2) Implement Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
# include <stdio.h>
# include <math.h>
# include <time.h>

void Merge(int a[], int low, int mid, int high) {
   int i, j, k, temp[high-low+1];
   i = low;
   k = 0;
   j = mid + 1;

   while (i <= mid && j <= high) {
      if (a[i] < a[j]) {
         temp[k] = a[i];
         k++;
         i++;
      } else {
         temp[k] = a[j];
         k++;
         j++;
      }
   }

   while (i <= mid) {
      temp[k] = a[i];
      k++;
      i++;
   }

   while (j <= high) {
      temp[k] = a[j];
      k++;
      j++;
   }
```

```
for (i = low; i <= high; i++) {
 a[i] = temp[i-low];
 }
}

void MergeSort(int a[], int low, int high) {
 int mid;
 if (low < high) {
 mid = (low + high) / 2;

 MergeSort(a, low, mid);
 MergeSort(a, mid + 1, high);

 Merge(a, low, mid, high);
 }
}

int main() {
 int n, a[100], k;
 clock_t st, et;
 double ts;

 printf("\n Enter How many Numbers: ");
 scanf("%d", &n);

 printf("\nThe Random Numbers are:\n");
 for (k = 1; k <= n; k++) {
 a[k] = rand() % (100 - 50 + 1) + 50;
 printf("%d\t", a[k]);
 }

 st = clock();
 for (k = 1; k < 10000; k++)
 MergeSort(a, 1, n);

 et = clock();
 ts = (double)(et - st) / CLOCKS_PER_SEC;
 ts = ts / 10000;
```

```
 printf("\nSorted Numbers are: \n ");
 for (k = 1; k <= n; k++)
 printf("%d\t", a[k]);

 printf("\nThe time taken is %e sec", ts);

 return 0;
}
```