## Q1. Implement the Radix Sort algorithm using a linked list. Start with the least significant digit (LSD) version.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define null NULL

struct node
{
    int num;
    struct node *next;
};

int max_num(struct node *head)
{
    int max = head->num;

    struct node *curr = head->next;

    while (curr != null)
    {
        if (curr->num > max)
        {
            max = curr->num;
        }
        curr = curr->next;
    }
    return max;
}

int get_digit(int num, int exp)
{
    return (num / exp) % 10;
}

struct node *counting_sort(struct node *head, int exp)
{
    struct node *bucket[10] = {null};
    struct node *tails[10] = {null};

    struct node *curr = head;
    while (curr != null)
    {
        int digit = get_digit(curr->num, exp);
        if (bucket[digit] == null)
```

```
        {
            bucket[digit] = curr;
            tails[digit] = curr;
        }
        else
        {
            tails[digit]->next = curr;
            tails[digit] = curr;
        }
        curr = curr->next;
    }
    struct node *new_head = null;
    struct node *new_tail = null;

    for (int i = 0; i < 10; i++)
    {
        if (bucket[i] != null)
        {
            if (new_head == null)
            {
                new_head = bucket[i];
                new_tail = tails[i];
            }
            else
            {
                new_tail->next = bucket[i];
                new_tail = tails[i];
            }
        }
    }
    new_tail->next = null;
    return new_head;
}

struct node *radix_sort(struct node *head)
{
    int max = max_num(head);
    int exp = 1;

    while (max / exp > 0)
    {
        head = counting_sort(head, exp);
        exp *= 10;
    }
    return head;
}
```

```c
void printlist(struct node *head)
{
    while (head != null)
    {
        printf("%d ", head->num);
        head = head->next;
    }
    printf("\n");
}

void insert(struct node **head, int num)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->num = num;
    new_node->next = *head;
    *head = new_node;
}

int main()
{
    struct node *head = null;
    insert(&head, 170);
    insert(&head, 54);
    insert(&head, 91);
    insert(&head, 90);
    insert(&head, 112);
    insert(&head, 34);
    insert(&head, 2);
    insert(&head, 79);
    insert(&head, 0);
    insert(&head, 100);
    printf("Unsorted list: ");
    printlist(head);

    head = radix_sort(head);
    printf("Sorted list: ");
    printlist(head);
    return 0;
}
```

OUTPUT:

```
Unsorted list: 100 0 79 2 34 112 90 91 54 170
Sorted list: 0 2 34 54 79 90 91 100 112 170
```

## Q2. Extend your linked list Radix Sort implementation to handle negative integers as well.

CODE:

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int num;
    struct node *link;
};

int getmax(struct node *head)
{
    int max = abs(head->num);
    struct node *current = head->link;
    while (current != NULL)
    {
        if (abs(current->num) > max)
        {
            max = abs(current->num);
        }
        current = current->link;
    }
    return max;
}

int getdigit(int number, int place)
{
    return (abs(number) / place) % 10;
}

void insertEnd(struct node **head, int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->num = data;
    new_node->link = NULL;

    if (*head == NULL)
    {
        *head = new_node;
    }
    else
    {
```

```
        struct node *current = *head;
        while (current->link != NULL)
        {
            current = current->link;
        }
        current->link = new_node;
    }
}

void insert(struct node **head, struct node *new_node)
{
    if (*head == NULL)
    {
        *head = new_node;
    }
    else
    {
        struct node *current = *head;
        while (current->link != NULL)
        {
            current = current->link;
        }
        current->link = new_node;
    }
}

void radixSortPositive(struct node **head)
{
    struct node *buckets[10];
    struct node *current;
    int max = getmax(*head);
    int place = 1;

    while (max / place > 0)
    {
        for (int i = 0; i < 10; i++)
        {
            buckets[i] = NULL;
        }

        current = *head;
        while (current != NULL)
        {
            int digit = getdigit(current->num, place);
            struct node *link = current->link;
            current->link = NULL;
```

```
            insert(&buckets[digit], current);
            current = link;
        }

        *head = NULL;
        for (int i = 0; i < 10; i++)
        {
            if (buckets[i] != NULL)
            {
                if (*head == NULL)
                {
                    *head = buckets[i];
                }
                else
                {
                    insert(head, buckets[i]);
                }
            }
        }

        place *= 10;
    }
}

void radixSortNegative(struct node **head)
{
    struct node *buckets[10];
    struct node *current;
    int max = getmax(*head);
    int place = 1;

    while (max / place > 0)
    {
        for (int i = 0; i < 10; i++)
        {
            buckets[i] = NULL;
        }

        current = *head;
        while (current != NULL)
        {
            int digit = getdigit(current->num, place);
            struct node *link = current->link;
            current->link = NULL;
            insert(&buckets[digit], current);
            current = link;
```

```
        }

        *head = NULL;
        for (int i = 9; i >= 0; i--)
        {
            if (buckets[i] != NULL)
            {
                if (*head == NULL)
                {
                    *head = buckets[i];
                }
                else
                {
                    insert(head, buckets[i]);
                }
            }
        }

        place *= 10;
    }
}

void mergeLists(struct node **head, struct node *negativeList, struct
node *positiveList)
{
    if (negativeList != NULL)
    {
        *head = negativeList;
        while (negativeList->link != NULL)
        {
            negativeList = negativeList->link;
        }
        negativeList->link = positiveList;
    }
    else
    {
        *head = positiveList;
    }
}

void radixSort(struct node **head)
{
    struct node *negativeList = NULL;
    struct node *positiveList = NULL;
    struct node *current = *head;
```

```
    while (current != NULL)
    {
        struct node *next = current->link;
        current->link = NULL;
        if (current->num < 0)
        {
            insert(&negativeList, current);
        }
        else
        {
            insert(&positiveList, current);
        }
        current = next;
    }

    if (negativeList != NULL)
    {
        radixSortNegative(&negativeList);
    }
    if (positiveList != NULL)
    {
        radixSortPositive(&positiveList);
    }

    mergeLists(head, negativeList, positiveList);
}

struct node *newNode(int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->num = data;
    new_node->link = NULL;
    return new_node;
}

void printList(struct node *head)
{
    while (head != NULL)
    {
        printf("%d ", head->num);
        head = head->link;
    }
    printf("\n");
}

int main()
```

```
{
    struct node *head = NULL;

    insertEnd(&head, -10);
    insertEnd(&head, 145);
    insertEnd(&head, -75);
    insertEnd(&head, 90);
    insertEnd(&head, -101);
    insertEnd(&head, 34);
    insertEnd(&head, 12);
    insertEnd(&head, -22);

    printf("Unsorted List: ");
    printList(head);

    radixSort(&head);

    printf("Sorted List: ");
    printList(head);
    return 0;
}
```

OUTPUT:

```
Unsorted List: -10 145 -75 90 -101 34 12 -22
Sorted List: -101 -75 -22 -10 12 34 90 145
```

## Q3. Implement a function to display the linked list representation of the sparse matrix. Print the row, column, and value of each non-zero element.

CODE:

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int row;
    int col;
    int value;
    struct Node *next;
};

void displaySparseMatrix(struct Node *head)
{
    struct Node *current = head;

    printf("Row\tColumn\tValue\n");
    while (current != NULL)
```

```
    {
        printf("%d\t%d\t%d\n", current->row, current->col, current->value);
        current = current->next;
    }
}

int main()
{
    struct Node *matrixHead = (struct Node *)malloc(sizeof(struct Node));
    matrixHead->row = 0;
    matrixHead->col = 2;
    matrixHead->value = 10;

    struct Node *matrixNode1 = (struct Node *)malloc(sizeof(struct Node));
    matrixNode1->row = 1;
    matrixNode1->col = 1;
    matrixNode1->value = 20;

    struct Node *matrixNode2 = (struct Node *)malloc(sizeof(struct Node));
    matrixNode2->row = 2;
    matrixNode2->col = 0;
    matrixNode2->value = 30;

    matrixHead->next = matrixNode1;
    matrixNode1->next = matrixNode2;
    matrixNode2->next = NULL;

    displaySparseMatrix(matrixHead);

    free(matrixHead);
    free(matrixNode1);
    free(matrixNode2);

    return 0;
}
```

OUTPUT: