Lab: 2

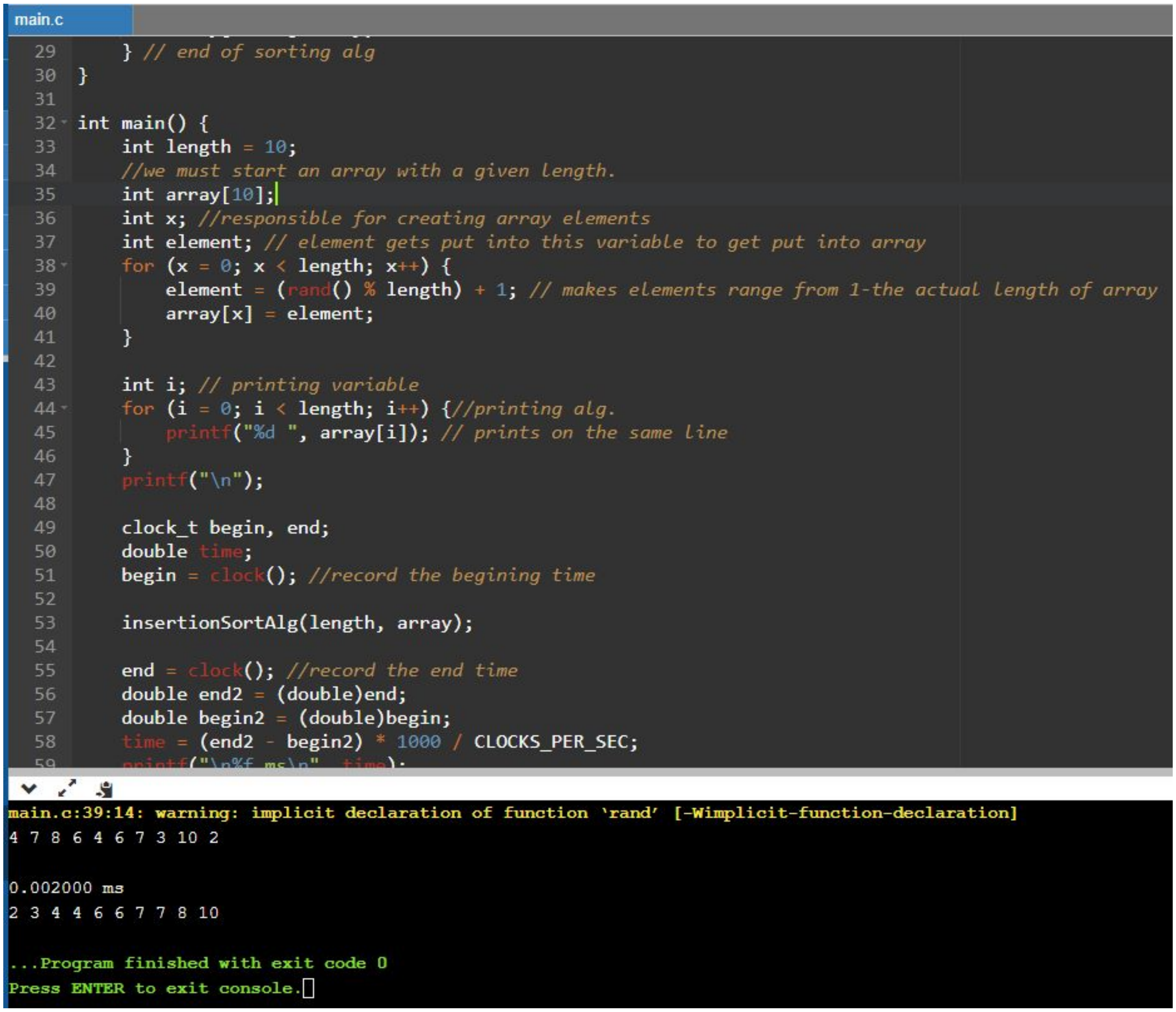
Execution Time and Algorithm Implementation

Due Date: January 14, 2021

Lab Session:Virtual/ Remote (A3, 16603)

Adrian Gomez
ID: 20119988

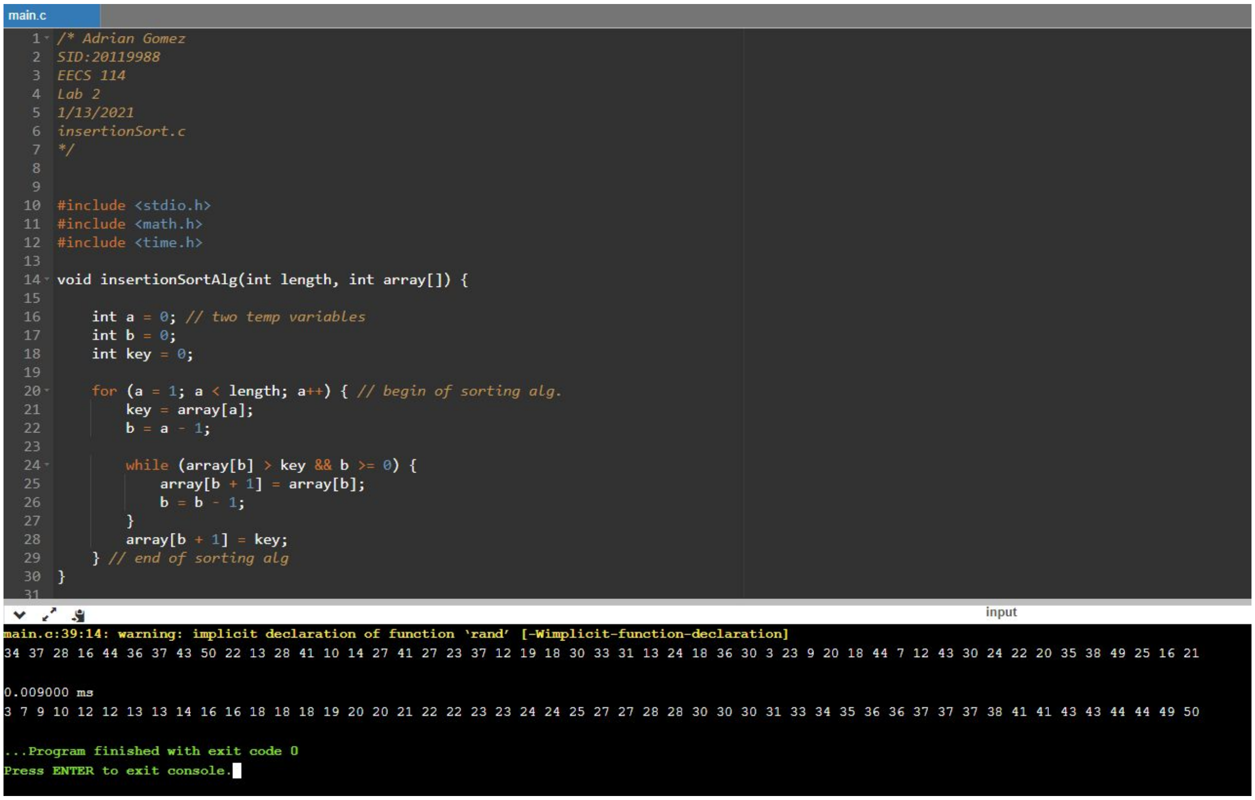
Insertion Sort Algorithm Simulation Result (n = 10 terms)



```c
    } // end of sorting alg
}

int main() {
    int length = 10;
    //we must start an array with a given length.
    int array[10];
    int x; //responsible for creating array elements
    int element; // element gets put into this variable to get put into array
    for (x = 0; x < length; x++) {
        element = (rand() % length) + 1; // makes elements range from 1-the actual length of array
        array[x] = element;
    }

    int i; // printing variable
    for (i = 0; i < length; i++) {//printing alg.
        printf("%d ", array[i]); // prints on the same line
    }
    printf("\n");

    clock_t begin, end;
    double time;
    begin = clock(); //record the begining time

    insertionSortAlg(length, array);

    end = clock(); //record the end time
    double end2 = (double)end;
    double begin2 = (double)begin;
    time = (end2 - begin2) * 1000 / CLOCKS_PER_SEC;
    printf("\n%f ms\n", time);
```

```
main.c:39:14: warning: implicit declaration of function ‘rand’ [-Wimplicit-function-declaration]
4 7 8 6 4 6 7 3 10 2

0.002000 ms
2 3 4 4 6 6 7 7 8 10

...Program finished with exit code 0
Press ENTER to exit console.
```

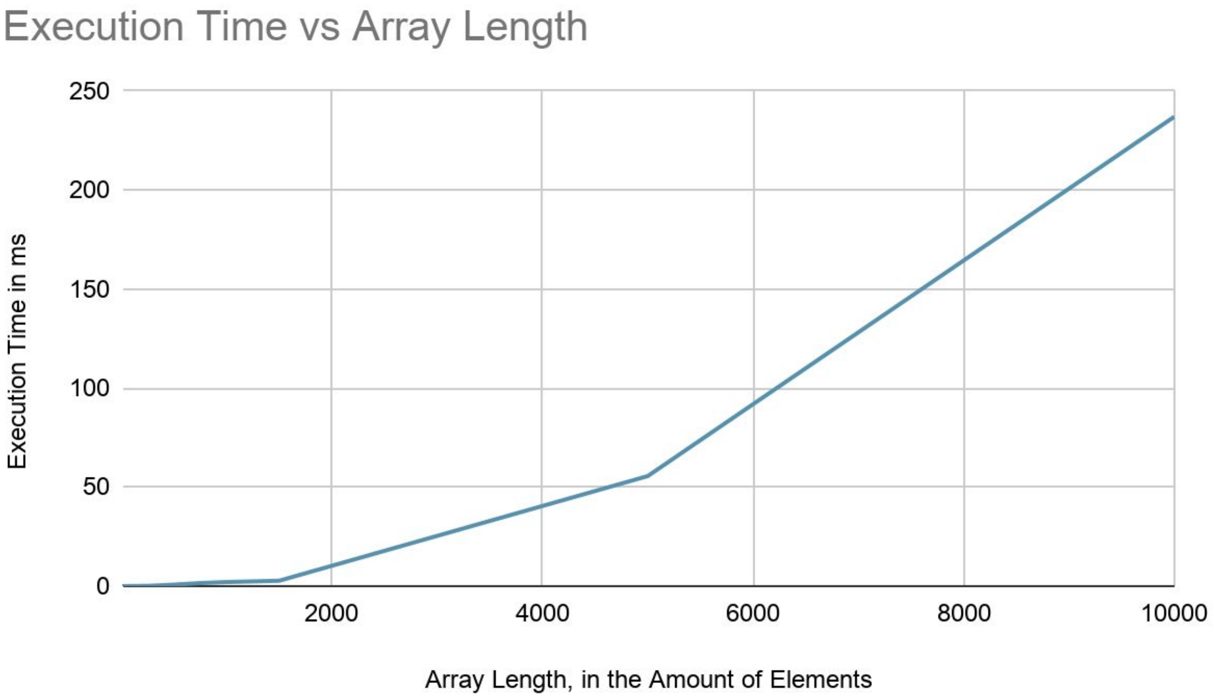
Insertion Sort Algorithm Simulation Result (n = 50 terms)

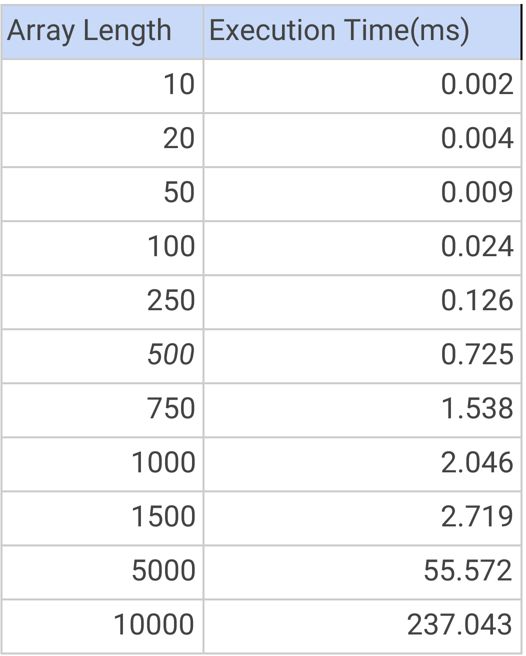


```c
/* Adrian Gomez
SID:20119988
EECS 114
Lab 2
1/13/2021
insertionSort.c
*/


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

void insertionSortAlg(int length, int array[]) {

    int a = 0; // two temp variables
    int b = 0;
    int key = 0;

    for (a = 1; a < length; a++) { // begin of sorting alg.
        key = array[a];
        b = a - 1;

        while (array[b] > key && b >= 0) {
            array[b + 1] = array[b];
            b = b - 1;
        }
        array[b + 1] = key;
    } // end of sorting alg
}
```

```
main.c:39:14: warning: implicit declaration of function ‘rand’ [-Wimplicit-function-declaration]
34 37 28 16 44 36 37 43 50 22 13 28 41 10 14 27 41 27 23 37 12 19 18 30 33 31 13 24 18 36 30 3 23 9 20 18 44 7 12 43 30 24 22 20 35 38 49 25 16 21

0.009000 ms
3 7 9 10 12 12 13 13 14 16 16 18 18 18 19 20 20 21 22 22 23 23 24 24 25 27 27 28 28 30 30 30 31 33 34 35 36 36 37 37 37 38 41 41 43 43 44 44 49 50

...Program finished with exit code 0
Press ENTER to exit console.
```

Insertion Sort Algorithm Graph

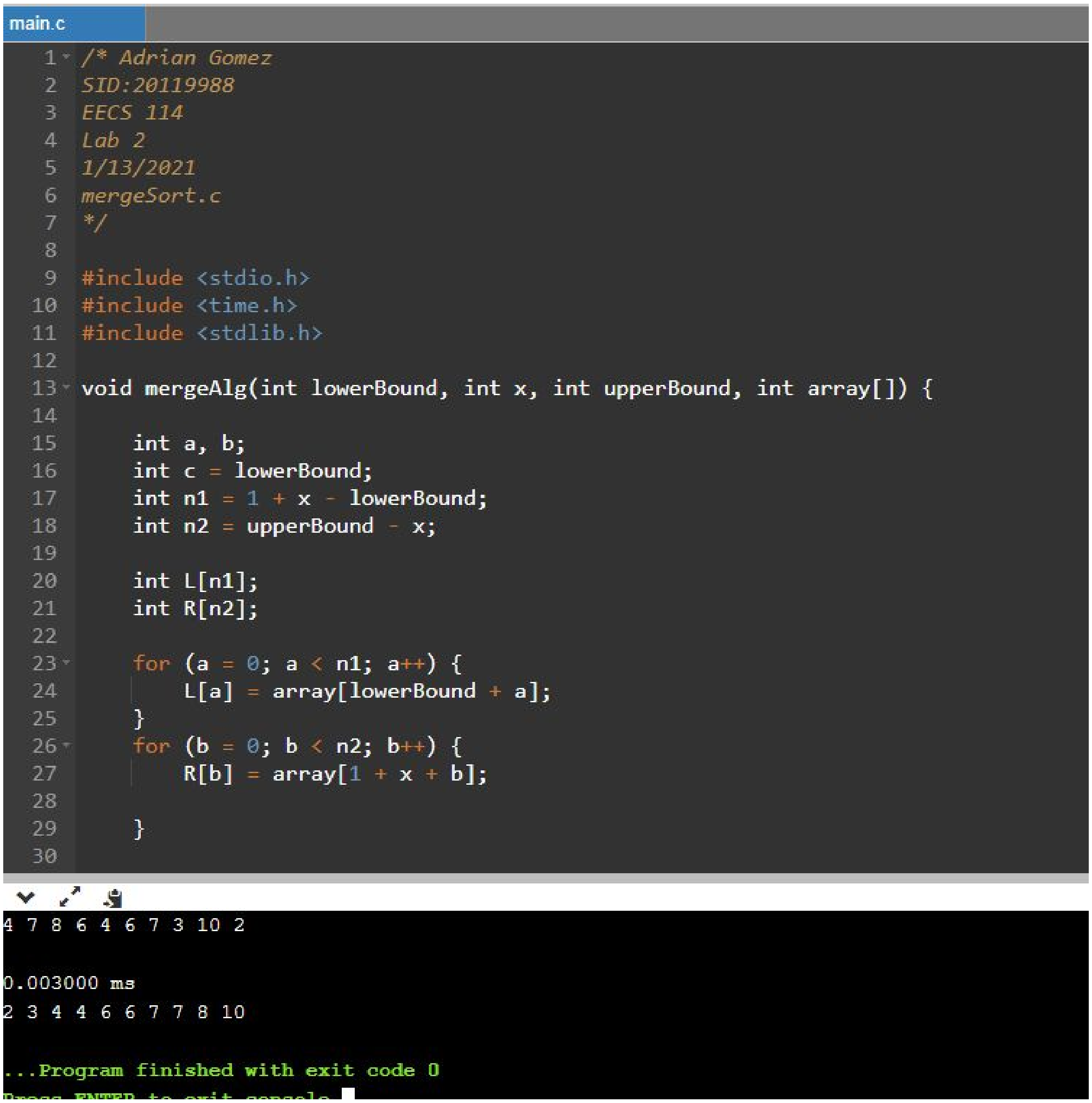




Array Length	Execution Time(ms)
10	0.002
20	0.004
50	0.009
100	0.024
250	0.126
500	0.725
750	1.538
1000	2.046
1500	2.719
5000	55.572
10000	237.043

Insertion sorting algorithm follows $T(n) = \theta(n^2)$ time that was given during lecture.

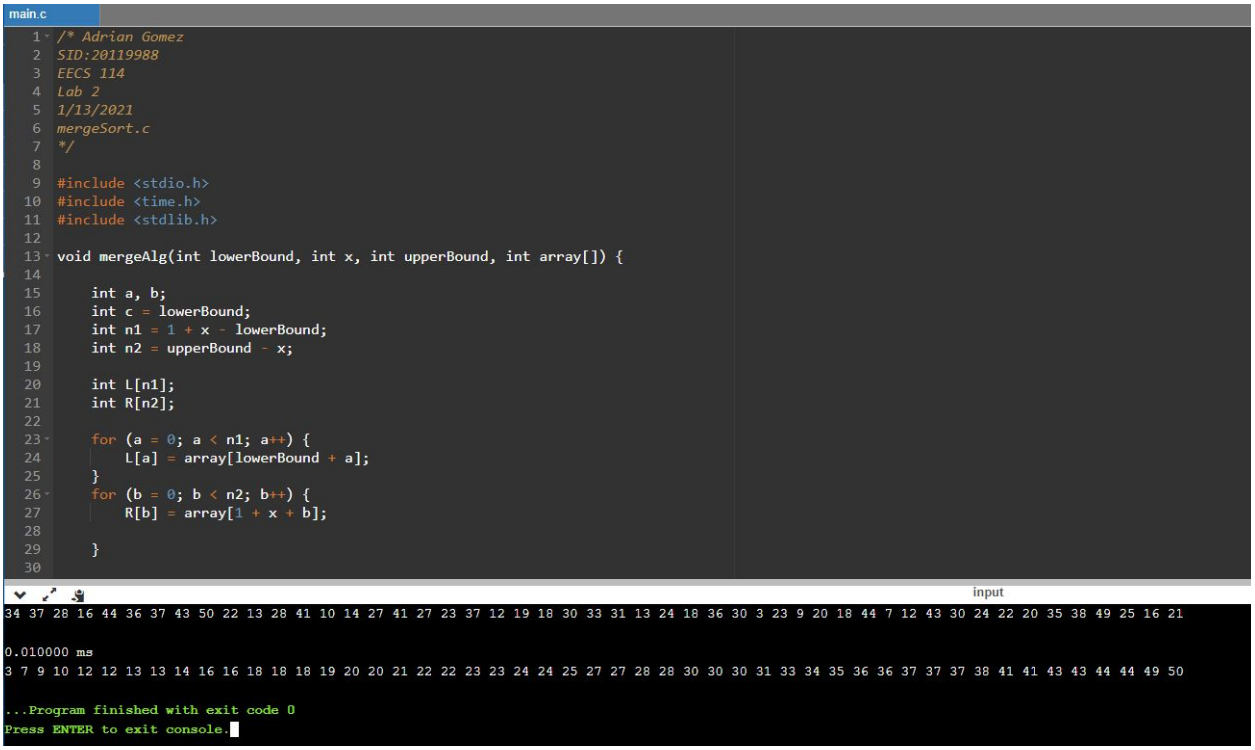
Merge Sort Algorithm Simulation Result (n = 10 terms)



```c
/* Adrian Gomez
SID:20119988
EECS 114
Lab 2
1/13/2021
mergeSort.c
*/

#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void mergeAlg(int lowerBound, int x, int upperBound, int array[]) {

    int a, b;
    int c = lowerBound;
    int n1 = 1 + x - lowerBound;
    int n2 = upperBound - x;

    int L[n1];
    int R[n2];

    for (a = 0; a < n1; a++) {
        L[a] = array[lowerBound + a];
    }
    for (b = 0; b < n2; b++) {
        R[b] = array[1 + x + b];

    }

```

```
4 7 8 6 4 6 7 3 10 2

0.003000 ms
2 3 4 4 6 6 7 7 8 10

...Program finished with exit code 0
Press ENTER to exit console.
```

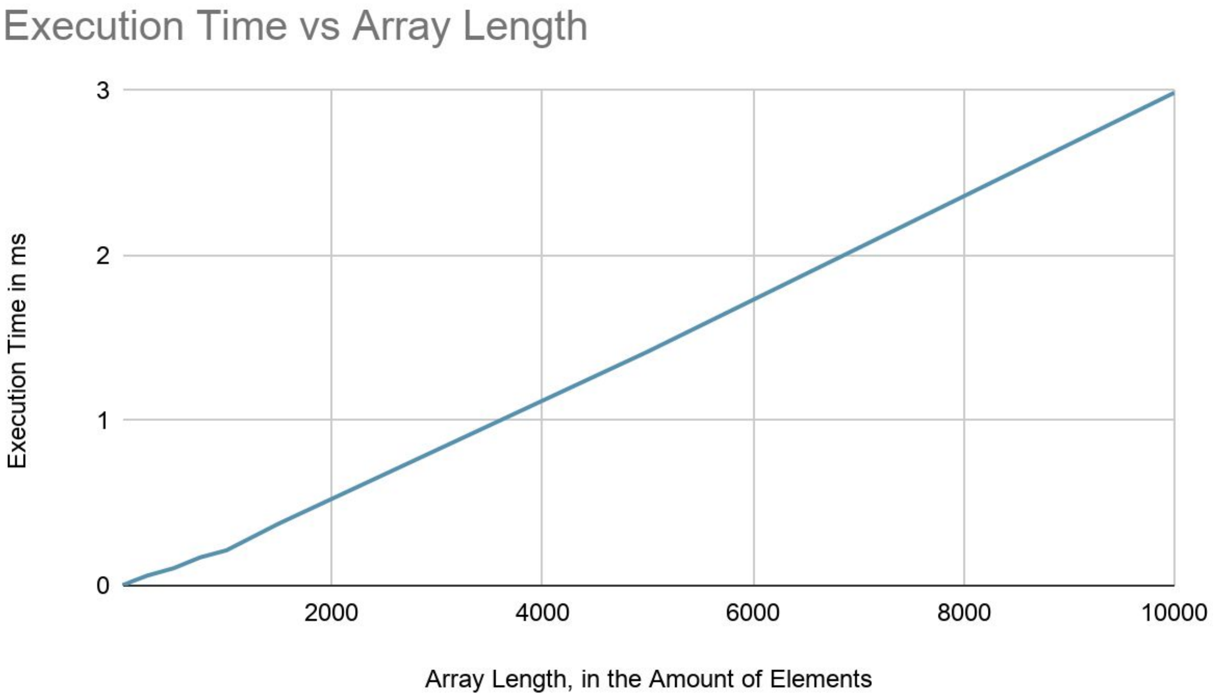
Merge Sort Algorithm Simulation Result (n = 50 terms)

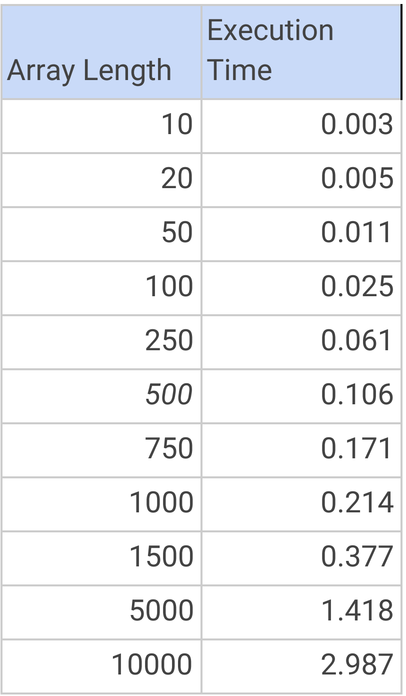


```c
/* Adrian Gomez
SID:20119988
EECS 114
Lab 2
1/13/2021
mergeSort.c
*/

#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void mergeAlg(int lowerBound, int x, int upperBound, int array[]) {

    int a, b;
    int c = lowerBound;
    int n1 = 1 + x - lowerBound;
    int n2 = upperBound - x;

    int L[n1];
    int R[n2];

    for (a = 0; a < n1; a++) {
        L[a] = array[lowerBound + a];
    }
    for (b = 0; b < n2; b++) {
        R[b] = array[1 + x + b];

    }

```

```
34 37 28 16 44 36 37 43 50 22 13 28 41 10 14 27 41 27 23 37 12 19 18 30 33 31 13 24 18 36 30 3 23 9 20 18 44 7 12 43 30 24 22 20 35 38 49 25 16 21

0.010000 ms
3 7 9 10 12 12 13 13 14 16 16 18 18 18 19 20 20 21 22 22 23 23 24 24 25 27 27 28 28 30 30 30 31 33 34 35 36 36 37 37 37 38 41 41 43 43 44 44 49 50

...Program finished with exit code 0
Press ENTER to exit console.
```

Merge Sort Algorithm Graph

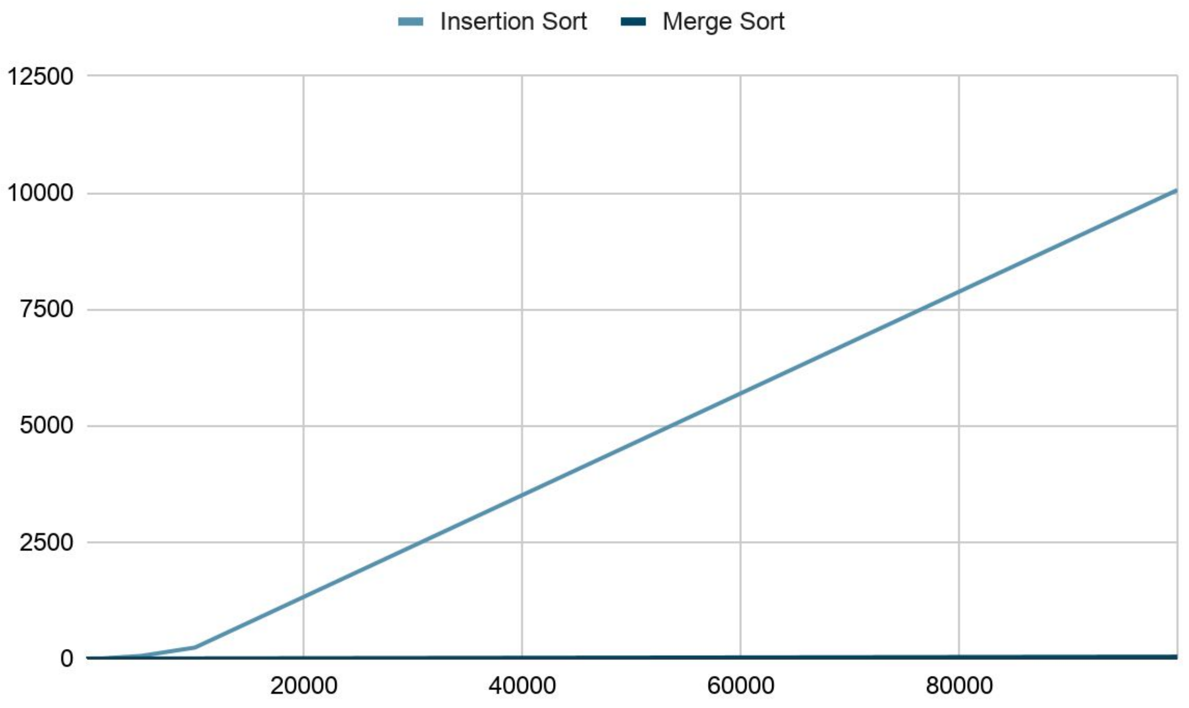




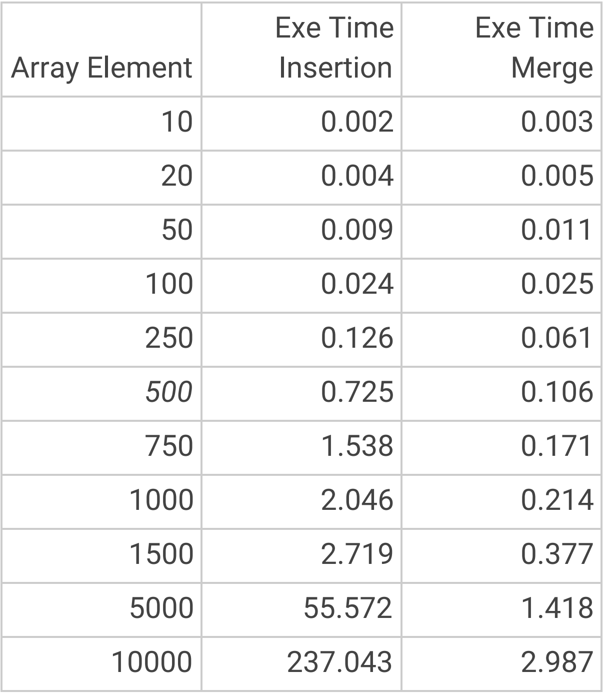
Array Length	Execution Time
10	0.003
20	0.005
50	0.011
100	0.025
250	0.061
500	0.106
750	0.171
1000	0.214
1500	0.377
5000	1.418
10000	2.987

Merge sorting algorithm follows $T(n) = \theta(n\ lg\ n)$ time that was given during lecture.

Comparison of Merge Sort vs Insertion Sort



Here we can see merge sort is much faster than insertion sort.



Array Element	Exe Time Insertion	Exe Time Merge
10	0.002	0.003
20	0.004	0.005
50	0.009	0.011
100	0.024	0.025
250	0.126	0.061
500	0.725	0.106
750	1.538	0.171
1000	2.046	0.214
1500	2.719	0.377
5000	55.572	1.418
10000	237.043	2.987