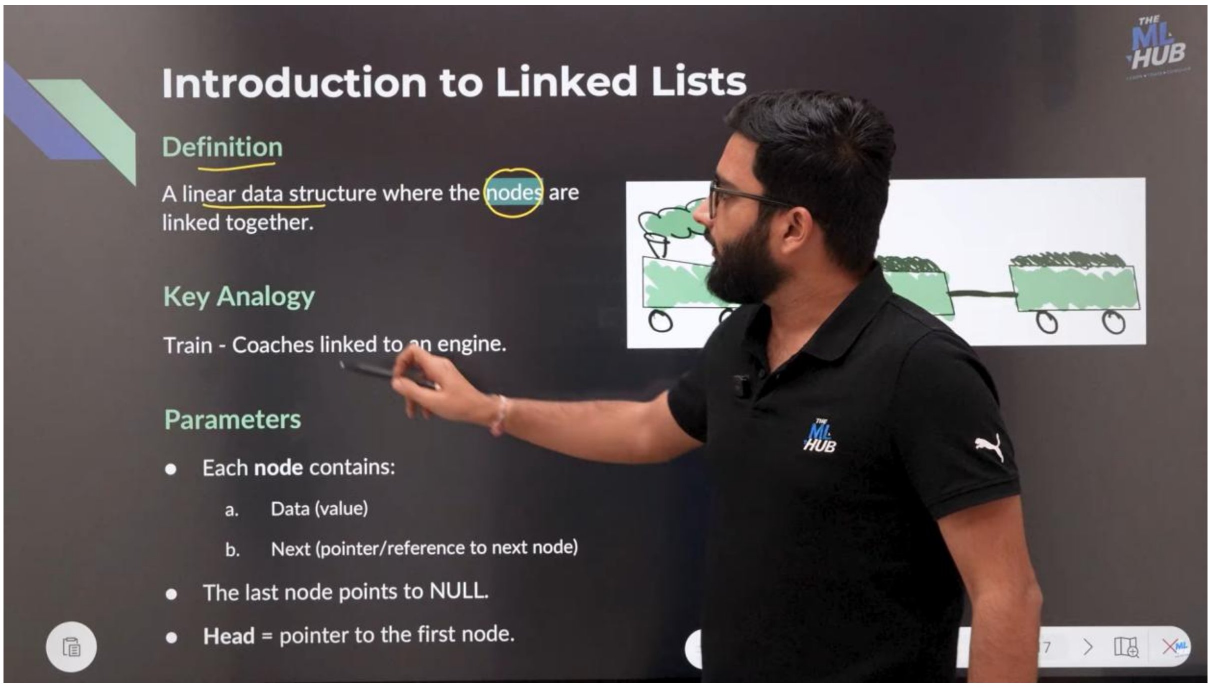


Introduction to Linked Lists

Definition

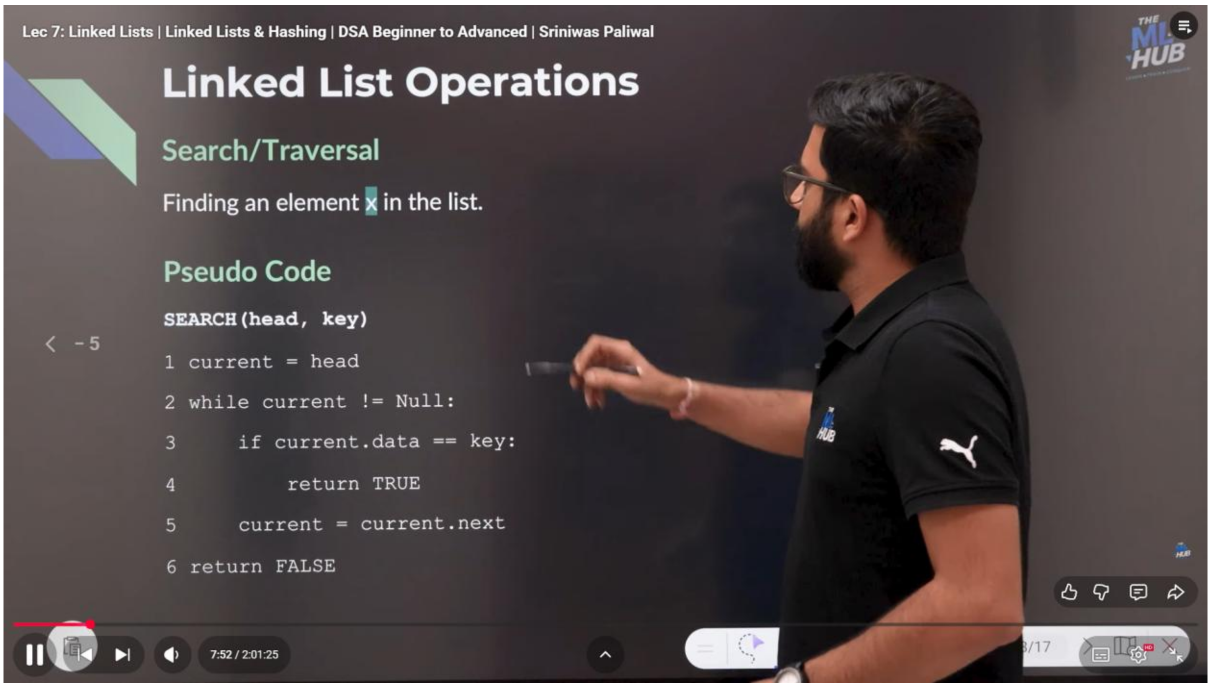
A linear data structure where the nodes are linked together.

Key Analogy

Train - Coaches linked to an engine.

Parameters

- Each **node** contains:
    a. Data (value)
    b. Next (pointer/reference to next node)
- The last node points to NULL.
- **Head** = pointer to the first node.



Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

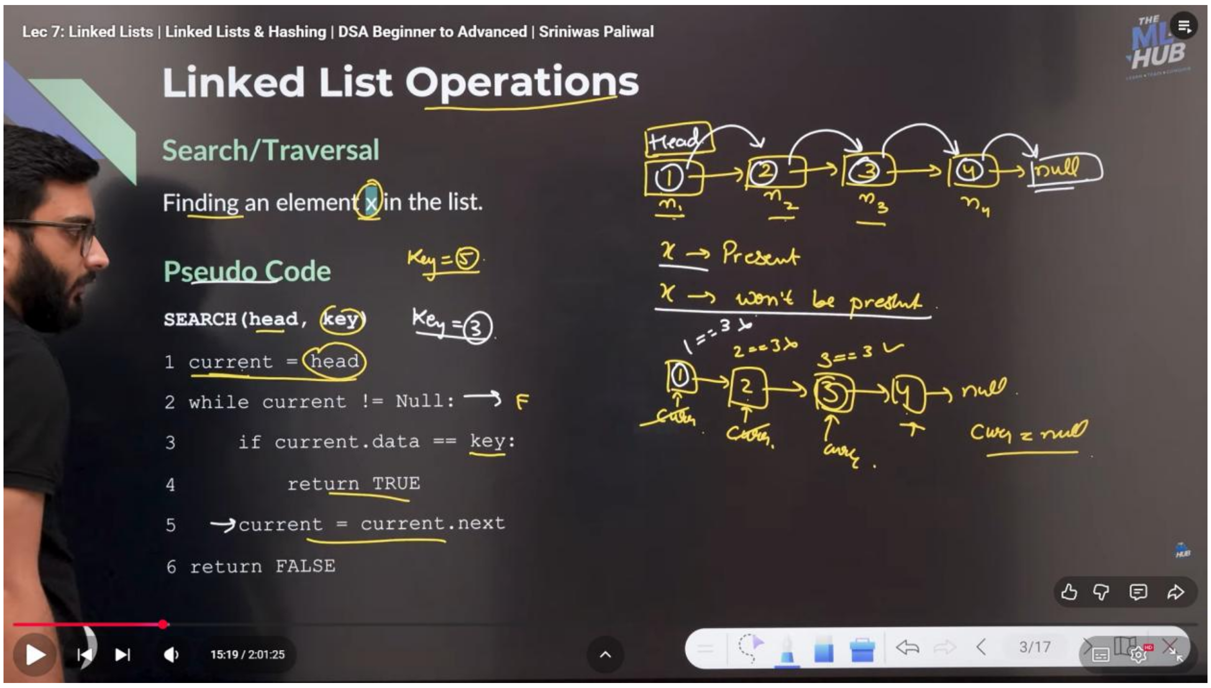
Linked List Operations

Search/Traversal

Finding an element x in the list.

Pseudo Code

```
SEARCH(head, key)
current = head
while current != Null:
    if current.data == key:
        return TRUE
    current = current.next
return FALSE
```


Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

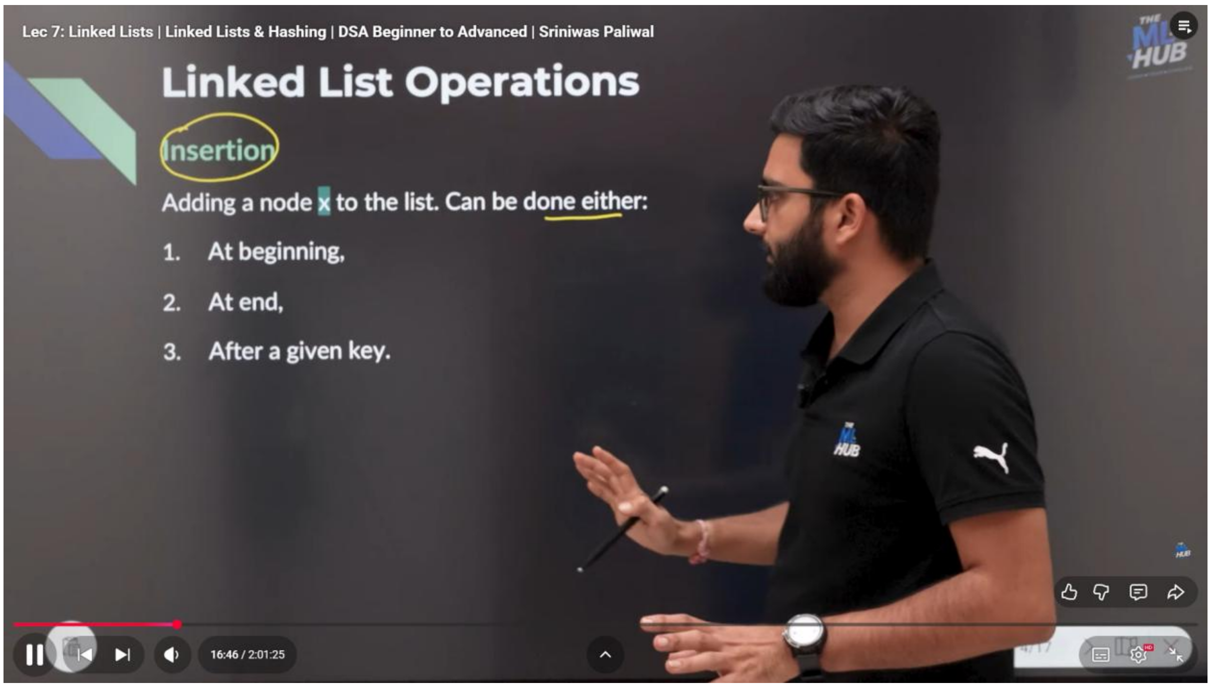
Linked List Operations

Search/Traversal

Finding an element x in the list.

Pseudo Code

```
SEARCH(head, key)
current = head
while current != Null:
    if current.data == key:
        return TRUE
    current = current.next
return FALSE
```

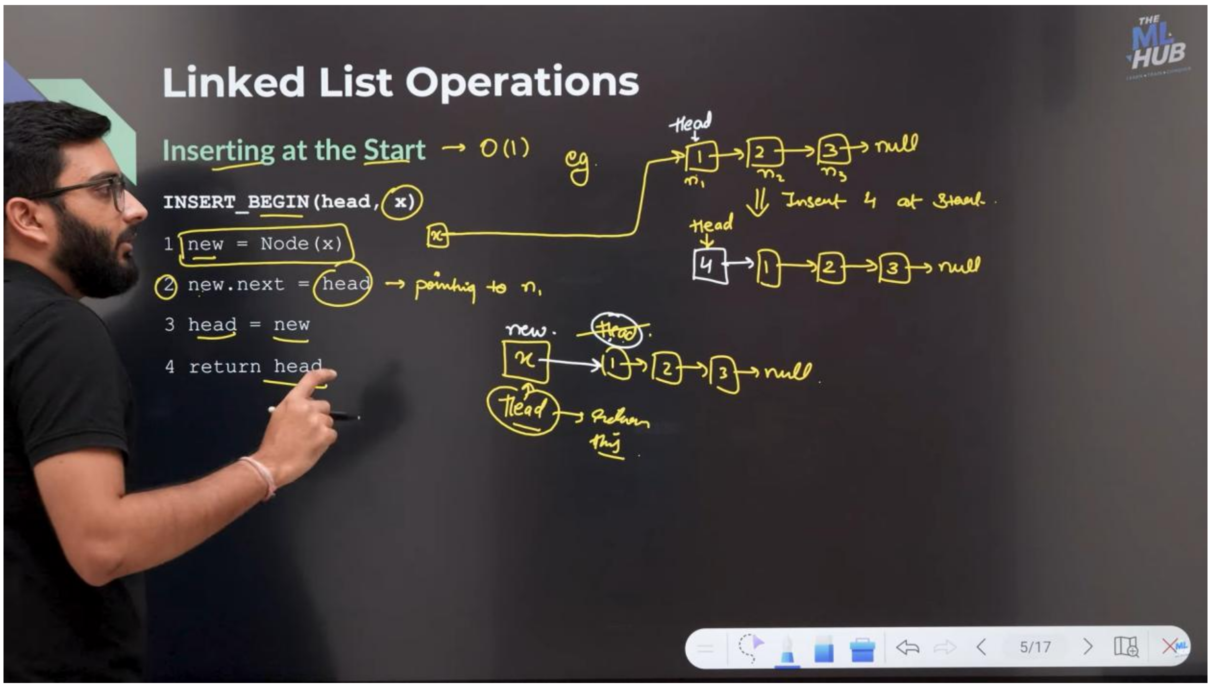

Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

Linked List Operations

Insertion

Adding a node x to the list. Can be done either:

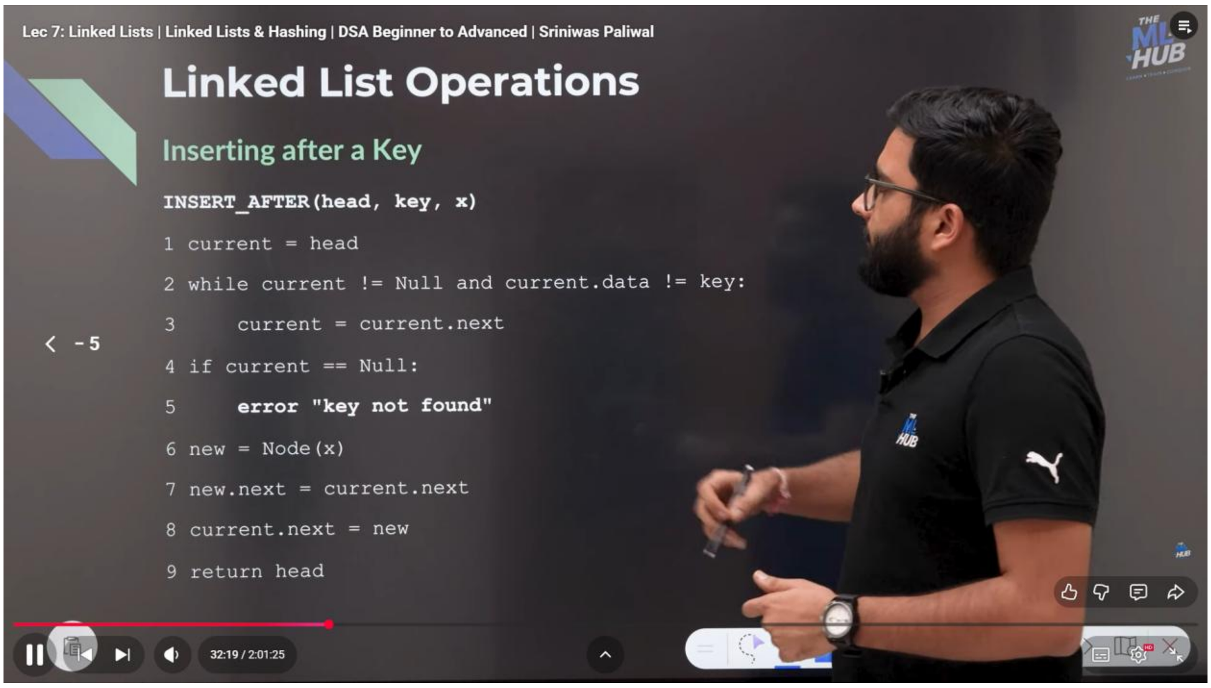
1. At beginning,
2. At end,
3. After a given key.



Linked List Operations

Inserting at the Start → O(1)

```
INSERT_BEGIN(head, x)
new = Node(x)
new.next = head
head = new
return head
```

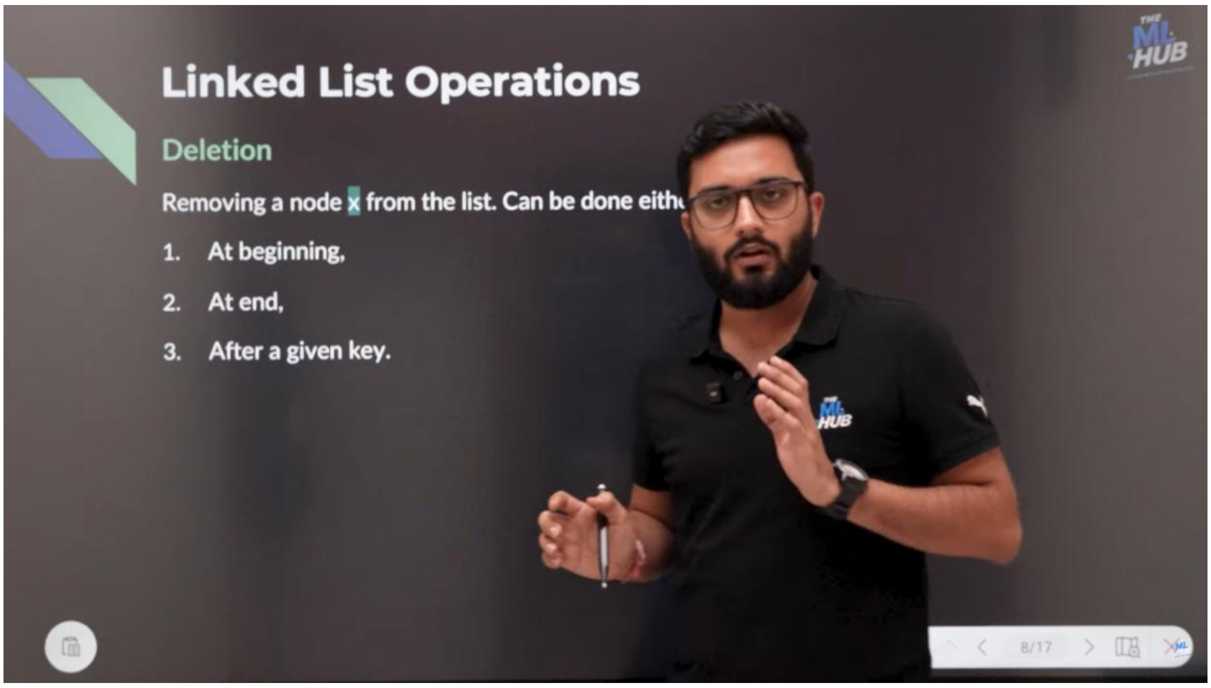


Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

Linked List Operations

Inserting after a Key

```
INSERT_AFTER(head, key, x)
current = head
while current != Null and current.data != key:
    current = current.next
if current == Null:
    error "key not found"
new = Node(x)
new.next = current.next
current.next = new
return head
```

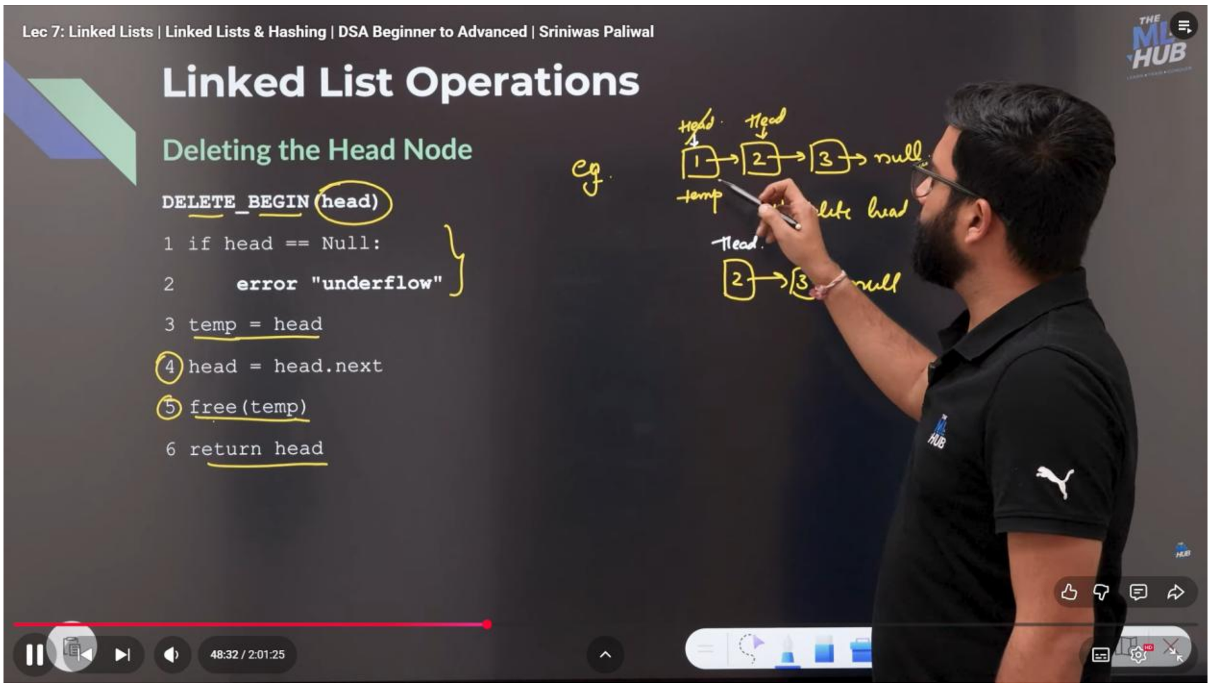


Linked List Operations

Deletion

Removing a node x from the list. Can be done eith

1. At beginning,
2. At end,
3. After a given key.



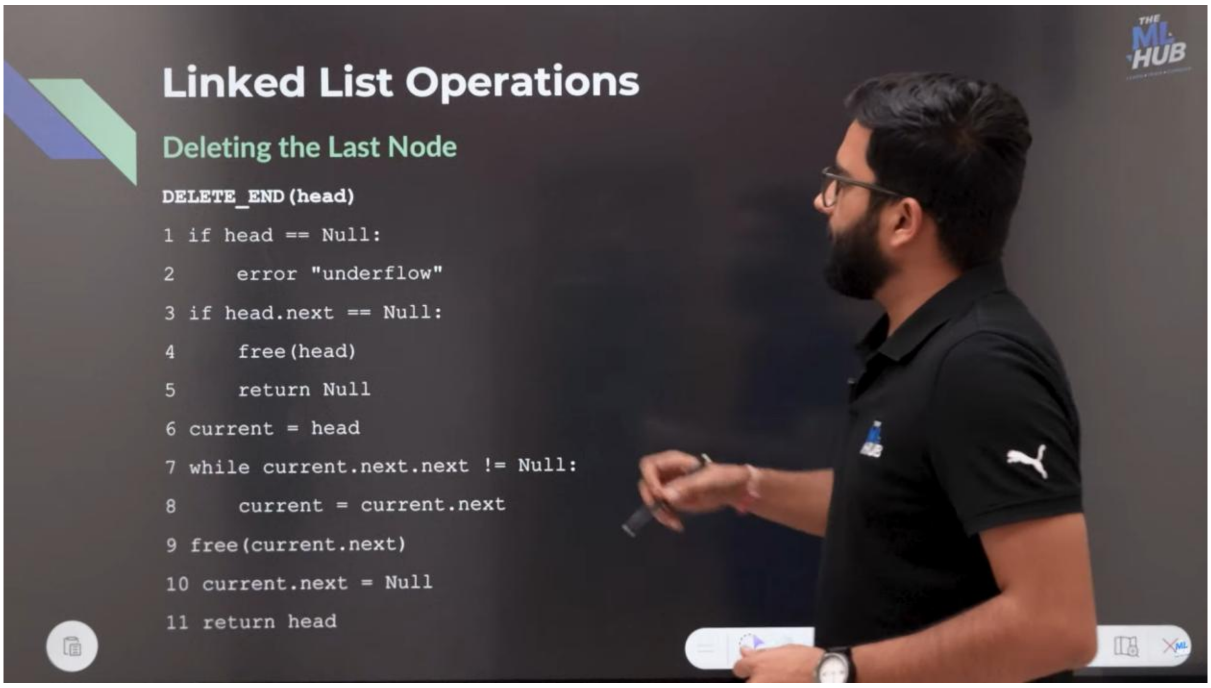
Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

Linked List Operations

Deleting the Head Node

```
DELETE_BEGIN(head)
1 if head == Null:
2     error "underflow"
3 temp = head
4 head = head.next
5 free(temp)
6 return head
```

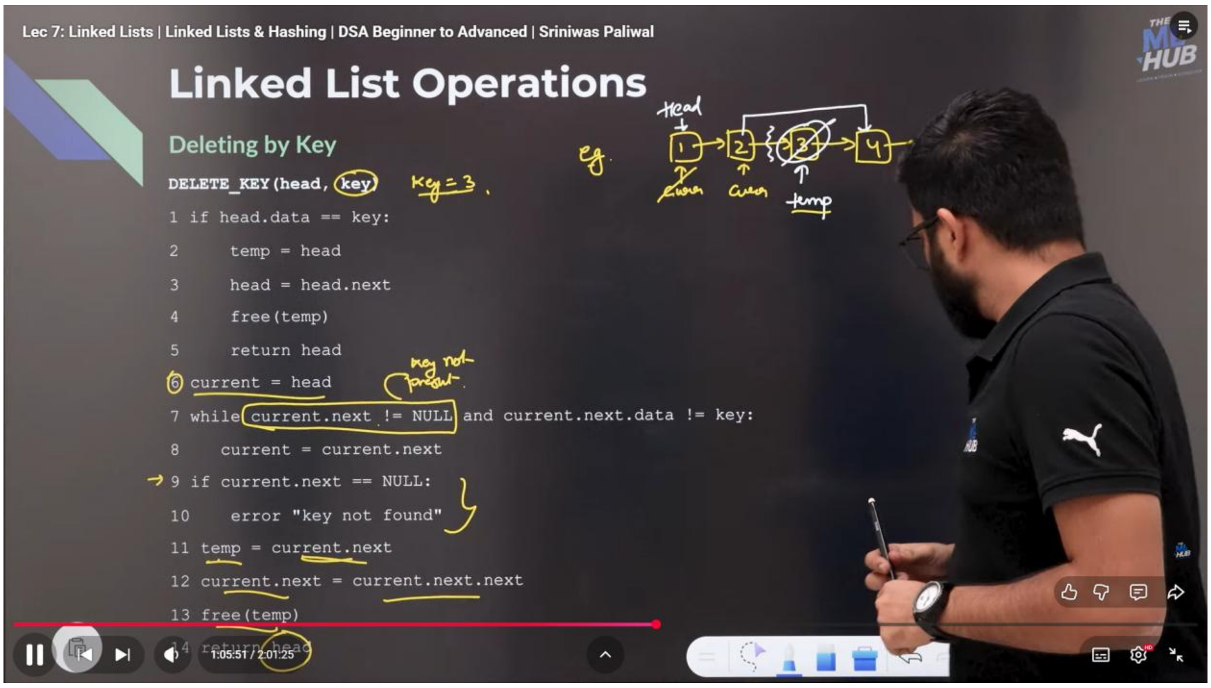
$$\begin{tikzcd} \boxed{1} \arrow[r] & \boxed{2} \arrow[r] & \boxed{3} \arrow[r] & \text{null} \end{tikzcd}$$



Linked List Operations

Deleting the Last Node

```
DELETE_END(head)
if head == Null:
    error "underflow"
if head.next == Null:
    free(head)
    return Null
current = head
while current.next.next != Null:
    current = current.next
free(current.next)
current.next = Null
return head
```

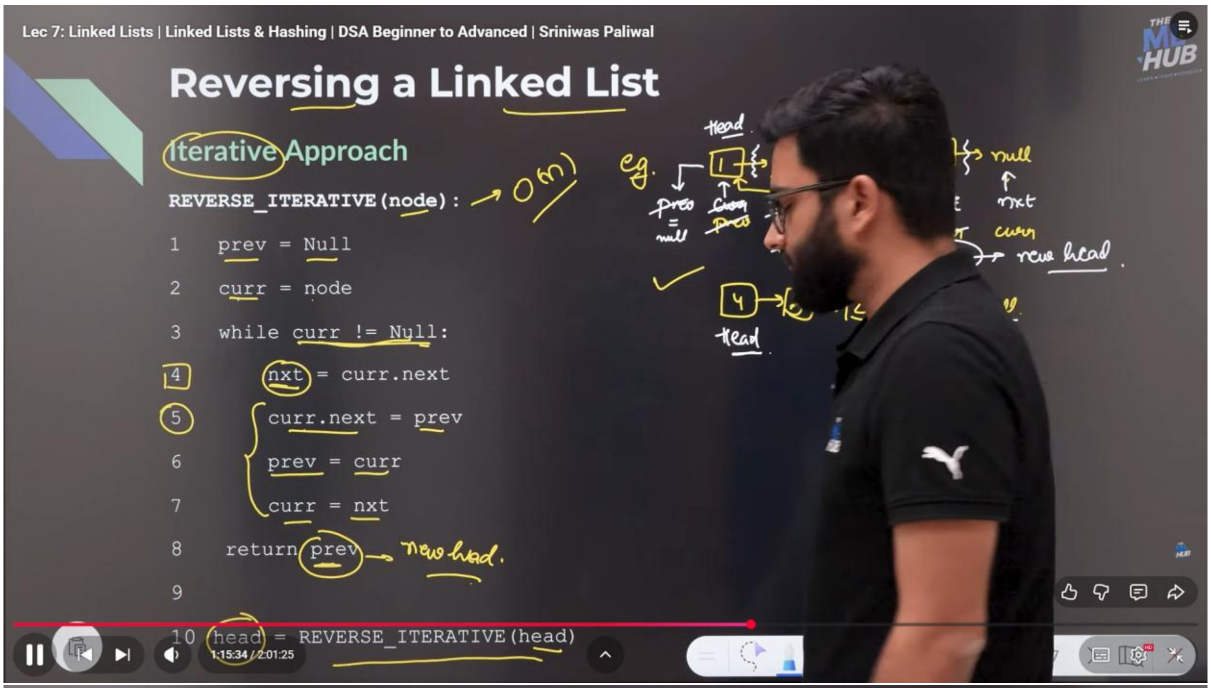


Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

Linked List Operations

Deleting by Key

```
DELETE_KEY(head, key)      Key = 3.
if head.data == key:
    temp = head
    head = head.next
    free(temp)
    return head
current = head
while current.next != NULL and current.next.data != key:
   current = current.next
if current.next == NULL:
    error "key not found"
temp = current.next
current.next = current.next.next
free(temp)
return head
```



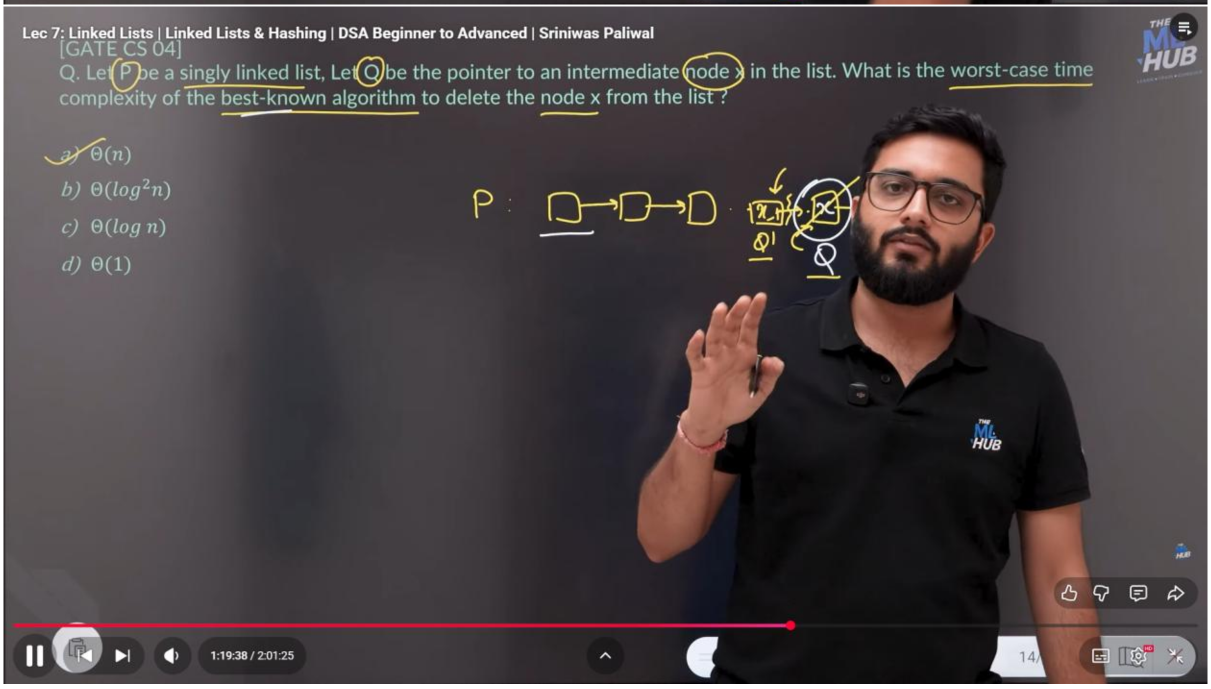
Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

Reversing a Linked List

Iterative Approach

```
REVERSE_ITERATIVE(node):  → O(n)
  prev = Null
  curr = node
  while curr != Null:
      nxt = curr.next
      curr.next = prev
      prev = curr
      curr = nxt
  return prev  → new head.

head = REVERSE_ITERATIVE(head)
```



Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

[GATE CS 04]

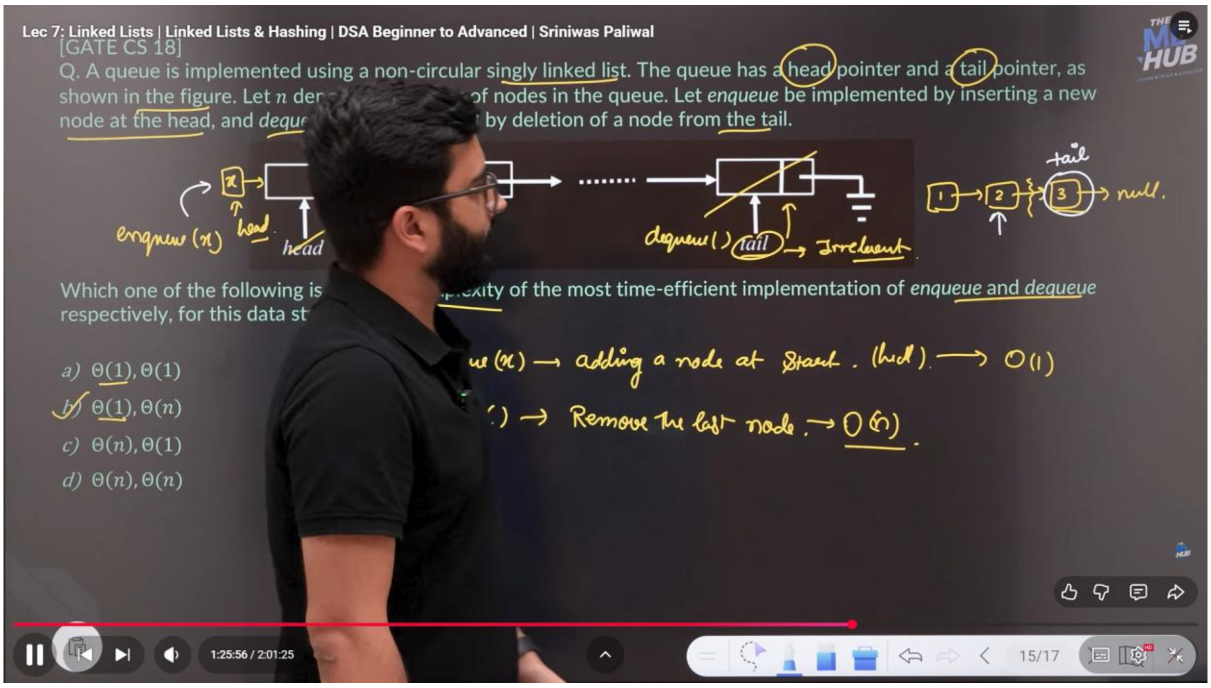
Q. Let P be a singly linked list, Let Q be the pointer to an intermediate node x in the list. What is the worst-case time complexity of the best-known algorithm to delete the node x from the list ?

[✓ a) $\Theta(n)$]

b) $\Theta(\log^2 n)$

c) $\Theta(\log n)$

d) $\Theta(1)$



Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

[GATE CS 18]

Q. A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n den … of nodes in the queue. Let *enqueue* be implemented by inserting a new node at the head, and *deque* … by deletion of a node from the tail.

Which one of the following is … …exity of the most time-efficient implementation of *enqueue and dequeue* respectively, for this data st…

a) $\Theta(1), \Theta(1)$

[✓ b) $\Theta(1), \Theta(n)$]

c) $\Theta(n), \Theta(1)$

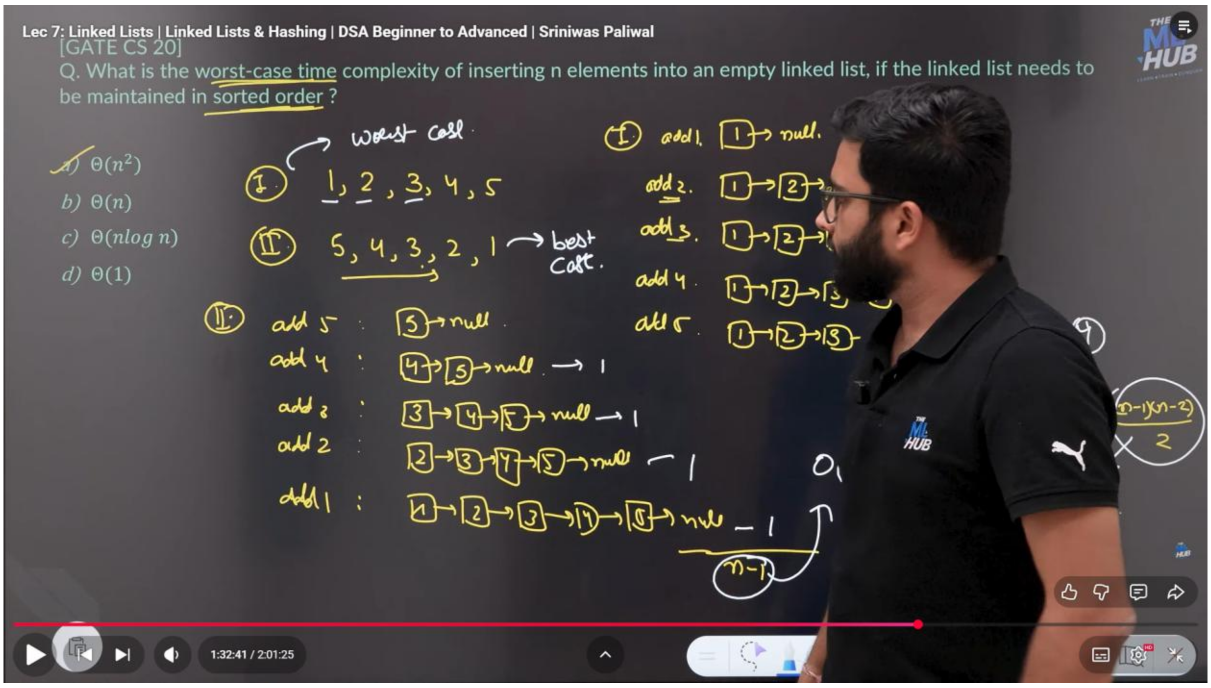
d) $\Theta(n), \Theta(n)$

…ue (x) → adding a node at Start. (head). ⟶ O(1)

…() → Remove the last node. → O(n).

1:25:56 / 2:01:25

15/17



Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal

[GATE CS 20]

Q. What is the worst-case time complexity of inserting n elements into an empty linked list, if the linked list needs to be maintained in sorted order ?

[✓ a) $\Theta(n^2)$]

b) $\Theta(n)$

c) $\Theta(n\log n)$

d) $\Theta(1)$

worst case.

(I) 1, 2, 3, 4, 5

(II) 5, 4, 3, 2, 1 → best case.

(II) add 5 : 5 → null.

add 4 : 4 → 5 → null. → 1

add 3 : 3 → 4 → 5 → null → 1

add 2 : 2 → 3 → 4 → 5 → null — 1

add 1 : 1 → 2 → 3 → 4 → 5 → null — 1

(n-1)

O(

(I) add 1. 1 → null.

add 2. 1 → 2 →

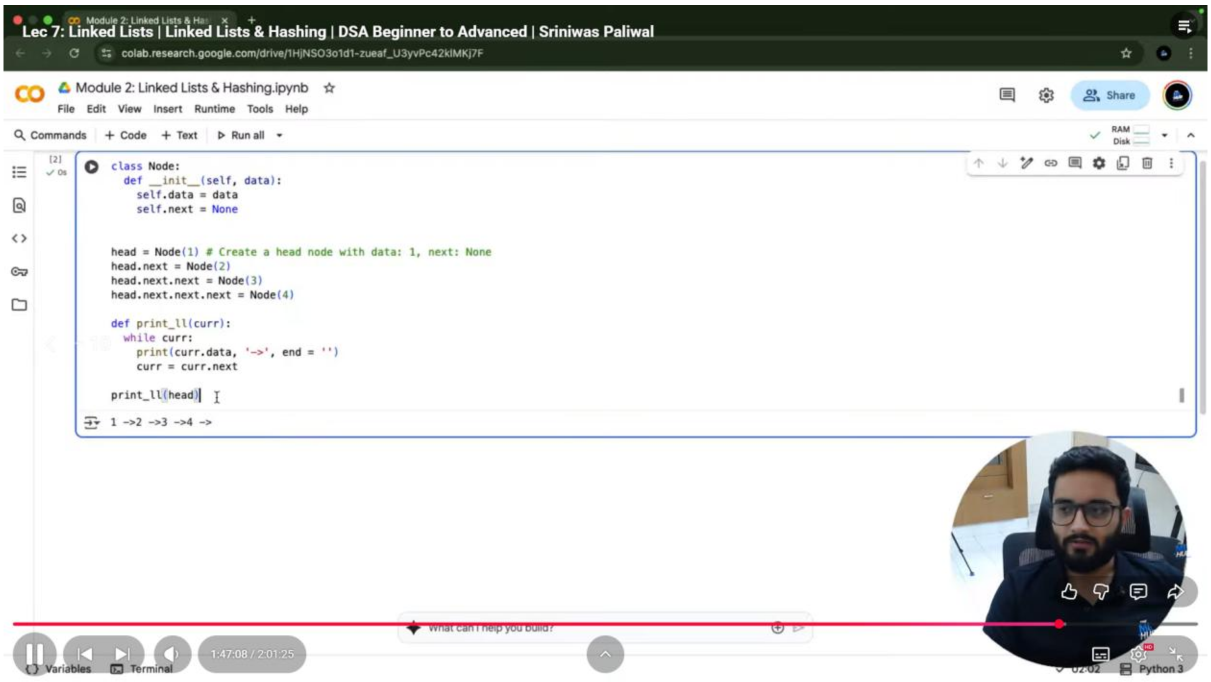
add 3. 1 → 2 →

add 4. 1 → 2 → 3

add 5. 1 → 2 → 3 →

$\frac{(n-1)(n-2)}{2}$

1:32:41 / 2:01:25


Lec 7: Linked Lists | Linked Lists & Hashing | DSA Beginner to Advanced | Sriniwas Paliwal
colab.research.google.com/drive/1HjNSO3o1d1-zueaf_U3yvPc42kIMKj7F
Module 2: Linked Lists & Hashing.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
1:47:08 / 2:01:25
Variables
Terminal
Python 3

```
class Node:
  def __init__(self, data):
    self.data = data
    self.next = None


head = Node(1) # Create a head node with data: 1, next: None
head.next = Node(2)
head.next.next = Node(3)
head.next.next.next = Node(4)

def print_ll(curr):
  while curr:
    print(curr.data, '->', end = '')
    curr = curr.next

print_ll(head)
```

```
1 ->2 ->3 ->4 ->
```