6 a) WAP to implement single linked list with following operations
Sort the linked list
Reverse the linked list,
concatenation of two linked list

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

struct node{
    int data;
    struct node* next;
    };
struct node* createNode(int data)
{
  struct node* newNode = (struct node*) malloc
(sizeof(struct node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
  }

void insertAtEnd(struct node** head, int data)
  {
    struct node* newNode = createNode(data);
    if(*head == NULL)
    {
      *head = newNode;
      return;
    }
```

```
struct node* temp = *head;
while ( temp->next != NULL)
  {
    temp = temp->next;
  }
  temp -> next = newnode;
}


void SortList ( struct Node** head)
{
   Struct node* i;
   Struct node* j;
   for(i=*head ; i!=NULL ; i = i->next)
   {
      for (j=i->next ; j!=NULL ; j=j->ne
      {
        if (i->data > j->data) {
          int temp = i->data;
          i->data = i->data;
          i-> data = temp;
      }
   }
}

void reverselist (Struct Node** head)
{
    Struct Node* prev = NULL
    Struct node* current = *head;
    Struct Node* next = NULL;
    while (current != NULL)
    {
       next = current->next;
       current->next = prev;
       prev = current;
       current = next;
    }
```

```
*head = prev;
}
void concatenatelist(struct Node** head1,
struct Node** head2)
{
    if (*head1 = NULL) {
        *head1 = *head2;
        return;
    }
    struct Node *temp = *head1;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = *head2;
}

void displaylist(struct Node* head)
{
    struct Node* temp = head;
    while (temp != NULL)
    {
        printf("%d", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
printf("\n 1. Insert into list1. \n 2. Insert
into List2 \n Sort list1 \n4. Reverse list1 \
```

```
5. Concatenate Liststring. Display List \n3. Exit
\n Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
 case 1:
  printf("Enter data to insert into list 1:
  scanf("%d", &data);
  insertAtEnd(&list1, data);
  break;
 case 2:
  printf("Enter data to insert into List2
  scanf("%d", &data);
  insertAtEnd(&list2, data);
  break;
 case 3:
      sortList(&list1);
      printf("List 1 sorted.\n");
      break;
 case 4:
      reverseList(&list1);
      printf("List 1 reversed.\n");
      break;
 case 5:
      concatenateLists(&list1, &list2);
      printf("Lists Concatenated.\n");
      break;
 case 6:
      printf("List 1:");
      displayList(list1);
      break;
```

```
case 7:
    printf ("List 2: ");
    displayList( list2);
    break;
case 8:
    exit(0)
    default:
        printf ("Invalid choice. Try again.\n")
    }
}
return 0;
}
```

OP

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse list 1
5. Concatenate List 2
6. Display List 1
7. Display List 2
8. Exit

Enter your choice : 1
Enter data to insert into List 1 : 32
Enter your choice : 1
Enter data to insert into List 1 : 23
Enter your choice : 2
Enter data to insert into List 2 : 11
Enter your choice : 2
Enter data to insert into List 2 : 34
Enter your choice :
List 1 sorted

4. Reverse List

```
Enter your choice : 6
List1 : 23 322
Enter your choice : 4
List 1 reversed.
List 1 : 322 23
Enter your choice : 5
List2 concatenated.
Enter your choice : 6
List1 : 322 23 11 34.
```

6b. WAP to implement single link list to simulate stack & Queue operations.

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

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

struct Node* createNode(int data)
{
    struct Node* createNode=(struct Node*)
    malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
};

void Push(struct Node** top, int data){
    struct Node * newNode = (createNode(da
    newNode->next = *top;
    *top = newNode;
```

```
if Pop( struct Node** top, int data) {
  struct Node* newNode = CreateNode
```

```
if Pop(struct Node** top)
{
  if (*top==NULL)
    {
      Printf ("stack underflow \n");
      return -1;
    }
  struct Node* temp = *top;
  int data = temp->data;
  *top = (*top)->next;
  free (temp);
  return data;
}
```

```
void enQueue(struct Node** front, struct Node
-e**  rear, int data)
  {
    struct Node* newNode = CreateNode(data);
    if (*rear == NULL){
    *front = *rear = newNode;
     return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
  }
```

```
int dequeue (struct Node** front, struct Node**
rear){
  if (*front == NULL)
    {
      printf ("Queue underflow \n");
      return -1;
```

ta);

```
}
struct Node* temp = *front;
int data = temp->data;
*front = (*front)->next;
if (*front == NULL)
{
  *rear = NULL;
}
free(temp)
return data;
}

void displayList(struct Node* head)
  {
   struct Node* temp = head;
   while (temp != NULL)
   {
    printf("%d", temp->data);
    temp = temp->next;
   }
   printf("\n");
}
int main()
 {
  struct node* Stack = NULL;
  struct Node* QueueFront = NULL;
  struct Node* QueueRear = NULL;
  int choice, data;

 while(1)
 {
  printf("\n1. Push to Stack \n2. Pop
```

from stack in 3. Display stack no. enQue
to Queue in 5. DeQueue from Queue in 7. Exi
Enter your choice:");
scanf("%d", &choice);

16/12

switch (choice) {
case 1:
printf("Enter data to PUSH to stack
scanf("%d", &data1);
PUSH (&stack, data1);
break;
case 2:
printf("Popped from stack: %d \n", POP
&stack));
break;
case 3:
printf("stack 3:");
displayList (stack);
case 4:
printf("Enter data to enQueue to
Queue:");
scanf("%d", &data);
enQueue (&QueueFront, &QueueRear, data);
case 5:
printf("DeQued from Queue: %d \n", de
-Queue (&QueueFront, &QueueRear);
break;
case 6:
printf("Queue:");
displayList (QueueFront);
break;
case 7:
exit(0);
default:

```
    }
  }
  return 0;
}
```

Output

```
1. Push to stack
2. Pop from stack
3. Display stack
4. Enqueue to Queue
5. dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 1
Enter data to push to stack: 11
Enter your choice: 1
Enter data to push to stack: 22
Enter your choice: 3
Stack: 22 11
Enter your choice: 2
Popped from stack: 22
Enter your choice: 3
Stack: 11
Enter your choice: 4
Enter data to enqueue to Queue: 56
Enter your choice: 4
Enter data to enqueue to Queue: 45
Enter your choice: 5
Dequeued from Queue: 56
Enter your choice: 6
Queue: 45
```