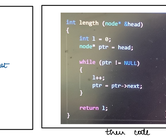
Single Pointer

(c) Create a linked list by inserting elements into it after crossing point all the elements present in the linked list.

There are several ways to insert elements into linked-list

Case 1:- Insert at start

Case 2:- Insert at end



Case 3:- Insert somewhere in middle (say after 3)

think math first

then code

(d) calculate length of a linked list

→ O(n) time complexity

while traversing the linked list list keep a count

think math

```
int length (node* &head)
{
    int l = 0;
    node* ptr = head;

    while (ptr != NULL)
    {
        l++;
        ptr = ptr->next;
    }

    return l;
}
```

then code

(e) In an already existing linked list, delete a node from that linked list from the specified location

Given linked list →

(common case) Case 1:- Delete from start

(common case) Case 2:- Delete from end

Case 3:- Delete from somewhere middle (say 2)

think math first

```
class node
{
    public:
    int data;
    node* next;

    node(int value)
    {
        data = value;
        next = NULL;
    }
};

void insert_at_start (node* &head, int val_to_insert)
{
    // writing new node and adding value into the node
    node* node_to_insert = new node (val_to_insert);

    // if there is no element in the ll then directly insert
    if (head == NULL)
    {
        head = node_to_insert;
        return;
    }

    // else i.e. if we already have elements in the linked list then
    // changing the links
    node_to_insert->next = head;
    head = node_to_insert;

    return;
}

void insert_at_end (node* &head, int val_to_insert)
{
    // writing new node and adding value into the node
    node* node_to_insert = new node (val_to_insert);

    // if there is no element in the ll then directly insert
    if (head == NULL)
    {
        head = node_to_insert;
        return;
    }

    // else i.e. if we already have elements in the linked list then
    // traversing until we reach the end of the linked list
    // and then finally changing links
    node* temp = head;
    while(temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = node_to_insert;

    return;
}

void insert_after (node* &head, int val_to_insert, int insert_after)
{
    // writing new node and adding value into the node
    node* node_to_insert = new node (val_to_insert);

    // traversing until we reach the element after which we need to insert
    node* temp = head;
    while (temp->data != insert_after)
    {
        temp = temp->next;
    }
    // inserting the new node
    node_to_insert->next = temp->next;
    temp->next = node_to_insert;

    return;
}

void display (node* &head)
{
    // if there is no element in the linked list then nothing to print
    if(head == NULL)
    {
        cout<< "No element in the linked list";
    }

    // else if there are elements present in the linked list then
    node* temp = head;
    while(temp != NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }

int main()
{
    // because initially since the ll is empty, head will be pointing to nothing
    node* head = NULL;

    // creating the ll
    insert_at_start(head,6);
    insert_at_start(head,5);
    insert_at_end(head,5);
    insert_at_end(head,6);
    insert_after(head,10,5);
    display(head);
    insert_at_start(head,4);
    insert_at_end(head,8);
    display(head);
    display(head);
}
```

then code

(f) Remove last node of linked list and put it as first node

think math first

```
void end_2_start(node* &head)
{
    node* ptr = head;

    // traversing to the second last node
    while(ptr->next->next != NULL)
    {
        ptr = ptr->next;
    }

    // removing last node
    node* last_node = ptr->next;
    ptr->next = NULL;

    // inserting last node at start
    last_node -> next = head;
    head = last_node;

    return ;
}
```

then code

(g) Remove last K nodes of linked list and put it at start.

Input:-

Linked list → 2 → 4 → 9 → 13 → 17 → 18 (tail)

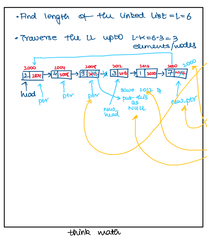
K → 3

Output:-

3 → 2 → 4 → 18 → 6 → 12 → 9

Question

- Find length of the linked list = L = 6
- Traverse the LL upto L-K=6-3=3 elements middle



think math

```
node* append (node* head, int K)
{
    // placing the ptr at its required location
    int L = length(head);
    node* ptr = head;  // pointer to iterate over the ll
    for (int I= 1; I< L-K; I++) // iterating to the required node
    {
        ptr = ptr -> next;
    }

    //// placing the new_ptr at its required position
    node* new_ptr = head; // pointer to iterate over the ll
    while (new_ptr-> next != NULL) //iterating to the last node
    {
        new_ptr =new_ptr->next;
    }

    //changing NULL to head in the last node
    new_ptr -> next = head;

    //doing required link changes at ptr position
    node* new_head = ptr->next; // saving
    ptr->next = NULL;  //changing links

    return new_head;
}

int main()
{
    // because initially since the ll is empty, head will be pointing to nothing
    node* head = NULL;

    // creating the ll
    insert_at_end(head,6);
    insert_at_end(head,12);
    insert_at_end(head,9);
    insert_at_end(head,8);
    insert_at_end(head,2);
    insert_at_end(head,4);
    insert_at_end(head,13);
    insert_at_end(head,2);
    insert_at_end(head,13);
    insert_at_end(head,17);
    display(head);
    cout << endl;

    node* new_head = append(head,5);
    display (new_head);
}
```

then code

(h) Given a sorted linked list, insert the element at the correct position

given → 2 → 14 → 5 → 20 → 8

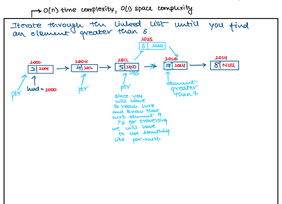
head

insert 6.

Question

→ O(n) time complexity, O(1) space complexity

Iterate through the linked list until you find an element greater than 6



think math first

```
void insert_in_sorted (node* &head, int val_to_insert)
{
    // creating the new node to insert
    node* node_to_insert = new node(val_to_insert);

    // pointer to first element of ll
    node* ptr = head;

    // traversing to find first element greater than val_to_insert
    while (ptr->next->data < val_to_insert)
    {
        ptr = ptr -> next;
    }

    // changing the links
    node_to_insert->next = ptr->next;
    ptr->next = node_to_insert;
    return;
}

int main()
{
    // because initially since the ll is empty, head will be pointing to nothing
    node* head = NULL;

    // creating the ll
    insert_at_start(head,6);
    insert_at_end(head,12);
    insert_at_end(head,9);
    insert_at_end(head,8);
    insert_at_end(head,2);
    insert_at_end(head,13);
    display(head);
    cout << endl;

    insert_in_sorted(head,5);
    display(head);
```

then code it

this code won't handle the edge case

insert 10 in 2 → 12 → 9 → 8

head, ptr, ptr, ptr, ptr

modification part will cause problem

since mostly your function now accordingly to handle edge case

```
void insert_in_sorted (node* &head, int val_to_insert)
{
    // creating the new node to insert
    node* node_to_insert = new node(val_to_insert);

    // pointer to first element of ll
    node* ptr = head;

    // traversing to find first element greater than val_to_insert
    while (ptr->next->data < val_to_insert )
    {
        ptr = ptr -> next;
    }

    if (ptr->next == NULL)
    {
        ptr->next = node_to_insert;
        return;
    }

    // changing the links
    node_to_insert->next = ptr->next;
    ptr->next = node_to_insert;
    return;
}

int main()
{
    // because initially since the ll is empty, head will be pointing to nothing
    node* head = NULL;

    // creating the ll
    insert_at_start(head,6);
    insert_at_end(head,12);
    insert_at_end(head,9);
    insert_at_end(head,8);
    insert_at_end(head,2);
    display(head);
    cout << endl;

    insert_in_sorted(head,10);
    display(head);
```

(i) Delete nodes having greater value on right in linked list

Input:-

12 → 5 → 10 → 11 → 5 → 6 → 2 → 3

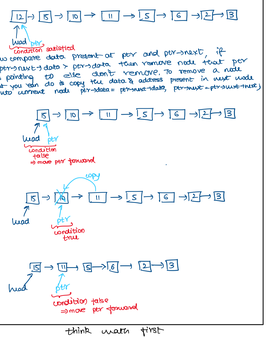
Output:-

15 → 11 → 6 → 3

Question

→ O(n) time complexity

put a pointer at head and traverse the LL using two pointer



think math first

```
void delete_greater (node* &head)
{
    // creating a pointer
    node* ptr = head;

    // traversing the LL
    while (ptr->next != NULL)
    {
        if(ptr->next->data > ptr->data)
        {
            ptr->data = ptr->next->data;
            ptr->next = ptr-> next->next;
        }

        ptr = ptr -> next;
    }

    return;
}
```

then code it