Analyzing the Runtime of Bubble and Selection Sort Algorithm
Lab #7

By
Michael Harden

CS 303 Algorithms and Data Structures
October 19, 2021

1. Problem Specification

The goal of this lab was to first, implement the Selection Sort and Bubble Sort Algorithms. Then, reverse every sorting algorithm implemented up to this point, and then, sort the content of provided files to see how each sorting algorithm compares.

2. Program Design

This program requires two classes, Sorts and SortsDriver. The Sorts class contains the reverse implementation for all the sorting algorithms explored in previous labs, implementations of Bubble Sort, Selection Sort and their reverse.

The SortsDriver class contains a method to parse the text files containing the integers to be sorted. The files can be parsed using any delimiter. The driver class also calculates the time taken for each sorting algorithm to sort each of the given files. It also contains methods to compare Selection and Bubble Sort. And finally, it provides methods for testing the reversed algorithms, Selection, and Bubble Sort.

The implement the reverse of each algorithm the same tactics were used as those in labs #2 through #5. To reverse these algorithms however, the comparisons had to be changed. Instead of checking if a number is less than another number, greater than was used. And when previously greater than was used to compare two numbers, less than was used in its place. Once the comparisons were reversed the order of the sort was reversed with it.

The Section Sort algorithm was implemented with the following process.

a) The goal is to find the smallest element in a searching space; once found that element is then placed at the start of the searching space. Then, decrease the search space by one until the whole array has been sorted.
b) This is executed by first setting the search space to the whole array.
c) Then assume the first element in the space is the smallest.
d) Next compare the following element to the current smallest. If it is smaller, then that becomes the new current smallest. This process is continued until the end of the searching space has been searched.
e) Then the smallest element is swapped with the element at the start of the search space and the size of the space is decremented by one to no longer include the trailing element.
f) This process is repeated until the search space has a length of 1. At this point we know the final one element is in its proper place and the whole array has been sorted

The Bubble Sorting algorithm was implemented with the following process. The idea behind Bubble sort is, starting with the first and second elements of an array compare them. If they are not sorted among themselves, then swap them. And then repeat this comparison for the second and third element. Continue this process until reaching the n and n-1th elements. At this point the last index will always be sorted. Repeat this process again however on the next iteration only go until reaching the n-1 and n-2th elements.

Continue repeating until there is only the first and second elements to sort. Once those have been sorted among themselves, the whole array is sorted.

To calculate the run times for each required file, the same process was followed as that in labs #2 through #5

3. Testing

To test all sorting algorithms seven test cases were used to attempts and break the heap sort algorithm in different ways. The first contains both negative and positive numbers. The second contains multiple duplicates of the same number. The third contains only negative numbers. The forth contains two numbers reverse sorted. The fifth only one single element. The sixth is reverse sorted, and the last is an empty list.

Test Number	**Input**	**Expected Output**
#1	[10, 4, 6, 3, 2, 9, 16, 0, 3, -1]	[-1, 0, 2, 3, 3, 4, 6, 9, 10, 16]
#2	[4, 3, 3, 3, 2, -2]	[-2, 2, 3, 3, 3, 4]
#3	[-3, -103, - 5, -2, -10, -44, -31]	[-103, -44, -31, -10, -5, -3, -2]
#4	[4, 2]	[2, 4]
#5	[10]	[10]
#6	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
#7	[]	[]

4. Timing

The times to sort the provided files are as follows:

File Size	**Insertion Sort Time** (seconds)	**Merge Sort Time** (seconds)	**Heap Sort Time** (seconds)	**Quick Sort Time** (seconds)
100	0.000544071	0.000162125	0.00041604	0.000300169
1,000	0.040580034	0.001969099	0.006635904	0.004061222
5,000	1.049294949	0.012735128	0.047384977	0.025377035
10,000	4.237375975	0.0302279	0.105453014	0.05537796
50,000	112.1073821	0.159678936	0.631783962	0.327415943

File Size	**Min Max Sort Time** (seconds)	**Selection Sort Time** (seconds)	**Bubble Sort Time** (seconds)
100	0.631783962	0.000714064	0.001302958
1,000	0.051505804	0.049670935	0.129776001
5,000	1.34592104	1.829334021	3.224568844
10,000	5.916672945	7.869405985	13.84754992
50,000	137.2978091	258.2478631	325.589319

The times to sort Randomly Sorted, Reverse Sorted, and Sorted files for Selection and Bubble Sort are as follows:

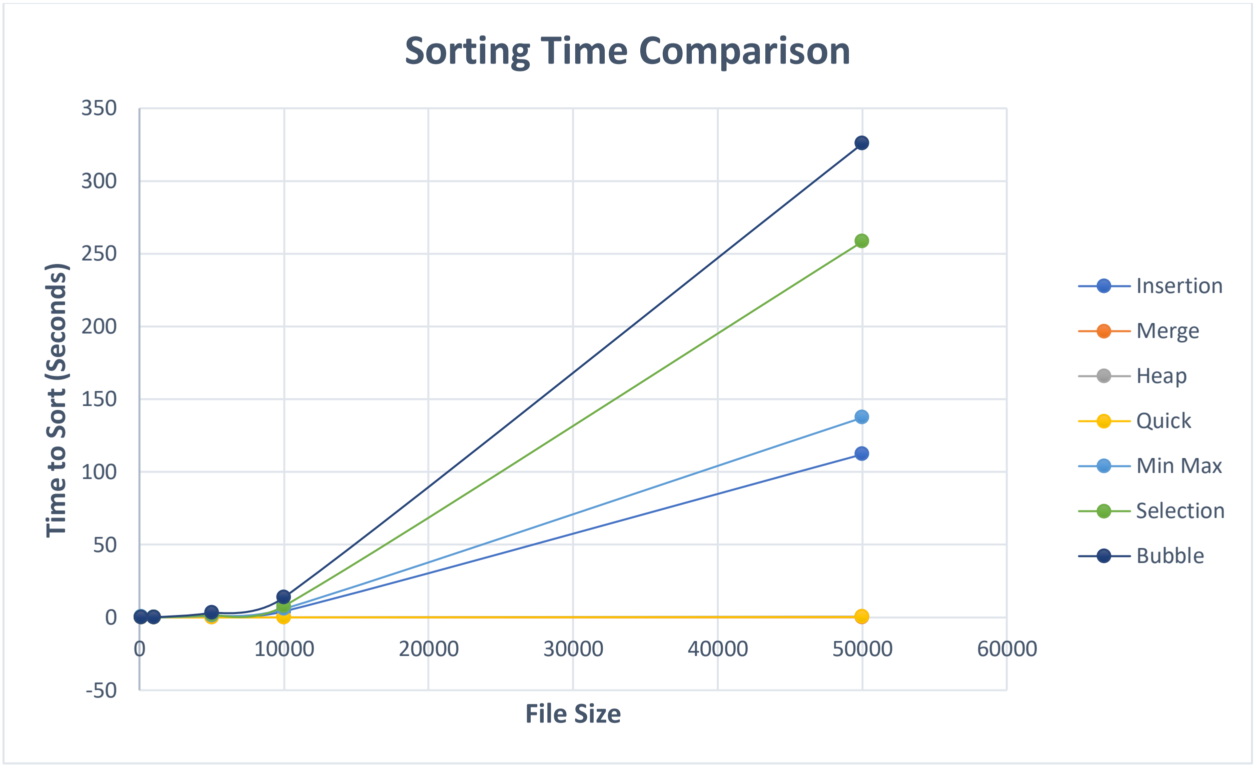
Presort Status	**Selection Sort** (seconds)	**Bubble Sort** (seconds)
Sorted	0.048203945	0.045151949
Reversed	0.053464174	0.213260889
Random	0.046848059	0.126723051

5. Analysis and Conclusions

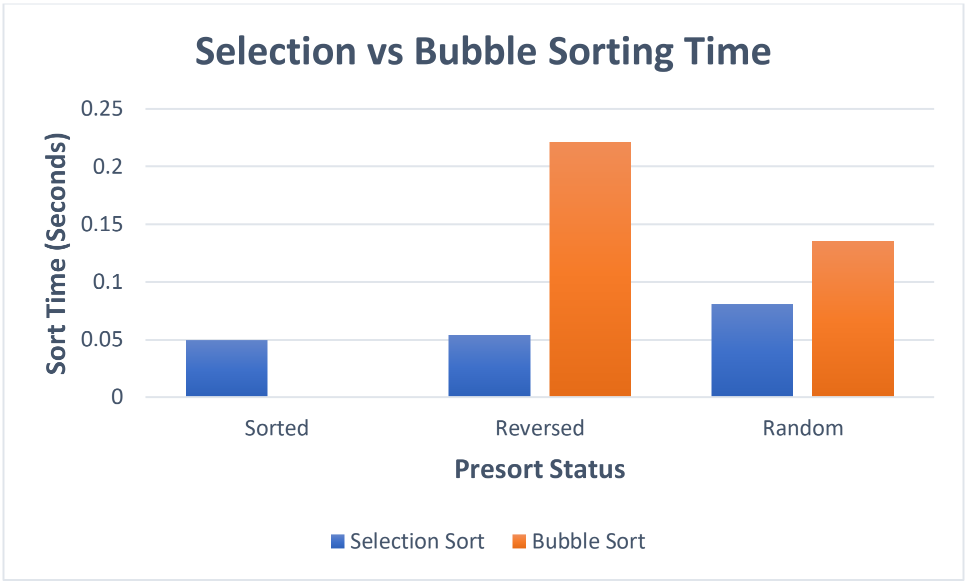
In conclusion Selection and Bubble Sort are slow sorting algorithms, when it comes to time complexity, both having an average and worse case of O(n^2). Bubble sort is slightly better by having a best case of O(n) when the array is almost entirely sorted, where Selection Sort remains at O(n^2). However, bubble sort falls behind again by having so many swaps.

To reverse a sorting method is a straightforward process. Just reverse the lines of code where the comparison takes place. This process does not change the time complexity in anyway.

The following table shows a chart comparison between each of the reversed sorting algorithms



The following chart compares Bubble Sort and Selection Sorts runtimes for files that are sorted, reverse sorted, and random.



As you can see in most cased Bubble Sort is worse than Selection Sort. However, when the data is presorted or almost sorted, Bubble Sort is far superior

6. Credits

- The implementations of the algorithms used in labs #2 through #6 were used in this lab report.
- The example lab report provided on canvas was used as a template for this lab report.