Grant Novota
CSCI 2270
March 24, 2017
Assignment 9

Assignment 9: Sorting Algorithms Analysis

Purpose:

The purpose of this experiment was to run multiple tests on three different sorting algorithms and analyze the results to create comparisons between them with the goal of being able to determine which is the most efficient. For the testing of these algorithms, each were used on arrays of differing sizes, containing randomly generated numbers. The largest arrays that were used for testing contained one million entries.

Procedure:

The algorithms that were chosen for this experiment were quick sort, shell sort, and select sort. To evaluate each algorithm's performance, I observed the number of loops that took place within each sorting algorithm when they were called to sort the arrays. I also observed the number of swaps that took place during the sorting of each array. The third method utilized for evaluating each algorithm's performance was implementing a timer into the code to count the number of milliseconds it took for each sort to finish.

The quick sort is a recursive algorithm that uses the method of dividing-and-conquering when performing the sorting and its average Big O notation is nlog(n). Quick sort first divides the array in half at a point called the "pivot" then rearranges the numbers so that all the numbers that are less than or equal to the pivot's number are placed to the left of the pivot and the numbers that are greater are placed to the right of the pivot. After this, the recursion of this algorithm begins as the function is called again and again to sort the sub-arrays.

Shell sort works by breaking down the array into multiple smaller arrays. Its average Big O notation is n(log(n))^2. These smaller arrays are then each sorting using the insertion sort method. It looks at one element in the sub-array and inserts it into its proper position. The algorithm goes through every single element and places it into its sorted position but since shell sort first divides up the array, it performs the sorting much, much faster.

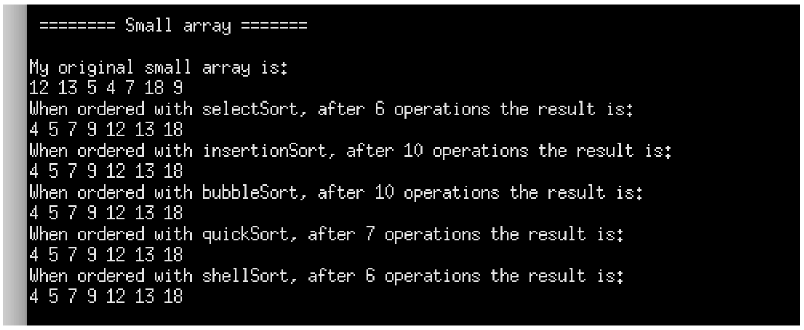
The select sort algorithm combines searching and sorting and has Big O notation n^2. It divides the array into two parts. The array that has been sorted and the array of numbers that have yet to be sorted. It finds the smallest element in the unsorted sub-array and swaps it with the leftmost unsorted element and moving the boundaries of the sub-array one element to the right.

Data:

To generate the data for the experiments. I conducted 3 tests, with array sizes of 7, 1000, and 1 million, where each array was filled with random numbers. I added loop counters to each of the three sorting algorithms to count how many times the functions were looped through. Each algorithm also had a counter to record the number of swaps that took place during the sorting of the arrays. To count the number of milliseconds that it took for each algorithm to sort the three arrays, a timer was added into the code that starts right before the sorting function is called and is stopped and recorded after the array has been sorted.

Results:

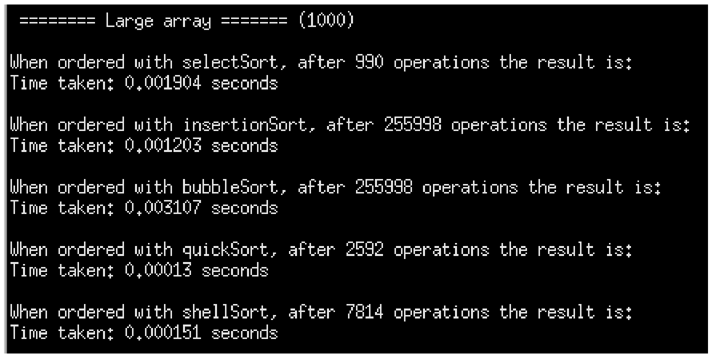
When tested with a small array, containing 7 numbers, all three of the sorting methods performed nearly the same. The quick sort finished after making 7 swaps; shell and select sort both made 6 swaps. When a small array was used, none of these sorting algorithms had a big advantage over the others.



```
 ======== Small array =======

My original small array is:
12 13 5 4 7 18 9
When ordered with selectSort, after 6 operations the result is:
4 5 7 9 12 13 18
When ordered with insertionSort, after 10 operations the result is:
4 5 7 9 12 13 18
When ordered with bubbleSort, after 10 operations the result is:
4 5 7 9 12 13 18
When ordered with quickSort, after 7 operations the result is:
4 5 7 9 12 13 18
When ordered with shellSort, after 6 operations the result is:
4 5 7 9 12 13 18
```

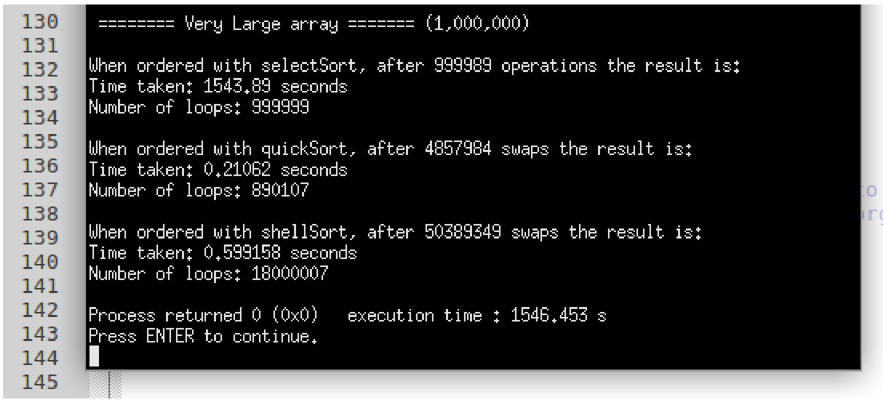
The sorting algorithms were then tested with a larger array, containing 1000 numbers. The quick sort performed the fastest at 0.00013 seconds, then the shell sort at 0.000151 seconds, and the select sort was the slowest, finishing in 0.001904 seconds. Select sort performed the least number of swaps at 990, quicksort made 2592, and shell made 7814. The select sort was slower but didn't have to perform as many swaps as the other two. Quick sort and shell sort were similar in their runtime but quick sort did not have to perform as many swaps.

Grant Novota
CSCI 2270
March 24, 2017
Assignment 9



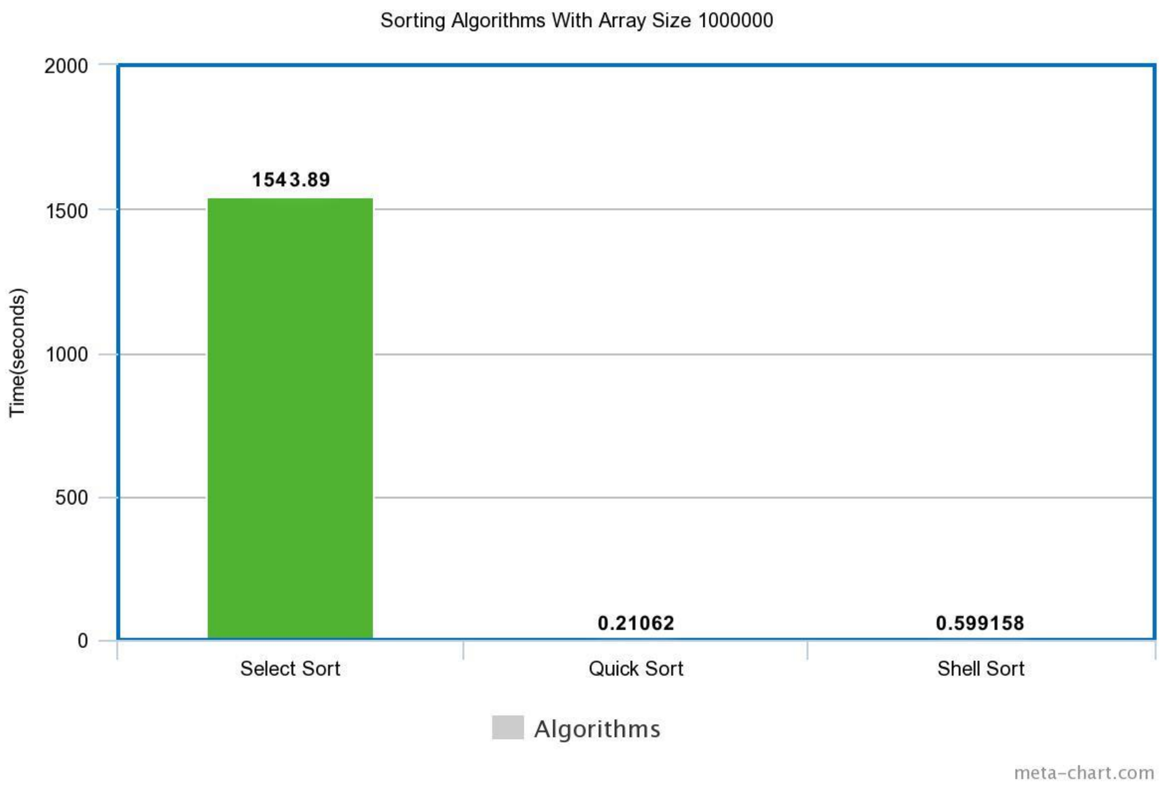
```
 ======== Large array ======= (1000)

When ordered with selectSort, after 990 operations the result is:
Time taken: 0.001904 seconds

When ordered with insertionSort, after 255998 operations the result is:
Time taken: 0.001203 seconds

When ordered with bubbleSort, after 255998 operations the result is:
Time taken: 0.003107 seconds

When ordered with quickSort, after 2592 operations the result is:
Time taken: 0.00013 seconds

When ordered with shellSort, after 7814 operations the result is:
Time taken: 0.000151 seconds
```

When the largest array of 1 million integers was used for testing, the differences in the efficiencies of the sorting algorithms became more apparent. The select sort was the slowest by far, taking about 25 minutes to complete whereas the quick sort and shell sort took 0.21 and 0.599 seconds respectively. The shell sort performed the most swaps, then quick sort, then select sort with the least number of swaps. The number of loops counted for each sorting methods were 999999 for select sort, 890107 for quick sort, and 18000007 for shell sort. This final test with the largest array shows that the quick sort is the most efficient for very large, unsorted arrays.



```
 ======== Very Large array ======= (1,000,000)

When ordered with selectSort, after 999989 operations the result is:
Time taken: 1543.89 seconds
Number of loops: 999999

When ordered with quickSort, after 4857984 swaps the result is:
Time taken: 0.21062 seconds
Number of loops: 890107

When ordered with shellSort, after 50389349 swaps the result is:
Time taken: 0.599158 seconds
Number of loops: 18000007

Process returned 0 (0x0)   execution time : 1546.453 s
Press ENTER to continue.
```

Grant Novota
CSCI 2270
March 24, 2017
Assignment 9


Sorting Algorithms With Array Size 1000000
Time(seconds)
Select Sort
Quick Sort
Shell Sort
Algorithms
meta-chart.com