Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. 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 <stdlib.h>
#include <time.h>
// Merge two subarrays of arr[]
void merge(int arr[], int I, int m, int r) {
  int i, j, k;
  int n1 = m - l + 1;
  int n2 = r - m;
  // Create temporary arrays
  int L[n1], R[n2];
  // Copy data to temporary arrays L[] and R[]
  for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
  for (j = 0; j < n2; j++)
     R[j] = arr[m + 1 + j];
  // Merge the temporary arrays back into arr[l..r]
  i = 0;
```

```
j = 0;
k = I;
while (i < n1 && j < n2) \{
  if (L[i] \le R[j]) {
    arr[k] = L[i];
    i++;
  } else {
    arr[k] = R[j];
    j++;
  }
  k++;
}
// Copy the remaining elements of L[], if there are any
while (i < n1) \{
  arr[k] = L[i];
  i++;
  k++;
}
// Copy the remaining elements of R[], if there are any
while (j < n2) {
  arr[k] = R[j];
  j++;
  k++;
```

```
}
}
// Merge Sort function
void mergeSort(int arr[], int I, int r) {
  if (I < r) {
    // Same as (I+r)/2, but avoids overflow for large I and r
    int m = I + (r - I) / 2;
    // Sort first and second halves
    mergeSort(arr, I, m);
    mergeSort(arr, m + 1, r);
    // Merge the sorted halves
    merge(arr, I, m, r);
  }
}
int main() {
  int n;
  printf("Enter the number of elements: ");
  scanf("%d", &n);
  // Generate n random numbers
  int arr[n];
```

```
srand(time(NULL));
for (int i = 0; i < n; ++i) {
    arr[i] = rand() % 10000; // Generate random numbers between 0 and 9999
}

// Measure the time taken for sorting
clock_t start = clock();
mergeSort(arr, 0, n - 1);
clock_t end = clock();

double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Time taken for sorting: %f seconds\n", time_taken);
return 0;
}</pre>
```