```c
// Name : Ayush Gupta
// Roll no: MNS2025027

// 1. Write a C program to sort the array of integers
using Insertion
// or selection sort algorithm.
#include <stdio.h>

int minimum(int a, int b)
{
    return (a > b) ? b : a;
}

int main()
{
    int n, choice;

    printf("\n--------------------");
    printf("\n1.Insertion Sort?");
    printf("\n2.Seleciton Sort?");
    scanf("%d", &choice);

    printf("\nNumber of elements?");
    scanf("%d", &n);
    int arr[n];

    printf("\nEnter the elements for the array:");

    for (int i = 0; i < n; i++)
    {
        int k;
        scanf("%d", &k);
```

```
        arr[i] = k;
    }

    switch (choice)
    {
        case 1:
            for (int i = 0; i < n; i++)
            {
                int k = arr[i];
                int j = i - 1;

                while (j >= 0 && arr[j] > k)
                {
                    arr[j + 1] = arr[j];
                    j--;
                }
                arr[j + 1] = k;
            }
            break;
        case 2:
            for (int i = 0; i < n - 1; i++)
            {
                int min = arr[i], index = i;
                for (int j = i + 1; j < n; j++)
                {
                    if (arr[j] < min)
                    {
                        min = arr[j];
                        index = j;
                    }
                }
                int temp = arr[i];
```

```
                arr[i] = arr[index];
                arr[index] = temp;
            }

            break;
    }
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }

    return 0;
}

/*
2. Implement bubble sort algorithm using singly linked
list.
*/

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

struct Node
{
    int data;
    struct Node* next;
};

// Function to insert a new node at the end
void insert(struct Node** head, int value)
{
    struct Node* newNode = malloc(sizeof(struct Node));
```

```
    newNode->data = value;
    newNode->next = NULL;

    if (*head == NULL)
    {
        *head = newNode;
    }
    else
    {
        struct Node* temp = *head;
        while (temp->next != NULL) temp = temp->next;
        temp->next = newNode;
    }
}

// Bubble sort on linked list
void bubbleSort(struct Node* head)
{
    if (!head) return;

    int swapped;
    struct Node* ptr;
    struct Node* lptr = NULL;

    do
    {
        swapped = 0;
        ptr = head;

        while (ptr->next != lptr)
        {
            if (ptr->data > ptr->next->data)
```

```
            {
                int temp = ptr->data;
                ptr->data = ptr->next->data;
                ptr->next->data = temp;
                swapped = 1;
            }
            ptr = ptr->next;
        }
        lptr = ptr;
    } while (swapped);
}

// Print the linked list
void printList(struct Node* head)
{
    while (head)
    {
        printf("%d ", head->data);
        head = head->next;
    }
}

int main()
{
    int n, value;
    struct Node* head = NULL;

    printf("Number of elements? ");
    scanf("%d", &n);

    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++)
```

```c
    {
        scanf("%d", &value);
        insert(&head, value);
    }

    bubbleSort(head);

    printf("Sorted list: ");
    printList(head);

    return 0;
}

// 3.Write a C program to sort the array of integers
using quicksort or mergesort algorithm.
#include <stdio.h>

void quickSort(int arr[], int low, int high);
int partition(int arr[], int low, int high);
void mergeSort(int arr[], int l, int r);
void merge(int arr[], int l, int m, int r);

int main()
{
    int n, choice;

    printf("\n--------------------");
    printf("\n1. Quicksort?");
    printf("\n2. Mergesort?");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);
```

```c
    printf("\nNumber of elements? ");
    scanf("%d", &n);
    int arr[n];

    printf("\nEnter the elements for the array:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    switch (choice)
    {
        case 1:
            quickSort(arr, 0, n - 1);
            break;
        case 2:
            mergeSort(arr, 0, n - 1);
            break;
        default:
            printf("Invalid choice.\n");
            return 1;
    }

    printf("\nSorted array: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

```
// -------------------- Quicksort --------------------
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}
```

```
// -------------------- Mergesort --------------------
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

    while (i < n1) arr[k++] = L[i++];
```

```
    while (j < n2) arr[k++] = R[j++];
}

/*
4. Implement radix sort algorithm using singly linked
list.
*/

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

#define BASE 10

struct Node
{
    int data;
    struct Node* next;
};

// Insert node at end
void insert(struct Node** head, int value)
{
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

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

```
        struct Node* temp = *head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }
}

// Get max value in list
int getMax(struct Node* head)
{
    int max = head->data;
    while (head)
    {
        if (head->data > max) max = head->data;
        head = head->next;
    }
    return max;
}

// Radix sort
void radixSort(struct Node** head)
{
    struct Node *buckets[BASE] = {NULL}, *tails[BASE] =
{NULL};
    int max = getMax(*head), exp = 1;

    while (max / exp > 0)
    {
        struct Node* curr = *head;
        while (curr)
        {
            int index = (curr->data / exp) % BASE;
            if (!buckets[index])
```

```
            {
                buckets[index] = tails[index] = curr;
            }
            else
            {
                tails[index]->next = curr;
                tails[index] = curr;
            }
            curr = curr->next;
        }

        *head = NULL;
        struct Node* last = NULL;
        for (int i = 0; i < BASE; i++)
        {
            if (buckets[i])
            {
                if (!*head)
                {
                    *head = buckets[i];
                    last = tails[i];
                }
                else
                {
                    last->next = buckets[i];
                    last = tails[i];
                }
                buckets[i] = tails[i] = NULL;
            }
        }
        last->next = NULL;
        exp *= BASE;
```

```
    }
}

// Print list
void printList(struct Node* head)
{
    while (head)
    {
        printf("%d ", head->data);
        head = head->next;
    }
}

int main()
{
    int n, value;
    struct Node* head = NULL;

    printf("Number of elements? ");
    scanf("%d", &n);

    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &value);
        insert(&head, value);
    }

    radixSort(&head);

    printf("Sorted list: ");
    printList(head);
```

```
    return 0;
}
```