GATE

CRASH COURSE

Data Science & AI

Subject

Data Structure & Algorithm
Linked Lists in Python
Lec No. 03

By – Satya Sir

Last Class

Quick Recap

1. Types Of Queues, Operations
2. Simple Queue, Circular Queue
3. Deque, Priority Queue
4. Hashing, Collision Resolution Techniques
5. Examples

Topics *to be covered*

1. Homework Questions Solution
2. Linked Lists - Operations
3. SLL, DLL – Time Complexities
4. Examples

Homework Question - 1

GATE
2024 – DA

#Q. The fundamental operations in a double-ended queue D are:

insertFirst(e) – Insert a new element e at the beginning of D.

insertLast(e) – Insert a new element e at the end of D.

removeFirst() – Remove and return the first element of D.

removeLast() – Remove and return the last element of D.

In an empty double-ended queue, the following operations are performed:

insertFirst(10)

insertLast(32)

a ←removeFirst() a ← 10

insertLast(28)

insertLast(17)

a ←removeFirst() a ← 32

a ← removeLast() a ← 17

The value of a is 17.

~~10~~	~~32~~	28	~~17~~

a ←

Homework Question - 2

#Q. Consider a double hashing scheme in which the primary hash function is h1(k) = k mod 21, and the secondary hash function is h2(k) = 1+(k mod 19). Assume that the table size is 21. Then the address returned by probe 1 in the probe sequence (assume that the probe sequence begins at probe 0) for key value k = 70 is

___________ .

[✓ A. 0]   B. 1   C. 2   D. 4

$h_1(70) = 70 \% 21$

$= 7$

$h_2(70) = 1 + 70 \% 19$

$= 1 + 13$

$= 14$

In Double Hashing, $h(k, i) = [h_1(k) + i * h_2(k)] \% \text{size}$

$h(70, 1) = [h_1(70) + 1 * h_2(70)] \% 21$

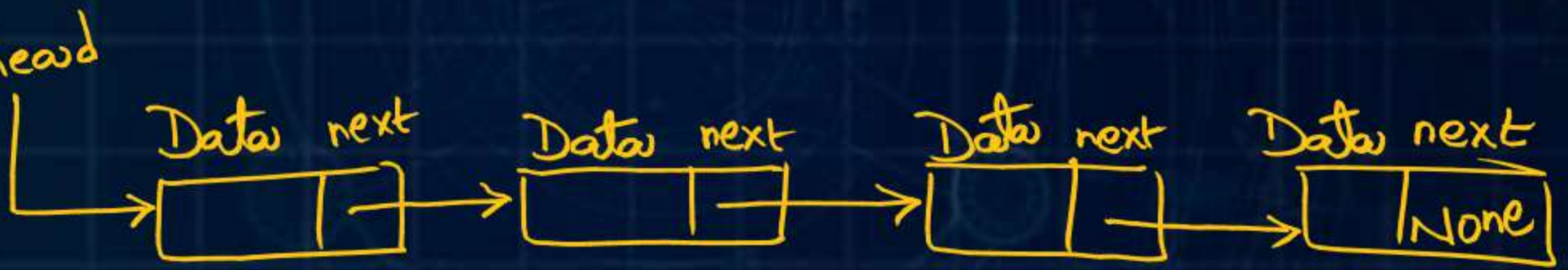
$= [7 + 1 * 14] \% 21 = 21 \% 21 = \underline{\underline{0}}$

Linked Lists

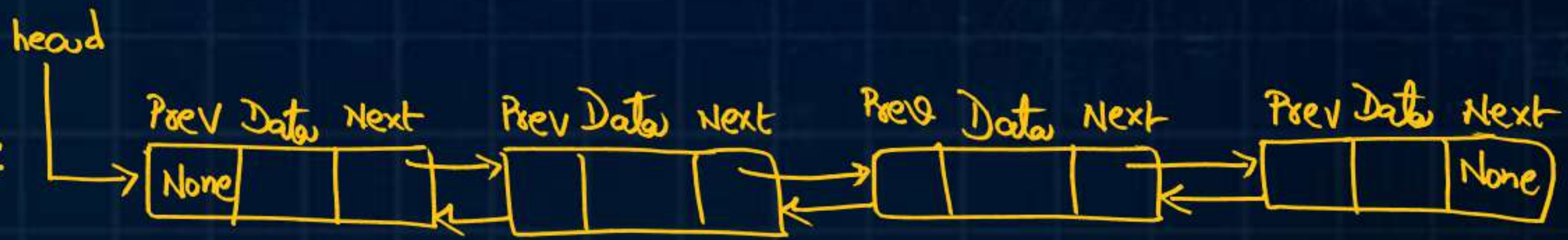
- A Linear DS, in which, Elements of the list are linked / Connected to another Element(s).

- 3 Types of Linked Lists :

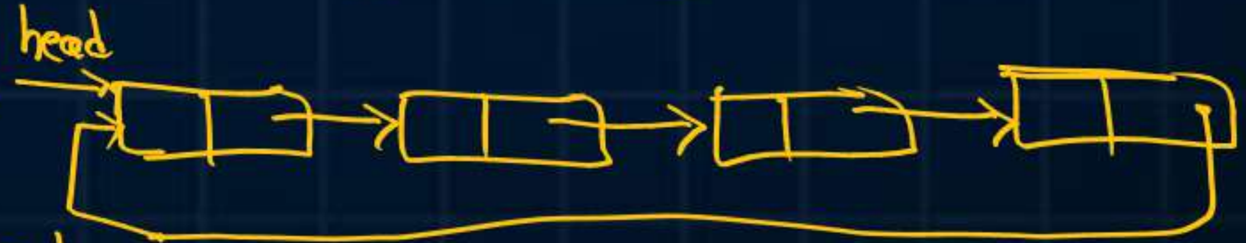
1) Singly Linked List (SLL) :



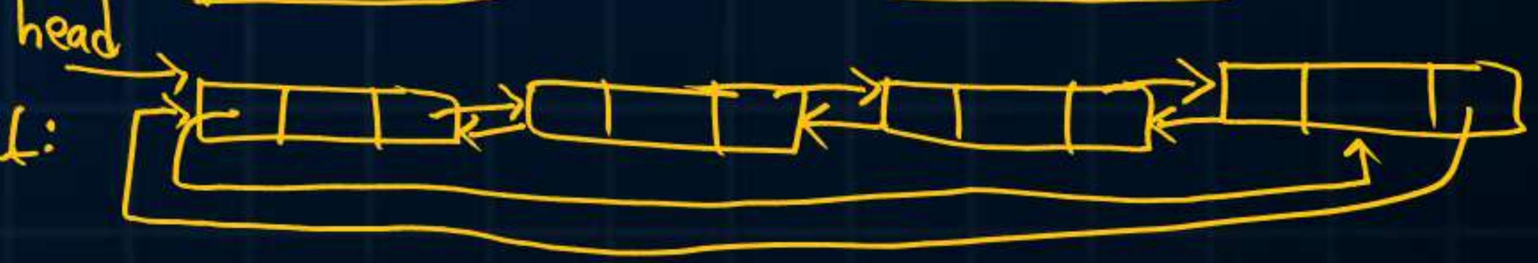
2) Doubly Linked List (DLL) :



3) Circular Linked List → Circular SLL :



→ Circular DLL:

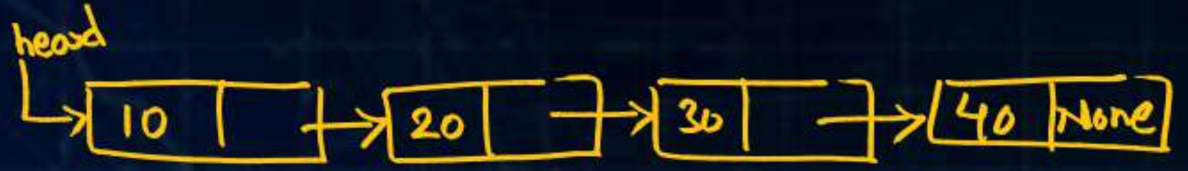


SLL - Insertion

```
class   SLLNode :

    def __init__(self, data):

        self.data = data

        self.next = None

class  sllhead :

    def __init__(self):

        self.head = None
```

Creation of List == Cumulative Insertion of Nodes.

SLL To be Constructed :



Insertion Sequence : 10, 20, 30, 40 ≅ Insertion at End

Insertion sequence : 40, 30, 20, 10 ≅ Insertion at beginning

Insertion Sequence : random sequence : ≅ Insertion at middle.

Insert_At_Beginning : Time Complexity : O(1)

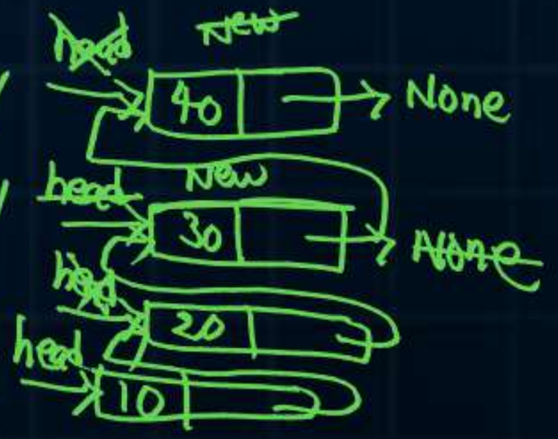
```
def  Insert_begin(Value):
      New = SLLNode(Value)
      if self.head is None : ✗✓
           self.head = New
           New.next = None
           return
                                    New.next = Self.head
                                    self.head = New
```

Insert_begin(40)✓

Insert_begin(30)✓

Insert_begin(20)

Insert_begin(10)

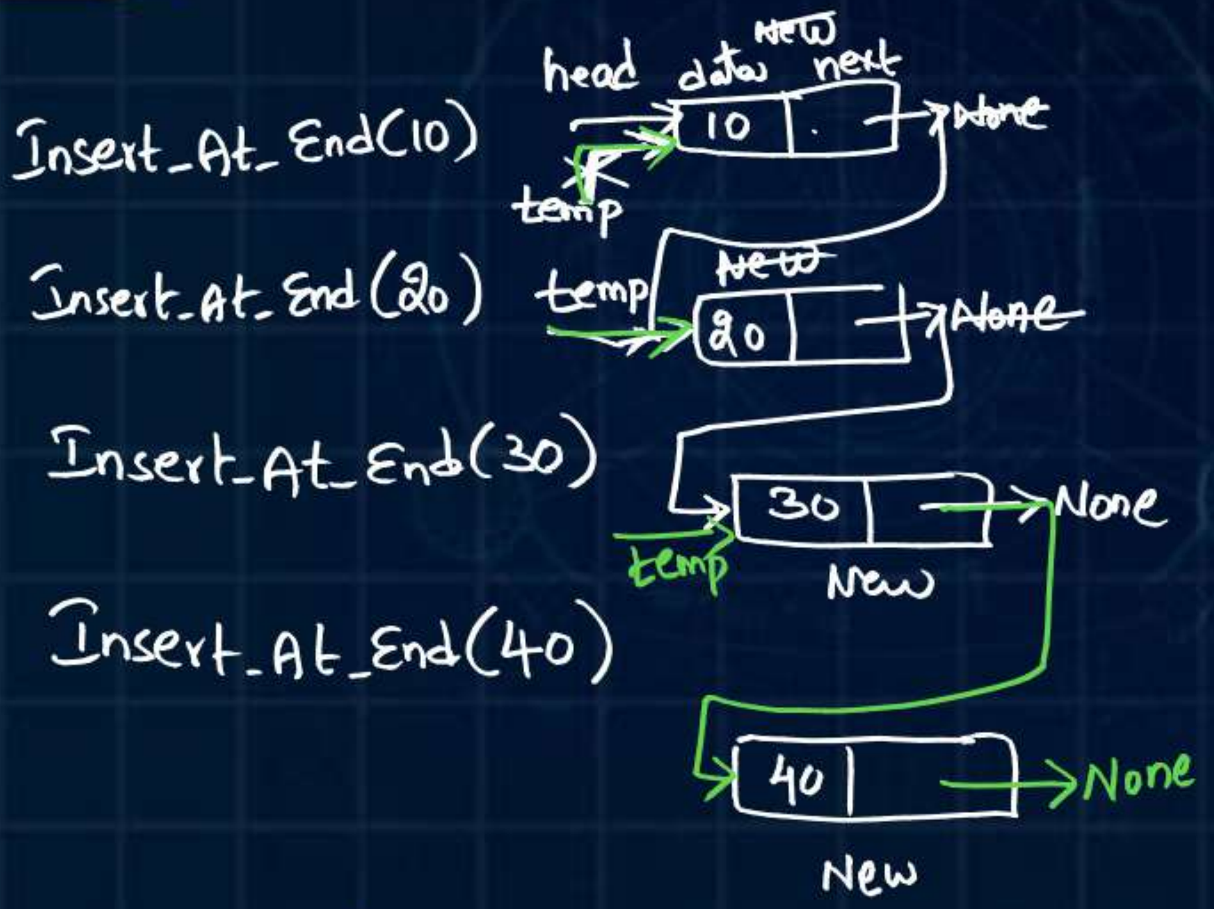


head →10 →20 →30 →40 →None.

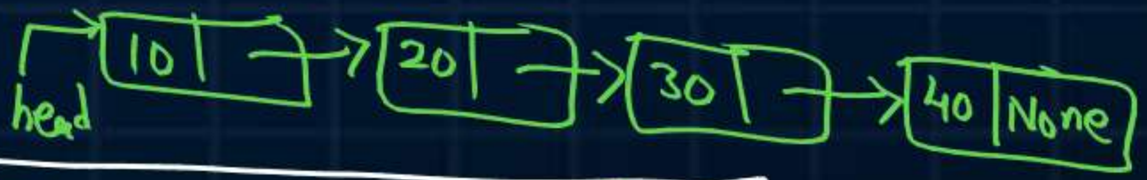
SLL - Insertion

Empty list : head == None.

```
def Insert_At_End(value) :
    new= SLLNode(value)
    if self.head is None:
        Self.head = new
        new.next = None
        return
    temp = SLLNode( )
    temp = Self.head
    while temp.next is not None:
        temp = temp.next
    temp.next = New
    new .next = None
```



Resultant List :



$\boxed{\text{Time complexity : } O(n)}$

```
def Insert_At_middle(value, pos):
    New = SLLNode( value )
    if self.head is None :
        Self.head = New
        New.next = None
        return
    Count = 1
    temp= SLLNode( )
    temp= Self.head
    while Count < Pos-1 :
        temp= temp.next
        Count= Count+1
    new.next = temp.next
    temp.next = new
```

$\boxed{\text{Time Complexity : } O(n)}$

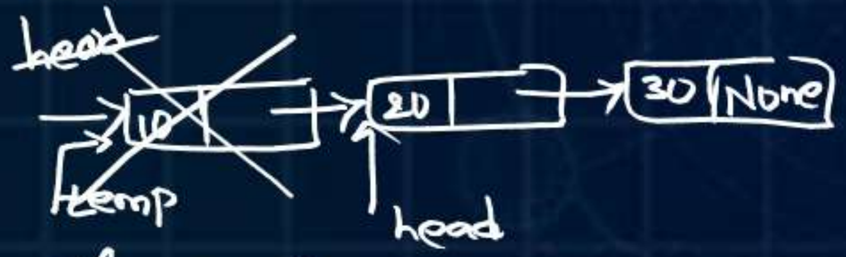
SLL - Deletion

Deletion : First Node / last Node | middle Node :

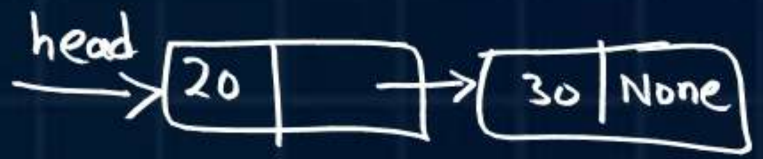
```
def delete_firstnode( ):

    if Self.head is None:
        Print ('Empty list')
        return

    temp= sll Node( )
    temp= self.head
    Self.head= self.head.next

    temp= None
```

Time Complexity : O(1)

Before Deletion



After Deletion

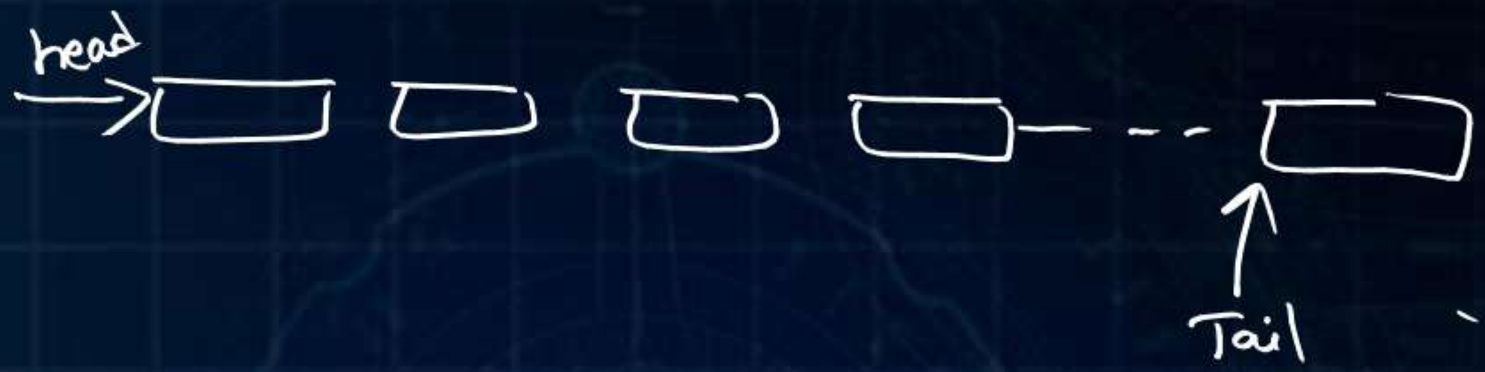


```
# only one element in list
if self.head.next is None:
    self.head = None
    temp = None
```

```
def delete_lastnode( ):

    if self.head is None:
        Print ('Empty List')
        return

    temp=SLLNode( )
    temp = self.head
    While temp.next.next is not None:
        temp= temp.next
    temp1 = temp.next
    temp.next = None
    temp1 = None
```

Time Complexity : O(n)

SLL - Deletion



Time Complexities

Linked List		Insertion			Deletion		
		At Beginning	At middle	At End	first Node	middle node	Last Node
SLL	without Tail	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)
	with Tail	O(1)	O(n)	O(n)	O(1)	O(n)	O(n) ✓
DLL	without Tail	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)
	with Tail	O(1)	O(n)	$\boxed{O(1)}$ ✓	O(1)	O(n)	$\boxed{O(1)}$ ✓
Circular SLL		O(1)	O(n)	O(n)	O(1)	O(n)	O(n)
Circular DLL		O(1)	O(n)	O(n)/O(1)	O(1)	O(n)	O(n)/O(1)

Question

#Q. What does the following function print for a given Linked List with input 1,2,3,4,5,6?

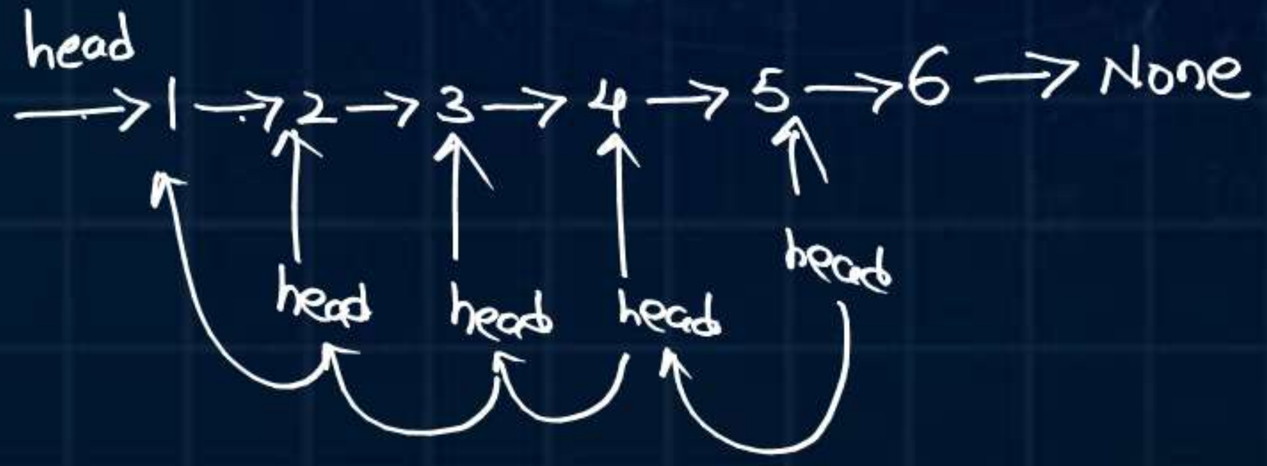
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def fun1(head):
    if head.next.next is None:
        return
    print(head.next.data, end=',')
    fun1(head.next)
    print(head.data, end=', ')
```

a) 2, 3, 4, 5, 6, 6, 5, 4, 3, 2

b) 2, 3, 4, 5, 5, 4, 3, 2

[✓ c)] 2, 3, 4, 5, 4, 3, 2,1

d) 3, 4, 5, 6, 6, 5, 4, 3



o/p: 2, 3, 4, 5, 4, 3, 2, 1

Question

#Q. Let SLLdel be a function that deletes a node in a singly-linked list given a pointer to the node and a pointer to the head of the list. Similarly, let DLLdel be another function that deletes a node in a doubly-linked list given a pointer to the node and a pointer to the head of the list.

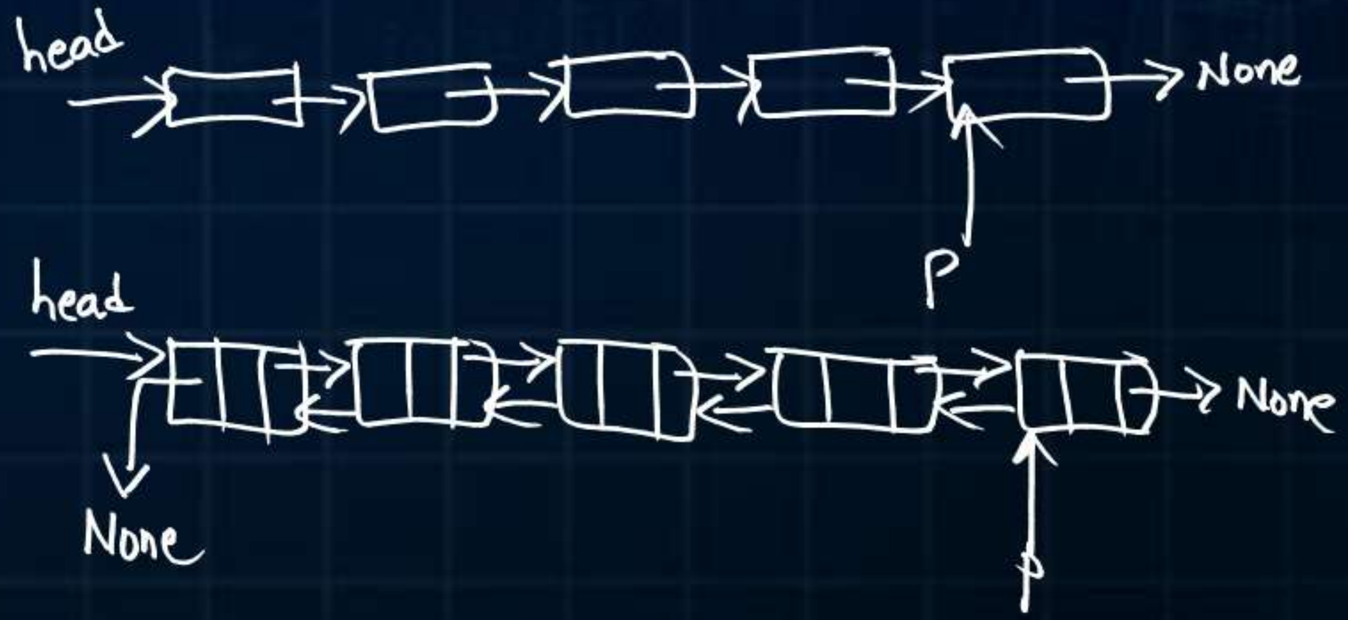
Let n denote the number of nodes in each of the linked lists. Which one of the following choices is TRUE about the worst-case time complexity of SLLdel and DLLdel?

A. SLLdel is O(1) and DLLdel is O(n)

B. Both SLLdel and DLLdel are O(log(n))

C. Both SLLdel and DLLdel are O(1)

[✓ D.] SLLdel is O(n) and DLLdel is O(1)



Summary

- Linked Lists
  - SLL operations
  - DLL operations
  - Time complexities

Thank

THANK

Keep Hustling!