Course Name: A Level (2nd Sem) **Subject:** Data Struture using C++
Topic: Merge Sort in C++ **Date:** 15-04-2020

Example1 : Sorting an Array using Merge Sort (Interactive Bottom-up**)**

```
/* Sorting an array using Merge Sort with bottom-up interactive algorithm
   (MergeSortInteractive.cpp) */
#include <iostream>
using namespace std;

void mergeSort(int a[], int size);
void merge(int a[], int iLeftHalfLeft, int iLeftHalfRight,
           int iRightHalfLeft, int iRightHalfRight, int work[]);
void print(const int a[], int iLeft, int iRight);

int main()
{
   const int SIZE_1 = 8;
   int a1[SIZE_1] = {8, 4, 5, 3, 2, 9, 4, 1};

   print(a1, 0, SIZE_1 - 1);
   cout << endl;
   mergeSort(a1, SIZE_1);
   print(a1, 0, SIZE_1 - 1);
   cout << endl << endl;

  const int SIZE_2 = 13;
   int a2[SIZE_2] = {8, 4, 5, 3, 2, 9, 4, 1, 9, 1, 2, 4, 5};

   print(a2, 0, SIZE_2 - 1);
   cout << endl;
   mergeSort(a2, SIZE_2);
   print(a2, 0, SIZE_2 - 1);
   cout << endl;
}

// Sort the given array of size
void mergeSort(int a[], int size)
{
   int work[size];                          // work space
   int width = 1;                           // width of sublists to merge

   while (width < size)
{
                                            // Merge 2 sublists of width
      for (int i = 0; i < size - width; i += 2*width)
      {
        // Get the bounds of left and right sublists
        int iLeftHalfLeft = i;
        int iLeftHalfRight = i + width - 1;
        int iRightHalfLeft = i + width;
        int iRightHalfRight = i + 2*width - 1;
```

```
        if (iRightHalfRight >= size - 1) iRightHalfRight = size - 1;

        merge(a, iLeftHalfLeft, iLeftHalfRight, iRightHalfLeft, iRightHalfRight, work);
      }
      width *= 2;
    }
  }

// Merge two halves in [iLeftHalfLeft, iLeftHalfRight] and [iRightHalfLeft, iRightHalfRight]
// Assume that iLeftHalfRight + 1 = iRightHalfLeft

void merge(int a[], int iLeftHalfLeft, int iLeftHalfRight,
          int iRightHalfLeft, int iRightHalfRight, int work[])
{
   int size = iRightHalfRight - iLeftHalfLeft + 1;
   int iResult = 0;
   int iLeft = iLeftHalfLeft;
   int iRight = iRightHalfLeft;
   while (iLeft <= iLeftHalfRight && iRight <= iRightHalfRight)
   {
      if (a[iLeft] <= a[iRight])
      {
         work[iResult++] = a[iLeft++];
      } else {
         work[iResult++] = a[iRight++];
    }
   }

// Copy the remaining left or right into work
   while (iLeft <= iLeftHalfRight) work[iResult++] = a[iLeft++];
   while (iRight <= iRightHalfRight) work[iResult++] = a[iRight++];

   // for tracing
   print(a, iLeftHalfLeft, iLeftHalfRight);
   print(a, iRightHalfLeft, iRightHalfRight);
   cout << "=> ";
   print(work, 0, size - 1);
   cout << endl;

   // Copy the work back to the original array
   for (iResult = 0, iLeft = iLeftHalfLeft; iResult < size; ++iResult, ++iLeft)
   {
      a[iLeft] = work[iResult];
   }
}
 // Print the contents of the given array from iLeft to iRight (inclusive)
void print(const int a[], int iLeft, int iRight)
{
   cout << "{";
   for (int i = iLeft; i <= iRight; ++i)
   {
      cout << a[i];
      if (i < iRight) cout << ",";
   }
   cout << "} ";
}
```