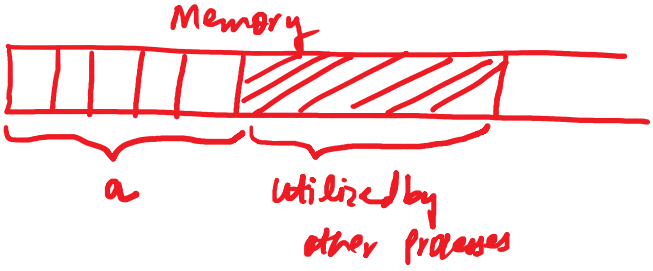
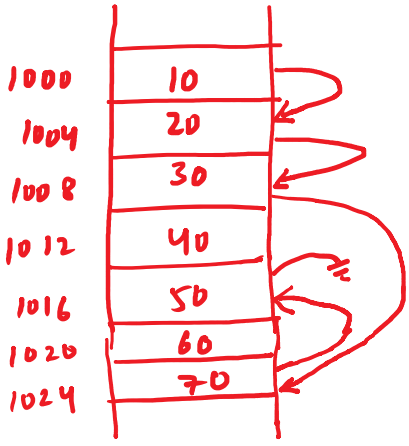
Linked List :-

Static Array – int a[100]; Wastage of memory

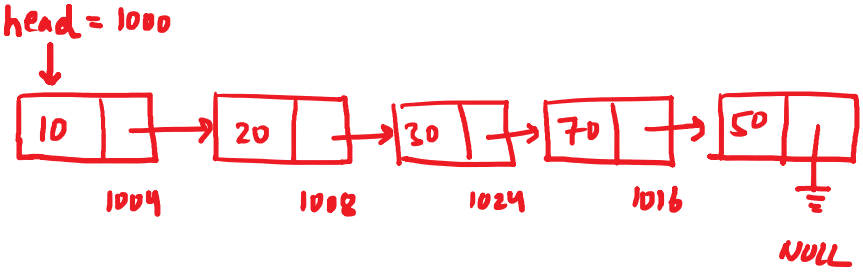
Dynamic Array – Array use Contiguous memory allocation



Linked List uses non-Contiguous memory allocation.



Singly Linked List :-



node

sizeof(int *) =
sizeof(float*) =
sizeof(char *) =
} sizeof(int)

```
struct node
{
    int data;
    struct node *next;
```

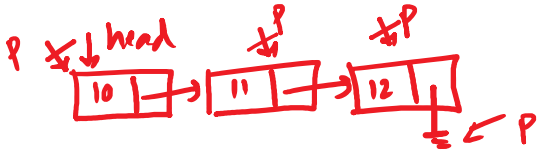
Self-referential pointer

```
    struct node *next;
};
typedef struct node NODE;
```

```
int a = 10;
typedef int ABC;
ABC a=10;
```

Simple Linked List

1) List Traversal :-




```
void display (NODE *head)
{
    NODE * p = head;
    while (p != NULL)
    {
        printf ("%d", p->data);
        P=p->next;
    }
}
```

2) Insertion :-

At start | At end | At a particular position | After a given node | Before a given node

```
NODE * insert_start (NODE * head, int x)
{
    NODE * q;
    q = (NODE *) malloc (sizeof (NODE));
    q->data = x;
    q->next = head;
```

In Main function :-

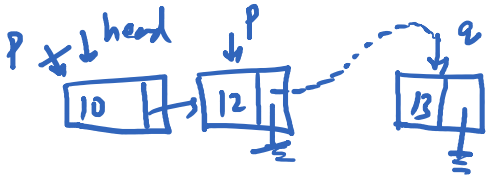
```
    q->next = head;
    return q;
}
```

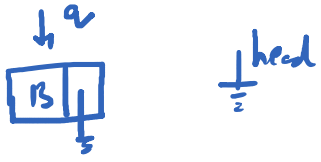
$\theta(1)$

In Main function :-

```
head = insert_start(head, 13);
```

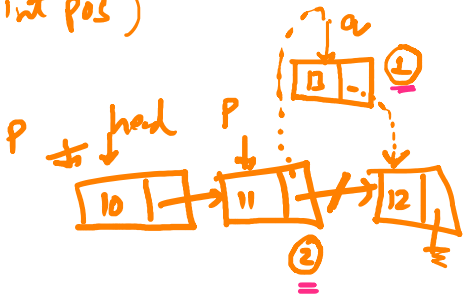
```
NODE * insert_end (NODE * head, int x)
{
    NODE * q, * p = head;
    q = (NODE *) malloc (sizeof (NODE));
    q -> data = x;
    q -> next = NULL;
    if head = NULL
        return q;
    while (p -> next != NULL)
        p = p -> next;
    p -> next = q;
    return head;
}
```

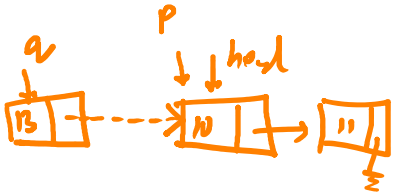




$\theta(n)$

```
NODE * insert_position (NODE * head, int x, int pos)
{   int i;
    NODE * q, * p = head
    q = (NODE *) malloc (sizeof (NODE));
    q -> data = x;
    for (i = 2; i < pos; i++)
        p = p -> next;
①   q -> next = p -> next;
②   p -> next = q;
    return head;
}
```






```
if (pos == 1)
{
    q -> next = head;
    return q;
```

```
    return head;
}
```

```
    q->next = head;
    return q;
}
```

Best case is when pos = 1 $\Omega(1)$

Worst Case is when pos = n+1 $O(n)$

Average case $\theta(n)$

At Start

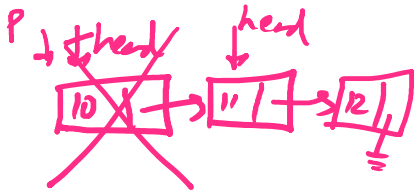
At End

At an arbitrary position

Deletion

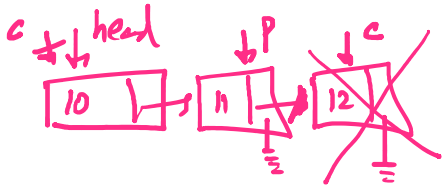
- First node
- Last Node
- Arbitrary Node at a position
- After a given node
- Before a given node

```
NODE * delete_start (NODE * head)
{
    NODE * p = head;
    head = head -> next;
    free (p);
    return head;
}
```

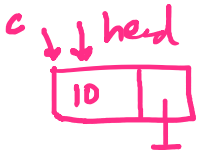


Include the concept of underflow.

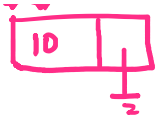
```
NODE * delete_end ( NODE * head)
{
    NODE * p , * c = head;
    while (c -> next != NULL)
    {
        p = c;
        c = c -> next;
    }
```



Include the concept of underflow



```
}
p->next = NULL;
free (c);
return head;
}
```



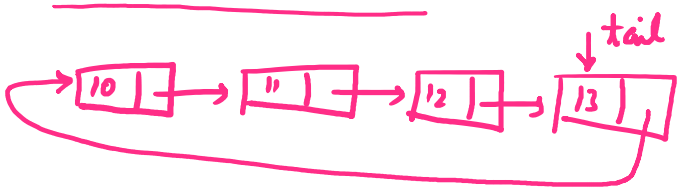

```
if(head->next == NULL)
{
    free (head);
    return NULL;
}
```

Deletion at a position

$\Omega(1)$	$O(n)$	$\theta(n)$
pos = 1	pos = n	pos = arbitrary position

Varients of Linked Lists :-

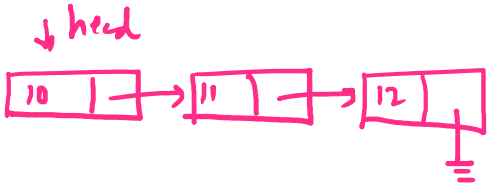
1) Circular Linked List :-



Any node can be the starting point. Instead of head, tail pointer can be used.

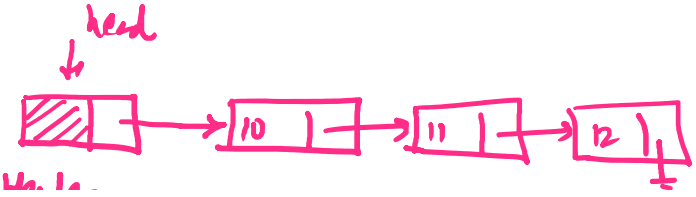
Insertion / Deletion of the last node can be done in constant time.

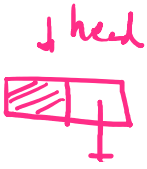
2) Header node based Linked list :-

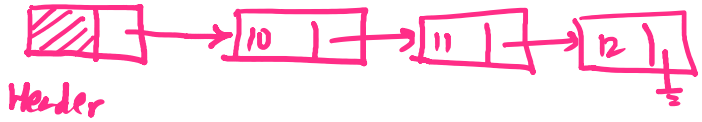


Standard SLL

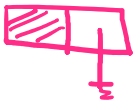
empty SLL





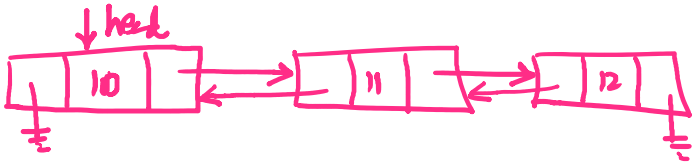


Header Node

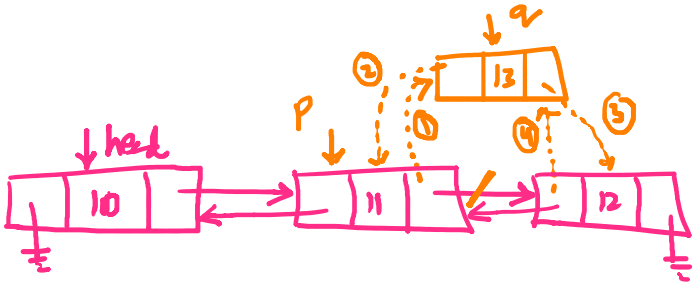


Global information

3) Doubly Linked List :-



Moving in both the directions.



② : q → prev = p ;

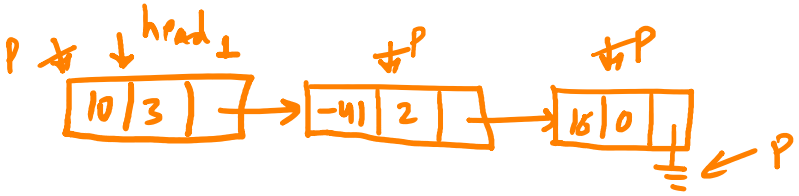
③ : q → next = p → next ;

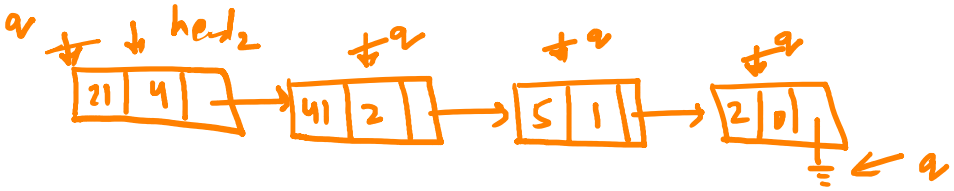
① : p → next = q ;

④ : (q → next) → prev = q ;

Polynomial Representation :-

$f(x) = 10x^3 - 41x^2 + 16$

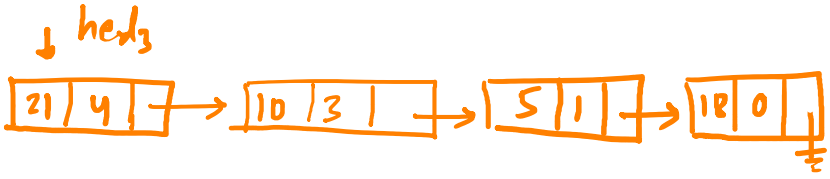




$10x^3 - 41x^2 + 16$

$+ \quad 21x^4 + 41x^2 + 5x + 2$

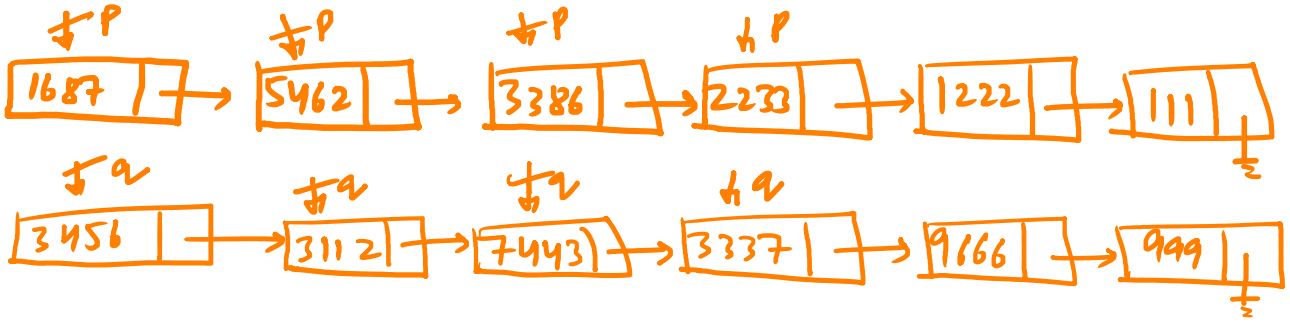
$21x^4 + 10x^3 + 5x + 18$



→ Addition of two very large numbers

p = 1111 222 22 3333 8 6 54 62 16 87

99 99 666 333 77 44 33 11 23 456




```
  1687
+ 3456
------
  4143
```

Carry = 0

```
  3386
+ 7443
------
 10829
```

Carry = 1

0829

. . . .