1. Explain Merge Sort.

Merge Sort

==========

Merge sort is a sorting algorithm.

- It uses a divide and conquer paradigm for sorting.
- It divides the problem into sub problems and solves them individually.
- It then combines the results of sub problems to get the solution of the original problem.

Step-01:

- Create two variables i and j for left (L[]) and right (R[]) sub arrays.
- Create variable k for sorted output array (A[]) .

Step-02:

- We have i = 0, j = 0, k = 0.
- Since L[0] < R[0], so we perform A[0] = L[0] i.e. we copy the first element from left sub array to our sorted output array.
- Then, we increment i and k by 1.

Step-03:

- We have i = 1, j = 0, k = 1.
- Since L[1] > R[0], so we perform A[1] = R[0] i.e. we copy the first element from right sub array to our sorted output array.
- Then, we increment j and k by 1.

Step-04:

- We have i = 1, j = 1, k = 2.
- Since L[1] > R[1], so we perform A[2] = R[1].
- Then, we increment j and k by 1.

Step-05:

- We have i = 1, j = 2, k = 3.
- Since L[1] < R[2], so we perform A[3] = L[1].
- Then, we increment i and k by 1.

Step-06:

- We have i = 2, j = 2, k = 4.
- Since L[2] > R[2], so we perform A[4] = R[2].
- Then, we increment j and k by 1.

Step-07:

- Since, all the elements from right sub array have been added to the sorted output array.
- So, we exit the first while loop with the condition while(i<nL && j<nR) since now j>nR.
- Then, we add remaining elements from the left sub array to the sorted output array using next while loop.

After finishing elements from any of the sub arrays, we can add the remaining elements from the other sub array to our sorted output array as it is. This is because left and right sub arrays are already sorted

Merge Sort Algorithm works in the following steps-

- It divides the given unsorted array into two halves- left and right sub arrays.
- The sub arrays are divided recursively.
- This division continues until the size of each sub array becomes 1.
- After each sub array contains only a single element, each sub array is sorted trivially.
- Then, the above discussed merge procedure is called.
- The merge procedure combines these trivially sorted arrays to produce a final sorted array.

2. Implement Merge sort.

Merge Sort

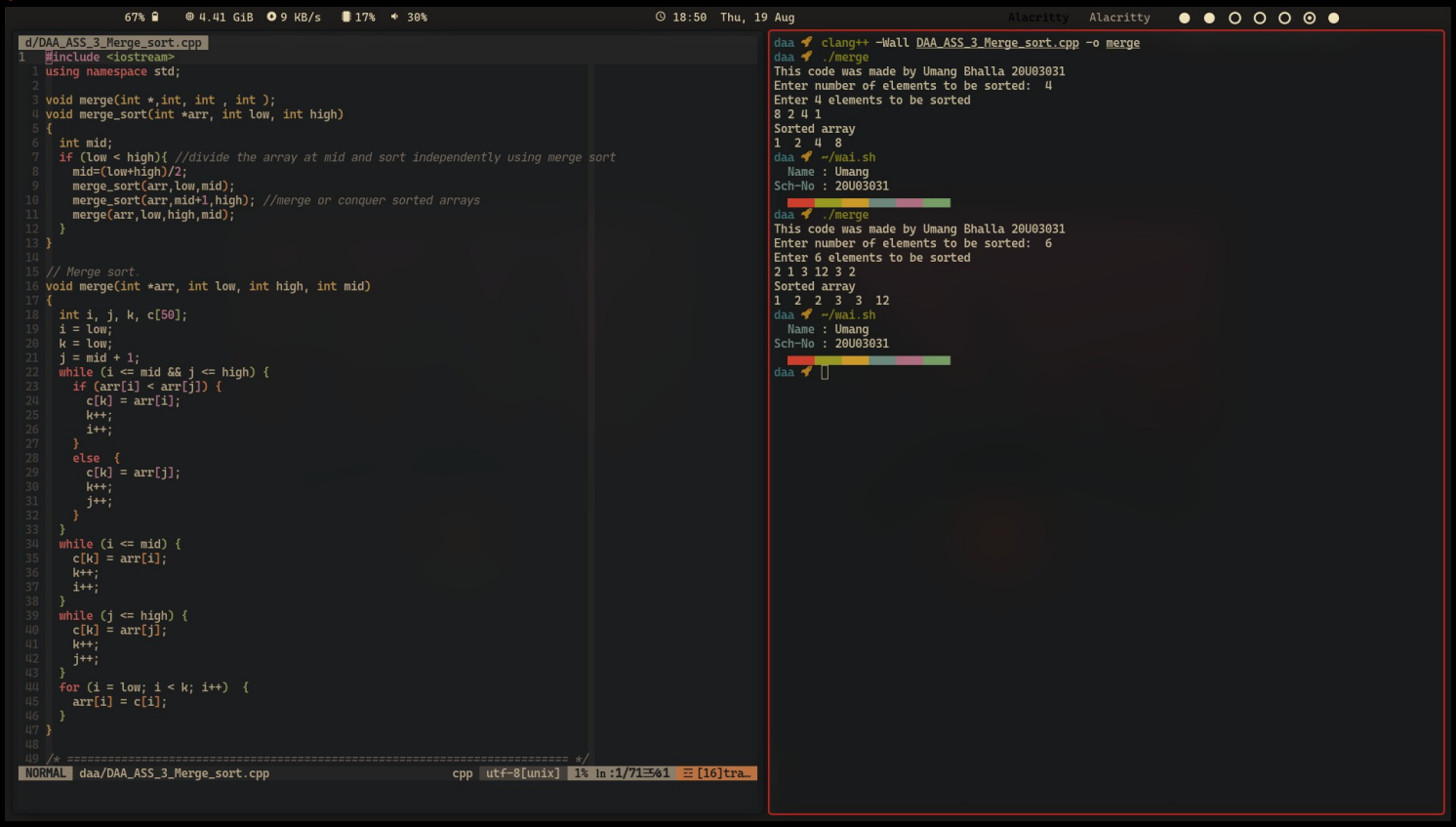
```
#include <iostream>
using namespace std;
void merge(int *,int, int , int );
void merge_sort(int *arr, int low, int high)
{
  int mid;
  if (low < high){ //divide the array at mid and sort independently using merge sort
    mid=(low+high)/2;
    merge_sort(arr,low,mid);
    merge_sort(arr,mid+1,high); //merge or conquer sorted arrays
    merge(arr,low,high,mid);
  }
}

// Merge sort
void merge(int *arr, int low, int high, int mid)
{
  int i, j, k, c[50];
  i = low;
  k = low;
  j = mid + 1;
  while (i <= mid && j <= high) {
    if (arr[i] < arr[j]) {
      c[k] = arr[i];
      k++;
      i++;
    }
    else  {
      c[k] = arr[j];
      k++;
      j++;
    }
  }
  while (i <= mid) {
    c[k] = arr[i];
    k++;
    i++;
  }
  while (j <= high) {
    c[k] = arr[j];
    k++;
    j++;
  }
  for (i = low; i < k; i++)  {
    arr[i] = c[i];
  }
```

```
}

/* ============================================================================ */

// read input array and call mergesort
int main()
{
  int myarray[30], num;
  cout<<"This code was made by Umang Bhalla 20U03031 \n";
  cout<<"Enter number of elements to be sorted: \t";
  cin>>num;
  cout<<"Enter "<<num<<" elements to be sorted \n";
  for (int i = 0; i < num; i++)
  {
    cin>>myarray[i];
  }
  merge_sort(myarray, 0, num-1);
  cout<<"Sorted array\n";
  for (int i = 0; i < num; i++)
  {
    cout<<myarray[i]<<"  ";
  }
  cout<<endl;
}
```



3.Explain their complexity.

In order to perform sorting using merge sort, we first divide the array into two equal halves. This is represented by “log n” which is a logarithmic function and the number of steps taken is log (n+1) at the most.

Next to find the middle element of the array we require single step i.e. O(1).

Then to merge the sub-arrays into an array of n elements, we will take O (n) amount of running time.

Thus the total time to perform merge sort will be n (log n+1), which gives us the time complexity of O (nlogn).

Worst case time complexity	O(nlog n)
Best case time complexity	O(nlog n)
Average time complexity	O(nlog n)
Space complexity	O(n)

The time complexity for merge sort is the same in all three cases (worst, best and average) as it always divides the array into sub-arrays and then merges the sub-arrays taking linear time.

Merge sort always takes an equal amount of space as unsorted arrays. Hence when the list to be sorted is an array, merge sort should not be used for very large arrays. However, merge sort can be used more effectively for linked lists sorting.