Linked List - Assignment Solutions

1.

```
// Time Complexity : O(N)
// Space Complexity : O(1)
function(head) {
    var p1 = head, p2 = head;
    while (p2 != null && p2.next != null) {
        p1 = p1.next;       // moving p1 by 1 node
        p2 = p2.next.next;  // moving p2 by 2 nodes
        // condition to find the cycle...
        if (p1 == p2)  break;
    }
    // In case there is no cycle or no meeting point...
    if (p2 == null || p2.next == null) return null;
    while (head != p1) {
        head = head.next;
        p1 = p1.next;
    }
    return head;
}
```

2.

```
// Time Complexity : O(N)
// Space Complexity : O(1)
function lengthOfLinkedList(head)
{
    var temp = head;
    var count = 0;
    while (temp!= null)
    {
        count++;
        temp = temp.next;
    }
    return count;
}
```

3.

```
// Time Complexity : O(N)
// Space Complexity : O(1)
   function Middle(head){
   var slow_ptr = head;
   var fast_ptr = head;
        if (head != null)
        {
            while (fast_ptr != null && fast_ptr.next != null)
            {
                fast_ptr = fast_ptr.next.next;
                slow_ptr = slow_ptr.next;
```

```
            }
            console.log("The middle element is [",  slow_ptr.data, "]”);
    }
```

4.

```
// Time Complexity : O(N) + O(N) = O(N)
// Space Complexity : O(1)
function 3rdLastNode(head)
{
    var temp = head, temp2 = head;
    var count = 0;
    while (temp!= null)
    {
        count++;
        temp = temp.next;
    }
    count = count - 3;
    while (count!= 0)
    {
        count --;
        temp2 = temp2.next;
    }
    return temp2;
}
```

5.

```
// Time Complexity  : O(N+M)
// Space Complexity : O(N)
class ListNode {
    constructor(val) {
        this.val = val;
        this.next = null;
    }
}
function (l1, l2) {
    let head = null;
    let temp = null;
    let carry = 0;
    while (l1 !== null || l2 !== null) {
        // At the start of each iteration, we should add carry from the last
iteration
        let sum = carry;
        // current node is null for one of the lists
        if (l1 != null) {
            sum += l1.val;
            l1 = l1.next;
        }
        if (l2 != null) {
            sum += l2.val;
            l2 = l2.next;
```

```
        }
        // At this point, we will add the total sum % 10 to the new node
        // in the resultant list
        let node = new ListNode(Math.floor(sum) % 10);
        // Carry to be added in the next iteration
        carry = Math.floor(sum / 10);
        // If this is the first node or head
        if (temp == null) {
            temp = head = node;
        }
        // For any other node
        else {
            temp.next = node;
            temp = temp.next;
        }
    }
    // After the last iteration, we will check if there is carry left
    // If it's left then we will create a new node and add it
    if (carry > 0) {
        temp.next = new ListNode(carry);
    }
    return head;
}
```

6.

```
// Time Complexity : O(N+M) where N and M are the length of two strings s1
and s2 represented in the form of list1 and list2
// Space Complexity : O(1)
function compare(list1, list2)
{
    let i = 0, j = 0;
    while (list1 != null && list2 != null) {
        while (i < list1.s.length && j < list2.s.length) {
            if (list1.s[i] != list2.s[j])
                return false;
            i++;
            j++;
        }
        if (i == list1.s.length) {
            i = 0;
            list1 = list1.next;
        }
        if (j == list2.s.length) {
            j = 0;
            list2 = list2.next;
        }
    }
    return list1 == null && list2 == null;
}
```

7.

```
// Time Complexity : O(NlogN)
// Space Complexity : O(logN)
var head = null;
class Node {
    constructor(val) {
        this.data = val;
        this.next = null;
    }
}
    // Function to get the middle of the linked list
    function getMiddle(head) {
        if (head == null)
            return head;
        var slow = head, fast = head;
        while (fast.next != null && fast.next.next != null)
        {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
      }
function merge(list1,  list2) {
var merged = new Node(-1);
var temp = merged;
        // While list1 is not null and list2 is not null
        while (list1 != null && list2 != null) {
            if (list1.data < list2.data) {
                temp.next = list1;
                list1 = list1.next;
            } else {
                temp.next = list2;
                list2 = list2.next;
            }
            temp = temp.next;
        }

        // While list1 is not null
        while (list1 != null) {
            temp.next = list1;
            list1 = list1.next;
            temp = temp.next;
        }

        // While list2 is not null
        while (list2 != null) {
            temp.next = list2;
            head2 = list2.next;
            temp = temp.next;
        }
        return merged.next;
    }

    function mergeSort(head) {
        // Base case : if head is null
        if (head == null || head.next == null) {
            return head;
```

```
        }
        // get the middle of the list
        var middle = getMiddle(head);
        var nextofmiddle = middle.next;

        // set the next of middle node to null
        middle.next = null;

        // Apply mergeSort on left list
        var left = mergeSort(head);

        // Apply mergeSort on right list
        var right = mergeSort(nextofmiddle);

        // Merge the left and right lists
        var sortedlist = merge(left, right);
        return sortedlist;
    }
```