WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

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

struct Node{
    int data;
    struct Node *next;
};
void insertAtBeginning(struct Node **head, int value){
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}
void insertAtEnd(struct Node **head, int value){
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    struct Node *temp = *head;
    newNode->data = value;
    newNode->next = NULL;

    if (*head == NULL){
        *head = newNode;
        return;
    }

    while (temp->next != NULL){
        temp = temp->next;
    }

    temp->next = newNode;
```

```
}

void insertAtPosition(struct Node **head, int value, int position)
{
    if (position <= 0)
    {
        printf("Invalid position\n");
        return;
    }

    if (position == 1 || *head == NULL)
    {
        insertAtBeginning(head, value);
        return;
    }

    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    struct Node *temp = *head;
    int count = 1;

    while (count < position - 1 && temp->next != NULL)
    {
        temp = temp->next;
        count++;
    }

    if (count < position - 1)
    {
        printf("Invalid position\n");
```

```
        return;
    }

    newNode->next = temp->next;
    temp->next = newNode;
}

void deleteAtBegining(struct Node **head)
{
    if (*head == NULL)
    {
        printf("The linkedlist is already empty\n");
        return;
    }
    else
    {
        struct Node *first = *head;
        *head = (*head)->next;
        free(first);
    }
}
void deleteAtEnd(struct Node **head)
{
    if (*head == NULL)
    {
        printf("The linkedlist is already empty\n");
        return;
    }
    else
    {
```

```
        struct Node *temp = *head;
        while (temp->next->next != NULL)
        {
            temp = temp->next;
        }
        struct Node *lastNode = temp->next;
        temp->next = NULL;
        free(lastNode);
    }
}
void deleteAtIndex(struct Node **head, int pos)
{
    if (*head == NULL)
    {
        printf("The Linked List is Empty \n");
    }
    else
    {
        struct Node *temp = *head;
        pos--;
        while (pos-- && temp != NULL)
        {
            temp = temp->next;
        }
        if (temp == NULL)
        {
            printf("pos not exist\n");
        }
        else
        {
```

```
            struct Node *nxt = temp->next->next;
            struct Node *del = temp->next;
            temp->next = temp->next->next;
            free(del);
        }
    }
}
void displayLinkedList(struct Node *head1)
{
    struct Node *temp = head1;

    if (temp == NULL)
    {
        printf("Linked list is empty.\n");
        return;
    }

    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }

    printf("NULL\n");
}

void reverseLL(struct Node **head)
{
    struct Node *prev = NULL;
    struct Node *current = *head;
```

```
    struct Node *front = NULL;

    while (current != NULL)
    {
        front = current->next;
        current->next = prev;
        prev = current;
        ;
        current = front;
    }
    *head = prev;
}
void concat(struct Node **head1, struct Node **head2)
{
    struct Node *t = *head1;
    while (t->next != NULL)
    {
        t = t->next;
    }
    t->next = *head2;
}
void sortlist(struct Node **head1)
{
    struct Node *temp, *i;
    for (temp = *head1; temp != NULL; temp = temp->next)
    {
        for (i = temp->next; i != NULL; i = i->next)
        {
            if (i->data < temp->data)
            {
```

```
                int tem = i->data;
                i->data = temp->data;
                temp->data = tem;
            }
        }
    }
}

int main()
{
    struct Node **heads = (struct Node **)malloc(sizeof(struct Node *)
* 2);
    heads[0] = NULL;
    heads[1] = NULL;
    int twice = 0;
    while (twice < 2)
    {

        printf("Enter total number of elements of linkedlist %d : ",
(twice + 1));
        int n;
        scanf("%d", &n);
        printf("Enter the   elements: \n");
        for (int i = 0; i < n; i++)
        {
            int a;
            scanf("%d", &a);
            insertAtEnd(&(heads[twice]), a);
        }
        twice++;
    }
```

```
    while (1)
    {
        int ch;
        printf("\n 1:reverll\t 2: sortlist \t 3. concat \t 4: display
\t 5: exit \n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            reverseLL(&heads[0]);
            break;
        case 2:
            sortlist(&heads[0]);
            break;
        case 3:
            concat(&heads[0], &heads[1]);
        case 4:
            displayLinkedList(heads[0]);
            break;
        case 5:
            exit(0);
        }
    }
}
```

OUTPUT :

```
Enter total number of elements of linkedlist 1 : 3
Enter the   elements:
2 6 8
Enter total number of elements of linkedlist 2 : 4
Enter the   elements:
10 20 30 40

 1:reverll       2: sortlist     3. concat       4: display      5: exit
1

 1:reversell       2: sortlist     3. concat       4: display      5:
exit
4
8 -> 6 -> 2 -> NULL

 1:reversell       2: sortlist     3. concat       4: display      5:
exit
2

 1:reversell       2: sortlist     3. concat       4: display      5:
exit
4
2 -> 6 -> 8 -> NULL

 1:reversell       2: sortlist     3. concat       4: display      5:
exit
3
2 -> 6 -> 8 -> 10 -> 20 -> 30 -> 40 -> NULL
```

1:reversell 2: sortlist 3. concat 4: display 5: exit

4

2 -> 6 -> 8 -> 10 -> 20 -> 30 -> 40 -> NULL

1:reversell 2: sortlist 3. concat 4: display 5: exit

5