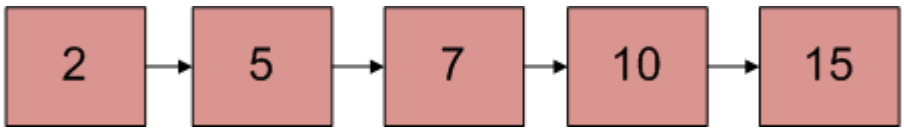
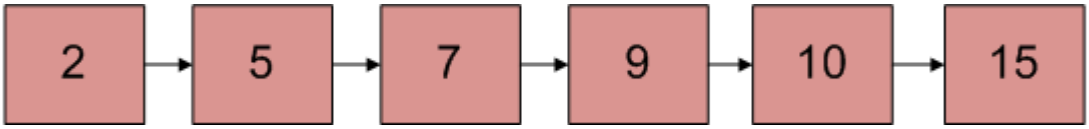
Sorted Insert in a Singly Linked List

Given a sorted linked list and a value to insert, write a function to insert the value in a sorted way.
Initial Linked List



Linked List after insertion of 9



Algorithm:
Let input linked list is sorted in increasing order.

```
1) If Linked list is empty then make the node as
   head and return it.
2) If the value of the node to be inserted is smaller
   than the value of the head node, then insert the node
at the start and make it head.
3) In a loop, find the appropriate node after
   which the input node (let 9) is to be inserted.
   To find the appropriate node start from the head,
   keep moving until you reach a node GN (10 in
   the below diagram) who's value is greater than
   the input node. The node just before GN is the
appropriate node (7).
4) Insert the node (9) after the appropriate node
   (7) found in step 3.
```

Implementation:

C++ Java

```
    int data;
    Node* next;
};

/* function to insert a new_node
in a list. Note that this
function expects a pointer to
head_ref as this can modify the
head of the input linked list
(similar to push())*/
void sortedInsert(Node** head_ref,
                  Node* new_node)
{
    Node* current;
    /* Special case for the head end */
    if (*head_ref == NULL
        || (*head_ref)->data
            >= new_node->data) {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
    else {
        /* Locate the node before the
point of insertion */
        current = *head_ref;
        while (current->next != NULL
&& current->next->data
< new_node->data) {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}

/* BELOW FUNCTIONS ARE JUST
UTILITY TO TEST sortedInsert */
```

```
create a new node */
    Node* new_node = new Node();

    /* put in the data */
    new_node->data = new_data;
    new_node->next = NULL;

    return new_node;
}

/* Function to print linked list */
void printList(Node* head)
{
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

/* Driver program to test count function*/
int main()
{
    /* Start with the empty list */
    Node* head = NULL;
    Node* new_node = newNode(5);
    sortedInsert(&head, new_node);
    new_node = newNode(10);
    sortedInsert(&head, new_node);
    new_node = newNode(7);
    sortedInsert(&head, new_node);
    new_node = newNode(3);
    sortedInsert(&head, new_node);
    new_node = newNode(9);
    sortedInsert(&head, new_node);
    cout << "Created Linked List\n";
```

```
    return 0;
```

Output:

```
Created Linked List
1 3 5 7 9 10
```

Complexity Analysis:

- **Time Complexity:** O(n).
  Only one traversal of the list is needed.
- **Auxiliary Space:** O(1).
  No extra space is needed.