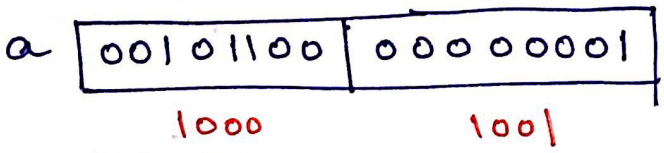
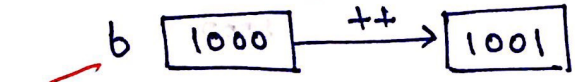
Q.

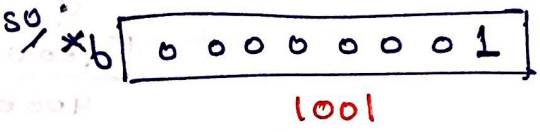
```
main()
{
    int a = 300;
    char *b = (char) &a;
    b++;
    *b = 7;
    printf("%d", a);
}
```

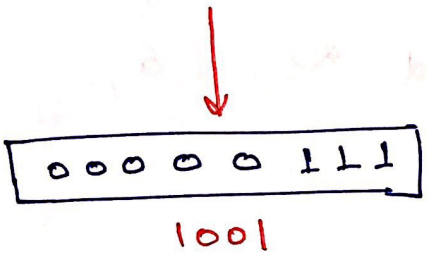




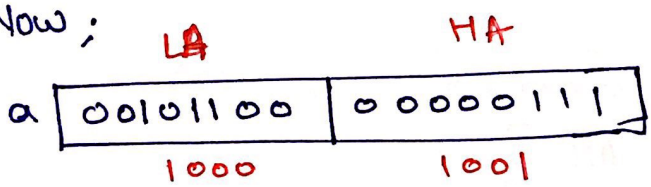
char pointer

*b = 7





Now;



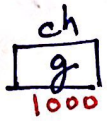
so, a = | HA | LA |

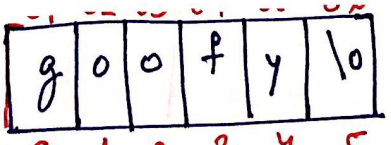
$= (1110010110 0)_2$

$= (1836)_{10}$

Q.

```
main()
{
    void *vp;
    char ch = 'g'
    char *cp = "goofy";
```



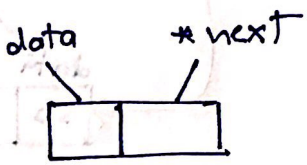


```
int j = 20;
vp = &ch;
printf("%c", *(char *)vp);  ⟶  g
vp = &j;
printf("%d", *(int *)vp);  ⟶  20
vp = cp;
printf("%s", (char *)vp + 3);  ⟶  ty
}
```

*vp: 1000 ; i: 20

Linked List :-

```
Struct node
{
    int data;  2B
    struct node *next;  8B
};
```



create note

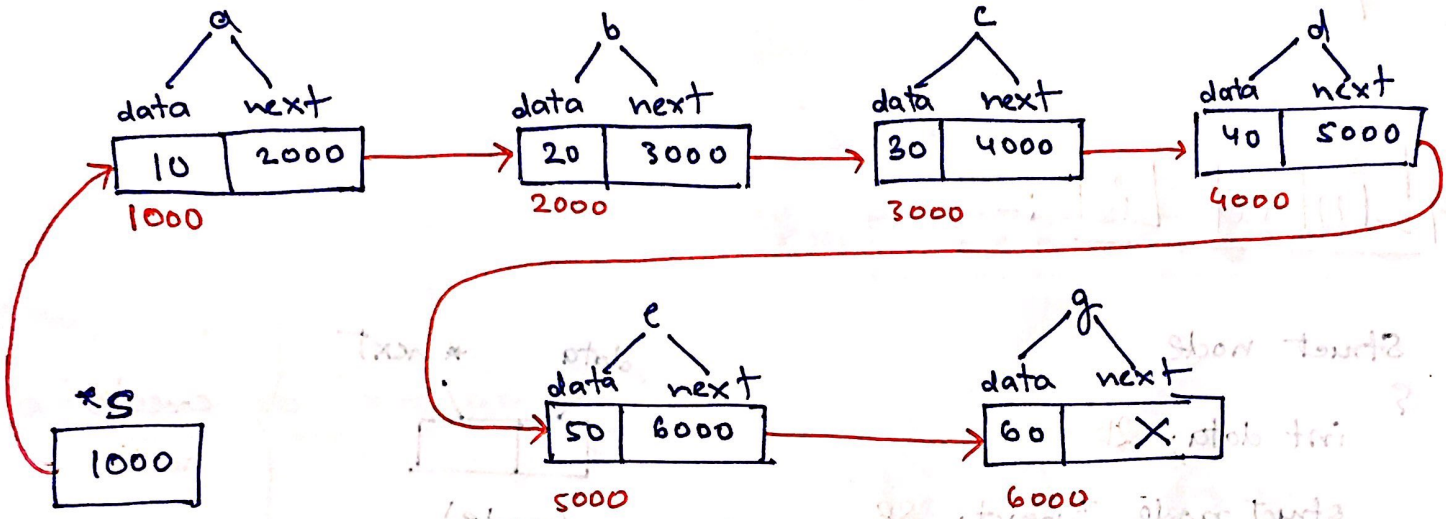
```
struct node a = {10, NULL};
struct node b = {20, NULL};
"      c = {30, NULL};
"      d = {40, NULL};
"      e = {50, NULL};
"      f = {60, NULL};
```

create variable (declare)

```
a.next = &b;
b.next = &c;
c.next = &d;
d.next = &e;
e.next = &g;
```

} Create link b/w nodes

Struct node *s = &a start pointer



NOTE:- Here, in this case nodes are created in STACK so, when function return (POP) all nodes are deleted, so, to make it save permanently, allocate them in heap area.

Q. Write a c-program to return address of second last node of a given Linked list.

Let s is start pointer;

so if s == NULL → Link list empty

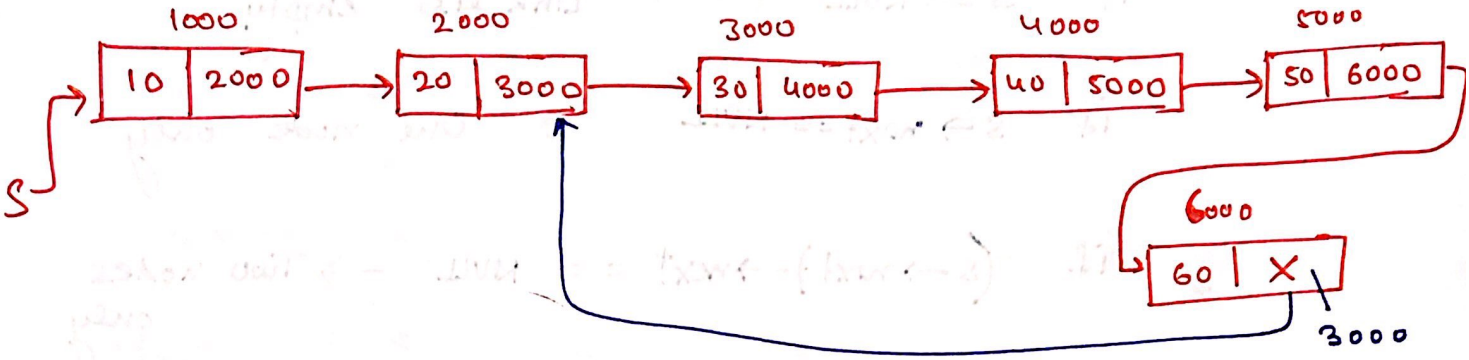
if s → next == NULL → One node only

// if (s → next) → next == NULL → Two nodes only

→ struct node * secondLastNode (struct node * s)

```
{   struct node * prev;
    if( s == null)
        return (NULL);
    else if( s → next == NULL);
        return (NULL);
    else
    {
        while ( s → next != NULL)
        {
            prev = s;
            s = s → next;
        }
        return ( p);
    }
}
```

Q: Consider the following linked list:



(i) struct node *p;

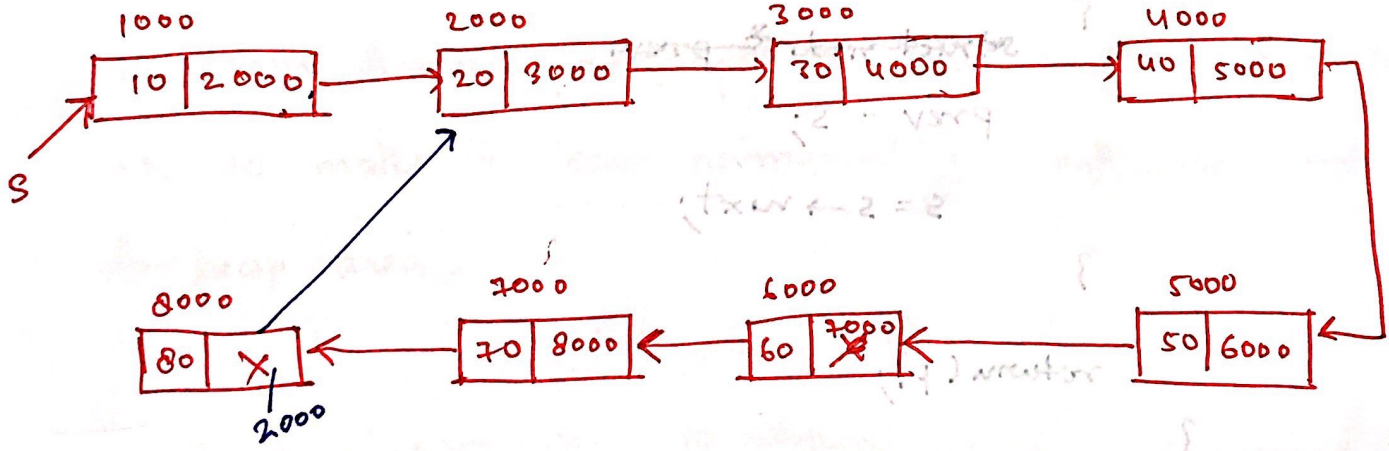
p = s → n → n → n → n;

p → n → n = s → n → n;

p = p → n → n → n → n → n → n → n → n → n;

s = p → n → n → n → n → n → n;

printf(s → n → n → n → n → data); ⟶ 40



(ii) struct node *p;

p = s → n → n → n → n → n → n;

s → n → n → n → n → n → n → n → n = s → n;

p = s → n → n → n → n → n → n → n → n;

s = p → n → n → n → n → n → n → n → n → n;

p = s → n → n → n;

printf(p → n → n → n → n → n → n → n → data); → 20

Q: write the e program to move last node of the linked list to the front of linked list.

```
struct node * movLast ( struct node *s)
{   struct node* temp, *prev;
    if( s == NULL)
        return(s);
    else if ( s->next = NULL.
            return(s);
    else{
            temp = s;
          while ( temp -> next. != NULL)
          {
              prev = temp;
              temp = temp -> next;
          }
        temp ->next = s;
          s = temp;
        prev->next = NULL;
}   }
```

Q. Write a C-program to INSERT a element (node) with data field x at the end of the given single linked list.

```
Insert_x_end ( struct node * s, int x)
{
    struct node *p;
    p = (struct node *) malloc (10B) ;   // dynamic allocation
```

sizeof(struct node)

(struct node *) — type casting

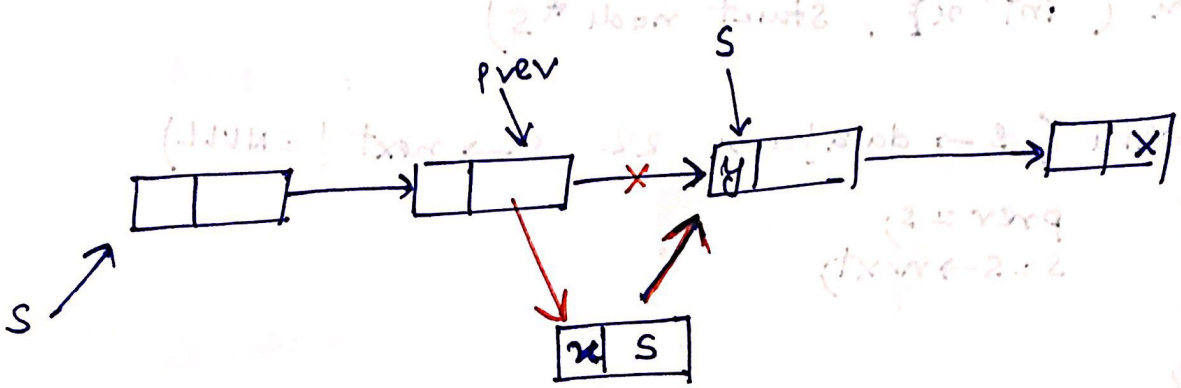
malloc (10B) — int pointer

```
    p→ data = x;
    p→ next = NULL;
    while ( s→next != NULL)
    {
        s = s→next;
    }
    s→next = p;
}
```

O(n)

NOTE:- Memory is allocated dynamically because we want node in heap area so that it will not delete when func^n over.

Q. Write a C program to insert a node with data x before a node with data y in the given single linked list.




```
insert_x ( struct node *s, int x)
{
    struct node *p;
    p = (struct node *) malloc( sizeof(struct node));
    p→data = x;
    struct node prev;
    while (s→ data != y && s→next !==NULL)
    {
        prev = s;
        s = s→next;
    }
    if(s→data ==y)
    {p→next = s;
    prev→next = p;
    }
    else
        printf("No y so no add");
}
```

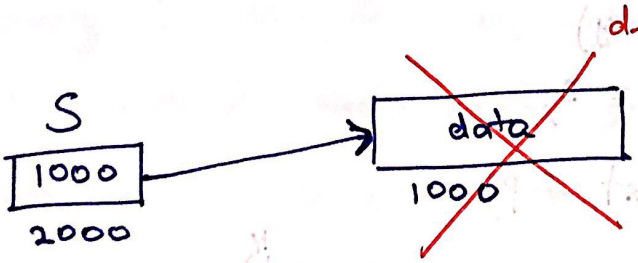
Q. Write a c-program to delete a node which contain data x in a single linked list.

```
delete_x ( int x , struct node *s)
{
    while ( s -> data != x && s -> next != NULL)
    {   prev = s;
        s = s -> next;

    }
    if ( s -> data == x)
    {
        prev -> next = s -> next;

        free(s);        // delete node pointed by s
                        and s is known as dangling
                        pointer
        s = NULL;
    }
}
```

⇒ Dangling Pointer :-

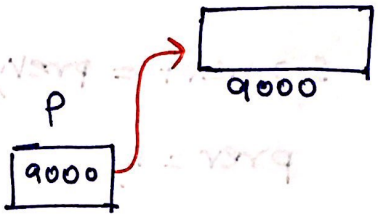


S still points to memory even after memory is de-allocated

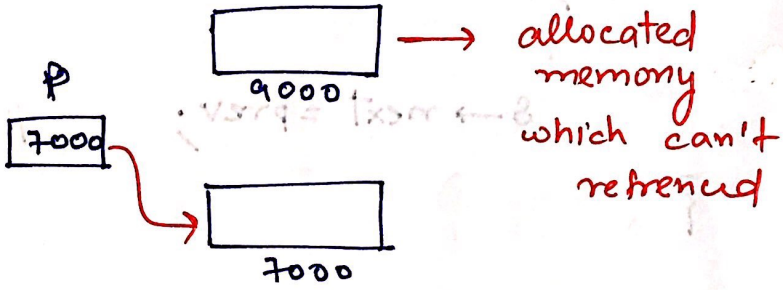
fun^{n}	Available mem	Allocated mem
malloc()	→	
free()	←	

Memory Leakage:-

p = malloc();



p = malloc();



so, when some allocated memory loss its refrence pointer which makes it un accessible, then this memory becomes useless, this is known as memory leakage.

Q. Write a c-program to reverse the linked list.

```
reverse ( struct node * s )
{
    struct node* prev = NULL;
    struct node * after = NULL;
    while (s->next != NULL)
    {
        after = s->next;        // find after node
        s->next = prev;         // set current node next pointer to prev
        prev = s;               // set prev to current
        s = after;              // set current to next node
    }
    s->next = prev;             // last node set to prev
}
```

Q. Write a c-program to find the address of middle node in the given linked list.

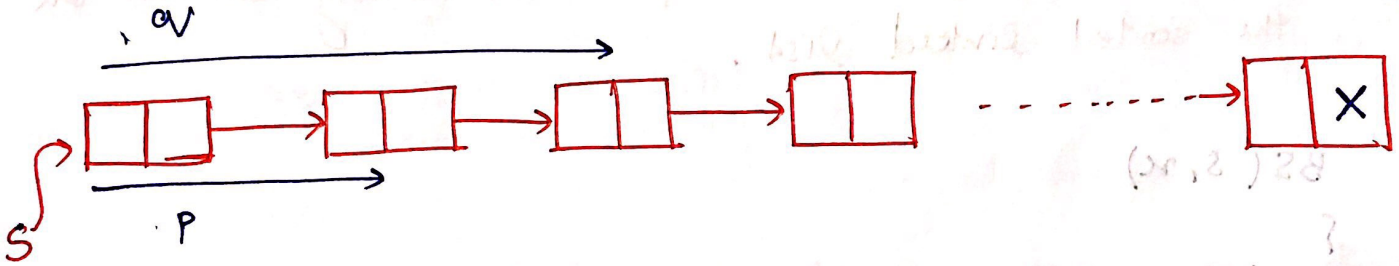
```
struct node * middle_node ( struct node *s)
{
    int count = 0;
    struct node * s1 = s;
```

```
while ( s1 = NULL)
{
    count++;                    // count total nodes
    s = s→next;
}

c = ⌈count/2⌉;

while ( c != 1)
{                               // traverse to middle
    s1 = s1→next;
    c--;
}
return s1;
}
```

OR



since we have to find middle element so q will move with step size of 2 whereas p will move with step size of 1.

middle element = $n/2$

for n/k^{th} element → step size of q is K

so, [q → find size
p → find middle]

```
struct node* middle_node ( struct node *s)
{
    struct node *p, *q;

    p = q = s;
    while ( q != NULL && q->next != NULL && q->next->next
                                          != NULL )
    {
        p = p->next;
        q = q->next->next;
    }
    return p;
}
```

mid algorithm

O(n)

Q. Write a program to perform binary search on the sorted linked list.

```
BS ( s, x)
{
  if( s->next == NULL)
  {
     if( s->data == x)
        return s;
     else
        return NULL;
  }
  else
  {
     p = middle_node (s);
  }
```

O(1)

O(n)

```
if ( P→ data ==x)
    return(p);

else if ( x < P→data)
  {
     P→next = NULL;
     BS(S, x);
  }
else
  {
     BS( P→next, x);
  }
}
```

O(1)

$T(n/2)$

$T(n/2)$

[Recurrence relation of $T(n)$ = $T(n/2) + n$]

so, $\boxed{T(n) = O(n)}$

NOTE:- Time complexity of Binary search on linked list is $O(n)$, while Binary search on array is $O(\log n)$ so,

(BS on LL is not efficient but possible.)

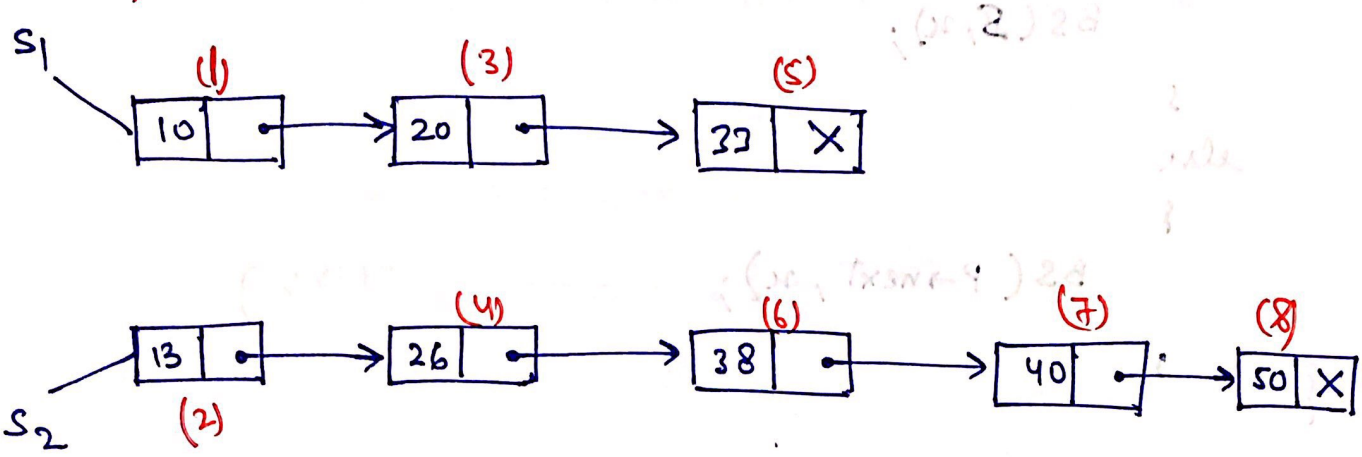
BEST $O(n)$

AVG or WORST $O(n)$

Q: Write a c-program to perform Merge sort on Linked List.

Merge algorithm:-

AVG or WORST:

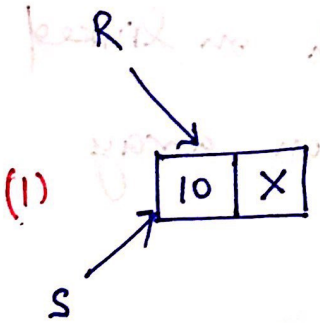


- To merge 2 sorted linked list:

compare ($S_1 \rightarrow$ data , $S_2 \rightarrow$ data)
10 13

Take out node with minimum value.

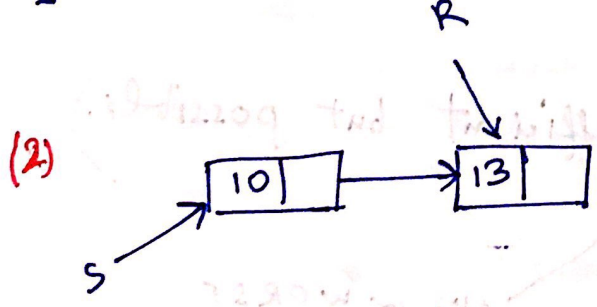
Continue above steps until all nodes are merged.



$R = S_1$ O(1)

$S_1 = S_1 \rightarrow next$

$R = R \rightarrow next$

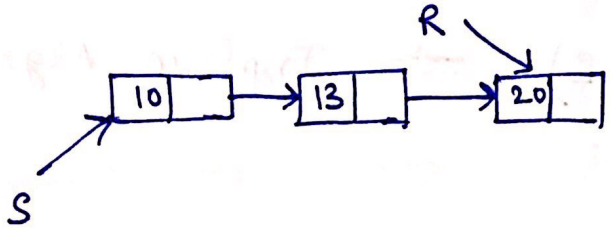


$R \rightarrow next = S_2$ O(1)

$S_2 = S_2 \rightarrow next$

$R = R \rightarrow next$

(3)



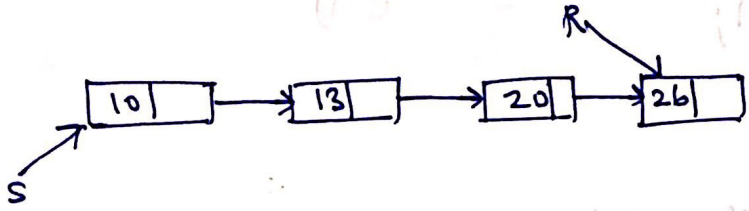
$R \rightarrow next = S_1$

$S_1 = S_1 \rightarrow next$

$R = R \rightarrow next$

O(1)

(4)



$R \rightarrow next = S_2$

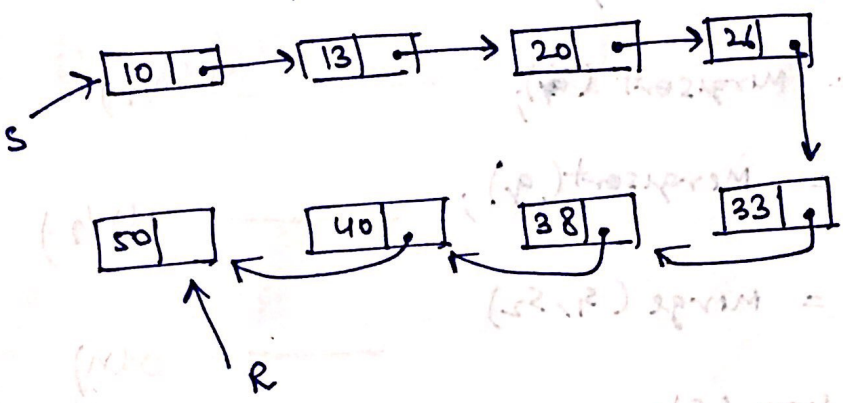
$S_2 = S_2 \rightarrow next$

$R = R \rightarrow next$

O(1)

.
.
.
.

(8)



$R \rightarrow next = S_2$

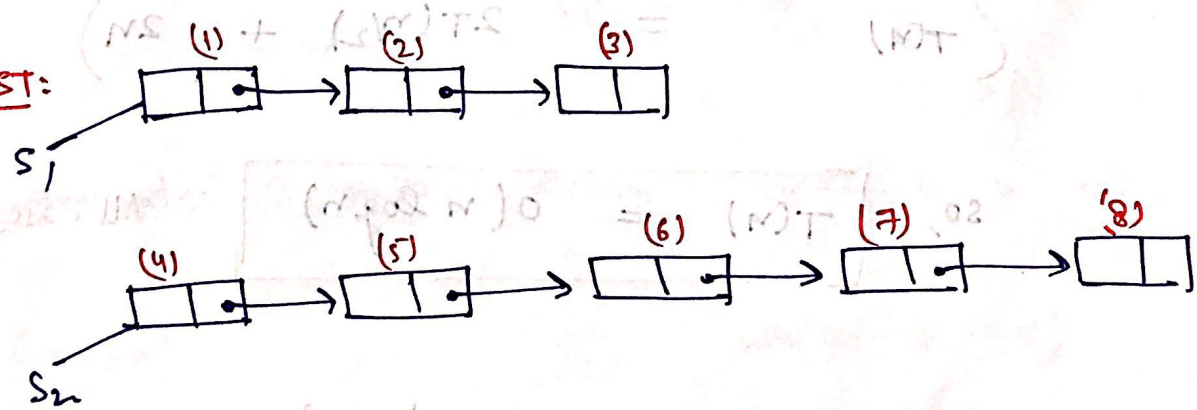
$S_2 = S_2 \rightarrow next$

O(1)

so, time complexity = $O(\ ,\ \times (m+n)$

$= O(m+n)$

BEST:



time complexity = $\min(m, n)$

mergesort (struct node * S) ⇒ Inplace Algorithm

```
{
    if( s->next == null)
        return(S);          ---- O(1)
    else
    {
        p = middle_node(S);     ---- O(n)
        q = p->next;
        p->next = NULL;
        S1 = Mergisort (S);     ---- T(n/2)
        S2 = Mergisort(q);      ---- T(n/2)
Inplace ---- S = merge (S1, S2)
                                ---- O(n)
        return (S);
    }
}
```

Recurrence relation :

$$T(n) = 2T(n/2) + 2n$$

so, $\boxed{T(n) = O(n \log n)}$ All cases

- Since mathematically mergisort algo :

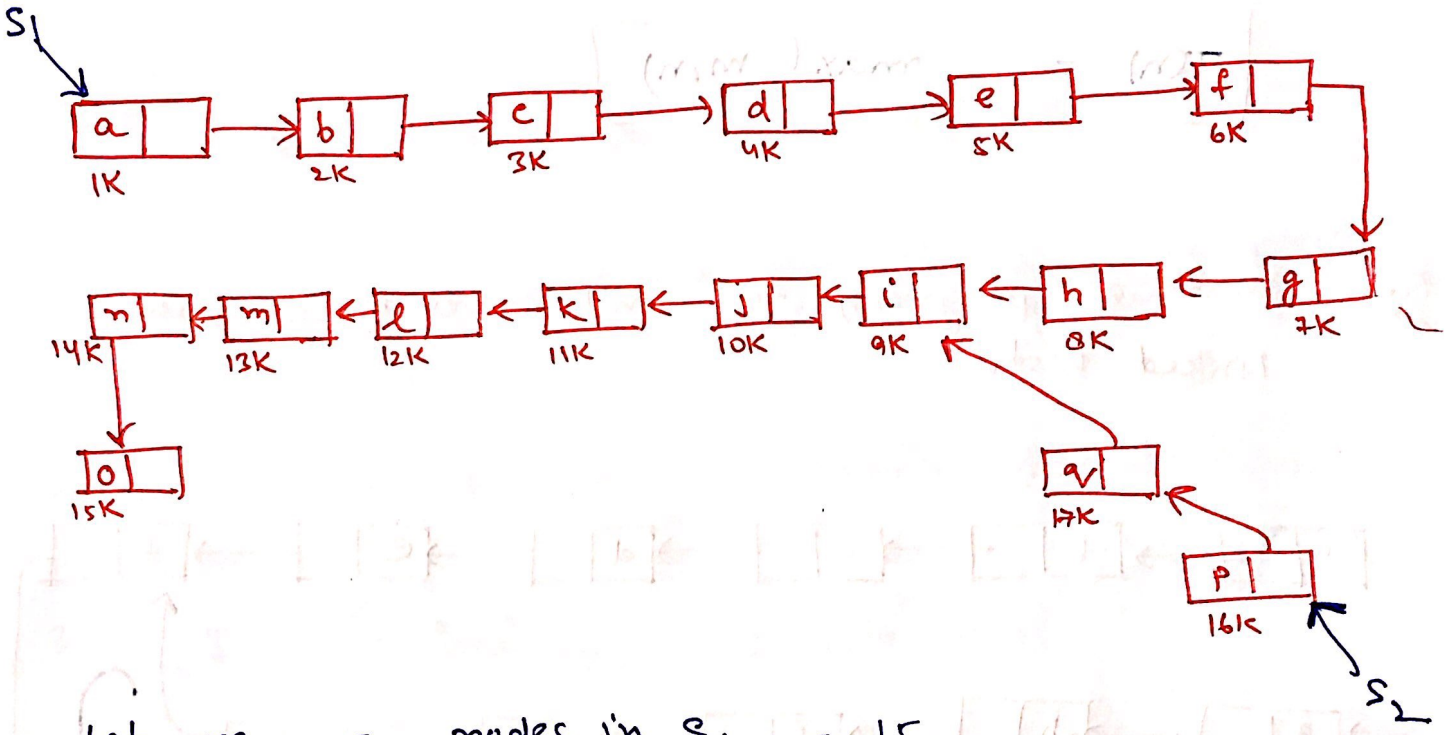
$$T(n)_{LL} > T(n)_{ARR}$$

- But asymptotically :

$$T(n)_{LL} \cong T(n)_{ARR}$$

- But Mergesort in LL is inplace which makes it efficient then outplace in case of array.

Q. Find first common link node in the given linked list.



Let m = nodes in S_1 = 15

n = nodes in S_2 = 9

$D = m - n = 15 - 9 = 6$

- eliminate D nodes from max(m, n).

① if (m > n)

D = m - n;

② while(D != 0)

{

$S_1 = S_1 \rightarrow next$;

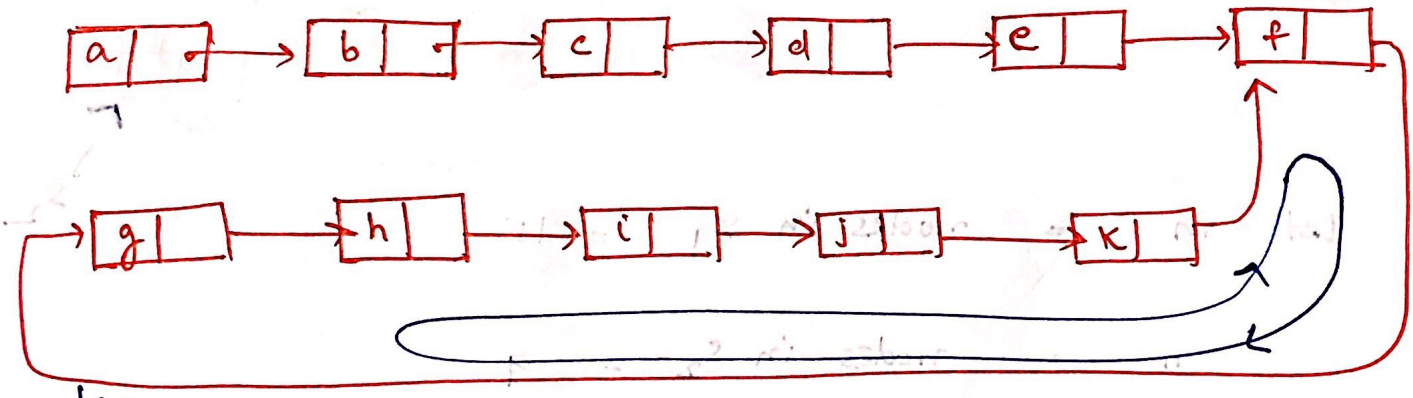
D--;

}

③
```
while ( s1 != s2)
{
    s1 = s1 → next;
    s2 = s2 → next;
}
```

$$[T(n) = \max(m,n)]$$

Q. Write a program to find a cycle in given Linked List.

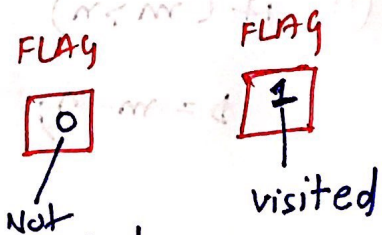


Algo!:-

1. Assume every node contains one more extra field flag which is zero by default.

2. By visiting every node and check the flag:

```
if ( flag == 0)
    flag = 1;
else
    printf("loop is found");
```



Time complexity $T(n) = O(n)$ single span of LL

But space complexity ↑ as extra flag data is used by every node.

Algo2:

1. Take two pointer p and q

   p moves with larger step size compare to q.

2. Now Both p and q starts simuntaneously

3. If $(p == q)$ ⟶ cycle exist

   else cycle not exist

[$T(n) = O(n)$

space complexity will not increase]

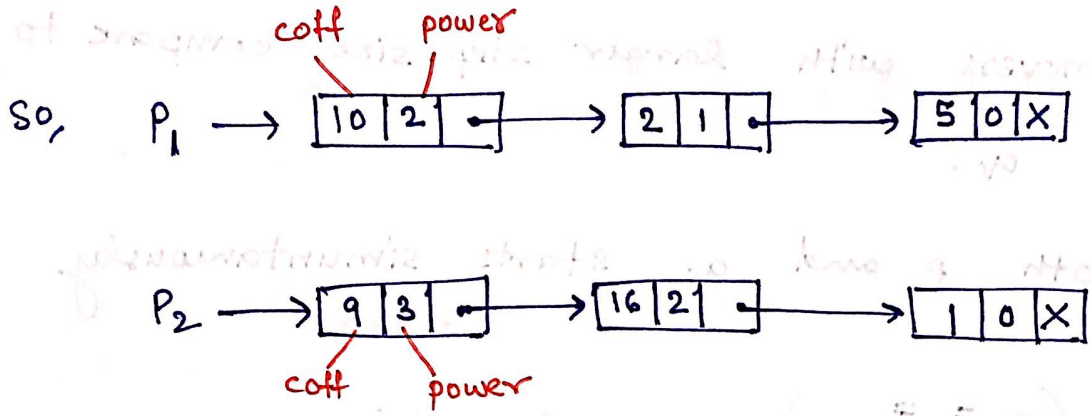
p		q
1000	≠	1000
3000	≠	2000
5000	≠	3000
7000	≠	4000
⋮		⋮
6000	=	6000 ⇒ cycle exist

Polynomial addition :-

Using linked list we can simulate addition of two polynomial. $P_1(x)$ & $P_2(x)$.

Let $P_1(x) = 10x^2 + 2x + 5$

$P_2(x) = 9x^3 + 16x^2 + 1$



Apply merge algorithm : $(T(n) = O(n))$

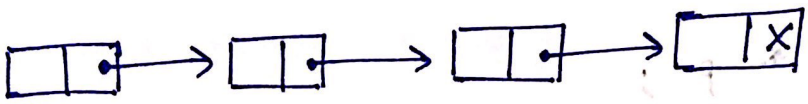
```
if ( P1 → pow > P2 → Pow)
{
        addnode (P1)
        P1 = P1 → next;
}
else if ( P1 → Pow < P2 → Pow)
{
        addnode (P2);
        P2 = P2 → next;
}
```

```
else
{  P1 → coff
   = P1 → coff +
        P2 → coff;

   addnode (P1);
   deletenode (P2);
}
```

Drawbacks of singly linked list:-

(i)



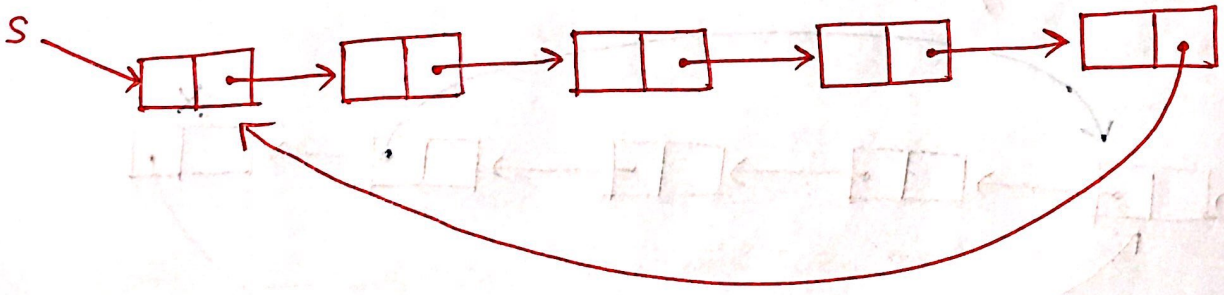
we can't move backward as there is only forward pointer.

(ii) In last node forward pointer is always NULL i.e we are not utilizing it properly.

To eliminate above drawbacks we implement circular single linked list.

Circular Single linked list:-

Last node (next) part contains first node address

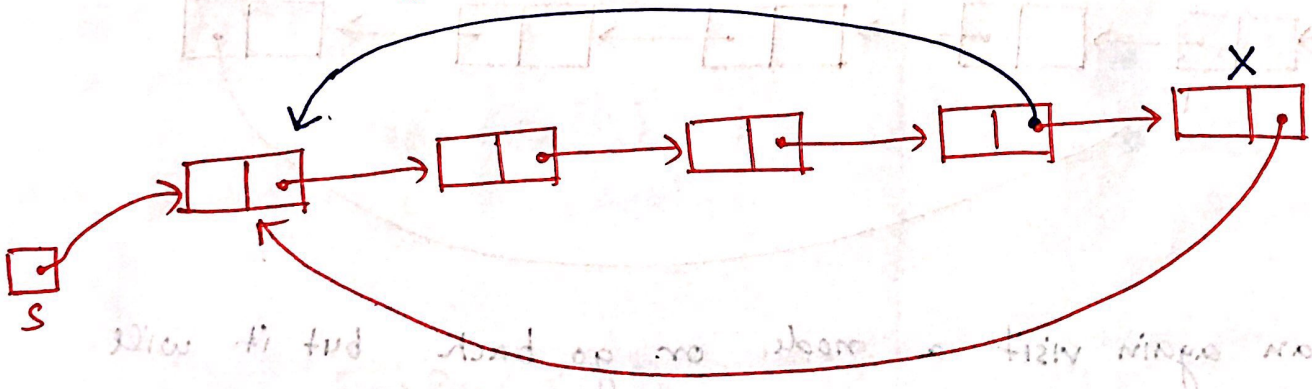


- we can again visit a node or go back but it will take O(n) time.

Q. Write a C++ program to insert a node with data x at the starting of circular single linked list.

```
insertnode_start (int x , struct node * s)
{
    struct node * p ;
    p = (struct node *) malloc ( sizeof(struct node));      ] O(1)
    p→data = x;
    p→next = s;
    s1 = s;
    while (s1→next != s)                                    ] O(n)
    {
        s1 = s1→next;
    }
    s1→next = p;                                            ] O(1)
    s = p;
}
```

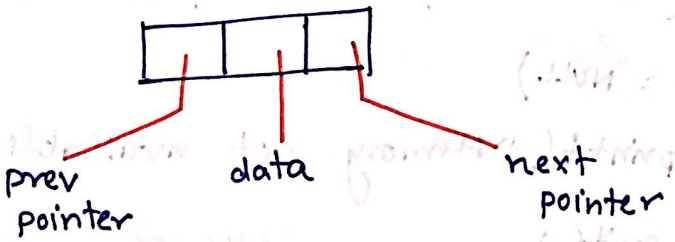
Q. Write a program to deleting last node in a given circular single linked list.



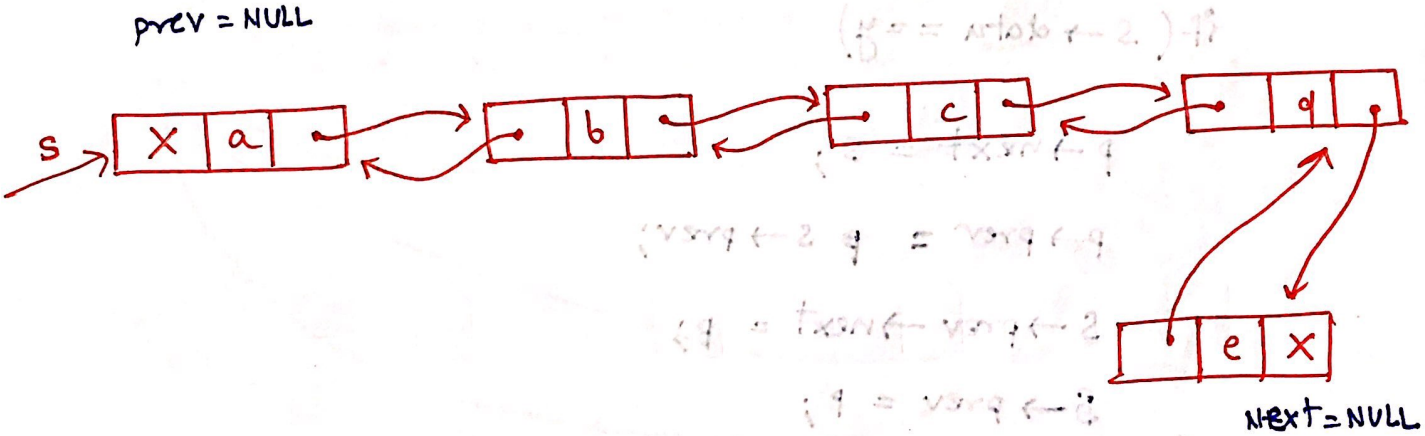

```
delete_lastnode (    struct node * s)
{
    s1 = s;
    while (s1→next != s)                                    ] O(n)
    {
        p = s1;
        s1 = s1→next;
    }
    p→next = s;
    free (s1);                                              ] O(1)
    s1 = NULL;
}
```

Doubly Linked List :-

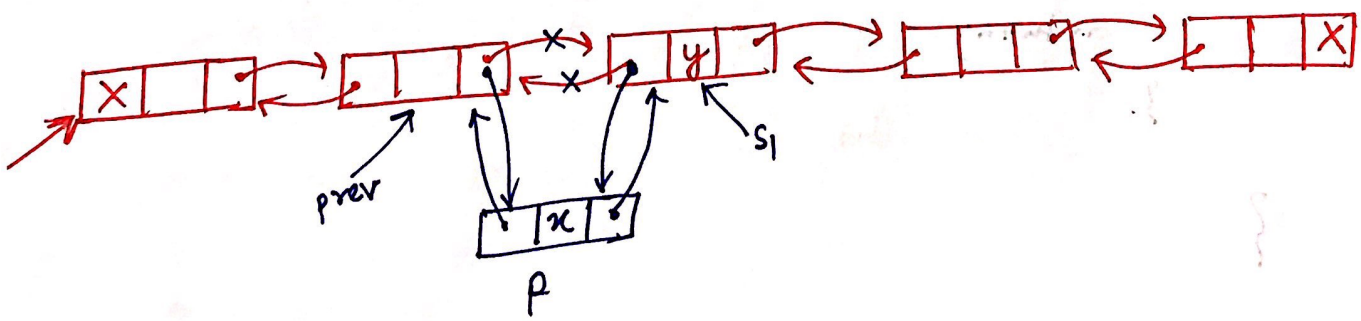
Every node contain 3 fields :




```
Struct DLL
{
    struct DLL *pre;
    int data;
    struct DLL *next;
};
```

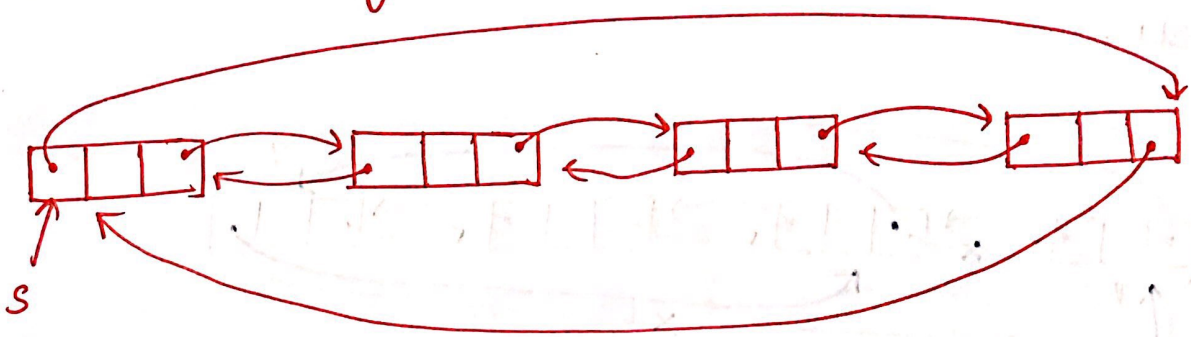


Q. write a program to add a node node with data x before node which contains y.



```
insert_node ( struct DLL* s , int x)
{
    struct DLL * p;
    p = (struct DLL*) malloc ( sizeof(struct DLL));
    if(p == NULL)
        { printf( "Memory not available ");
          exit(1);                    // Terminate program
        }
    p→data = x;
    while ( s→data != y && s→next != NULL)
    {
        s = s→ next;
    }
    if( s→data == y)
    {
        p→next = s;
        p→prev =  s→prev;
        s→prev→next = p;
        s→prev = p;
    }
    else
    {
        printf(" can't add");
        return;
    }
}
```

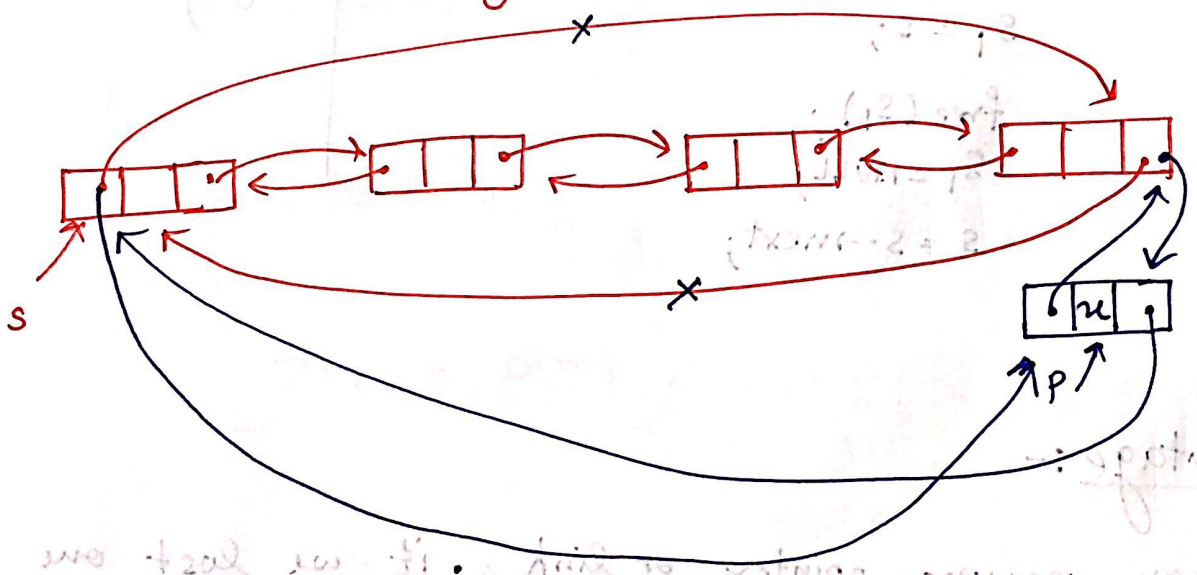
Circular Doubly Linked List :-



firstnode → prev = lastnode

lastnode → next = firstnode

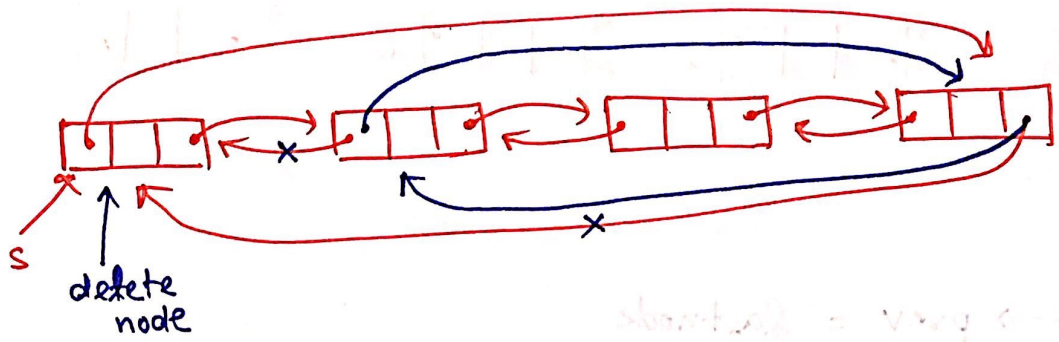
Q. Write a program to add node (x) at the end of circular doubly linked list.




```
insertnode_end ( struct node *s, int x)
{
    struct DLL *p = (struct DLL *) malloc ( size of (struct DLL));
    p → data = x;
    p → next = s;
    p → prev = s → prev;
    s → prev → next = p;
    s → prev = p;
}
```

O(1)

Q. Write a program to delete first node in circular DLL.

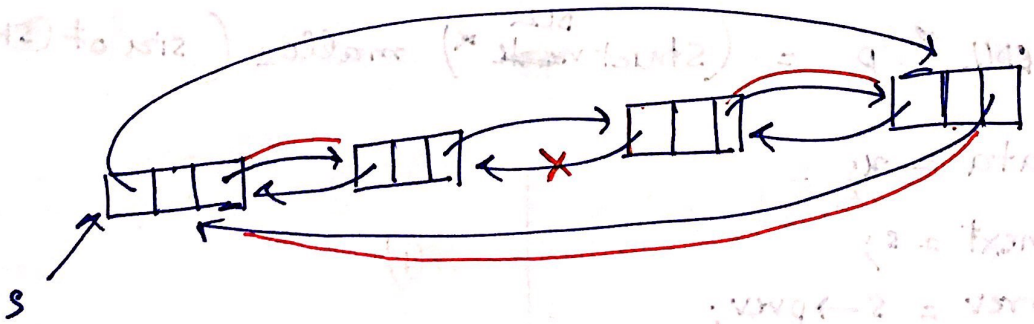



```
delete_firstnode ( struct DLL * s)
{
        s → prev → next = s → next;
        s → next → prev = s → prev;
         s1 = s;
         free (s1);
         s1 = NULL;
         s = s → next;
}
```

O(1)

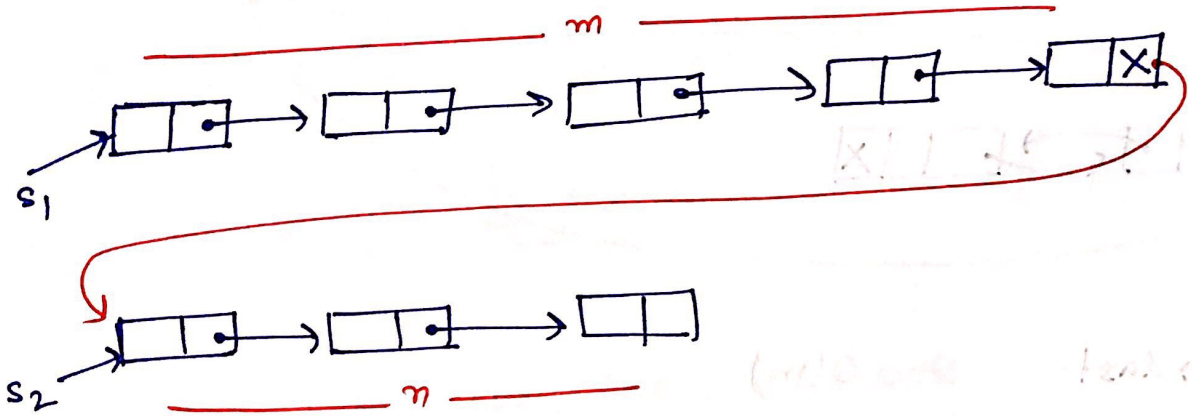
Advantage :-

- we can recover pointer or link, if we lost one pointer or link.



Concatenation in Linked List :-

(i) Singly Linked List :-




```
while ( s1 → next != NULL)
{
    s1 = s1 → next;
}
```

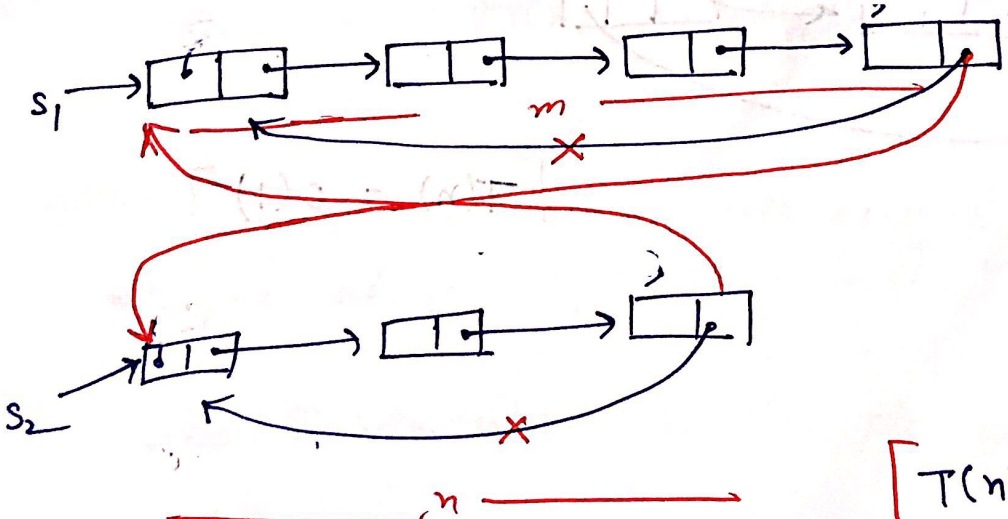
] O(n)

```
s1 → next = s2 ;
```

// $s_1 \cdot s_2$

$$[T(n) = O(m)]$$

(ii) Circular singly linked List :-



$s_1 \rightarrow$ last $O(m)$

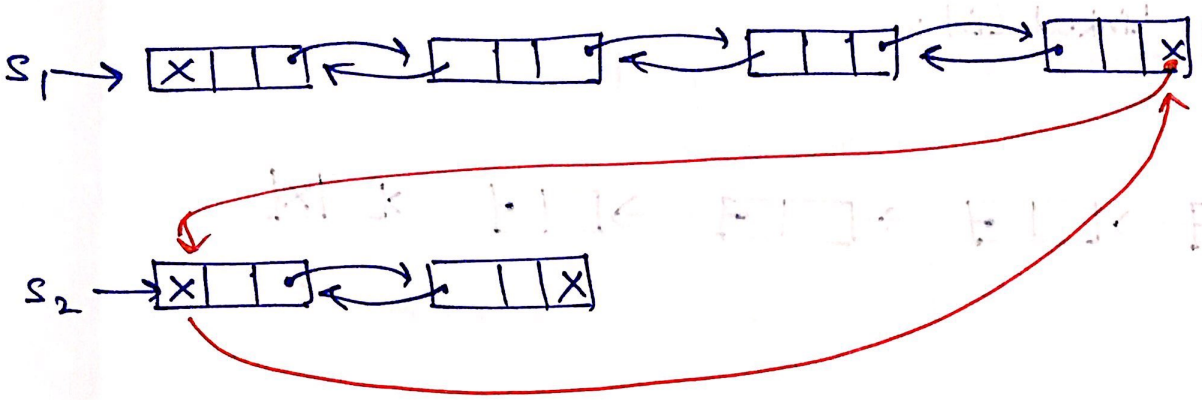
$s_1 \rightarrow next = s_2$

$s_2 \rightarrow$ last = $O(m)$

$s_2 \rightarrow next = s_1$

$$[T(n) = O(m+n)]$$

(iii) Doubly Linked List:-



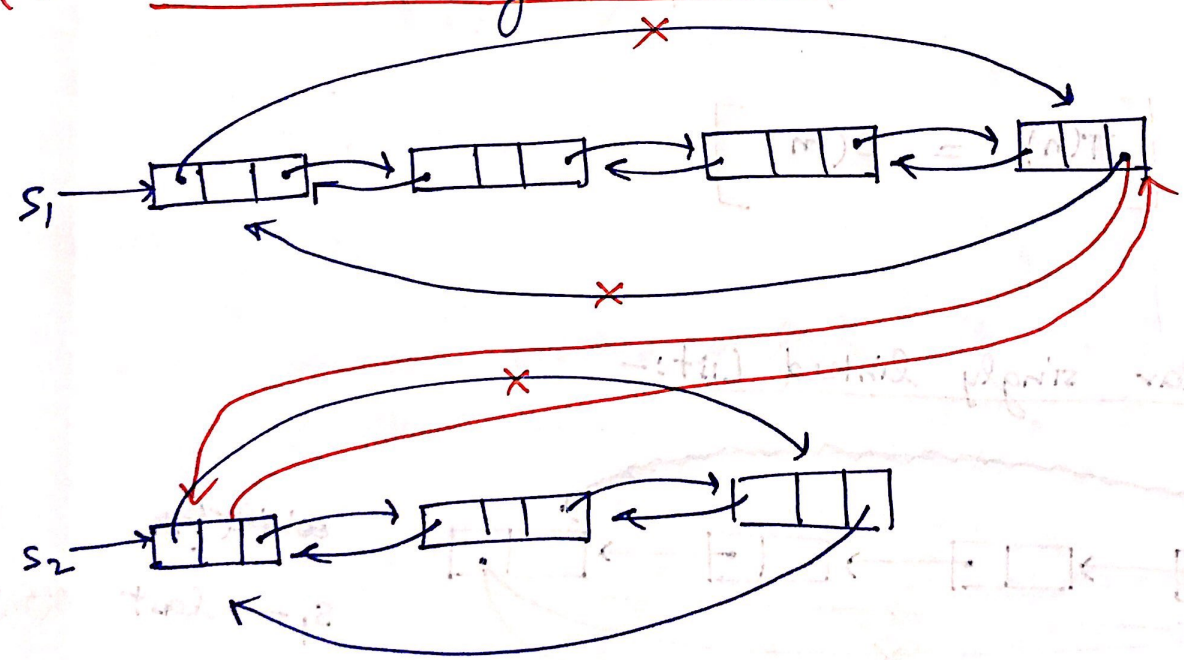
$S_1 \rightarrow$ last — O(m)

$S_1 \rightarrow$ next = S_2

$S_2 \rightarrow$ prev = S_1

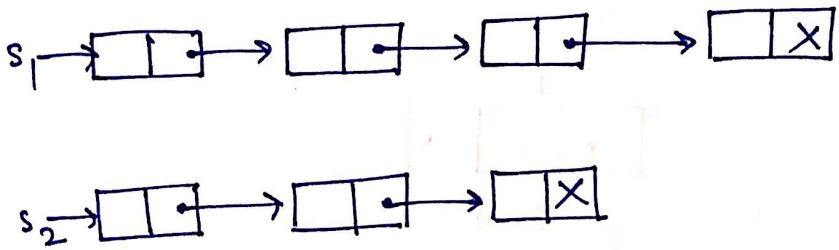
[T(n) = O(m)]

(iv) Circular Doubly Linked List:-



[T(n) = O(1)]

⇒ Intersection & Union in Linked List :-



Union ⇒ sort two linked lists $O(n \log n)$

merge two sorted LL $O(n)$

$$T(n) = O(n \log n) + O(n)$$
$$= O(n \log n)$$

Intersection ⇒ sort two linked list $O(n \log n)$

apply or merge (only common) $O(n)$

$$T(n) = O(n \log n)$$

NOTE :- T(n) for union & intersection is same for all types of LL.

Union & intersection are the slowest operation on LL.