```cpp
#include <iostream>
#include <climits>
#include <ctime>
#include <vector>
#include <string>
#include <chrono>

using namespace std;

void mergeSort(int * A, int p, int q, int r);

int main() {
    int arrSize[11] = {100, 500, 1000, 5000, 10000, 50000, 100000, 500000,
      1000000, 5000000, 10000000};
    printf("Merge Sort\n");
    for(int i = 0; i < 11; ++i) {
        // Generate the random array of numbers to be sorted
        srand(clock());
        int * A = new int[arrSize[i]];
        for(int j = 0; j < arrSize[i]; ++j) {
            A[j] = rand();
        }
        // Get the start time
        auto init = chrono::high_resolution_clock::now();
        // Run the algorithm
        mergeSort(A, 0, (arrSize[i] - 1) / 2, arrSize[i] - 1);
        // Get then end time
        auto end = chrono::high_resolution_clock::now();
        // calculate the elapsed time
        auto duration = end - init;
        int sec = chrono::duration_cast<chrono::seconds>(duration).count();
        int nano = chrono::duration_cast<chrono::nanoseconds>(duration).count() %
          1000000000;

        printf("%i, %i.%09i\n", arrSize[i], sec, nano);
        // Make sure the output was sorted
        for (int j = 1; j < arrSize[i]; j++) {
            if (A[j] < A[j - 1]) {
                cout << "WRONG " << j;
            }
        }
        delete[] A;
    }
    // Wait for user input.
    string tmp;
    getline(cin, tmp);
}

void mergeSort(int * A, int p, int q, int r) {
    if(p < r) {
        mergeSort(A, p, (p + q) / 2, q);
        mergeSort(A, q + 1, (q + 1 + r) / 2, r);

        // this portion will be the actual merging of the arrays
        int * L = new int[q - p + 1];
        for(int i = 0; i < q - p + 1; ++i) {
            L[i] = A[p + i];
        }

        int * R = new int[r - q];
        for(int i = 0; i < r - q; ++i) {
            R[i] = A[q + 1 + i];
        }

        int i = 0;
        int j = 0;
        for(int k = p; k <= r; ++k) {
            // instead of using sentinel "infinite" values, we check to see if i and j
            // are in range
            if(( L[i] <= R[j] && i != (q - p + 1) ) || j == r - q) {
                A[k] = L[i];
                ++i;
            } else {
                A[k] = R[j];
                ++j;
            }
        }
        delete[] L;
        delete[] R;
    }
}

```