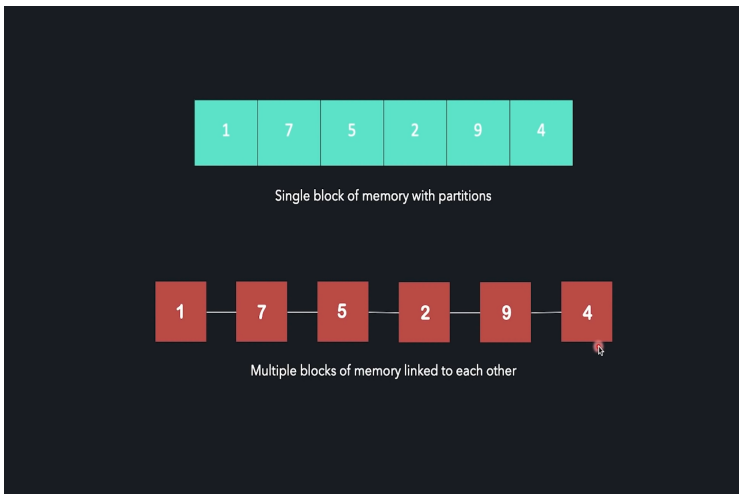
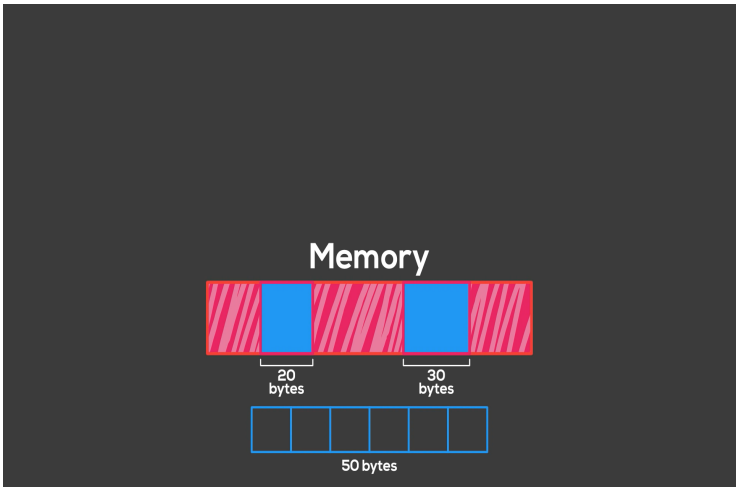
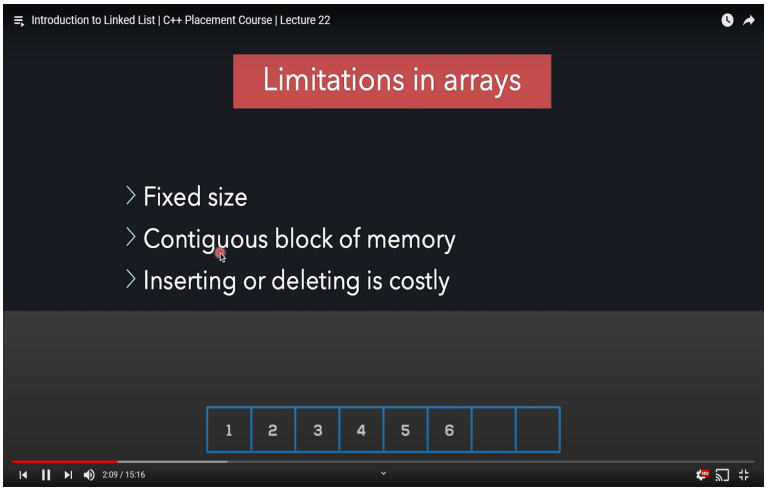
LINKED LIST



Limitations in arrays

- Fixed size
- Contiguous block of memory
- Inserting or deleting is costly



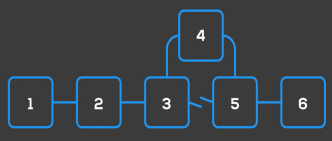


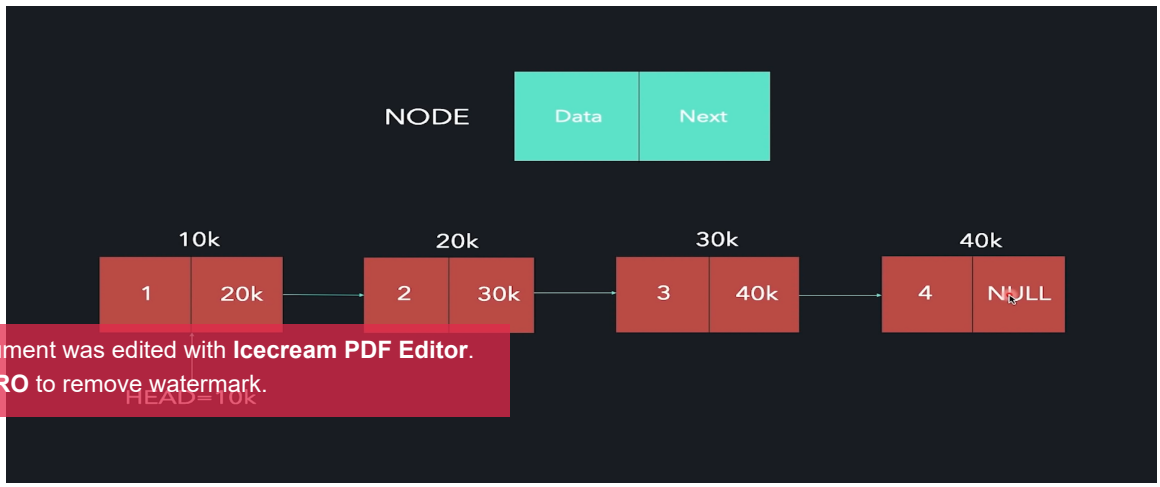
Properties of Linked List

- Size can be modified
- Non-contiguous memory
- Insertion and deletion at any point is easier

Properties of Linked List

- Size can be modified
- Non-contiguous memory
- Insertion and deletion at any point is easier





This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Linked List [IMP]

Note: These questions could be easily done using Arrays/ Vectors, However in the interview, you'll have to do it using Linked List only.

Node structure

Each node contains the link to the next node and some data variables.

```
class node {
public:
    int data;
    node* next;
    node(int val) {
        data = val;
        next = NULL;
    }
};
```

Implementation

1. Make a node named head. This will act as the start of the linked list.

To insert at the end of the Linked List, iterate to the last node. In the below code temp is the last node. So temp -> next = new node(val).

```
void insertAtTail(node* &head, int val) {
    node* n = new node(val);
    if (head == NULL) {
        head = n;
        return;
    }
    node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = n;
}
```

AFTER KAKSHA

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Printing a LinkedList

We pass the head of the node, and then keep on iterating till we do not reach the end of the node.

```
void display(node* head) {
    node* temp = head;
    while (temp != NULL) {
        cout << temp->data << "->";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}
```

Inserting at the head

The next of the new node should be the old linked list (head). So the head will now point to the new node because it now becomes the new head.

```
void insertAthead(node* &head, int val) {
    node* n = new node(val);
    n->next = head;
    head = n;
}
```

Try yourself:

Do a Linear Search in the Linked List.

APNI KAKSHA

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Code for Linear Search:

```
bool search(node* head, int key) {
    node* temp = head;
    while (temp != NULL) {
        if (temp->data == key) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}
```

APNI KAKSHA

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Deletion of a node

To delete a node having node->data = val

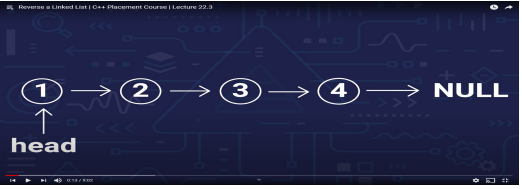
There are two cases:

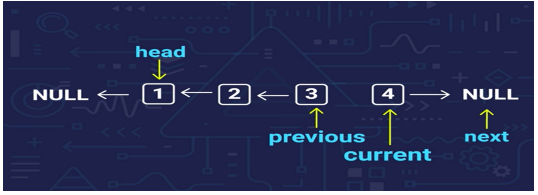
1. We have to delete the node. In that case, we simply increment head.
2. Else, we search for the node having node->next->data == val. And then delete it.

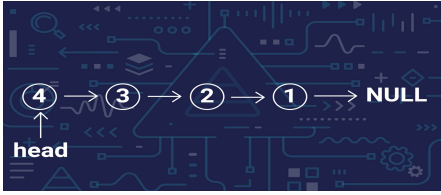
```
void deleteAtHead(node* &head) {
    node* todelete = head;
    head = head->next;
    delete todelete;
}
void deletion(node* &head, int val) {
    if (head == NULL)
        return;
    if (head->next == NULL) {
        deleteAtHead(head);
        return;
    }
    node* temp = head;
    while (temp->next->data != val) {
        temp = temp->next;
    }
    //we have to delete the temp->next node
    node* todelete = temp->next;
    temp->next = temp->next->next;
    delete todelete;
}
```

APURVA AKSHA

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.







Reverse a Linked List [Both methods are very important]

There are two methods:

1. Iterative

   Idea: We will keep three-pointers, for the previous, current, and next node.

   Hint: We have to connect current->next to the previous node, and then move to the next node.

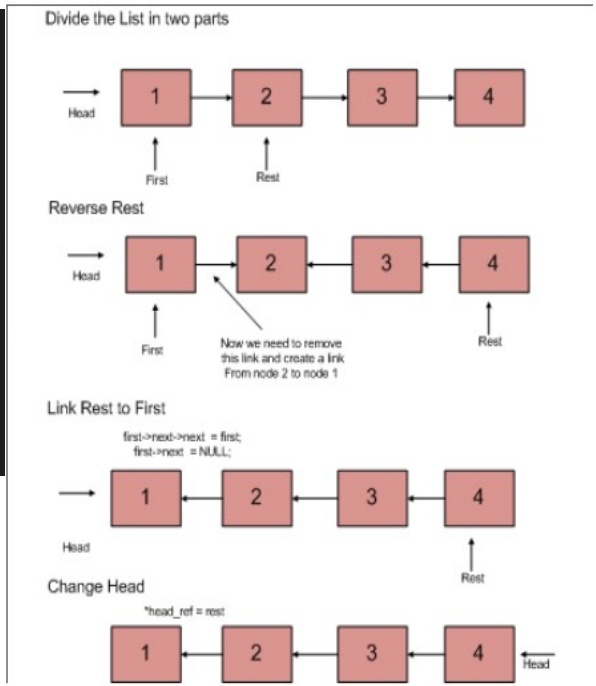
```
node* reverse(node* &head) {
    node* prevptr = NULL;
    node* currptr = head;
    node* nextptr;
    while (currptr != NULL) {
        nextptr = currptr->next;
        currptr->next = prevptr;
        prevptr = currptr;
        currptr = nextptr;
    }
    return prevptr;
}
```

2. Recursive

   Idea: We are at the head, recursively reverse the remaining list.

   The (head->next) is the next node, this node should be the second last node in the reversed list. head would be the last node in the reversed list, so its next should be NULL. And then we will return the reversed list.

```
node* reverseRecursive(node* &head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }
    node* newhead = reverseRecursive(head->next);
    head->next->next = head;
    head->next = NULL;
    return newhead;
}
```



This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

NIKAKSHA

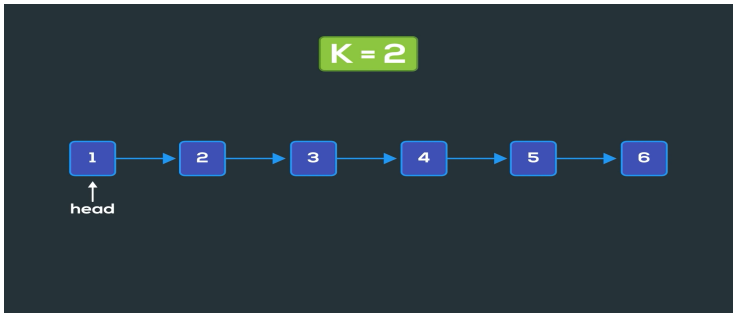
K reverse[IMP]

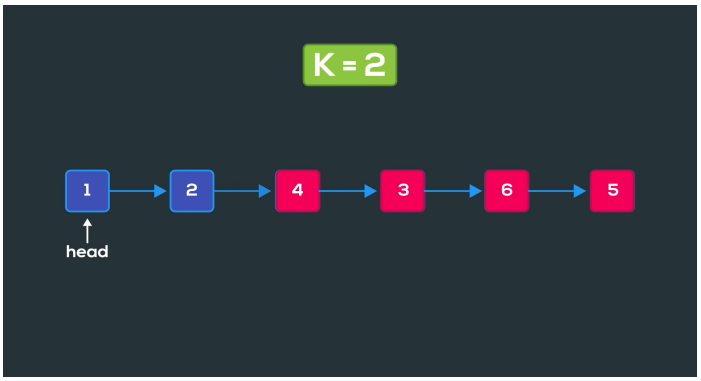
Reverse k nodes in a linked list

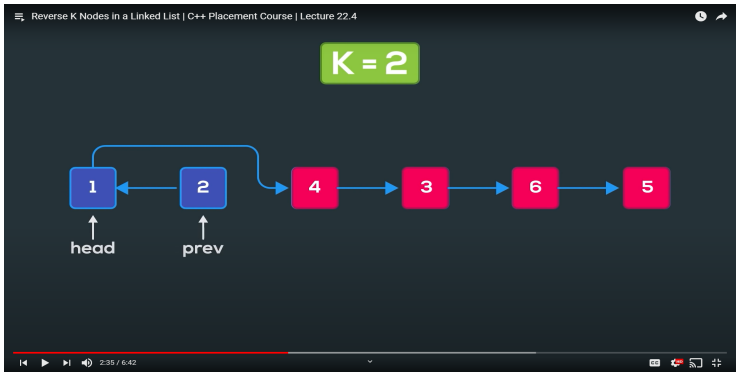
Given a linked list and an integer K. Reverse the nodes of a linked list k at a time and return its modified list. If the number of nodes is not a multiple of K then left-out nodes, in the end, should remain as it is.

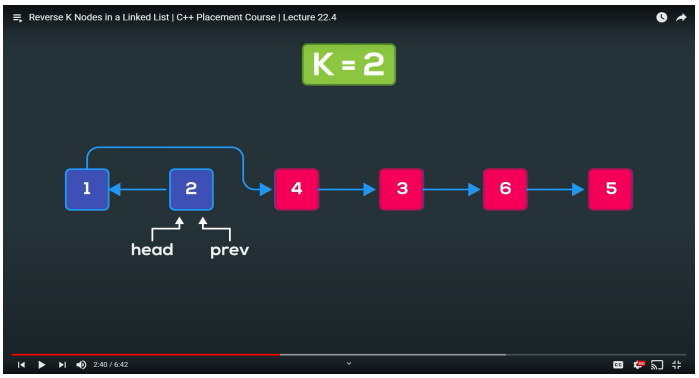
Idea: There are 2 cases:

1. The size of the linked list is less than K. In this case, return the list as it is.
2. The size of the linked list is more than K. Reverse the first K nodes and recurse for the remaining list.



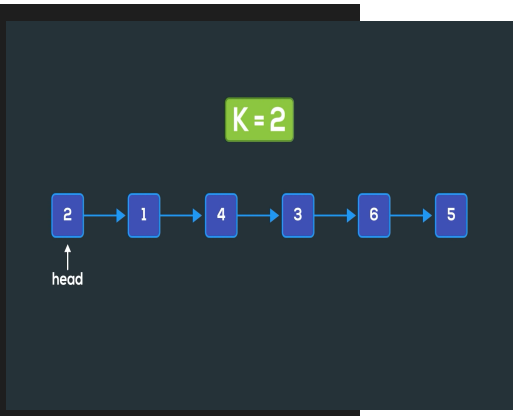







```
node* reversek(node* &head, int k) {
    node* prevptr = NULL;
    node* currptr = head;
    node* nextptr;
    int count = 0;
    while (currptr != NULL && count < k) {
        nextptr = currptr->next;
        currptr->next = prevptr;
        prevptr = currptr;
        currptr = nextptr;
        count++;
    }

    if (nextptr != NULL) {
        head->next = reversek(nextptr, k);
    }
    return prevptr;
}
```



This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

FollowUp: Try to do it iteratively as well.

Cycle Detection[IMP][without map]

Floyd's cycle detection

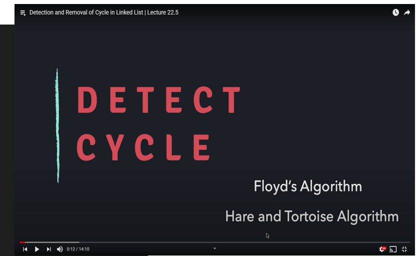
Also know as hare and tortoise method, and slow and fast pointer method.

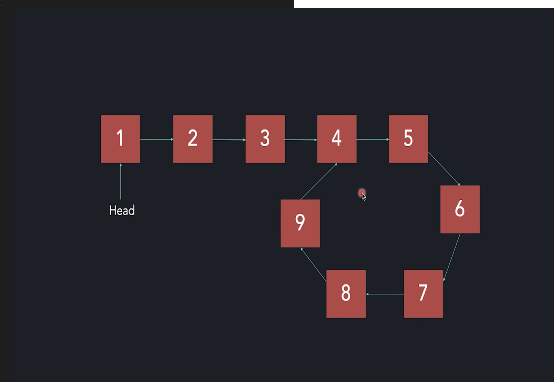
Idea: Slow and pointers are initially at the start. Slow takes one step. Fast takes two-step at a time.

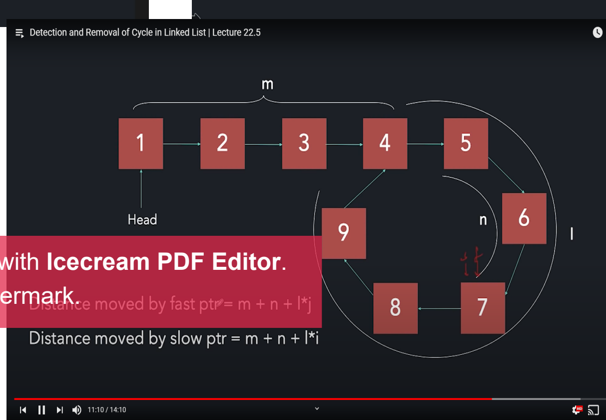
Time Complexity: O(n)

```
bool detectCycle(node* &head) {

    node* slow = head;
    node* fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
        if (fast == slow) {
            return true;
        }
    }
    return false;
}
```







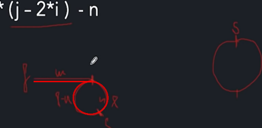
Distance moved by fast ptr = 2 x Distance moved by slow ptr

m + n + l*j = 2 * (m + n + l*i)

m + n = l * (j - 2*i)

m = l * (j - 2*i) - n

i -> no. of times slow moves
j -> no. of times fats moves



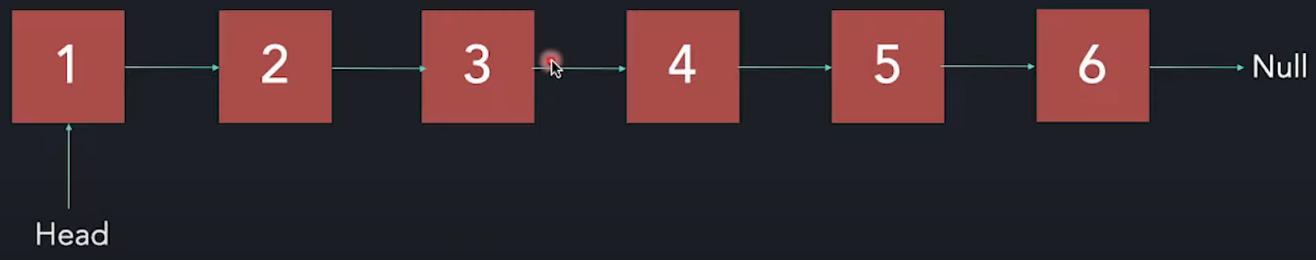
This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Append Last K Nodes of a Linked List | C++ Placement Course | Lecture 22.7

Append last k nodes to start of linked list

Append Last K Nodes of a Linked List | C++ Placement Course | Lecture 22.7

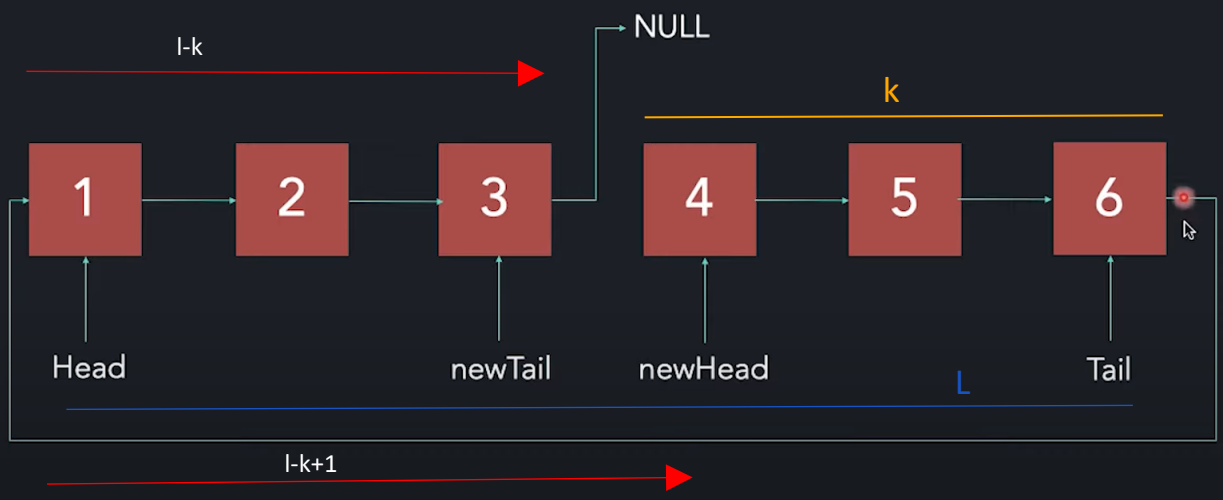
K = 3



Append Last K Nodes of a Linked List | C++ Placement Course | Lecture 22.7

L = Lenght of the linked list

K = 3

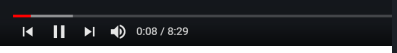


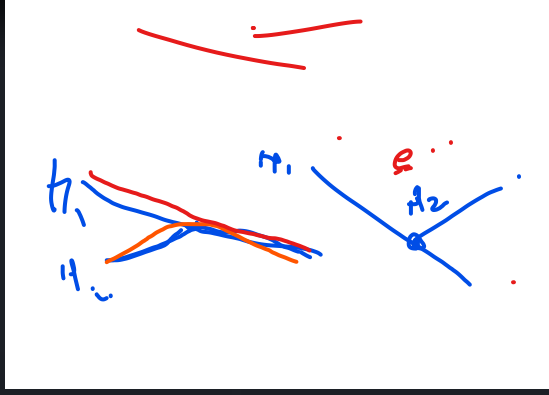
This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

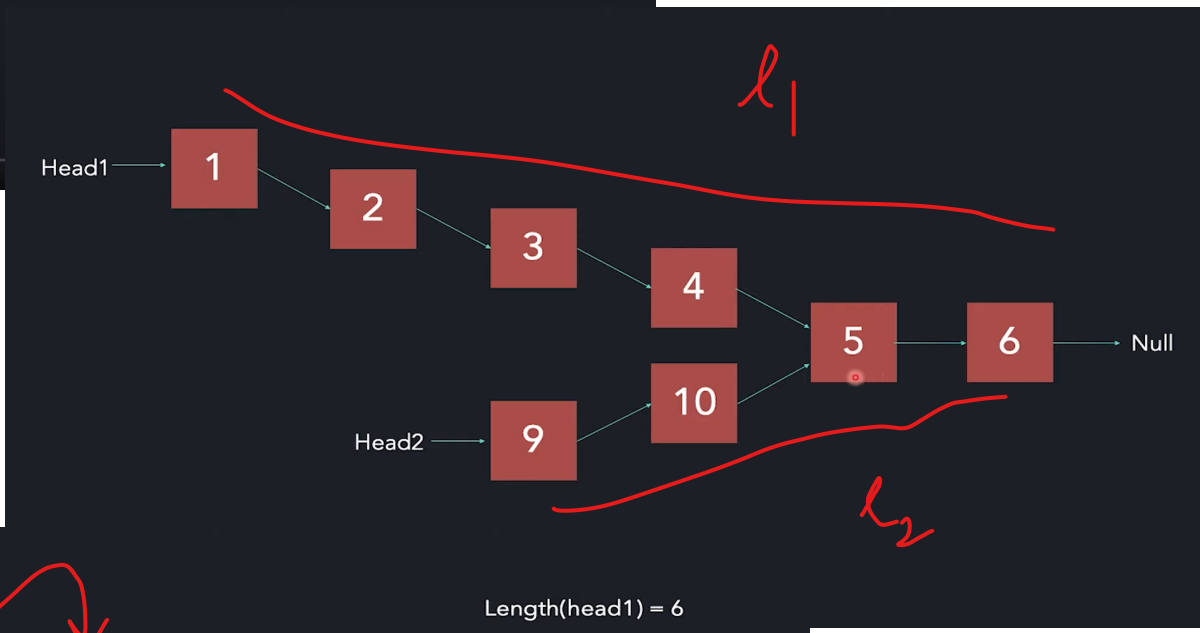
Find Intersection point of 2 Linked List | Lecture 22.8

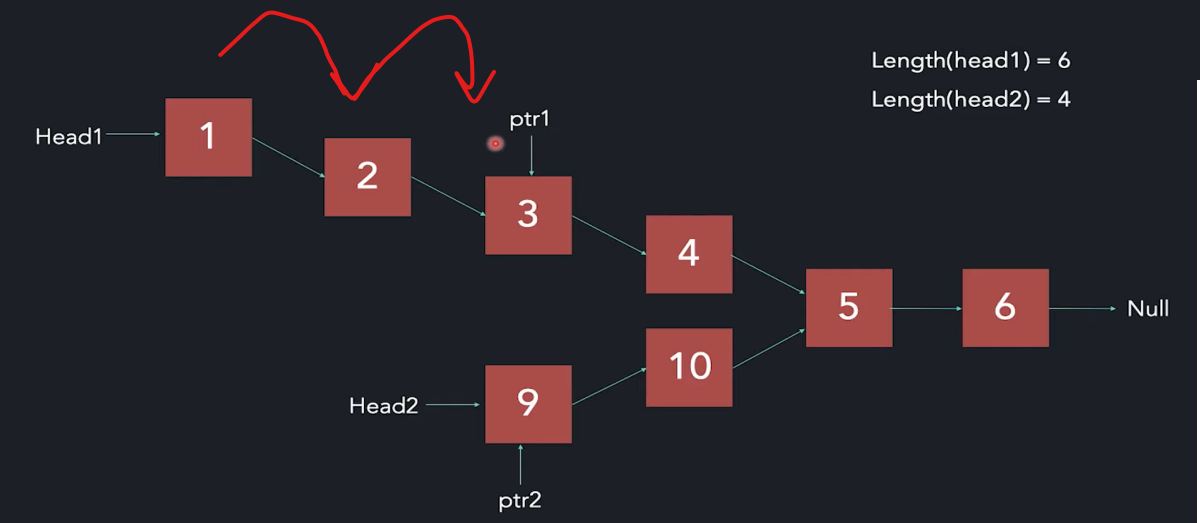
Press Esc to exit full screen

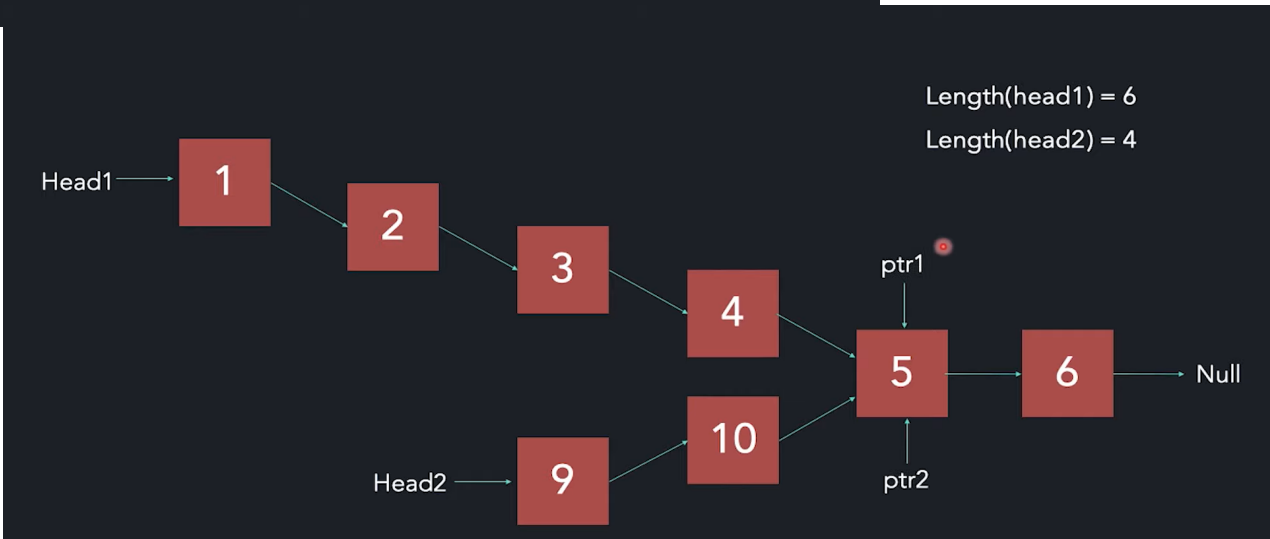
Find intersection point of 2 linked lists









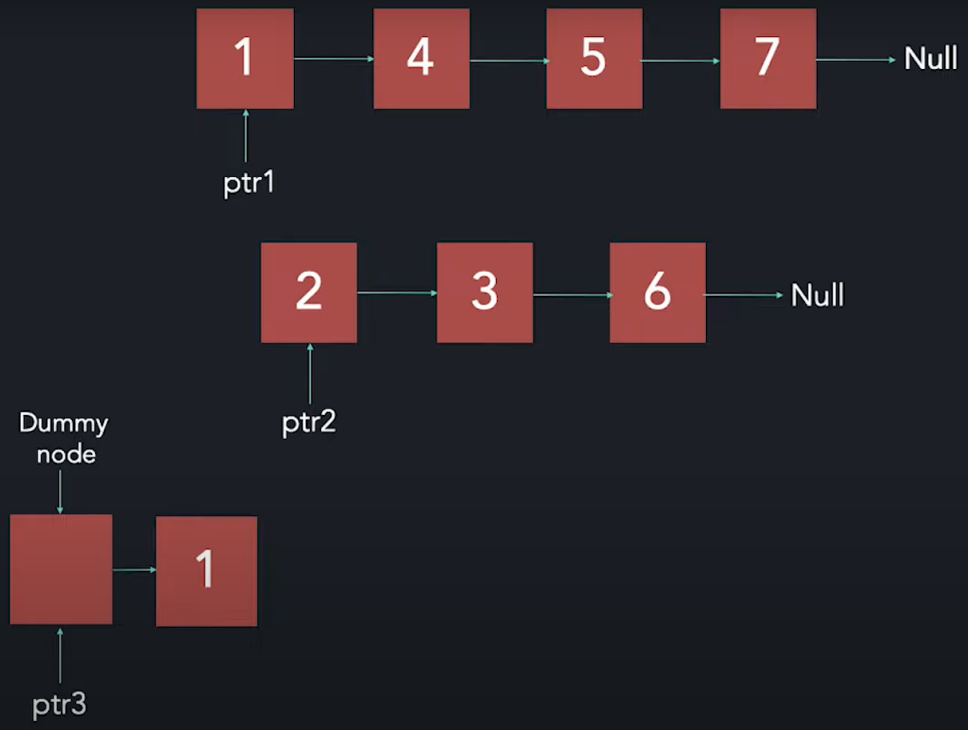


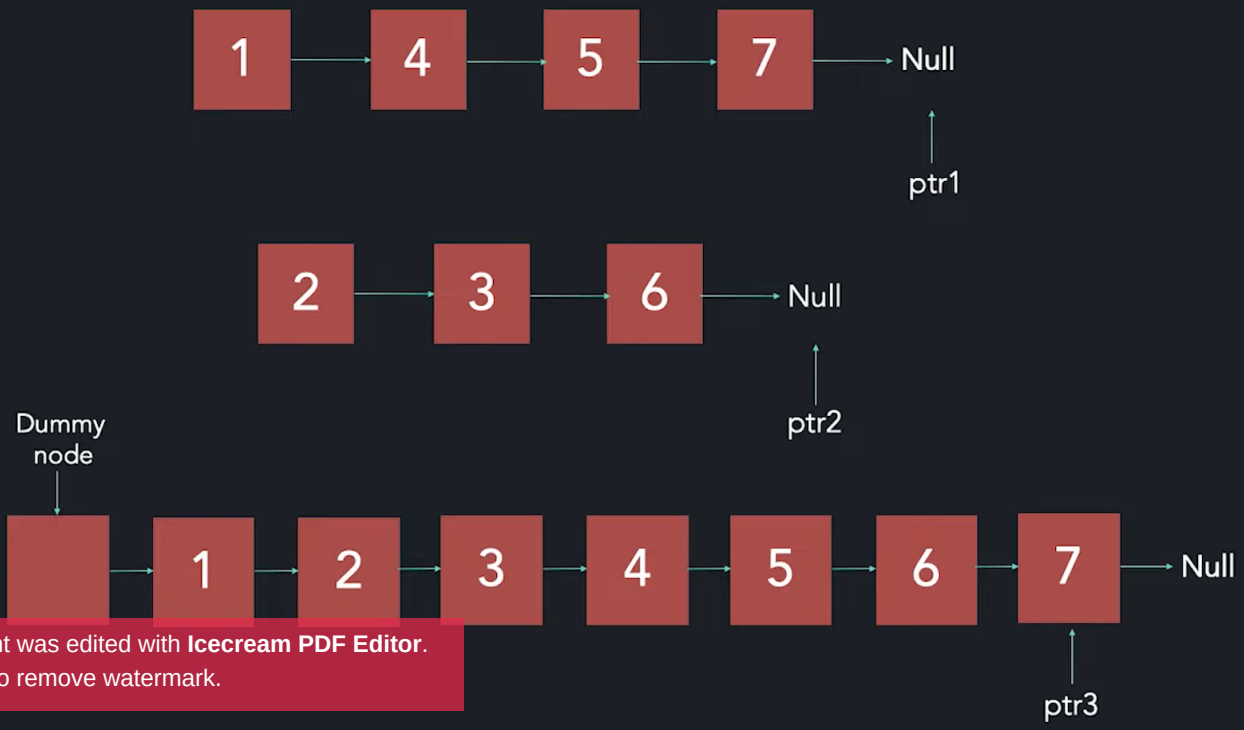
This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Merge 2 Sorted Linked List | C++ Placement Course | Lecture 22.9

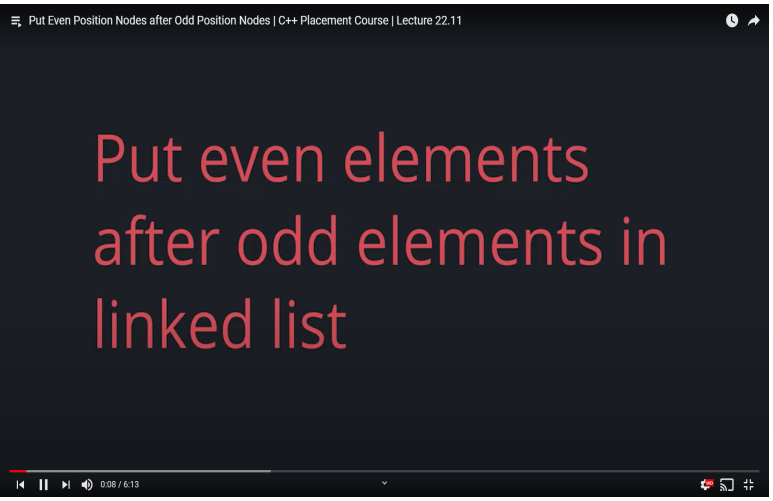
Merge 2 sorted linked lists

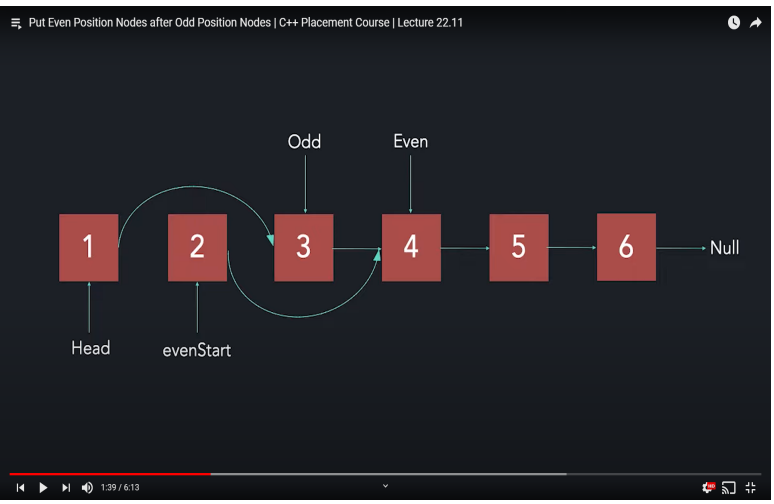
Merge 2 Sorted Linked List | C++ Placement Course | Lecture 22.9

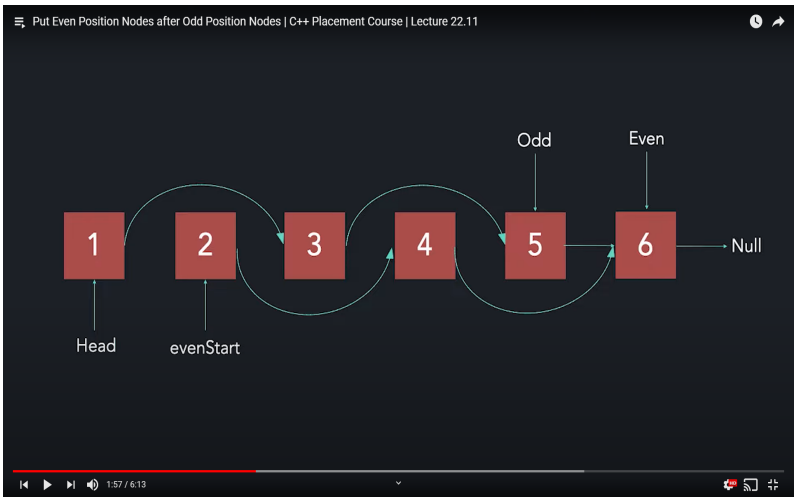


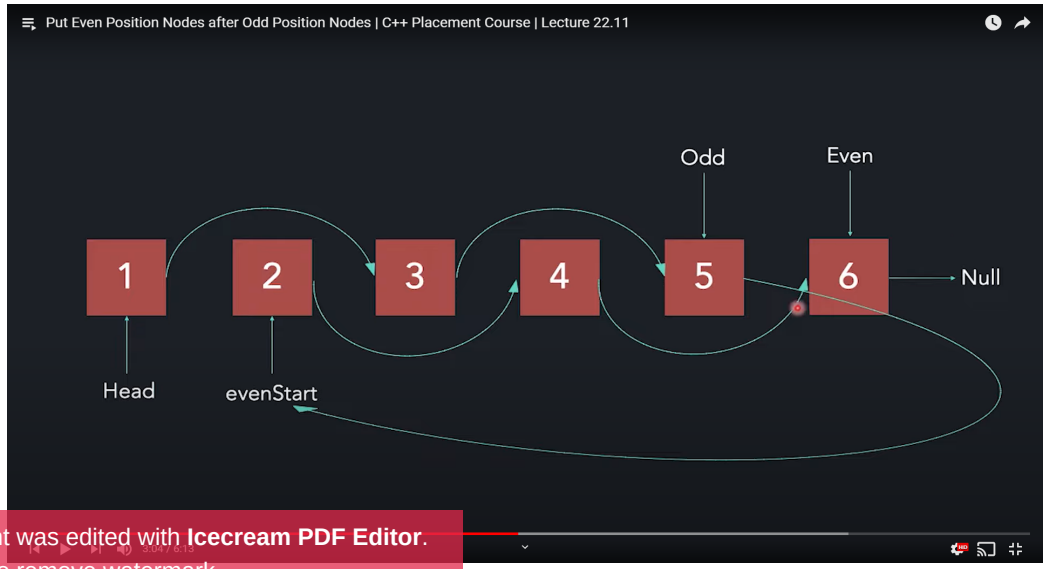


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

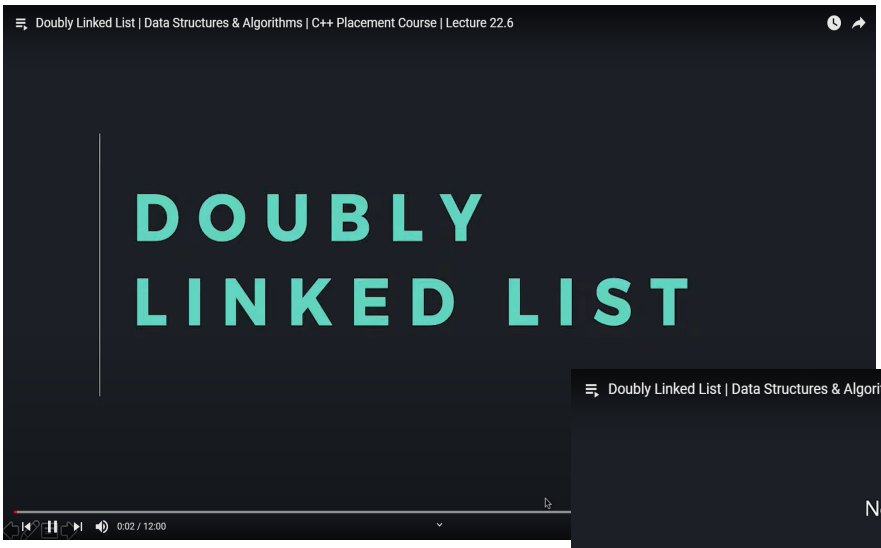


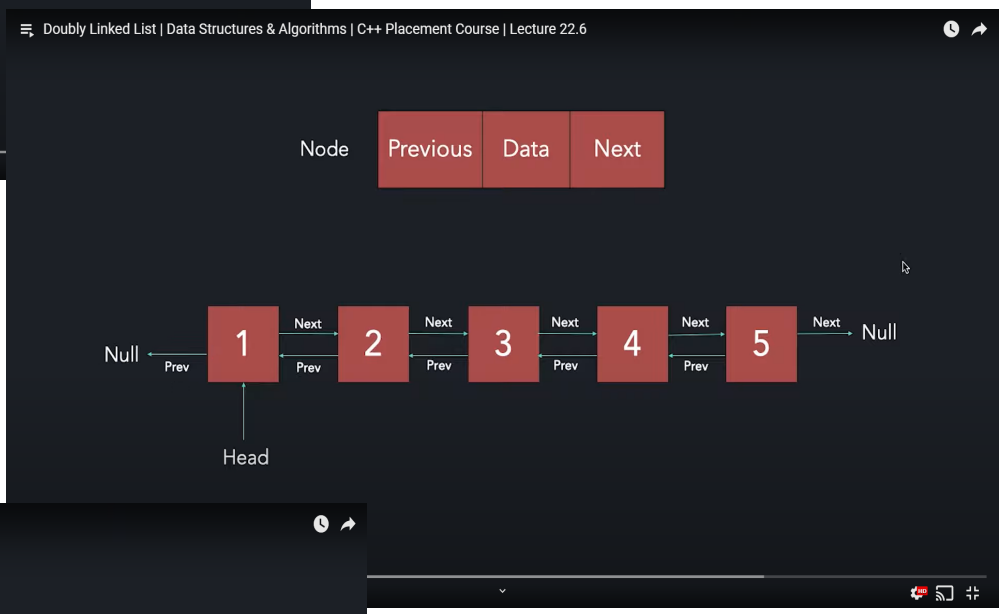


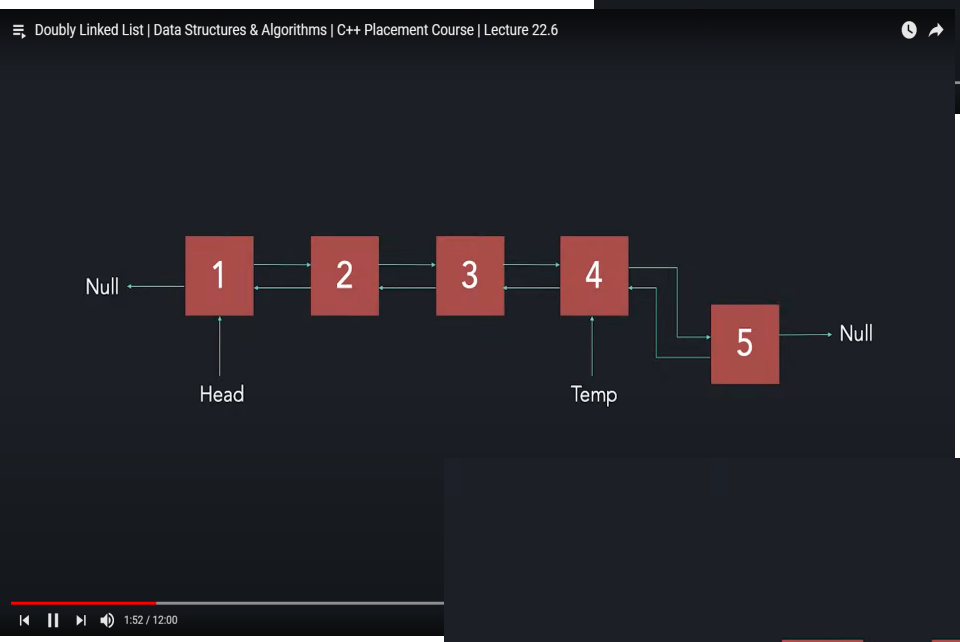


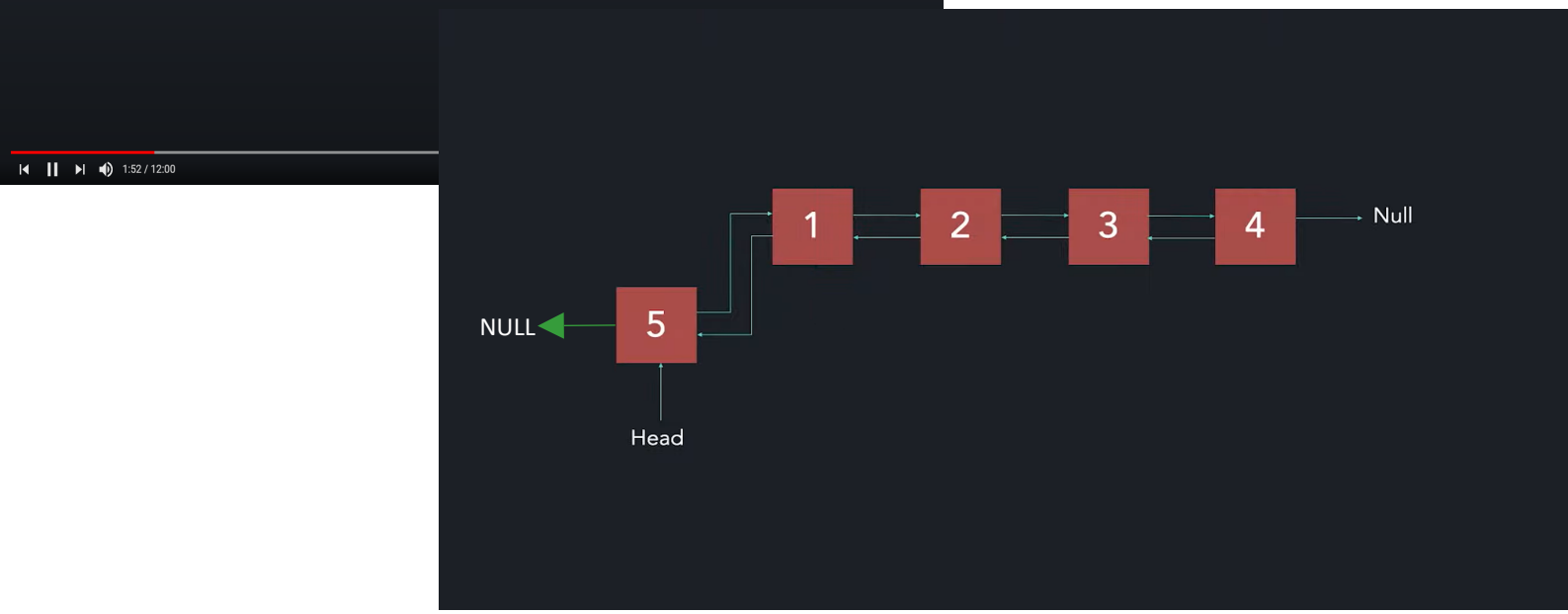


This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.









This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.

Doubly linked list

The structure of the linked list gets modified. In a doubly linked list, each node will also store the link to the previous node.

```
class node {
public:
    int data;
    node* next;
    node* prev;
    node(int val) {
        data = val;
        next = NULL;
        prev = NULL;
    }
};
```

Implementation

Circular Linked List

In a circular linked list, the last node is connected to the first node in the linked list.

Implementation

Additional Important Questions:

1. Even Odd List
2. Middle of the Linked List
3. Delete a Node in Linked List
4. Palindrome Linked List
5. Swap nodes in pair
6. N'th node from the last
7. Add 2 numbers
8. Merge Sort in Linked List

This PDF document was edited with **Icecream PDF Editor**.
Upgrade to PRO to remove watermark.