David Giacobbi
HW2-Sorting: Report

Program Output:

```
./main 100
--------------------------------------------------------------------------------
Sort type: Bubble Sort
Bubble Sort, input data: Ascending, n=100, average time: 0.0001172 secs
Bubble Sort, input data: Descending, n=100, average time: 0.0002468 secs
Bubble Sort, input data: Random, n=100, average time: 0.0001818 secs
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Sort type: Bubble Sort (Optimized)
Bubble Sort (Optimized), input data: Ascending, n=100, average time: 2.4e-06 secs
Bubble Sort (Optimized), input data: Descending, n=100, average time: 0.0001934 secs
Bubble Sort (Optimized), input data: Random, n=100, average time: 0.0001812 secs
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Sort type: Selection Sort
Selection Sort, input data: Ascending, n=100, average time: 6.48e-05 secs
Selection Sort, input data: Descending, n=100, average time: 8.36e-05 secs
Selection Sort, input data: Random, n=100, average time: 7.78e-05 secs
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Sort type: Insertion Sort
Insertion Sort, input data: Ascending, n=100, average time: 3.4e-06 secs
Insertion Sort, input data: Descending, n=100, average time: 0.0001122 secs
Insertion Sort, input data: Random, n=100, average time: 6.36e-05 secs
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Sort type: Std::sort
Std::sort, input data: Ascending, n=100, average time: 1.8e-06 secs
Std::sort, input data: Descending, n=100, average time: 1.6e-06 secs
Std::sort, input data: Random, n=100, average time: 6.2e-06 secs
--------------------------------------------------------------------------------
./main 1000
--------------------------------------------------------------------------------
Sort type: Bubble Sort
Bubble Sort, input data: Ascending, n=1000, average time: 0.005221 secs
Bubble Sort, input data: Descending, n=1000, average time: 0.0072936 secs
```

Bubble Sort, input data: Random, n=1000, average time: 0.0061814 secs
---

Sort type: Bubble Sort (Optimized)
Bubble Sort (Optimized), input data: Ascending, n=1000, average time: 8e-06 secs
Bubble Sort (Optimized), input data: Descending, n=1000, average time: 0.0064338 secs
Bubble Sort (Optimized), input data: Random, n=1000, average time: 0.0062626 secs
---

Sort type: Selection Sort
Selection Sort, input data: Ascending, n=1000, average time: 0.0019408 secs
Selection Sort, input data: Descending, n=1000, average time: 0.0025882 secs
Selection Sort, input data: Random, n=1000, average time: 0.0020272 secs
---

Sort type: Insertion Sort
Insertion Sort, input data: Ascending, n=1000, average time: 9.2e-06 secs
Insertion Sort, input data: Descending, n=1000, average time: 0.0036998 secs
Insertion Sort, input data: Random, n=1000, average time: 0.0018188 secs
---

Sort type: Std::sort
Std::sort, input data: Ascending, n=1000, average time: 1.6e-06 secs
Std::sort, input data: Descending, n=1000, average time: 2.8e-06 secs
Std::sort, input data: Random, n=1000, average time: 2.78e-05 secs
---
./main 10000

Sort type: Bubble Sort
Bubble Sort, input data: Ascending, n=10000, average time: 0.280581 secs
Bubble Sort, input data: Descending, n=10000, average time: 0.612227 secs
Bubble Sort, input data: Random, n=10000, average time: 0.584323 secs
---

Sort type: Bubble Sort (Optimized)
Bubble Sort (Optimized), input data: Ascending, n=10000, average time: 5.76e-05 secs
Bubble Sort (Optimized), input data: Descending, n=10000, average time: 0.617744 secs
Bubble Sort (Optimized), input data: Random, n=10000, average time: 0.585933 secs
---

Sort type: Selection Sort
Selection Sort, input data: Ascending, n=10000, average time: 0.186913 secs
Selection Sort, input data: Descending, n=10000, average time: 0.241049 secs
Selection Sort, input data: Random, n=10000, average time: 0.187621 secs
---

Sort type: Insertion Sort
Insertion Sort, input data: Ascending, n=10000, average time: 8.76e-05 secs
Insertion Sort, input data: Descending, n=10000, average time: 0.348585 secs
Insertion Sort, input data: Random, n=10000, average time: 0.175237 secs
---

Sort type: Std::sort
Std::sort, input data: Ascending, n=10000, average time: 1.04e-05 secs
Std::sort, input data: Descending, n=10000, average time: 1.68e-05 secs
Std::sort, input data: Random, n=10000, average time: 0.0003578 secs
---
./main 20000

Sort type: Bubble Sort
Bubble Sort, input data: Ascending, n=20000, average time: 1.1178 secs
Bubble Sort, input data: Descending, n=20000, average time: 2.45922 secs
Bubble Sort, input data: Random, n=20000, average time: 2.36492 secs
---

Sort type: Bubble Sort (Optimized)
Bubble Sort (Optimized), input data: Ascending, n=20000, average time: 0.0001172 secs
Bubble Sort (Optimized), input data: Descending, n=20000, average time: 2.49833 secs
Bubble Sort (Optimized), input data: Random, n=20000, average time: 2.34315 secs
---

Sort type: Selection Sort
Selection Sort, input data: Ascending, n=20000, average time: 0.746091 secs
Selection Sort, input data: Descending, n=20000, average time: 0.963231 secs
Selection Sort, input data: Random, n=20000, average time: 0.748066 secs
---

Sort type: Insertion Sort
Insertion Sort, input data: Ascending, n=20000, average time: 0.0001764 secs
Insertion Sort, input data: Descending, n=20000, average time: 1.39359 secs

Insertion Sort, input data: Random, n=20000, average time: 0.696975 secs
---

Sort type: Std::sort
Std::sort, input data: Ascending, n=20000, average time: 2.04e-05 secs
Std::sort, input data: Descending, n=20000, average time: 3.46e-05 secs
Std::sort, input data: Random, n=20000, average time: 0.0007842 secs
---
./main 40000

Sort type: Bubble Sort
Bubble Sort, input data: Ascending, n=40000, average time: 4.46986 secs
Bubble Sort, input data: Descending, n=40000, average time: 9.7938 secs
Bubble Sort, input data: Random, n=40000, average time: 9.34578 secs
---

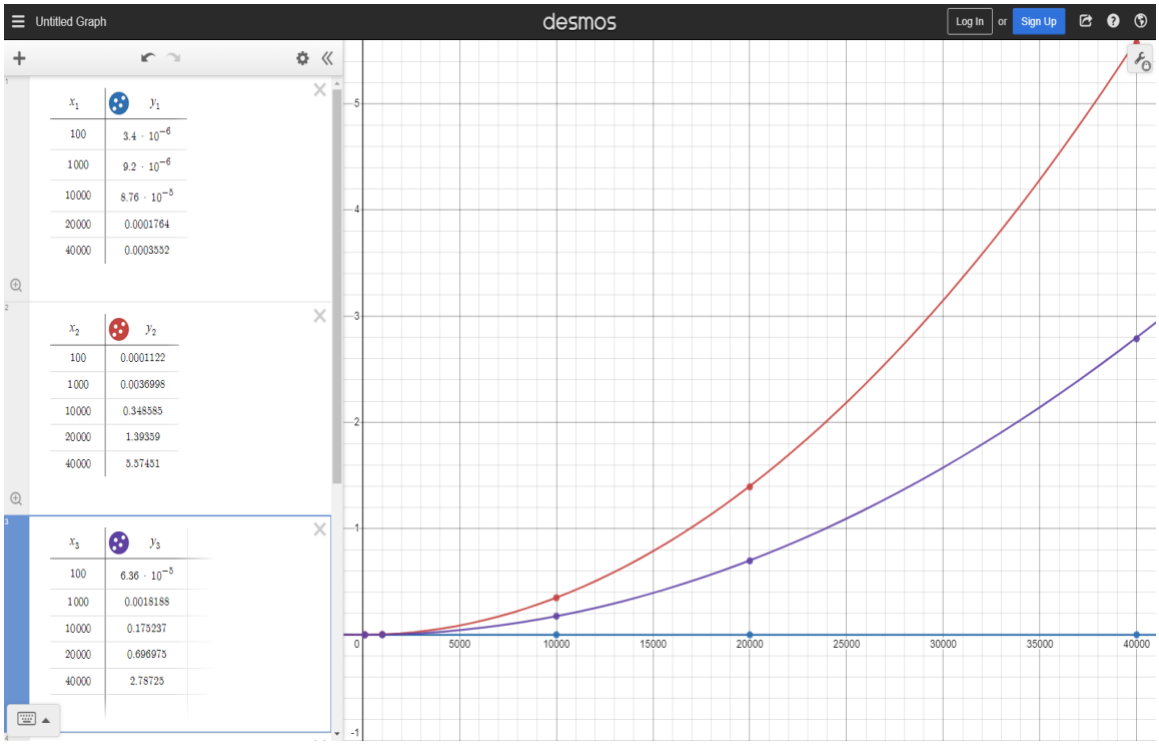
Sort type: Bubble Sort (Optimized)
Bubble Sort (Optimized), input data: Ascending, n=40000, average time: 0.0002316 secs
Bubble Sort (Optimized), input data: Descending, n=40000, average time: 9.79604 secs
Bubble Sort (Optimized), input data: Random, n=40000, average time: 9.35067 secs
---

Sort type: Selection Sort
Selection Sort, input data: Ascending, n=40000, average time: 2.98079 secs
Selection Sort, input data: Descending, n=40000, average time: 3.8571 secs
Selection Sort, input data: Random, n=40000, average time: 2.98457 secs
---

Sort type: Insertion Sort
Insertion Sort, input data: Ascending, n=40000, average time: 0.0003552 secs
Insertion Sort, input data: Descending, n=40000, average time: 5.57451 secs
Insertion Sort, input data: Random, n=40000, average time: 2.78725 secs
---

Sort type: Std::sort
Std::sort, input data: Ascending, n=40000, average time: 4.04e-05 secs
Std::sort, input data: Descending, n=40000, average time: 7.16e-05 secs
Std::sort, input data: Random, n=40000, average time: 0.0016332 secs
---

Insertion Sort Graphed Results:



Blue Line: Ascending (y = 0.000000009x)
Red Line: Descending (y = 0.0000000035x^2)
Purple Line: Random (y = 0.00000000175x^2)

The graph above helps to demonstrate how the time cost increases significantly from best case scenario (ascending) to worst case scenario (descending). For insertion sort, the lines derived from the plotted points show that at best case, this algorithm has a complexity of *n*. The worst case scenario has a complexity of n^2. Even though the coefficients are different on all the tests, the insertion sort still follows the theoretical growth rates mentioned in the text. This implies that even though insertion sort could be quite fast for nearly sorted data or smaller data set, insertion sort should not be used for larger data sets with potential of descending data.

Questions:

A. The number of inversions present directly affected the time cost for each of the algorithms. This is because as the number of iterations required to sort the list increased, the time cost increases as well to carry out the iterations. This is seen most significantly in both bubble sorts. Even though an already sorted list could pass through the algorithm quite efficiently, inverted, and randomized lists took far longer and increased quite quickly as the dataset increased. Sorting time reached up to 9 seconds for the largest

dataset on both bubble sorts. Similarly, insertion and standard sort both drew the same time complexity paths for best case and worst case. As the dataset approached a worst-case scenario of a completely inverted list, time cost increased to a squared function. Lastly, selection sort had an interesting type of behavior with the randomized datasets. When the dataset was small, the time cost trended closer towards the worst-case scenario time; however, as the dataset grew, the time cost trended closer towards the best-case scenario.

B. The time cost follows very close to the expected growth for their input. The optimized bubble sort had acted as expected; however, the efficacy of a clean pass variable was a lot greater than expected. For example, for the n = 40000 ascending case, the optimized version still took only 0.0002316 seconds on average. On the other side of the spectrum, the bubble sort performed quite poorly compared to the other algorithms with a Big-O complexity of n^2. This can especially be observed in the n = 20000 descending cases for bubble sort and insertion sort. While insertion sort, which also has a $O(n^2)$, only took 1.39359 seconds, bubble sort nearly doubled this number with an average of 2.45922 seconds. Lastly, the standard library sort seemed to have astounding efficiency in sorting numbers, regardless of the dataset size. In a sort size of n = 10000 randomized, the standard sort outperformed all the other sorts significantly with an average time of 0.0003578 seconds.

C. The standard sort follows a very similar complexity to the insertion sort. Even though insertion sort had overall greater time costs than the standard sort, the Big-O and Big-Omega complexities share a similar pattern. I think that the built-in standard sort could be more effective because there are more stopping parameters, similar to the optimization of bubble sort. Much like insertion sort, the standard sort's points plot a y = x^2 for Big-O as well as a y = x for Big-Omega. The only thing that is different is the scale of the coefficient used to trace out the theoretical lines on the empirical data.

D. Some challenges that I faced was trying to understand how exactly each sort works. It is hard to write up code when you are only provided with written descriptions and figures of how the sort works. However, the visualgo.net website was incredibly helpful in visualizing the sorts for me and helping me to generate some pseudocode to base my algorithms off. I think that analysis of the different time costs was the easiest part of this assignment. We have already discussed in detail the difference between worst-case and best-case scenario time complexities. In fact, knowing these detail beforehand made it easier to write the algorithms, knowing how many loops and what kinds of loops were required to fulfill the theoretical time cost goals.