```
//Created By Ritwik Chandra Pandey on 30th March 2021
//183215
//Radix Sort Using Linked List


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

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

struct node *start = NULL;

void radix_sort();
int larger_digit();
int digit_finder(int number, int k);

int main()
{
    struct node *temp, *x;
    int count, limit, element;
    printf("\nEnter Total Number of Elements:\t");
    scanf("%d", &limit);
    for(count = 0; count < limit; count++)
    {
        printf("Element No. %d:\t", count + 1);
        scanf("%d", &element);
        temp = malloc(sizeof(struct node));
        temp->data = element;
        temp->link = NULL;
        if(start == NULL)
        {
            start = temp;
        }
        else
        {
            x = start;
            while(x->link != NULL)
            {
                x = x->link;
            }
            x->link = temp;
        }
    }
    radix_sort();
    printf("\nSorted List\n");
    x = start;
    while(x != NULL)
    {
        printf("%3d", x->data);
        x = x->link;
    }
    printf("\n");
    return 0;
}

void radix_sort()
{
    int count, k, digit, least_significant, most_significant;
    struct node *rear[10], *front[10], *p;
    least_significant = 1;
    most_significant = larger_digit(start);
    for(k = least_significant; k <= most_significant; k++)
    {
        for(count = 0; count <= 9; count++)
        {
            rear[count] = NULL;
            front[count] = NULL ;
        }
        for(p = start; p != NULL; p = p->link)
        {
            digit = digit_finder(p->data, k);
            if(front[digit] == NULL)
            {
                front[digit] = p;
            }
            else
            {
                rear[digit]->link = p;
            }
            rear[digit] = p;
        }
        count = 0;
        while(front[count] == NULL)
        {
            count++;
        }
        start = front[count];
        while(count < 9)
        {
            if(rear[count + 1] != NULL)
            {
                rear[count]->link = front[count + 1];
            }
            else
            {
                rear[count + 1] = rear[count];
            }
            count++;
        }
        rear[9]->link = NULL;
    }
}

int larger_digit()
{
    struct node *p = start;
    int temp = 0, digit = 0;
    while(p != NULL)
    {
        if(p ->data > temp)
        {
            temp = p->data;
        }
        p = p->link ;
    }
    while(temp != 0)
    {
        digit++;
        temp = temp / 10;
    }
    return(digit);
}

int digit_finder(int number, int k)
{
    int term, count;
    for(count = 1; count <= k; count++)
    {
        term = number % 10;
        number = number / 10;
    }
    return(term);
}
```

Radix Sort Algorithm Analysis

The **run time** complexity of the radix sorting algorithm is **O(p * n)** where p is the number of iterations of the outer loop and n is the number of iterations of the inner loop. The **worst case** scenario complexity of this algorithm is **O(n)** whereas the **best case** scenario complexity is **O(n log n)**.

Radix Sort is a **stable sort** and is also an **in-place sort**. However, this algorithm takes extra space for maintaining **queue overheads**. This algorithm is preferable when the number of digits are small.