Linked List 1

DS advantages & Vectors

1) Deleting an element is actually a pain compar for Last Index

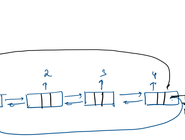
In your ram 4 4GB remembering
int : 4B : to array size
4GB are not continuous location
We cannot allocate array size + 10

```
class Node {
  public:
    int data;
    Node *next;
    Node(int n) {
      this->next = NULL
      this->data = n
    }
    int main() {
      Node *n = new Node(10);
    }
```

Pointer + Node type

Constructor — To initialize data members

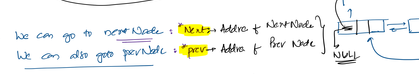
Self Referencing Pointer | Points to Same Node



n → next = new Node(20)

Linked List :

SLL :

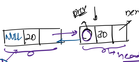
CSLL : 

DLL : We can go to next Node : *next → Address of Next Node
We can also goto prev Node : *prev → Address of Prev Node

If CDLL
4 → next = 1
1 → prev = 4

```
class Node {
  public:
    int data;
    Node *next, *prev;
    Node(int n) {
      this->data = n
      this->next = NULL
      this->prev = NULL
    }
}
```

this points to itself



head → next

head → data
head → next
head → prev

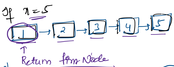
head → next = new Node(20)
head → next → prev = head

head is a pointer

Node *head = new Node(20);

CDLL :

If n = 5



Return from Node

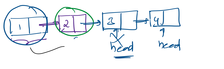
```
class Node {
  public:
    int data;
    Node *next;
    Node(int n) {
      this->data = n
      this->next = NULL
    }
};
```

n ≥ 1

```
Node CN(int n) {
  Node *head = new Node(1);
  Node *temp = head;
  for( i = 2; i <= n; i++) {
    head->next = new Node(i)
    head = head->next
  }
  return head; return temp;
}
```

Return start Node address

If it contains last Node




```
main() {
  Node *h = CN(10);
}
```

Ex:

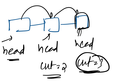

```
int size(Node *head) {
  int cnt = 0;
  while( head != NULL){
    cnt++;
    head = head->next
  }
  return cnt;
}
```

TC : O(N)

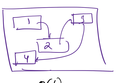
Way 1

```
if(head == NULL) return 0;
int cnt = 1
while( head->next != NULL){
  cnt++;
  head = head->next
}
```

Way 2



→ Linked List : Make sure edge cases are maintained



O(1)

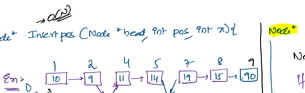
```
Node* InsertStart(Node *head, int n){
  //Create a node with data n
  & Insert the node at start
  & return new Node
  Node *temp = new Node(n);
  temp->next = head;
  return temp;
}
int main() {
  head = InsertStart(head, 20);
}
```

local variable

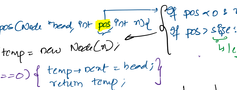
If head == NULL code works

Code

```
Node* InsertPos(Node *head, int pos, int n){
  if( pos < 0 : return head
  if pos > size : return head
  & length + InsertPos
  Node *temp = new Node(n);
  if(pos == 0) { temp->next = head;
                 return temp;
  Node *temp2 = head;
  //Say we want to Insert at pos / pos-1
  for(int i = 1; i < pos-1; i++)
    head = head->next
  temp->next = head->next
  head->next = temp;
  return temp2
}
```

Ex: 

Insert 15 at Index 5 : Insert at start
Insert 29 at Index 2 : Between 1 & 2
Insert 40 at Index 4 : Between 3 & 4
Insert 80 at Index 0 : Insert at start
Insert 90 at Index 7 : Insert at last



head = NULL
pos = 0

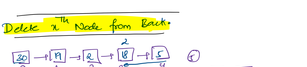
pos = 4

Delete node & given Pos → There is a chance that head Node changes

```
Node* DeletePos(Node *head, int pos){
  if(pos < 0 || pos ≥ size(head)){
    return head;
  }
  if(pos == 0) { return head->next }
    Node *temp = head;
  for(int i = 1; i < pos-1; i++)
    head = head->next
  head->next = head->next->next
  return temp;
}
```

free(temp)
delete(temp) : free head

6a) Delete n th Node from Back.

Ex: 

n = 2 : Delete Node 18 : 20 → 19 → 2 → 5

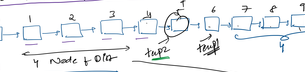
```
Node* Delete(Node *head, int pos) {
  // Delete size() - pos from start
  return DeletePos( head, size(head) - pos);
}
```

head
Node
return
Unary

N

Iterations: N

To solve the above problem in a single traversal : Asked by Lot of Interviews



Ex: pos = 6

temp2 =	7	8	9	10	11
temp1 =	0	1	2	3	4

Say we want to delete 4th Node from back:
If we want to delete 7th Node
temp2 somehow it should be at 6th Node.

temp1	temp2
0	5
1	6
2	7
3	8
4	9
5	10
6	11 = NULL

temp1 → At required pos

temp2 = NULL
temp1 = should be at required position

We need to iterate

```
for int mov temp2 = (pos+1)
  while( temp2 != NULL){
    temp2 = temp2->next
    temp1 = temp1->next
  }
```

It is a single traversal of single list