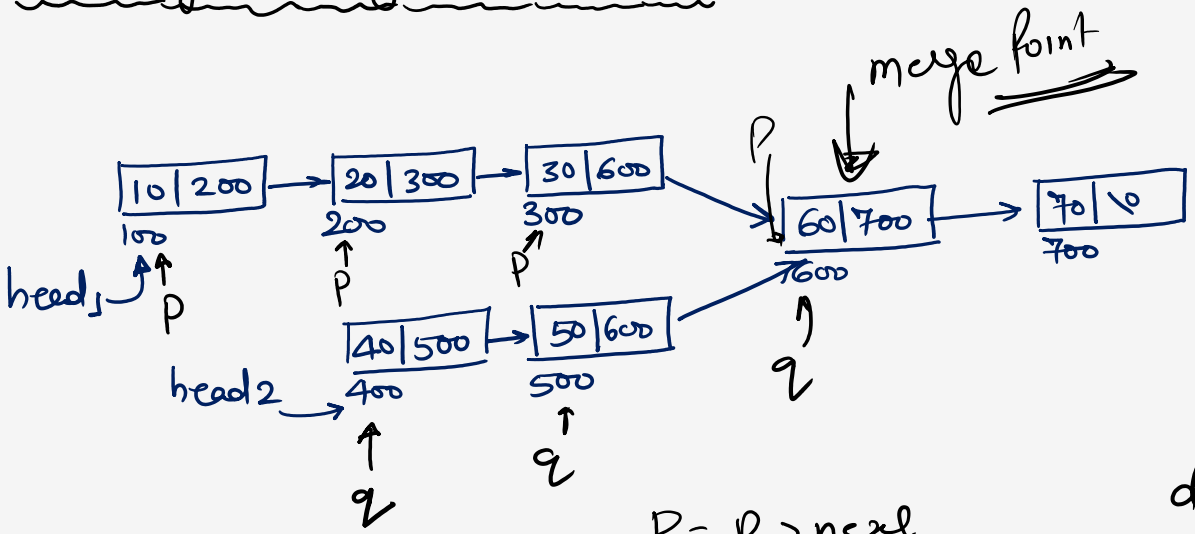
yes

DSA

Find start node of the loop

```
if(loop exist)
  slow = head
  while(slow != fast)
  { fast = fast -> next
    slow = slow -> next;
  }
```

Find merge Point in given two linked list:-



$\Rightarrow O(m+n)$

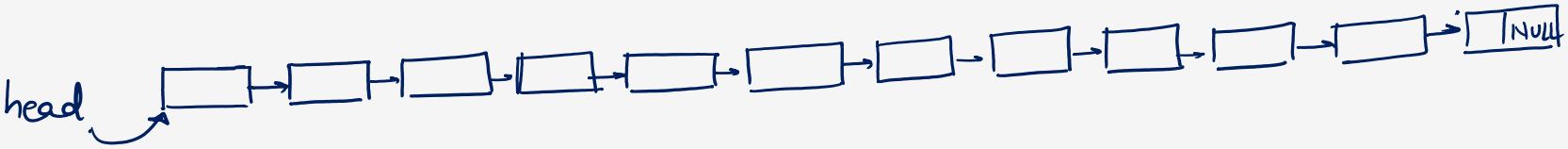
$|L_1| = 5 = m$

$|L_2| = 4 = n$ $\Rightarrow O(m+n)$

$d = |L_1 - L_2| = 1$

```
P = P -> next
Q = Q -> next
if (P == Q)
{ printf(" ... ")
```

Find n^{th} node from the end of a linked list:-



Singly linked list (SLL)

⇒

	Begin	end
Insert	O(1)	O(n)
Delete	O(1)	O(n)