Insertion Sort for Singly Linked List

We have discussed Insertion Sort for arrays. In this article same for linked list is discussed.

Below is simple insertion sort algorithm for linked list.

```
1) Create an empty sorted (or result) list
2) Traverse the given list, do following for every node.
......a) Insert current node in sorted way in sorted or result list.
3) Change head of given linked list to head of sorted (or result) list.
```

The main step is (2.a) which has been covered in below post.

Sorted Insert for Singly Linked List

Below is C implementation of above algorithm

```
/* C program for insertion sort on a linked list */
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

// Function to insert a given node in a sorted linked list
void sortedInsert(struct node**, struct node*);

// function to sort a singly linked list using insertion sort
void insertionSort(struct node **head_ref)
{
    // Initialize sorted linked list
    struct node *sorted = NULL;

    // Traverse the given linked list and insert every
    // node to sorted
    struct node *current = *head_ref;
    while (current != NULL)
    {
        // Store next for next iteration
        struct node *next = current->next;

        // insert current in sorted linked list
        sortedInsert(&sorted, current);

        // Update current
        current = next;
    }

    // Update head_ref to point to sorted linked list
    *head_ref = sorted;
}


/* 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(struct node** head_ref, struct node* new_node)
{
    struct 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 */

/* Function to print linked list */
void printList(struct node *head)
{
    struct node *temp = head;
    while(temp != NULL)
    {
        printf("%d  ", temp->data);
        temp = temp->next;
    }
}

/* A utility function to insert a node at the beginning of linked list */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node = new node;

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

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}


// Driver program to test above functions
int main()
{
    struct node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    printf("Linked List before sorting \n");
    printList(a);

    insertionSort(&a);

    printf("\nLinked List after sorting \n");
    printList(a);

    return 0;
}
```

```
Linked List before sorting
30  3  4  20  5
Linked List after sorting
3  4  5  20  30
```