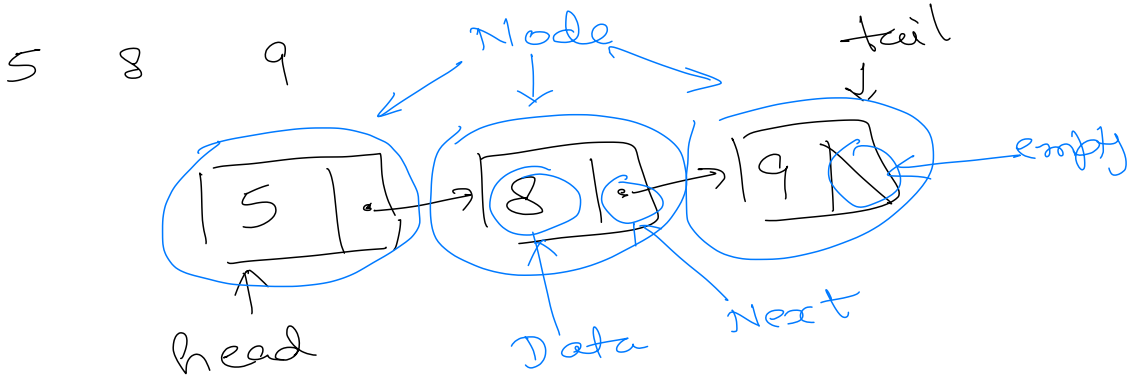
# Singly linked list




**Traversal →** Starting from first element, access each element one at a time, till the last element.

```
for (i=0; i < a.size(); ++i)
    a[i] ...
```

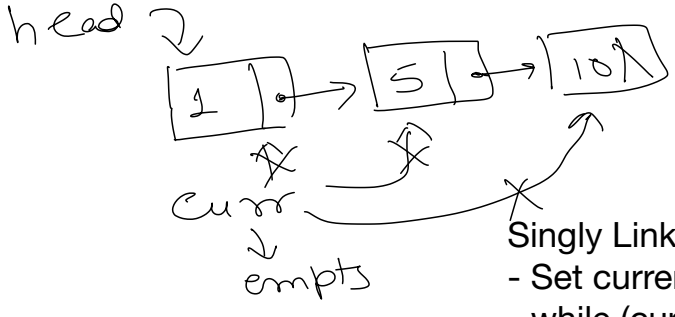
Array Traversal

**Empty list**

head → empty

[tail →]

Singly LinkedList Traversal
- If list is empty then stop.
- Set current to first node of list.
- while (current is not empty) do
  - Process current node.
  - Set current to current node's next.




Singly LinkedList Traversal (Optimised)
- Set current to first node of list.
- while (current is not empty) do
  - Process current node.
  - Set current to current node's next.

For empty list
head → empty
curr → empty.

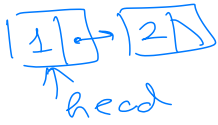
# Create a linked list

↳ Adding elements to be stored in list.

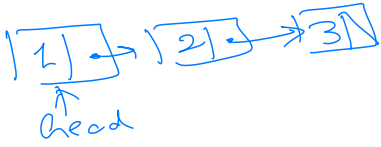
→ Add a new element at start of list


Add 3 at start of list

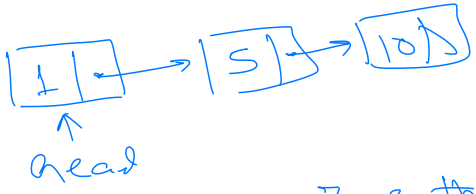

→ Add a new element at end of list



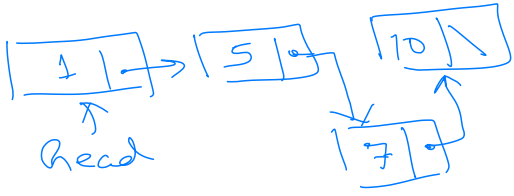

Add 3 at end of list




→ Insert in the list.

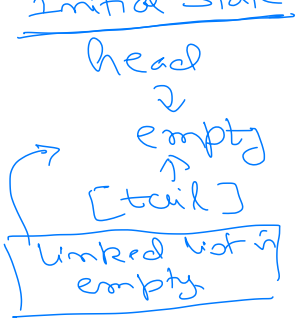


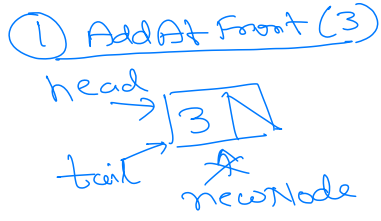

Insert / Add 7 so that list remains sorted.

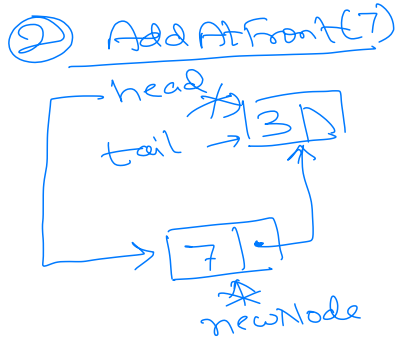


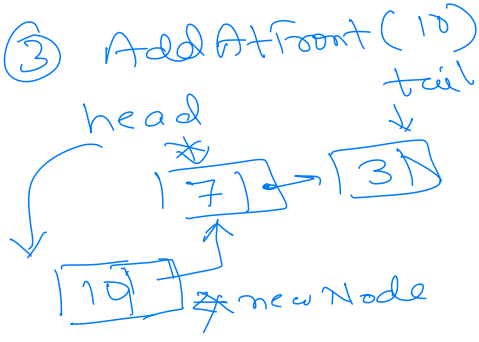
1. Add a new element at the start of list.

AddAtFront(element)
- Make space for new element, say newNode.
- Store element in newNode's data.
- Set newNode's next to empty.
- if list is empty then
  - Set head and tail to newNode.
  - Stop.
- Set newNode's next to head.
- Set head to newNode.



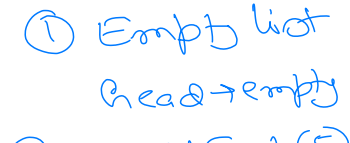



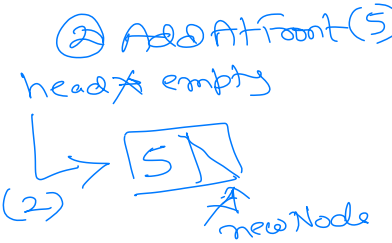



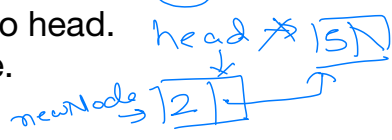




---

AddAtFront(element) - List having only head pointer.
- Make space for new element, say newNode.
- Store element in newNode's data.
- Set newNode's next to empty.
- if list is empty then
  - Set head to newNode.
  - Stop.
- Set newNode's next to head.
- Set head to newNode.

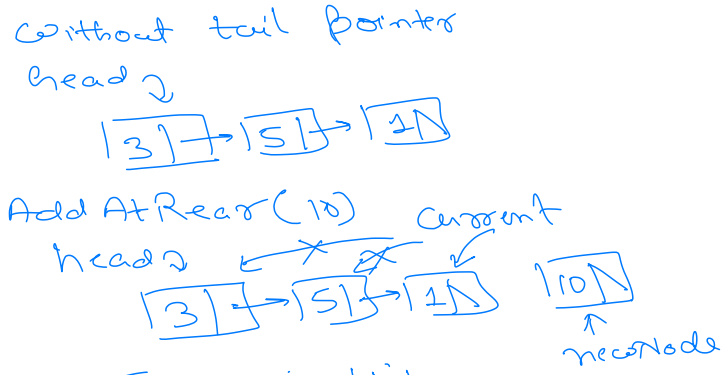


2- Add a element at the end of list.

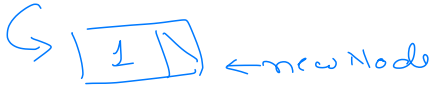
→ Without tail pointer




→ Traverse list to reach last node.

→ Set newNode as next of last node.

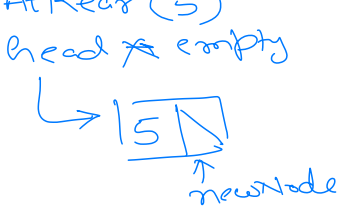

Empty list ← // AddAtRear(1)



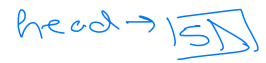

① Emty list

head → empty

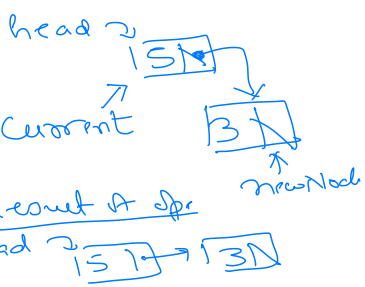
② AddAtRear(5)




Result of operation




③ AddToRear (3)

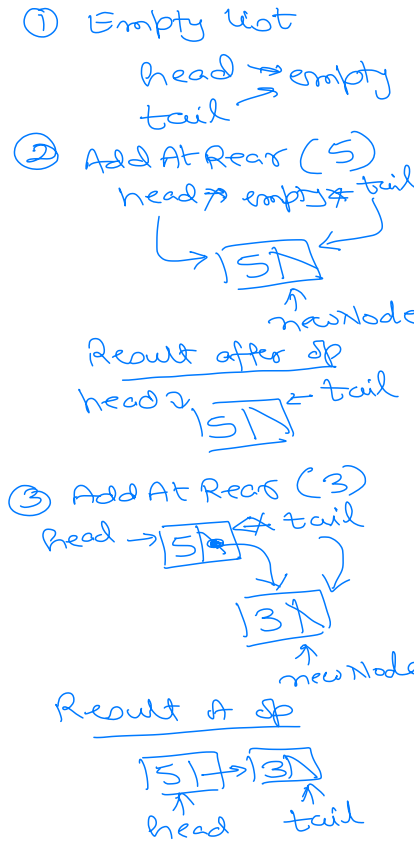



AddAtRear(element) - with no tail pointer.
- Make memory for new element, say newNode.
- Store element in newNode's data.
- Set newNode's next to empty.
- if list is empty then
  - Set head to newNode.
  - Stop.

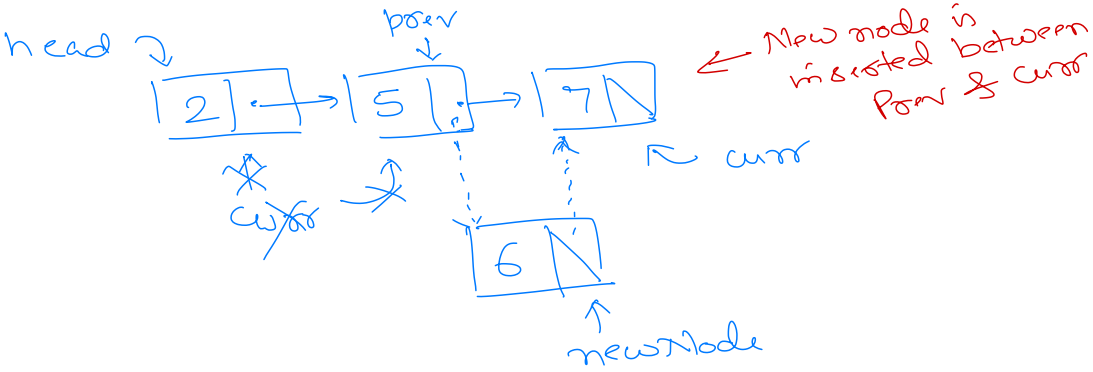
// List is not empty. Traverse the list to find the last node.
- Set current to head.
- while (current's next is not empty)
  - Set current to current's next.
- Set current node's next to newNode.

→ with tail pointer.

AddAtRear(element) - with tail pointer.
- Make memory for new element, say newNode.
- Store element in newNode's data.
- Set newNode's next to empty.
- if list is empty then
  - Set head and tail to newNode.
  - Stop.
- Set tail node's next to newNode.
- Set tail to newNode.



## ③ Insert element in a sorted list.



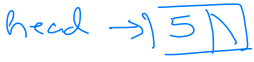

### ① Empty list

head → empty

### ② Insert (5)

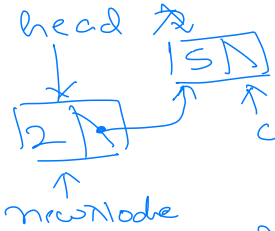
head → empty


Result after op




### ③ Insert (2)

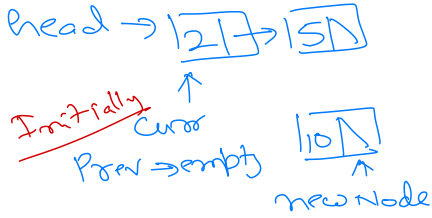
Result in adding new element before first node.

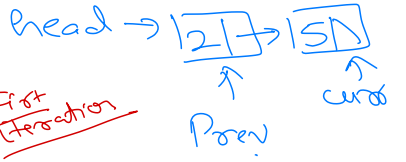


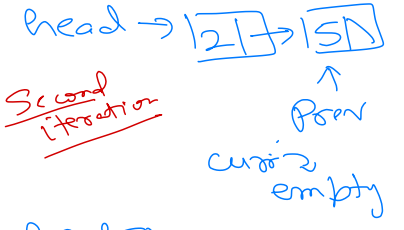

Result after op

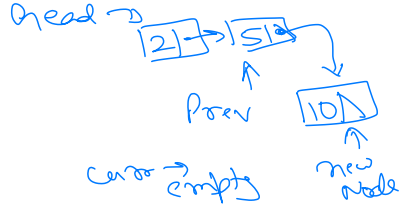

### ④ Insert (10)

Result in adding new element after last node.



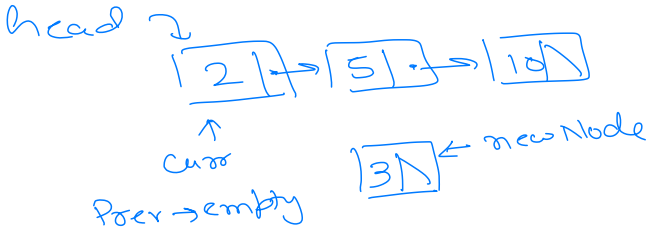



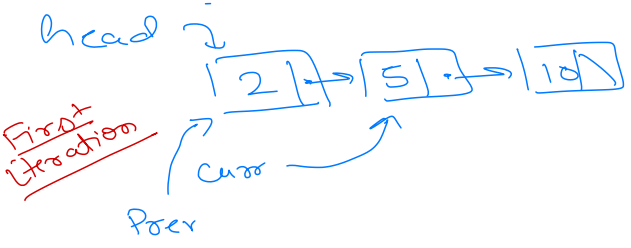



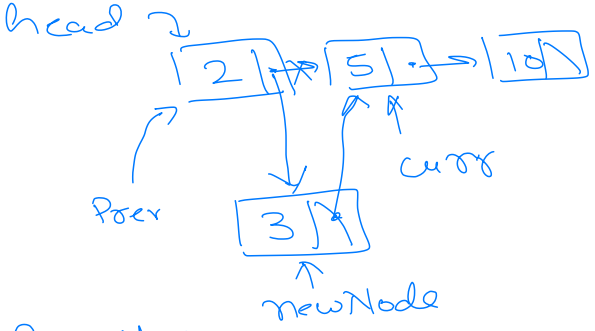




Result of op

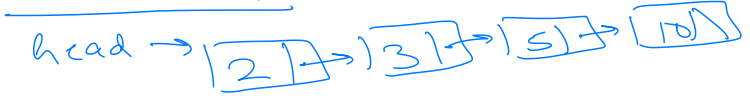
(5) Insert ( 3)




Result of SP



Insert(element) - with no tail pointer.
- Make memory for new element, say newNode.
- Store element in newNode's data.
- Set newNode's next to empty.
- if list is empty then
  - Set head to newNode.
  - Stop.

// List is not empty. Traverse the list to fin previous and current nodes, because newNode will be inserted between previous and current nodes..

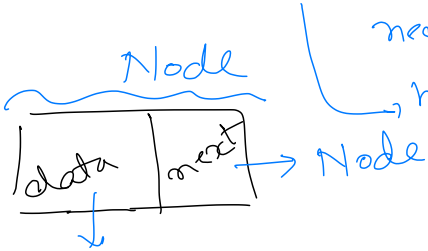
- Set current to head.
- Set previous to empty.
- while ( (current is not empty) and (current node's data < element) )
  - Set previous to current.
  - Set current to current's next.

// Adding before first node? (Adding smallest element).

- if current is head node then
  - Set newNode's next to head.
  - Set head to newNode.
  - Stop.
- Set previous node's next to newNode.
- Set newNode's next to current.

AddAtFront(element) - List having only head pointer.

- Make space for new element, say newNode. → Node newNode = new Node()
- Store element in newNode's data. → newNode.data = element
- Set newNode's next to head. → newNode.next = head
- Set head to newNode. → head = newNode




```
public class Node{
    public int data;
    public Node next;
}
```

head → null