Topics:

1) Linked lists
2) Stack
3) Queue

Linked lists

Qn: You are given a sequence of elements given some queries (deletion of element). For each query In what time complexity can you do it in terms of size of array n?

Trivial idea: I store them in an array then

Takes in worst case (when we delete first element) O(n) time.

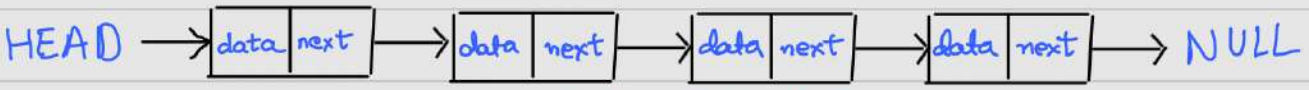
With linked list this could be done in O(1)
Similarly insertion in a sequence can be done in O(1).
(Think how to do this in array?)

What is Linked list?

A linear list consisting of a series of connected nodes.

Node = { Data
Address of next node (pointer)

Types: Singly Linked, Doubly Linked, Circular Linked list



(Singly Linked)

(Doubly Linked)

(circular Linked List)

(circular Doubly Linked List)

Representation

```
class Node{
public:
  int data;
  Node* next;
};  //Singly Linked
```

```
class Node {
public:
   int data;
   Node* next;
   Node* prev;
};  // Doubly Linked
```

* Main power: power to break and rejoin the chain

Linked list Operations: (Singly Linked lists)

* Traversal
* Insertion
* Deletion

TRAVERSAL:

* HEAD points to 1st node.
* NEXT pointer of last node is NULL.

If you have the HEAD you can traverse whole list.

Pseudocode:

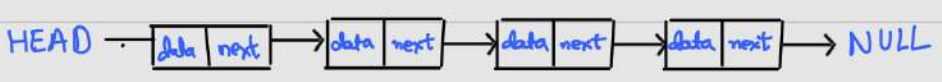

```
Node* temp = HEAD;
while ( temp != NULL)
   {  cout << temp -> data << " ";
      temp = temp -> next;
   }
```

* INSERTION

- allocate memory for new node
- store data
- store appropriate location in the next field } VVI

Insert at beginning

Pseudocode:

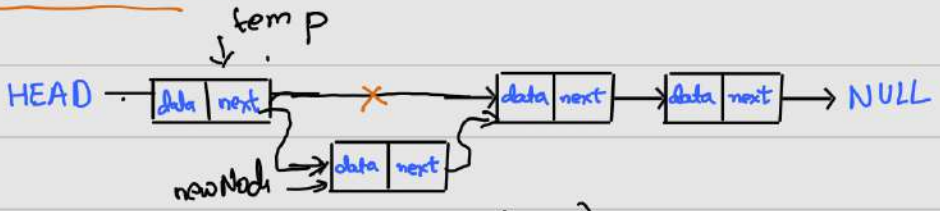



```
Node* newNode = malloc (sizeof (Node)); // new Node;
    newNode -> data = 4;
    new Node -> next = HEAD;
=   HEAD = newNode;
```

Similarly we can do for last Node; (Always make rough sketches of which pointer has to go where)

Insert anywhere in middle

Pseudocode:




```
Node* newNode = malloc( sizeof( Node) ) ;
newNode ->data1 = 4;

Node* temp = head;
for ( int i = 1 ; i < position ; i++)   // position is acc. to 1-based indexing
 { if ( temp -> next != NULL)
      { temp = temp -> next; }
 }

newNode -> next = temp -> next ;
temp -> next = newNode ;
```

O(N) O(Q,N) O(N²)

O(1) O(1) O(N)

As you can feel this code does insertion in O(N) time. Then how is linked list useful?

When we talk about insertion & deletion then we assume that we have a direct access to position like in array we have index & in Linked list we have ptr of element where insertion takes place & thereafter we calculate the time taken.

E.g.- Deletion of duplicate elements of a sequence

DELETION

- Just set next field of previous node to the node which is present just after the node to be deleted.

delete from anywhere in middle

- traverse to its before and do as said above.

Pseudocode:

```
Node* temp = HEAD;
for( int i=1 ; i<pos ; i++ )
  { if (temp->next != NULL)
      {temp = temp->next; }
  }
temp->next = temp->next->next ;
```

Doubly Linked list

Node { *prev, data, *next }

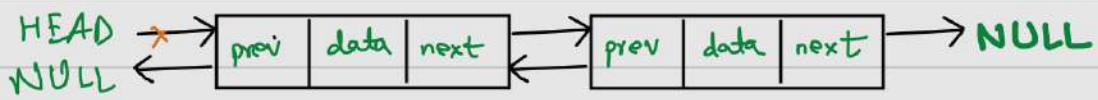
Representation:

```
class Node {
public:
  int data;
  Node* next;
  Node* prev };
```

(Doubly Linked)

INSERTION:

Insertion at beginning:



HEAD->prev = newNode

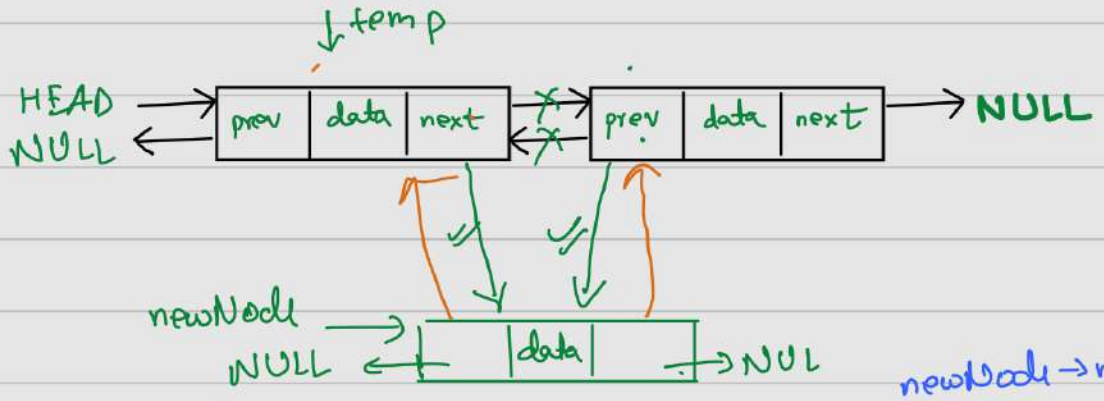
newNode->next = HEAD;

- Create a New node, set its pointers accordingly

Pseudocode:

```
Node* newNode = malloc (sizeof(Node));
newNode → data = 4;
newNode → prev = NULL;
newNode → next = HEAD;
HEAD → prev = newNode
HEAD = newNode
```

Insertion anywhere in middle



newNode → next = temp → next

```
newNode → prev = temp;
temp → next → prev = newNode;
temp → next = newNode;
```

Pseudocode:

```
Node* newNode = malloc (sizeof(Node));
newNode → val = 4;
Node* temp = HEAD;
for(int i=1; i<pos; i++)
  { if( temp → next != NULL )
        { temp = temp → next;}
  }
```

```
newNode -> next = temp -> next;
newNode -> prev = temp;
temp -> next -> prev = newNode;
temp -> next = newNode;
```

DELETION

Deletion of inner node

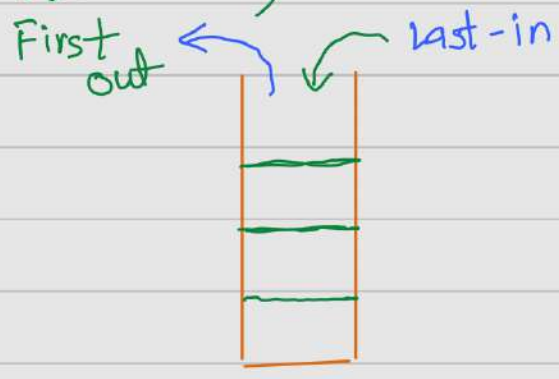

```
temp->next = temp -> next -> next;
temp->next->prev = temp;
```

} pointer changes

```
HEAD = func ( )
temp -> next = newNode;
```

STACK

Linear data structure following principle of (LIFO)
(Last-in First-out)



Putting an item on top is called "push" & removing is called "pop"

Main Operations:

* Push - adds element on top