

Case 1:

Insertion in a Linked List

13 August 2020 13:42

Head → 8 → 9 → 11 → 7 → 2 → NULL

2 → 1 → 2 → ptr

→ Case 1: Insert at the beginning ✓  $O(1)$

→ Case 2: Insert in between

→ Case 3: Insert at the end

→ Case 4: Insert after a Node

5 →  
50001 →

struct Node \* ptr = (struct Node \*) malloc(  
Sizeof(struct Node))

ptr → next = head;

head = ptr;

return head;

Time complexity =  $O(1)$

## Case 2:

Insertion in a Linked List

13 August 2020 13:42

5 →  
50000 →

Struct Node \* ptr = (Struct Node \*) malloc( sizeof(Struct Node) )

① ptr → next = p → next;  
p → next = ptr;

- Case 1: Insert at the beginning ✓  $O(1)$
- Case 2: Insert in between ✓  $O(n)$
- Case 3: Insert at the end
- Case 4: Insert after a Node

Time Complexity =  $O(n)$  → for the case when we have to insert the element at position 4 b/c we have to traverse through all the elements via pointer to get there i.e. all  $n$  elements

### Case 3:

Insertion in a Linked List

13 August 2020 13:42

Head → 8 → 9 → 11 → 7 → 2 → NULL

5 →  
50001 →

ptr

Struct Node \* ptr = (Struct Node \*) malloc( Sizeof(Struct Node) )

ptr → next = ptr  
ptr → next = NULL;

- Case 1: Insert at the beginning ✓  $O(1)$
- Case 2: Insert in between ✓  $O(n)$
- Case 3: Insert at the end ✓  $O(n)$
- Case 4: Insert after a Node

Time Complexity =  $O(n)$  → reason same as in case 2

Case 4:

Address of **q** is given

Insertion in a Linked List

13 August 2020 13:42

Head → **p** → 1 → 2 → 3 → 4 → 5 → 60000 →

8 → 9 → 11 → 7 → 2 → NULL

q → 11

Struct Node \* ptr = (Struct Node \*) malloc( sizeof(Struct Node) )

→ Case 1: Insert at the beginning ✓  $O(1)$

→ Case 2: Insert in between ✓  $O(n)$

→ Case 3: Insert at the end ✓  $O(n)$

→ Case 4: Insert after a Node ✓  $O(1)$

$$\begin{cases} ptr \rightarrow next = q \rightarrow next \\ q \rightarrow next = ptr \end{cases}$$

Time Complexity =  $O(1)$  → since address of 'q' is given