LAB-7

Vaishwarth Kiran S
1BM19CS187
7/12/2020

1. Sorting linked list

```
Void sort (node * start) {

    int flag, i;
     node * ptr1;
     node * ptr2;
     ptr2 = NULL;
    if (start == NULL)
        return;
     do
    {
      flag = 0;
      ptr1 = start;
     while (ptr1 -> next != ptr2)
     {
      if (ptr -> value -> ptr1 -> next -> value)
      {
        swap (ptr1, ptr1 -> 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) {
        printf ("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 (head1, head2, head3),
//head1 & head2 are head pointer to two linked list
//head3 is head ptr of merged list
//alternatively merge can be called as
//merge( head1, head2, NULL);
```

```
int void merge (node * curr1, node * curr2, node *
                prev) {

int flag1 =(curr1 == NULL);
int flag3 = (curr2 == NULL);
 if (flag1 && flag3)
        return;
node * newNode =(Node *) malloc(sizeof(node));
   newNode -> next =NULL;
      if (prev == NULL){
       sort (head1); // algorithm in part 1
       sort(head2); // algorithm in part(

       head3 = newNode;
   }

   int flag2 =1, flag4 =1;
 if ( !flag1 && !flag3)  // both curr1 & curr2
                            not null
  flag2= curr1 -> value >= curr2 -> value;
}
   if(flag1)
       flag4 =0;
    if (flag3)
       flag2 =0;
```

```
if (flag1 || flag2) {
    newNode -> value = curr2 -> value
    curr2 = curr2 -> next;
}

else if (flag3 || flag4) {
    newNode -> value = curr1 -> value;
    curr1 = curr1 -> next;
}
if (prev != NULL)
    prev -> next = newNode;
    prev = newNode;
    merge (curr1, curr2, prev);
}
```