Types of Linked List - Part I

Course on C-Programming & Data Structures: GATE - 2024 & 2025

Vishvadeep Gothi • Lesson 29 • Jan 4, 2023

Data Structure:
Linked List 2

By: Vishvadeep Gothi

Hello!

I am Vishvadeep Gothi

I am here because I love to teach

Traversing in Linked List

```
p = start
while (p)
{
  process p -> data
  p = p -> link
}
```

Number of Elements in Linked List

Address of Last Node in Linked List

```
p = start
if (p == NULL) return NULL;
while (p -> link)
{

  —


}
```

Question

What would be the output after the following code segment is executed on a valid NULL terminated singly linked list.

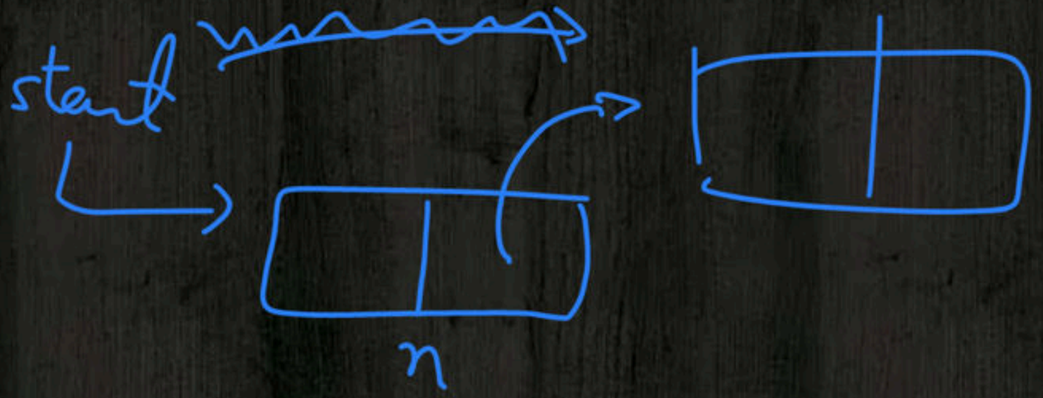
```
struct node *p;
p = start ;
while(p → link → link)
{
p  = p→ link;
}
printf("%c", p → data );
```

Insertion in Linked List

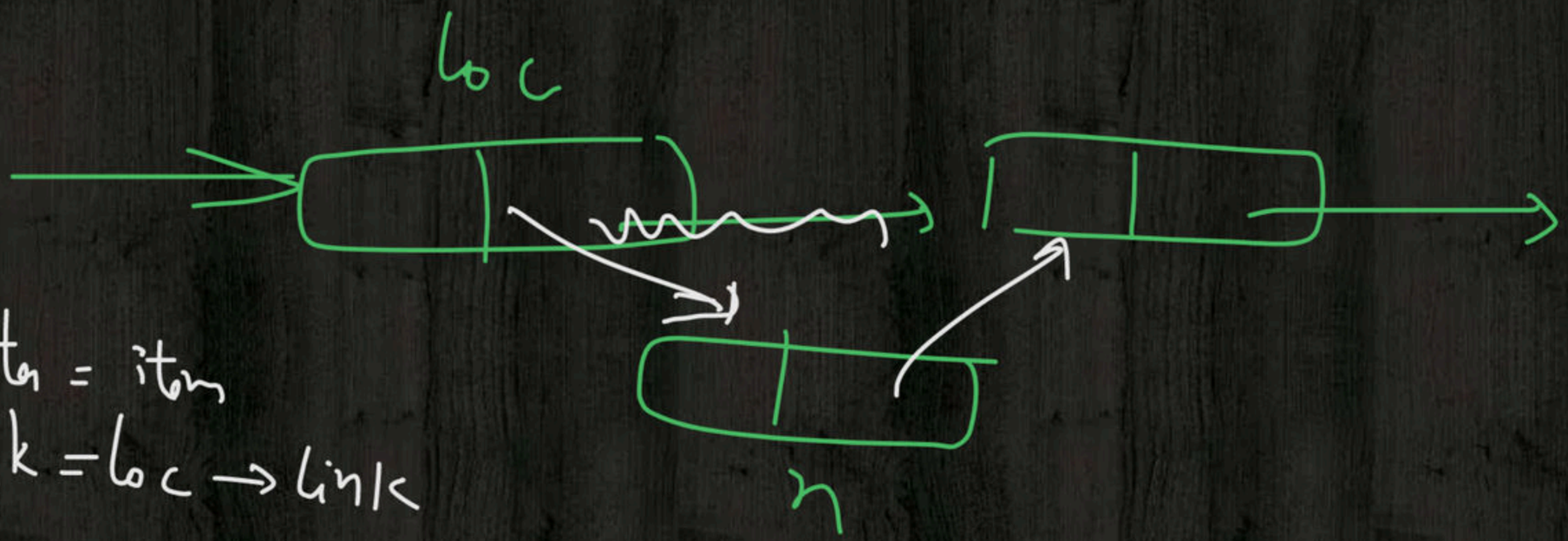
1. Insertion at beginning
2. Insertion after given node
3. Insertion at the end

Insertion at beginning

```
n -> data = item
n -> link = start
start = n
```



Insertion After Given Node



n→data = item

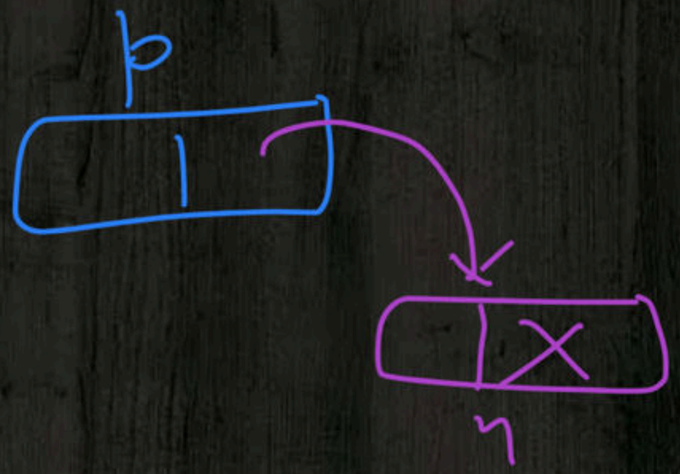
n→link = loc→link

loc→link = n

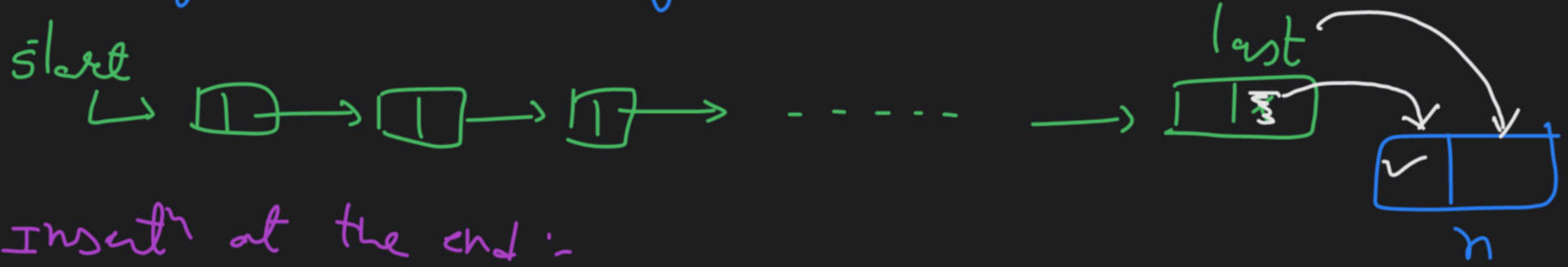
Insertion At the End

```
p = start
while (p → link)
{
   p = p → link;
}
n → data = item
n → link = NULL
p → link = n
```

$O(n)$



if add of last node is given

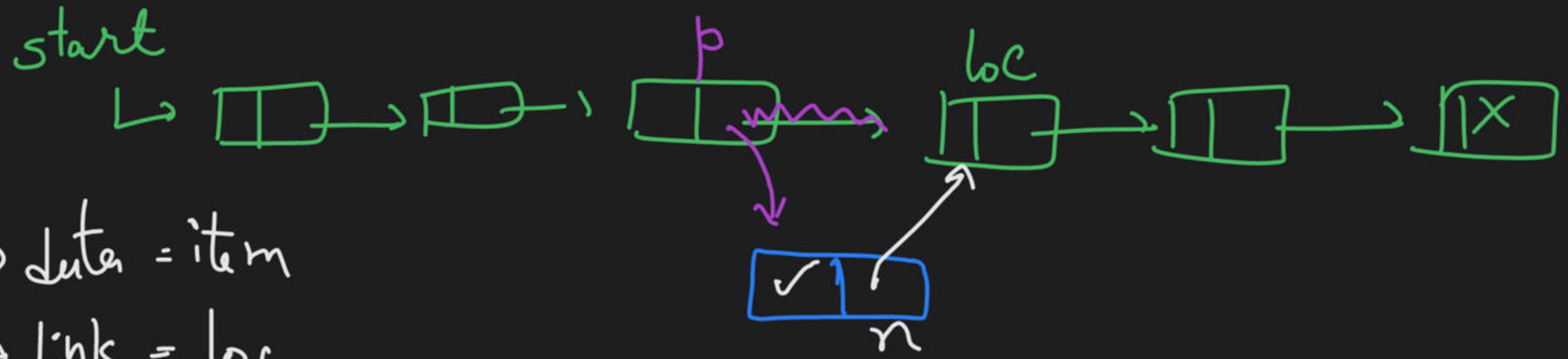


Insertn at the end :-

```
n -> data = item
n -> link = NULL
last -> link = n
last = n
```

$\theta(1)$

Insertion before a given node:-



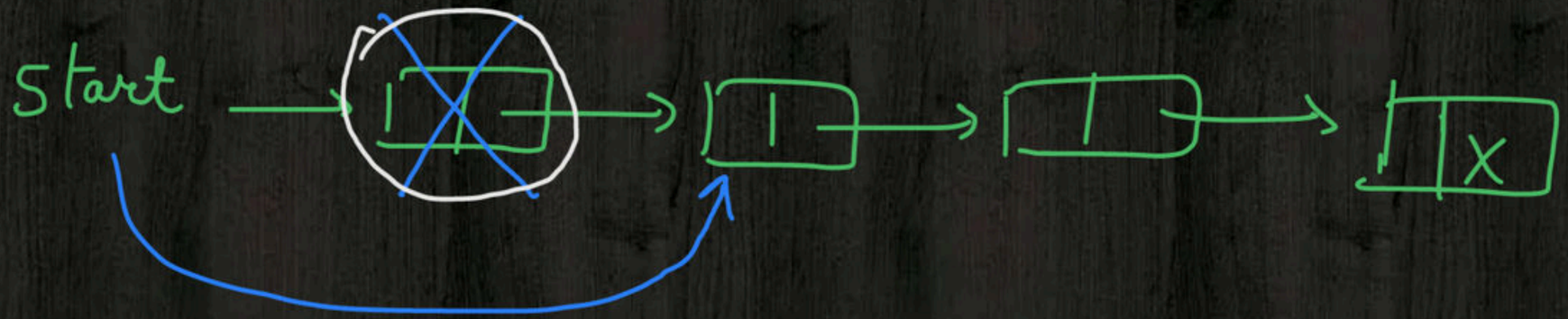

```
n -> data = item
n -> link = loc
struct node *p = start
while (p -> link != loc)
{
  p = p -> link
}
p -> link = n
```

R.T. Complexity = O(n)

Deletion in Linked List

1. Deletion at beginning
2. Deletion of given node
3. Deletion at the end

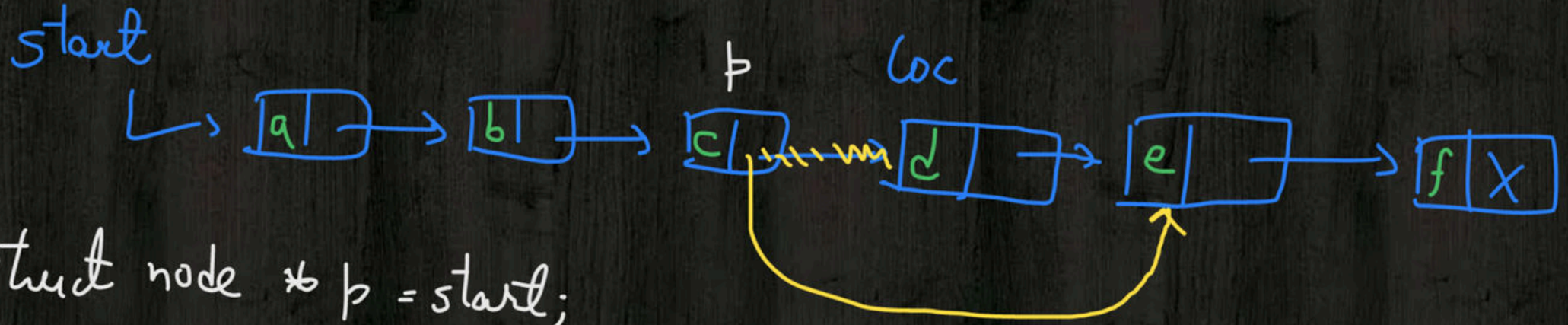
Deletion At Beginning




```
struct node *p = start
start = start -> link;
free(p);
```

$O(1)$

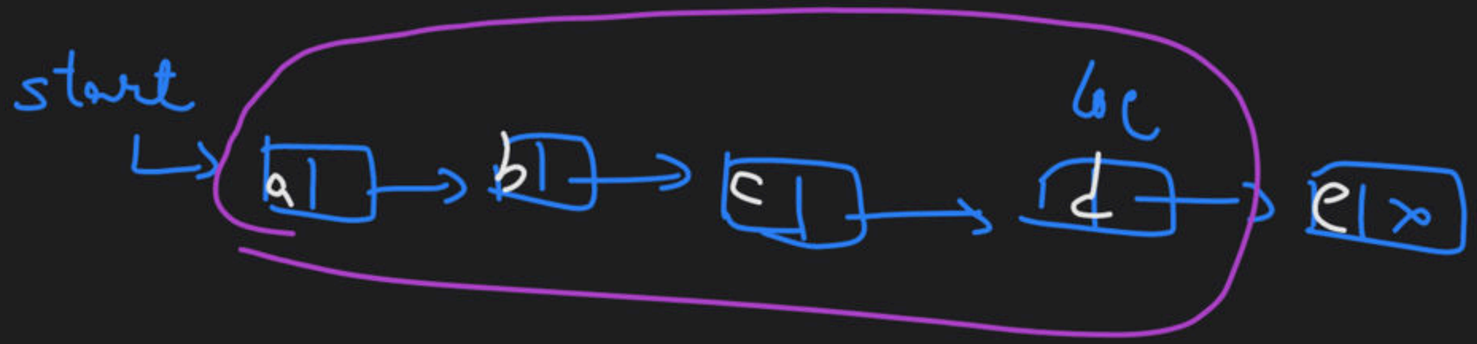
Deletion Of a Given Node



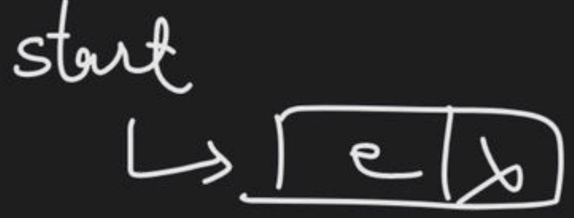

```
struct node *p = start;
while (p -> link != loc)
{
   p = p -> link;
}
p -> link = loc -> link
free (loc)
```

R.T.C. $\Rightarrow O(n)$

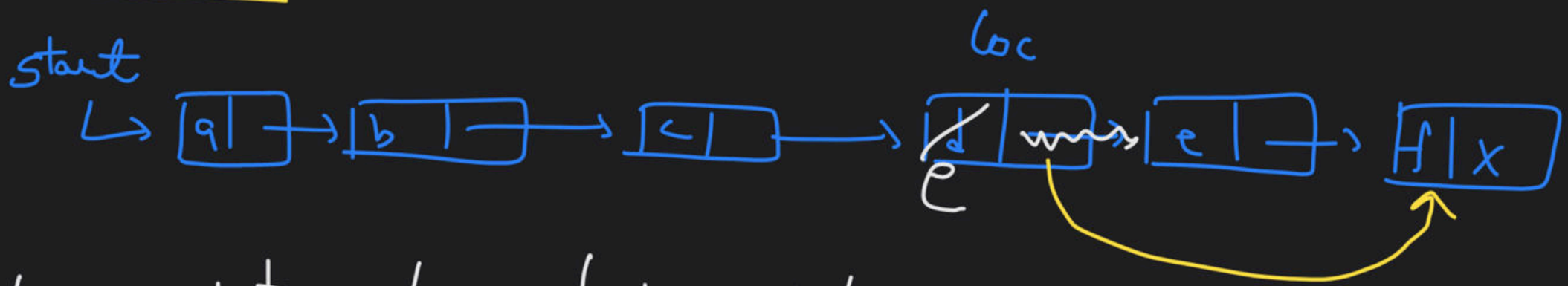
Invalid : (1)



start = loc → link



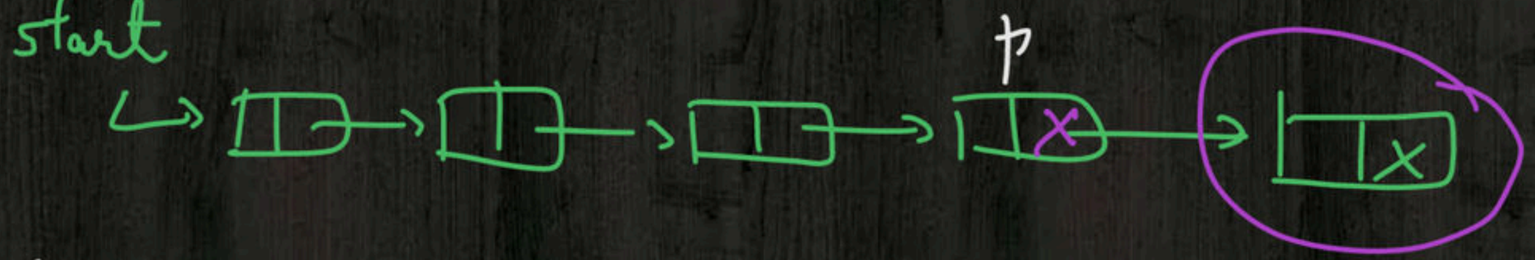
Invalid 2 :-



loc → data = loc → link → data

loc → link = loc → link → link

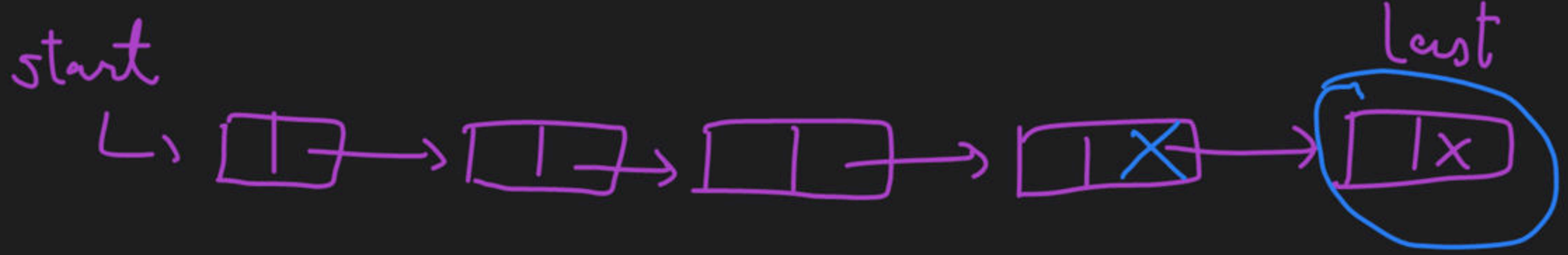
Deletion At the End




```
struct node *p = start;
while (p->link->link)
{
   p = p->link;
}
p->link = NULL;
```

R.T. Complexity = $\Theta(n)$

Deletⁿ of last node, when address of last node is given :-



R.T. Complexity = $\theta(n)$

Searching in linked-list :-

Successful search => return add. of node where element found

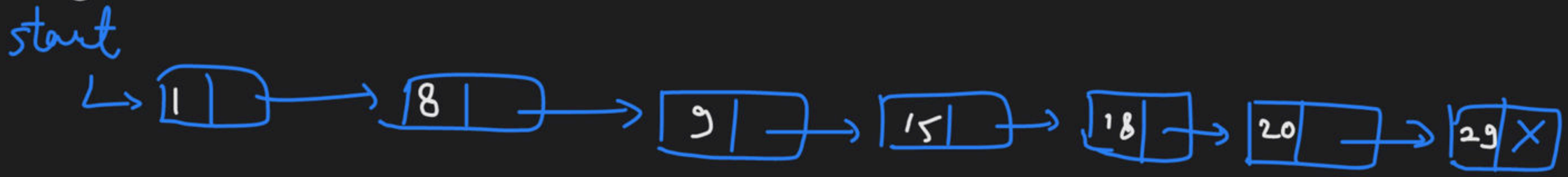
unsuccessful search => return NULL

Linear search in LL. :-

```
struct node *p = start;
while (p)
{
   if (p -> data == item)
        return p;
   p = p -> link;
}
return p or return NULL;
```

R.T. complexity = O(n)

Binary search in LL :-



B.S. in LL can not be as efficient as it is on array because in LL, reaching to mid node is not possible in constant time.

sets:-

$S1 = \{a, b, c, d\}$

$S2 = \{c, d, e, f\}$

Union	Intersection
$S1 \cup S2$	$S1 \cap S2$
$\Rightarrow \{a, b, c, d, e, f\}$	$\{c, d\}$

Question 1 GATE-2004

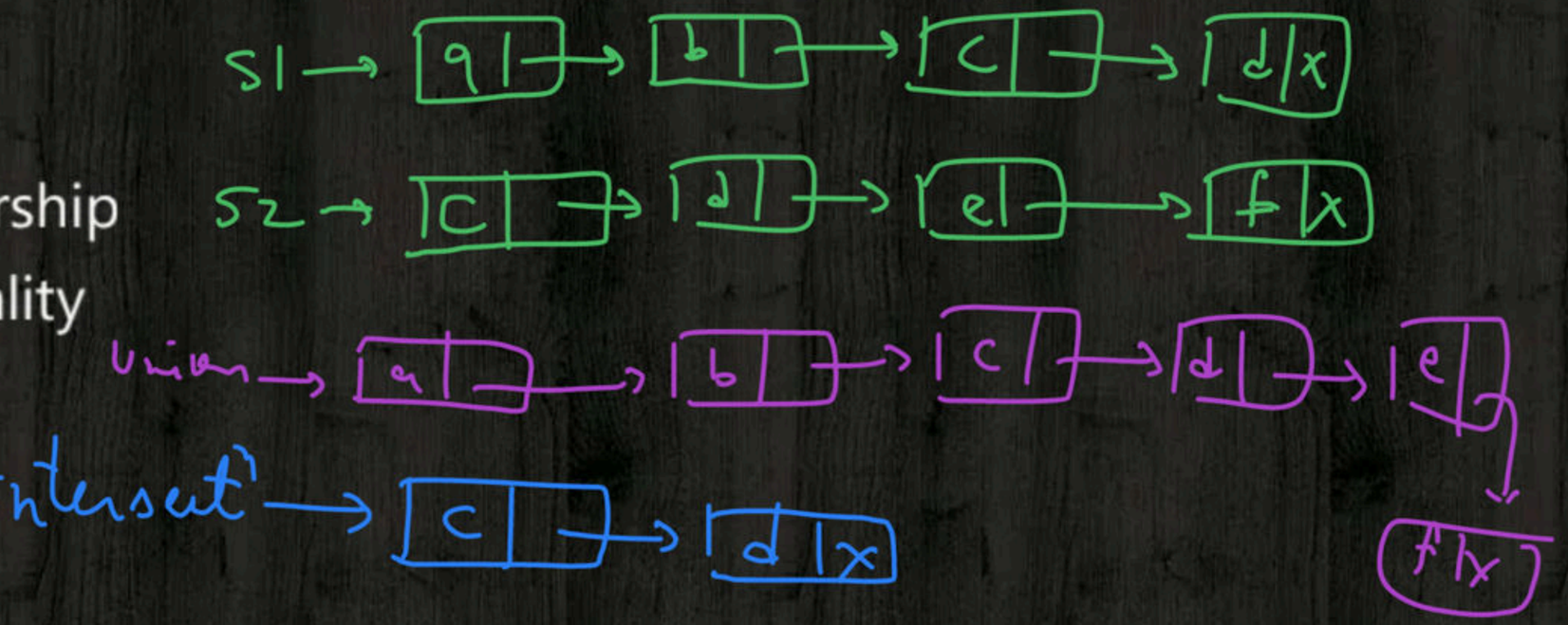
Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership and cardinality will be the slowest?

(A) Union only

(B) Intersection, membership

(C) Membership, cardinality

(D) Union, intersection



$S1 = \{a, b, c, d\}$

$S2 = \{c, d, e, f\}$

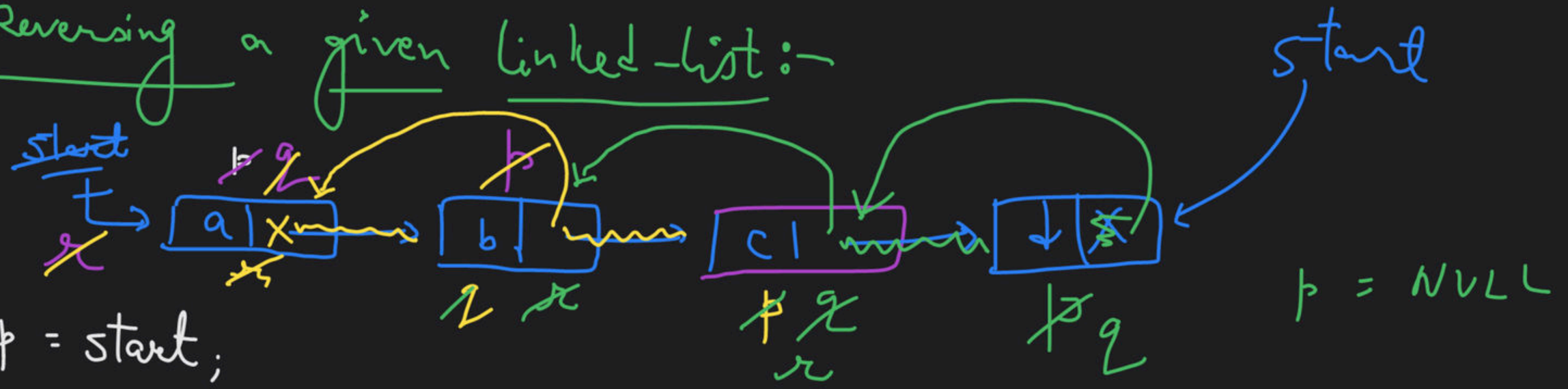
membership ⇒ searching ⇒ $O(n)$

cardinality ⇒ finding no. of elements ⇒ $\Theta(n)$

union ⇒ ⇒ $O(n^2)$

intersect ⇒ ⇒ $O(n^2)$

Reversing a given linked list :-




```
p = start;
q = NULL;
r;
while (p)
{
    r = q;
    q = b;
    b = p->link;
    q->link = r;  q  start = q;
```

R.T. Complexity = $\theta(n)$

space —"— = $\theta(1)$

Application of Linked List

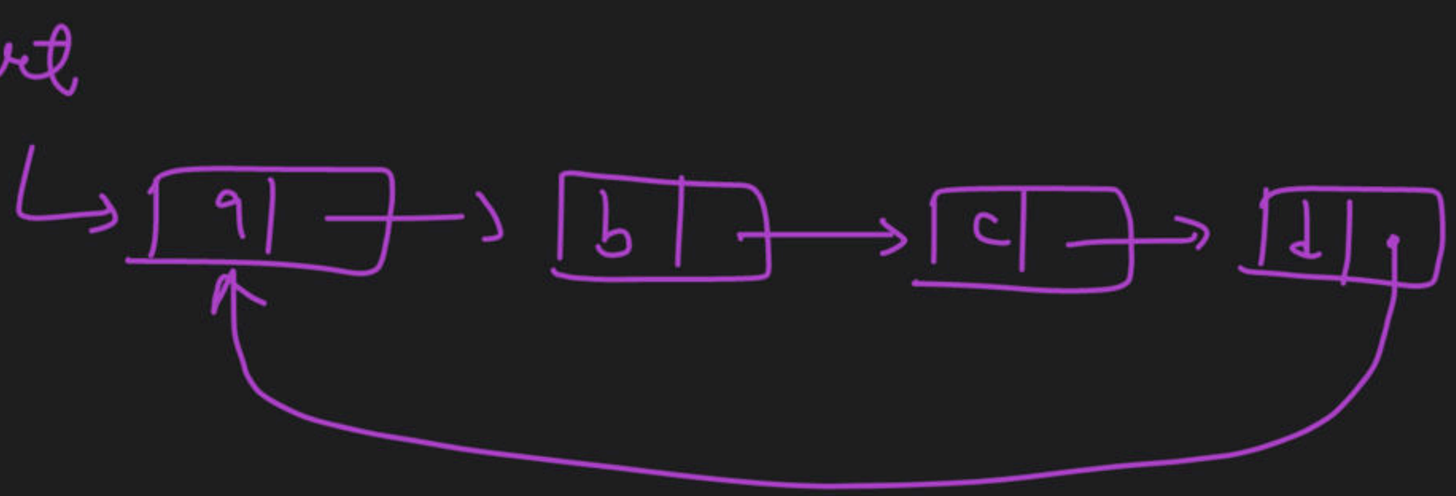
- To store polynomials
  - Univariate
  - Bivariate

Disadvantage of Singly Linked List

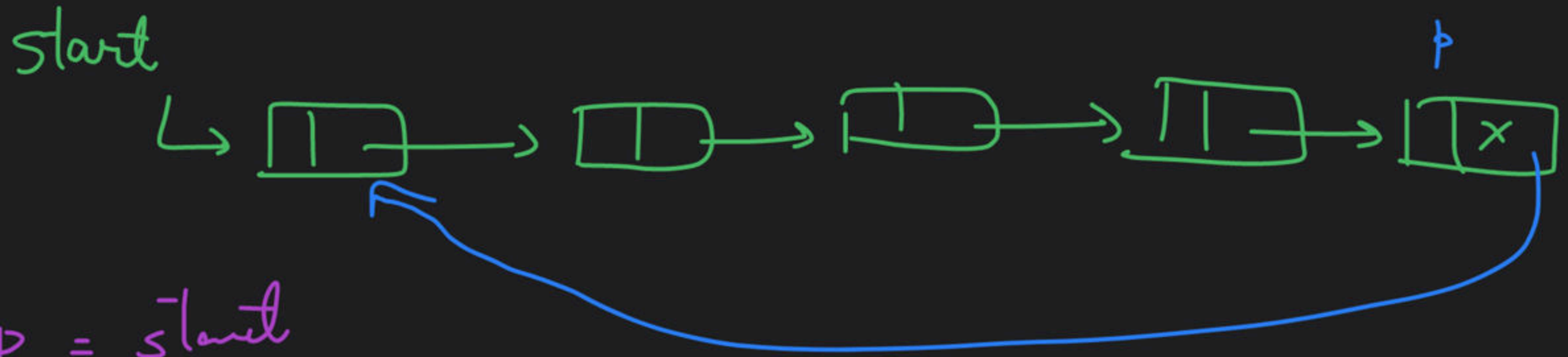
- The link part of last node is not utilized → circular list
- The address of predecessor is not known
- Stepping backward is not possible

→ doubly linked list

Circular List:-



Convert a NULL terminated list into a circular list:-




```
p = start
while (p -> link)
{
   p = p -> link;
}
p -> link = start;
```

R.T. Complexity = $\Theta(n)$

Header List

- Contains special first node called **Header node**.
- Header node contains some summery information.

Types of Header List

1. Grounded
2. Circular

Doubly Linked List

Insertion in Doubly Linked List

Deletion in Doubly Linked List

Question 1

- How many links updated when insertion and deletion done in doubly linked list?

Question 2

The following code is given to reverse the doubly linked list. Fill the blank?

```
void reverse(Dnode *head)
{
struct Dnode *p = head;
while(p ! = head)
{
swap(p → forw, p → back);
_____________
}
swap( head → forw, head →  back)
}
```

DPP

Question 1

Given a singly linked list, check whether there is any loop in the list or not?

Question 2

Find the intersection point of a Y-shaped List?

Question 3

Write an algorithm to reverse a singly linked list?

Happy Learning