Lab 5: Sorting

In this lab we covered sorting and efficiency. We were required to make 6 different sorting methods. They were bubble sort, insertion sort, merge-sort, quicksort, counting sort, and radix sort. Then, we had to use the chrono library to record the time it takes for each sorting method to sort through a list of a certain size. The time values were recorded in a table for an array size of 10, 100, 500, 25,000, and 100,000 values. Then, we were required to use 3 of the 6 sorting methods to sort an unordered linked list. We used bubble sort to sort the list using M numbers, insertion sort to sort the list using first names in an alphabetical order, and merge sort to sort the list using last names in an alphabetical order. The coding skills acquired from this lab will help us excel in our future career as computer or software engineers.

Contributions:

Raman Hliabovich – Brainstorming and code revision. Cleaning up mains. Code testing and lab report. Fixed merge sort and insertion sort for linked list.

Sean Sliter – Brainstorming and code revisions. Most of Task 3. Made the table for task 2.

Zack Johnson – Brainstorming and code revisions. Most of Task 1. Helped with Task 3.

Task 1:

Task one output screen gives the option to select the array size, prints the original array, and outputs the time it takes for each sorting algorithm to sort the selected array (size 10) in nanoseconds.

```
Select an array size:
 1. 10
 2. 100
 3. 500
 4. 5000
 5. 25,000
 6. 100,000

 Enter: 1

 Testing...

 Original
 13 8 2 20 10 11 6 8 15 1

 Bubble Time: 4923 nanoseconds
 1 2 6 8 8 10 11 13 15 20

 Insertion Time: 410 nanoseconds


 Merge Time: 3693 nanoseconds


 Quick Time: 821 nanoseconds


 Count Time: 1641 nanoseconds
 1 2 6 8 8 10 11 13 15 20

 Radix Time: 1641 nanoseconds


 Test Again?(y/n)
```

Output times for a 100 size array.

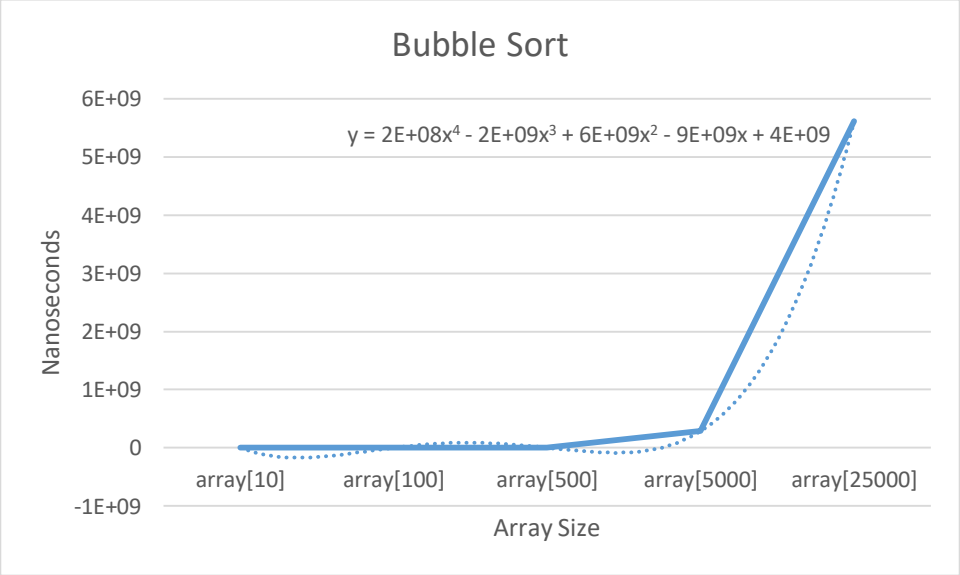
```
Select an array size:
 1. 10
 2. 100
 3. 500
 4. 5000
 5. 25,000
 6. 100,000

 Enter: 2

 Testing...

 Original
 45 17 191 9 30 92 22 185 42 157 94 71 157 178 46 18 104 191 191 168 93 119 187 130 16 65 175 77 122 76 24 42 76 163 69 182 179 191
44 106 136 170 87 54 105 72 8 40 34 181 81 116 57 156 87 119 14 83 173 186 94 138 160 59 67 63 135 72 160 68 105 161 116 99 171 76 3
 129 43 129 132 5 61 159 91 109 6 184 25 138 81 153 122 117 103 25 141 129 95 92

 Bubble Time: 51282 nanoseconds
 3 5 6 8 9 14 16 17 18 22 24 25 25 30 34 40 42 42 43 44 45 46 54 57 59 61 63 65 67 68 69 71 72 72 76 76 76 77 81 81 83 87 87 91 92 9
2 93 94 94 95 99 103 104 105 105 106 109 116 116 117 119 119 122 122 129 129 129 130 132 135 136 138 138 141 153 156 157 157 159 160
 160 161 163 168 170 171 173 175 178 179 181 182 184 185 186 187 191 191 191 191

 Insertion Time: 10256 nanoseconds


 Merge Time: 71385 nanoseconds


 Quick Time: 12308 nanoseconds


 Count Time: 6975 nanoseconds
 3 5 6 8 9 14 16 17 18 22 24 25 25 30 34 40 42 42 43 44 45 46 54 57 59 61 63 65 67 68 69 71 72 72 76 76 76 77 81 81 83 87 87 91 92 9
2 93 94 94 95 99 103 104 105 105 106 109 116 116 117 119 119 122 122 129 129 129 130 132 135 136 138 138 141 153 156 157 157 159 160
 160 161 163 168 170 171 173 175 178 179 181 182 184 185 186 187 191 191 191 191

 Radix Time: 14769 nanoseconds


 Test Again?(y/n)
```

Task 2:

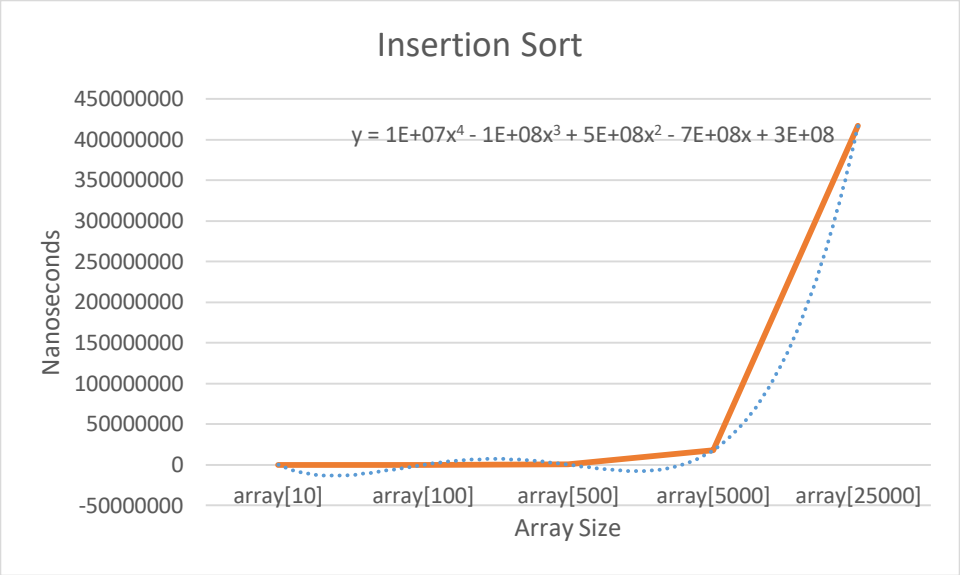
Table of times it took to sort the array of different sizes for each sorting algorithm.

	array[10]	array[100]	array[500]	array[5000]	array[25000]
Bubble Sort	1763	87801	2198919	287217391	5611251159
Insertion Sort	352	11637	280331	18276486	417268848
Merge Sort	5289	95559	662215	3315656	18267318
Quick sort	1058	21509	166788	1903073	10372931
Counting Sort	2116	5995	34204	142810	651989
Radix Sort	3526	14457	118126	699592	4117860

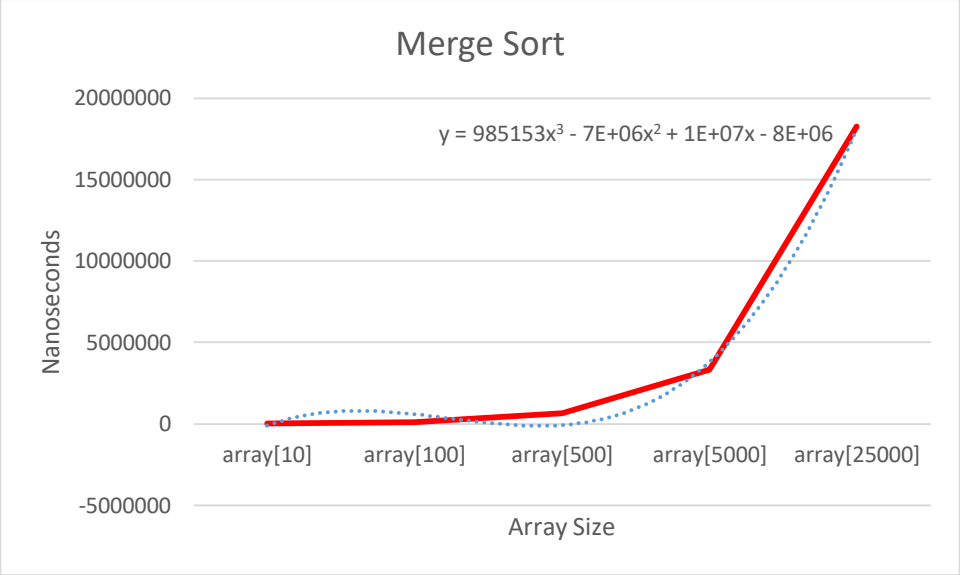
Graphs for each for each of the sorting algorithms showing the relationship between array size and sorting time.



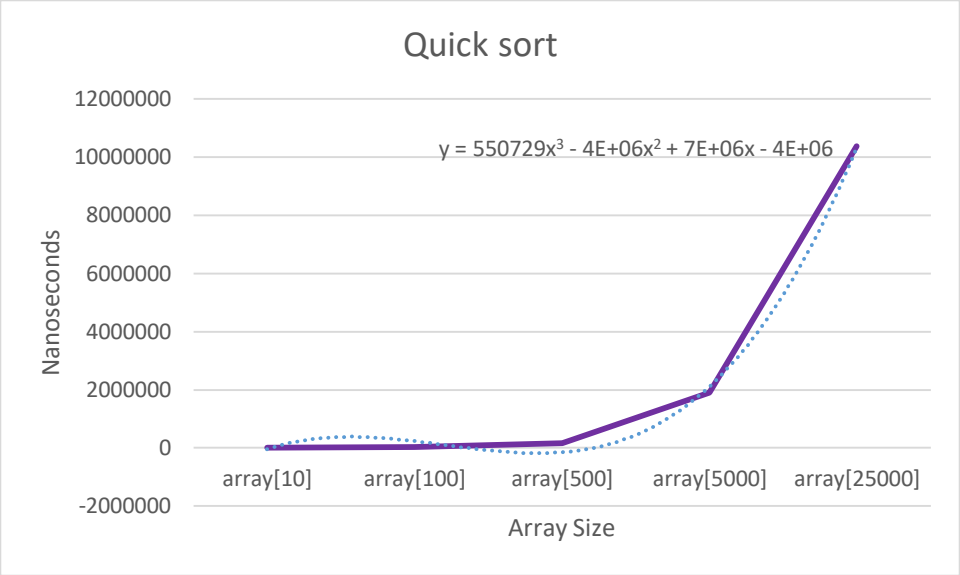
Bubble Sort Performance: $O(n^4)$



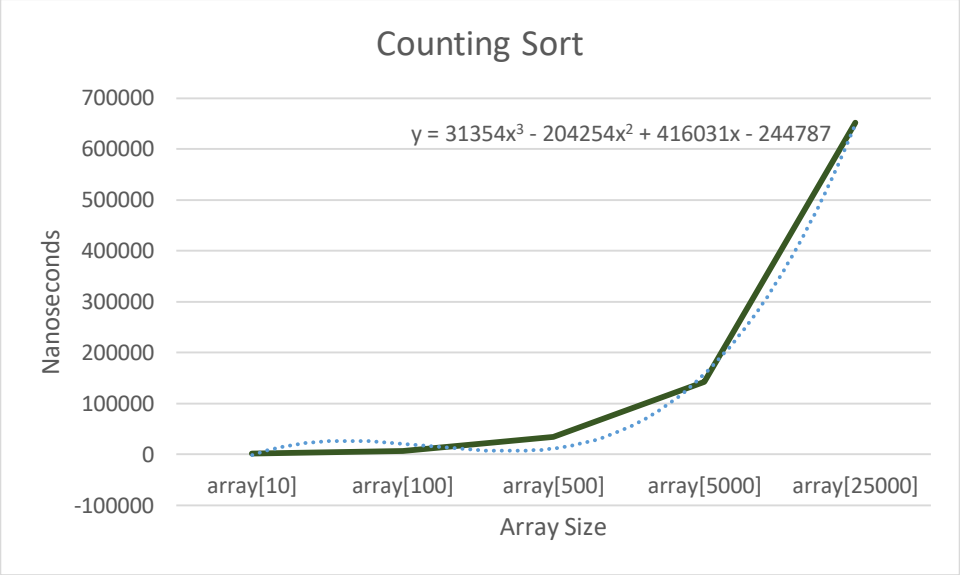
Insertion sort Performance: $O(n^4)$



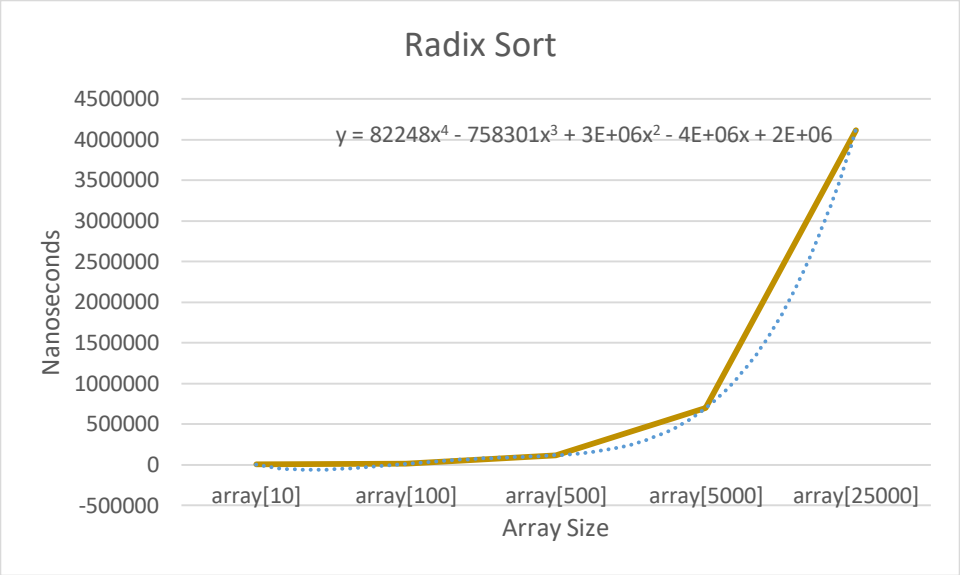
Merge Sort Performance: $O(n^3)$



Quick Sort Performance: $O(n^3)$



Counting Sort Performance: $O(n^3)$



Counting Sort Performance: $O(n^4)$

Task 3:

In Task 3 the program generates a linked list of 50 random Mnumbers, first names, and last names. Then it asks you to choose which sorting algorithm you want to use. Next, it asks what direction you want to print the sorted list in and outputs the 50 values in the linked list accordingly.

This output shows the shows the sorted linked list that was sorted with a Bubble sort using the Mnumbers to sort the list. Then it outputs the 50 students sorted in the linked list in an ascending direction. (Only the first 6/50 sorted students are shown in the picture because the list is too long to fit in the image below)

```
Seeding list...
List seeded!

Which sorting algorithm would you like to use?
1. Bubble Sort (mNum)
2. Insertion Sort (first name)
3. Merge Sort (last name)

Enter: 1

Which direction would you like the list to be ordered in?
1. Ascending
2. Descending

Enter: 1

Traversal in forward direction
**********************
* First Name: YlNlfdXf
*  Last Name: HqghUmea
*    Mnumber: M42
**********************

**********************
* First Name: BPpWEQtg
*  Last Name: GrPUNkdM
*    Mnumber: M1201
**********************

**********************
* First Name: hzRIWKBa
*  Last Name: rimPMYHc
*    Mnumber: M1540
**********************

**********************
* First Name: etlyhNkO
*  Last Name: mnMgqOok
*    Mnumber: M1588
**********************

**********************
* First Name: RtkJPREP
*  Last Name: wfnfOzvS
*    Mnumber: M1870
**********************

**********************
* First Name: vDwvxhRC
*  Last Name: NFwlQiJT
*    Mnumber: M3435
**********************
```

Sorting list in an alphabetical order with insertion sort using firs names. Printing sorted list in an ascending order.

```
 Seeding list...
 List seeded!

 Which sorting algorithm would you like to use?
 1. Bubble Sort (mNum)
 2. Insertion Sort (first name)
 3. Merge Sort (last name)

 Enter: 2
Sorting first name alphabetically.


 Which direction would you like the list to be ordered in?
 1. Ascending
 2. Descending

 Enter: 1

 Traversal in forward direction
**********************
* First Name: BiWybxDU
*  Last Name: QmKiMwzo
*    Mnumber: M13032
**********************

**********************
* First Name: BPpWEQtg
*  Last Name: GrPUNkdM
*    Mnumber: M1201
**********************

**********************
* First Name: boyGpoey
*  Last Name: OeJUvPva
*    Mnumber: M16513
**********************

**********************
* First Name: buSBMbOr
*  Last Name: lDVgylwG
*    Mnumber: M12950
**********************

**********************
* First Name: BwKfnqDu
*  Last Name: rcvsCxgG
*    Mnumber: M11943
**********************

**********************
* First Name: CJtnzxUg
*  Last Name: UdLTFZot
*    Mnumber: M17550
**********************
```

Sorting list in an alphabetical order with merge sort using last names. Printing sorted list in a descending order.

```
 Seeding list...
 List seeded!

 Which sorting algorithm would you like to use?
 1. Bubble Sort (mNum)
 2. Insertion Sort (first name)
 3. Merge Sort (last name)

 Enter: 3
Sorting last name alphabetically.


 Which direction would you like the list to be ordered in?
 1. Ascending
 2. Descending

 Enter: 2

 Traversal in reverse direction
**********************
* First Name: pSajlfVg
*  Last Name: yyzpVScM
*    Mnumber: M22647
**********************

**********************
* First Name: FNuXxZrz
*  Last Name: xjsWnkkU
*    Mnumber: M20601
**********************

**********************
* First Name: yhjUgIXW
*  Last Name: xBgFcBce
*    Mnumber: M8520
**********************

**********************
* First Name: JzwucWlj
*  Last Name: XOThowKb
*    Mnumber: M5707
**********************

**********************
* First Name: hdIZcoBz
*  Last Name: wSRtEsJW
*    Mnumber: M10286
**********************

**********************
* First Name: RtkJPREP
*  Last Name: wfnfOzvS
*    Mnumber: M1870
**********************
```