Linked List

int A[5];

int *P = new int[5];

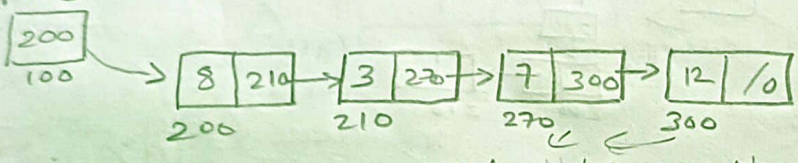
-> size can ↑ (or) ↓ in LL

-> Node = elet pointer.

-> for each element in LL memory is allocated separately if we remove an element then memory also.

def:- LL is a collection of Nodes where each node contains data & pointer to next node.

first (or) Head.



all nodes aren't side by side but through links they maintain. That continuity

-> LL created in Heap (bcz it's not static)

-> bcz they can dynamically expand when needed without worrying about memory consumption.

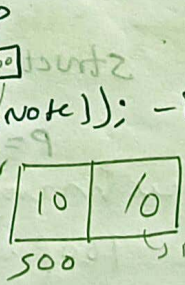
Struct Node *P;

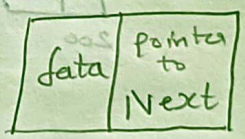
P = (struct Node*) malloc (sizeof (struct Node)); - in c lang

P = new Node; - in c++.

P -> data = 10;

P -> next = 0;





Struct Node

{

int data;

struct Node *next;

};

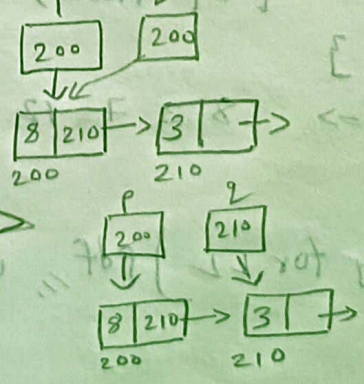
More about LL

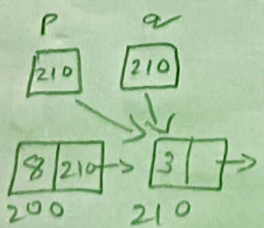
Struct Node *P, *Q;

1. Q = P;

2. Q = P -> Next;

3. P = P -> next;





Struct Node *P = 0/NULL;

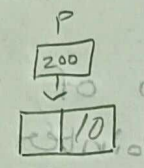
1. P → 0

2. P → 200 → [| 210] → [|] → 200 210

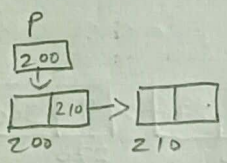
if (P==NULL)
if (P==0)
if (!P) → True if it is NULL

if (P!=NULL)
if (P!=0)
if (P)

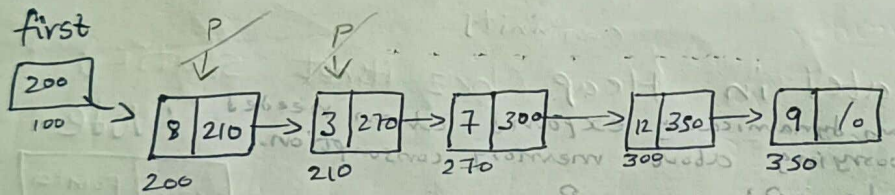
if (P->next==NULL)



if (P->next!=NULL)



Display LL



Struct Node *P = first; (Traversing)

P = P->next;

we don't know how many times; to repeat it

So use: while loop.

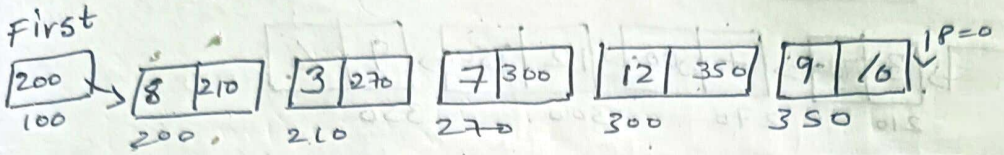
```
{
while(P!=0)  → (or) NULL
{   printf("%d", P->data);
    P = P->next;
}
```

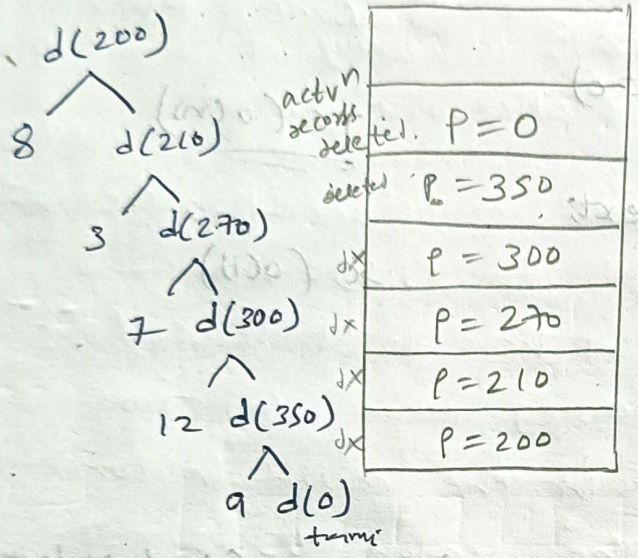
(o/p => 8 3 7 12 9)

* (Code Display for LL) pdf // understand ✓

Recursive Display of LL

→ recursion uses system stack.





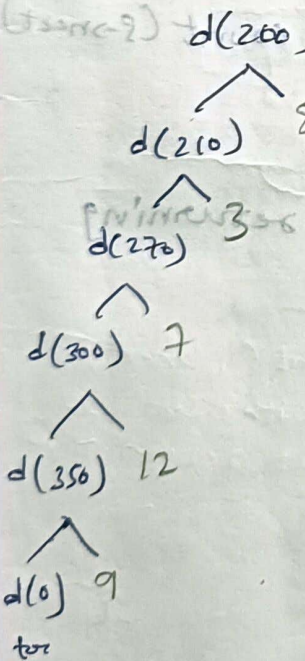

```
void Display (struct Node *p)
{
    if (p != NULL)
    {
        printf("%d", p->data);
        Display(p->next);
    }
}
```

Display(first).

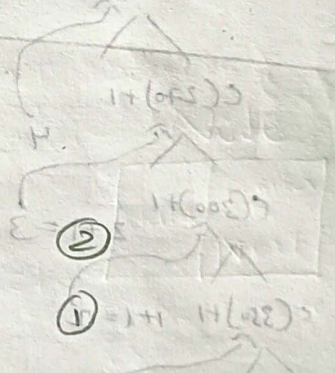
TC = O(n)

SC = O(n) → n+1

o/p = 8, 3, 7, 12, 9



stack same



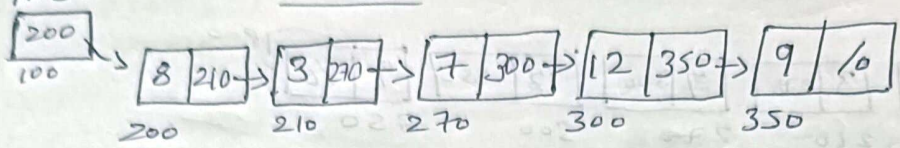
o/p = 9, 12, 7, 3, 8

TC) same

SC)

(code) pdf, uniform

Counting Nodes in a LL

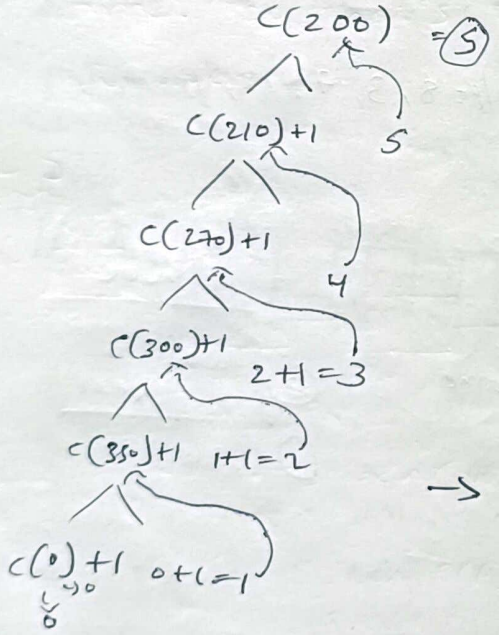



```
int count(struct Node *p)
{ int c=0;
  while(p!=0)
  {
    c++;
    p=p->next;
  }
  return c;
}
```

TC (O(n))

SC (O(1))

Recursive process




```
int count (struct Node *p)
{
  if(p==0)
    return 0;
  else
    return count(p->next)+1;
}
```

→ add 1 done at returning time

T → O(n)

S → O(n)

Sum of all Elements in a LL

Ass (first)

```
int Add(struct Node *p)
{
  int sum=0;
  while(p)
  {
    sum=sum+p->data;
    p=p->next;
  }
  return sum
}
```

Now using recursion

```
int Add(struct Node *p)
{ if(p==0)
    return 0;
  else
    return Add(p->next)+p->data;
}
```

code for count & Sum PDF ✓

Max ele in a LL

```
int max(Node *p)
{
    int m=MIN_INT;
    while(p)
    { if(p->data>m)
        m=p->data;
      p=p->next;
    }
    return m;
}
```

max = MIN_INT ~~& (12)~~

recursive

```
int max(Node *p)
{   int x=0;
    if(p==0)
        return MIN_INT;
    else
    {
        x=max(p->next);
        if(x>p->data)
            return x;
        else
            return p->data;
    }
}
```

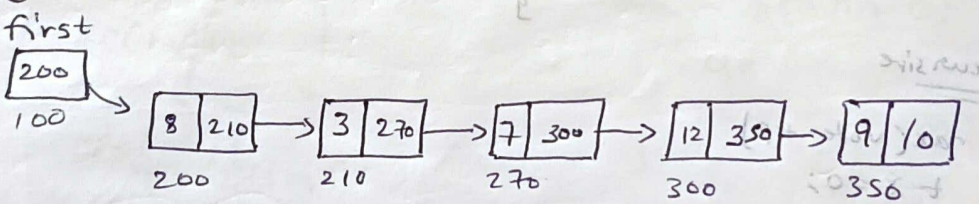
code PDF ✓

Searching in a LL

1. Linear search → it checks one by one.
2. Binary search → This will work only on Sorted list & it'll check in the middle of a list. if ele is found, it's okay. if small check Left otherwise right

That BS procedure we cannot perform upon LLs, bcz we cannot directly go into the middle of a LL. we have to traverse from first node. So Traversing takes O(n) Time we cannot reach in the middle in constant time. So that's the reason bs is not suitable.

LS




```
Node * search(Node *P, int key)
{
    while(P!=NULL)
    {
        if(key== P->data)
        {
            return P;
        }
        P=P->next;
    }
}   return NULL;
```

recursive

```
Node* Search(Node *p, int Key)
{
    if (p==NULL)
        return NULL;
    if (Key == p->data)
        return p;
    return Search(p->next, Key);
}
```

Improve Searching in LL

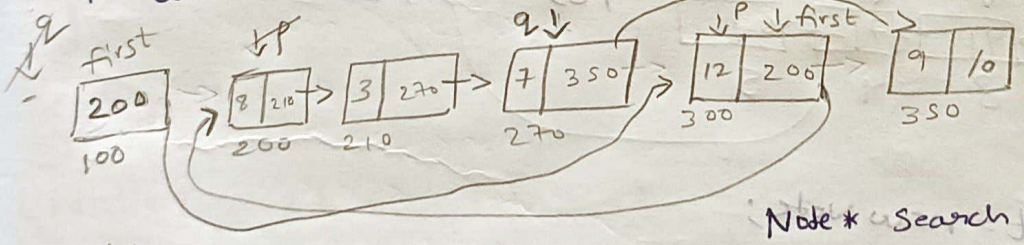
1. Transposition
2. Move to Head/front

-> These are useful, if we again search for a key we can found it in less time.

-> So transposition was a method where we interchange the value with the previous value

-> Move to head means the key value should be brought in the beginning.

-> in LL we don't do transposition bcz we avoid movement of data we prefer movement of nodes. so its better if we take out this node & add in into front.

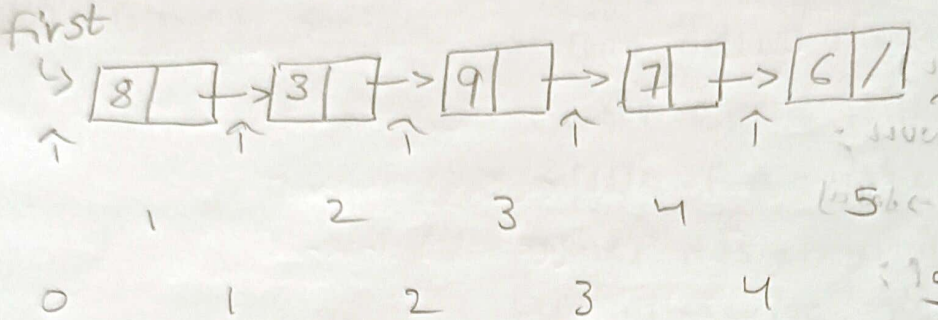


Key = 12

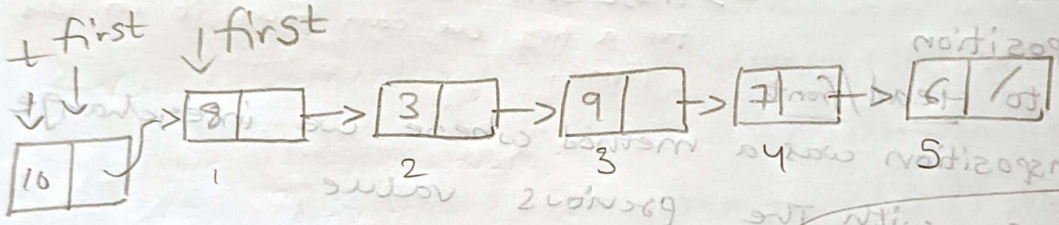
```
Node * Search (Node *p, int Key)
{
    Node *q = NULL;
    while(p!=NULL)
    {
        if (Key == p->data)
        {
            q->next = p->next;
            p->next = first;
            first = p;
        }
        q = p;
        p = p->next;
    }
}
```

code searching in LL pdf ✓

Inserting in a LL



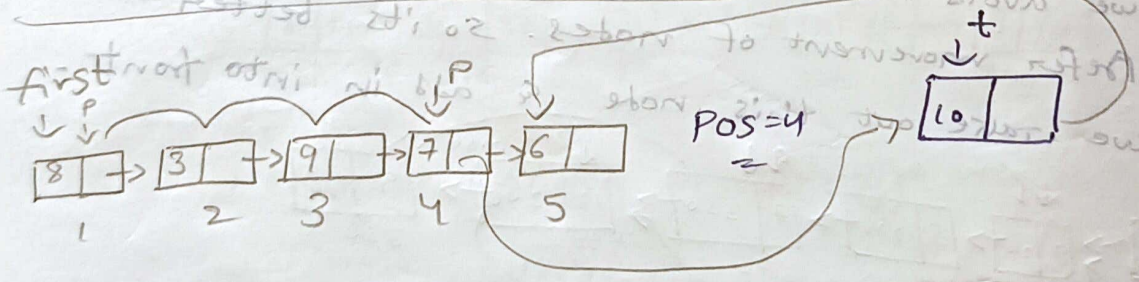
1. insert before first node
2. inserting after given position.



Before first node

```
Node *t = new Node;
t->data = x;
t->next = first;
first = t;
```

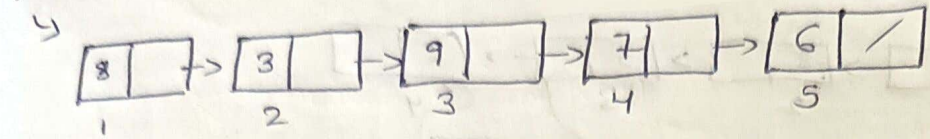
O(1) min
max




```
Node *t = new Node;
t->data = x;
p = first;
for(i=0; i<pos-1; i++)
{
    p = p->next;
}
t->next = p->next;
p->next = t;
```

O(n). max

O(1). min




```
void Insert(int pos, int x)
{
    Node *t, *p;
    if (pos==0)
    {
        t=new Node;
        t->data=x;
        t->next=first;
        first=t;
    }
    else if (pos>0)
    {
        p=first;
        for(i=0; i<pos-1 && p; i++)
            p=p->next;
        if (p)
        {
            t=new Node;
            t->data=x;
            t->next=p->next;
            p->next=t;
        }
    }
}
```

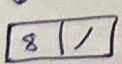
Code ✓ insert for LL

Creating a LL using insert

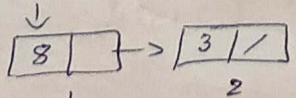
first
↓
=

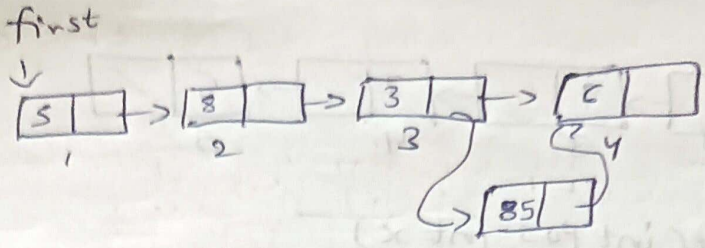
Pointer first in null

Insert(0,8); (when NULL we can insert at only '0')

first
↓


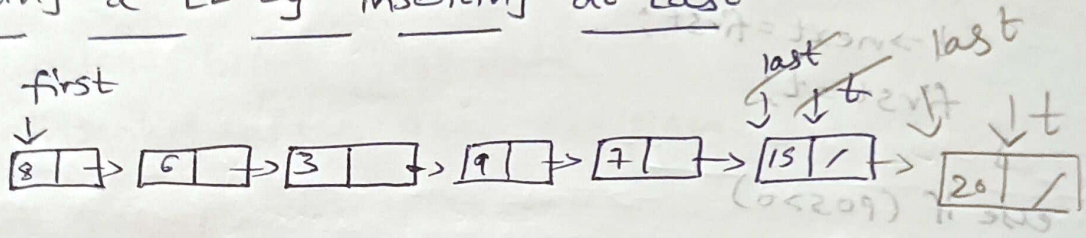
Insert (1, 3);

first
↓


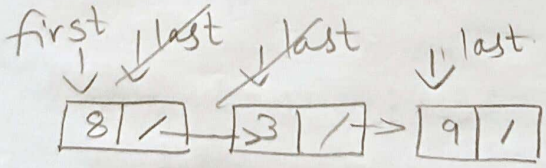



```
Insert(2, 6);
Insert(0,5);
Insert(3,85);
```

creating a LL by inserting at Last

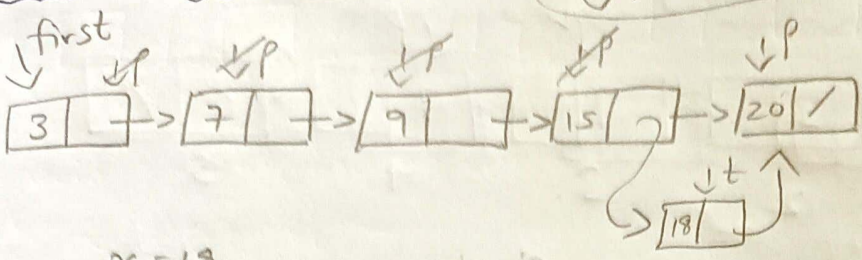



```
Void Insert(last, int x)
{
   Node *t=new Node;
   t->data=x;
   t->next=NULL;
   if (first==NULL)
   {
      first=last=t;
   }
   else
   {
     last->next=t;
     last=t;
   }
}
```




```
Insertlast(6)
Insertlast(3)
Insertlast(9)
```

Inserting in a Sorted LL

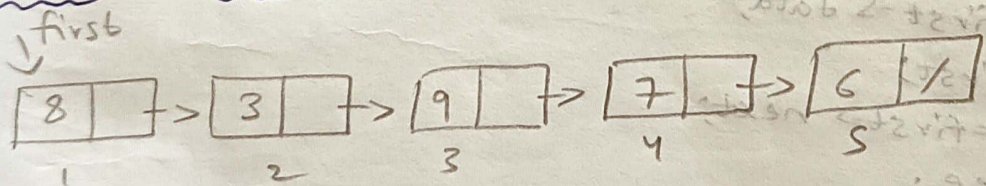



```
p = first;
q = NULL;
while(p && p->data<x)
{
  q = p;
  p = p->next;
}
t = new Node;
t->data = x;
t->next = q->next
q->next = t;
```

-> Some edge cases code still not done in program code it will be completed.

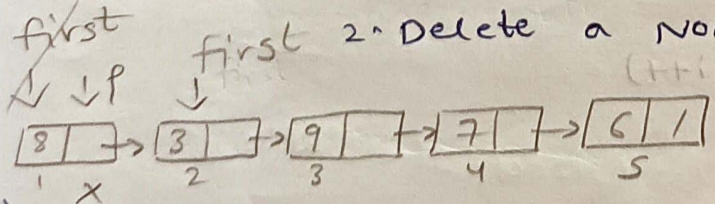
code insert in sorted LL ✓

Deleting from LL



1) Delete first Node

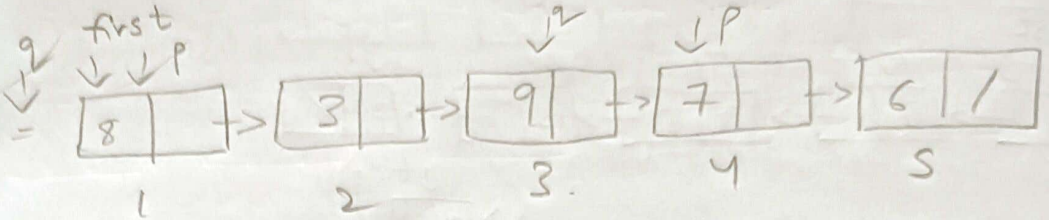
2. Delete a Node at given position.



1)

```
Node *p = first;
first = first->next;
x = p->data;
delete p;
```

2) pos = 4.




```
Node *p = first;
Node *q = NULL;
for (i=0; i<pos-1; i++)
{ q = p;
  p = p->next;
}
q->next = p->next;
x = p->data;
delete p;
```

(min o(1), max o(n))

function

```
int Delete(int pos)
{
  Node *p, *q;
  int x=-1, i;
  if (pos==1)
  {
    x = first->data;
    p = first;
    first = first->next;
    delete p;
  }
  else
  {
    p = first;
    q = NULL;
    for (i=0; i<pos-1 && p; i++)
    {
      q = p;
      p = p->next;
    }
    if (p)
    {
      q->next = p->next;
      x = p->data;
      delete p;
    }
  }
  return x;
```