SUSHMITHA Y.V
1BM19CS165
D1

1. Sorting linked list

```
void sort (node * sort) {
    int flag, i;
     node *ptr 1;
      node * ptr 2;
       ptr 2 = NULL;
       if (start = =NULL)
            return;
         do
   {
    flag =0;
    ptr 1 = start;
    While (ptr 1 -> next! =ptr 2)
   {
    if (ptr -> value -> ptr2 -> next -> value)
    {
     swap(ptr 1, ptr 1 -> next);
          flag = 1;
     }
ptr1 = ptr1 → next;
}

 ptr2 = ptr1;
}
  while (flag);
}
  void swap (node *a, node *b){
     int temp = a → value;
       a → value = b → value;
        b → value = temp;
   }
```

2. Reversing linked list

```
   void reverse (){
       if (head == NULL){
        printf ("linked list is empty");
             return;
       }
```

```
if (head -> next == NULL){
      print ("Reversed");
         return;
 }

node * temp;
 node * current = head -> next;
  node * previous = head;
  while (current! =NULL){
        temp = current -> next;
        current -> next = previous;
         previous = current
          current = temp;
     }
     head -> next = NULL;
      head = previous;
       printf (" Reversed");
          return;
    }
```

3. Merging in ascending order

```
  // Recursive implementation
 // Called initially as merge(head 1, head 2, head 3)
 // head 1 & head 2 are head pointer to two linked
                                              list
 // head 3 is head ptr of merged list
 // alternatively merge can be called as
 // merge ( head 1, head 2, NULL);

   void merge (node * curr 1, node * curr 2,
                node * prev){
  int flag 1 =(curr 1 == NULL);
  int flag 3 = (curr 2 = = NULL);
     if (flag 1 && flag3)
           return;
 node * newNode = (Node *) malloc (size of (node));
   newNode -> next = NULL;
      if (prev == NULL) {
   Sort (head1); // algorithm in part 1
   Sort (head2); // algorithm in part 1
      head 3 = newNode;
  }
```

```
int flag 2 = 1, flag 4 = 1;
 if (! flag 1 && ! flag 3)  // both curr 1 & curr 2 not
                                                   full
 flag 2 = curr1 -> value -> = curr 2 -> value;
 }
   if (flag 1)
       flag 4 = 0;
     if (flag 3)
       flag 2 = 0;
     if (flag 1 || flag 2) {
       New Node -> value = curr 2 -> value
          curr 2 = curr 2 -> next;
     }
     else if (flag 3 || flag 4) {
        New Node -> value = curr 1 -> value;
           curr 1 = curr 1 -> next;
      }
     if (prev != NULL)
             prev -> next = new Node;
             prev = new Node;
      }      merge (curr 1, curr 2, prev);
```