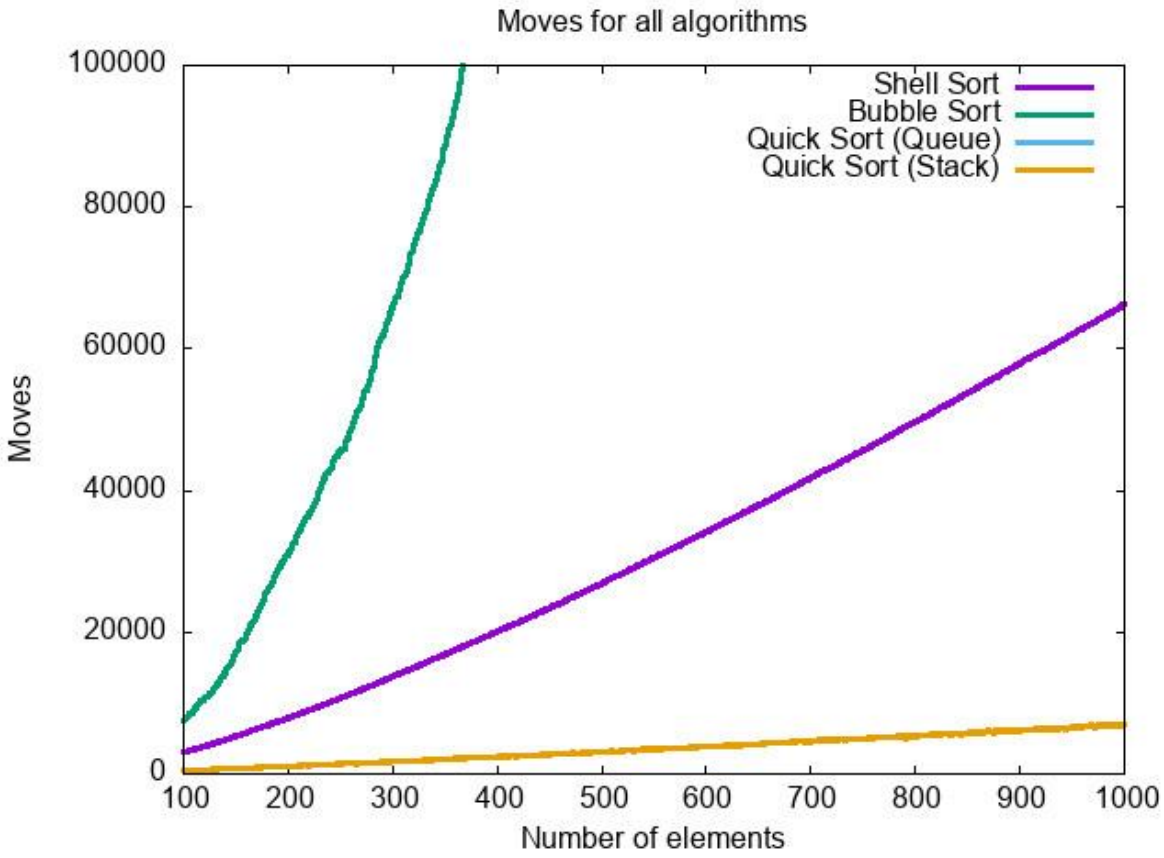
Rahul Vaidun
rvaidun@ucsc.edu
17 April 2021

CSE 13S Spring 2021
Assignment 3: Putting your affairs in order
Design Document

The program sorts an array using different sorting algorithms. The sorting algorithm can be specified via command line arguments. The program can sort with bubble sort, shell sort, and quick sort (with stacks and queues).

I. Time Complexity

To help calculate the time complexity for all the sorting algorithms I created a plot with the number of elements on the y-axis and the number of moves on the x-axis



With this plot we can clearly see the differences between the sorting algorithms.

A. Bubble Sort

It is immediately apparent that bubble sort has the worst time complexity of all the sorting algorithms. As the number of elements increases the number of moves increases exponentially. Bubble sort has a time complexity of $O(n^2)$. In some cases if the array is already almost sorted the time complexity can be $O(n$)

B. Shell Sort

The worst case time complexity for shell sort depends entirely on the sequence of gaps that we are using. The program utilizes a gap sequence known as the Pratt Sequence. The worst case complexity of Shell sort is $O(n^2)$However, shell sort on average performs at $O(n^{\frac{5}{3}})$ complexity if the gap sequence is optimal

C. Quick Sort

Both the stack and queue implementations of quicksort have the same number of moves and queues and therefore have the same time complexity. We can see from the graph plot that quick sort performs greatly outperforms the other two sorting algorithms since the moves rises at a much slower rate. The time complexity of quicksort is $O(n\ log(n))$

II. What I learned

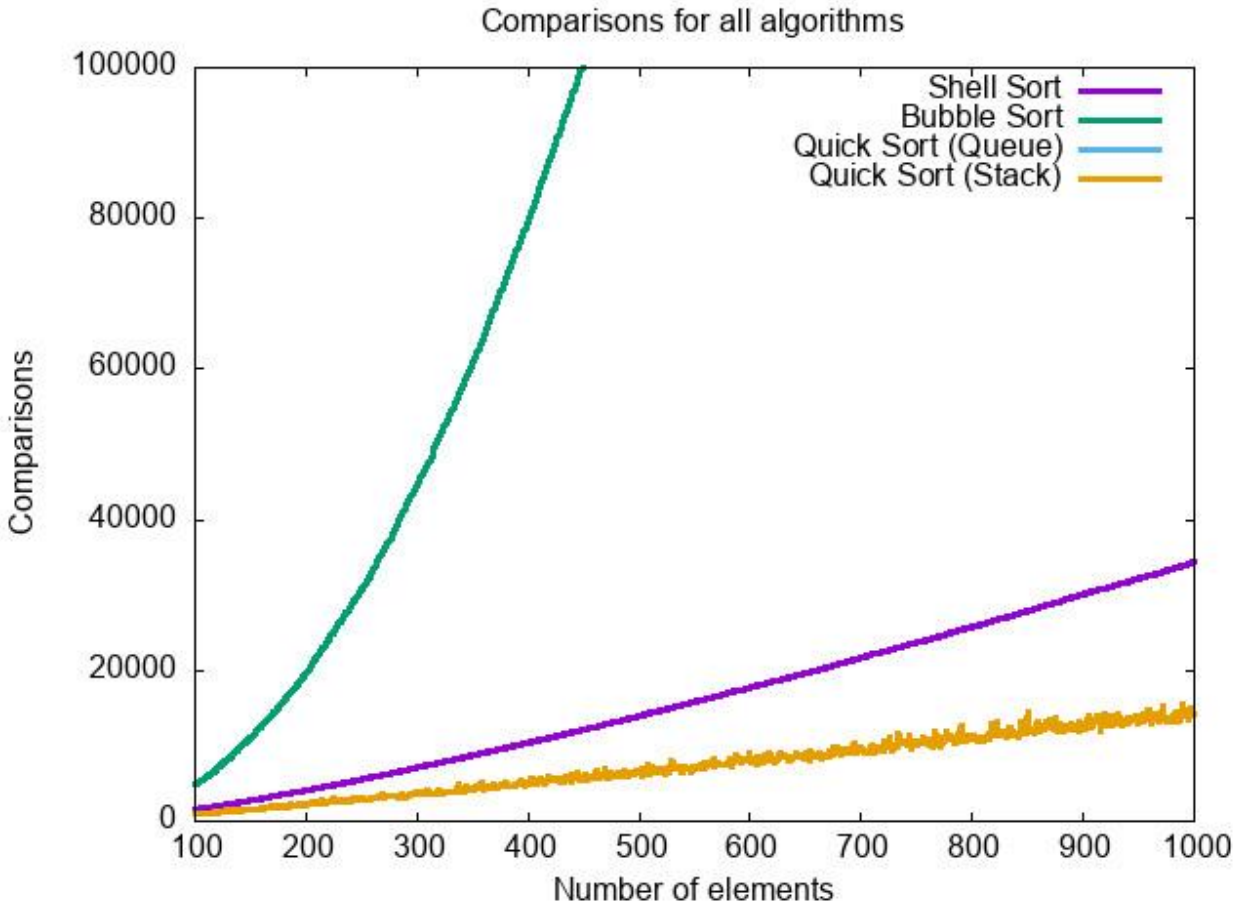
By implementing all these sorting algorithms in C I learned more about the different sorting algorithms. Before I started the assignment I didn't fully realize how bad bubble sort was and used it. Now, after plotting the amount of moves and comparisons each sort makes I see just how much bubble sort struggles compared to the rest of the sorts. I even tried to sort 100,000 elements. Both shell sort and quicksort were able to do this nearly instantly however bubble sort took nearly 30 seconds.

Other than learning about time complexity I also learned about the logic behind implementing different sorts. In shell sort I learned about the gap sequence and about how changing the gaps can greatly affect the time complexity of the sort. When I was implementing quicksort I learned how stacks and queues could be utilized effectively to implement quicksort.

III. Graphs and Experimentations

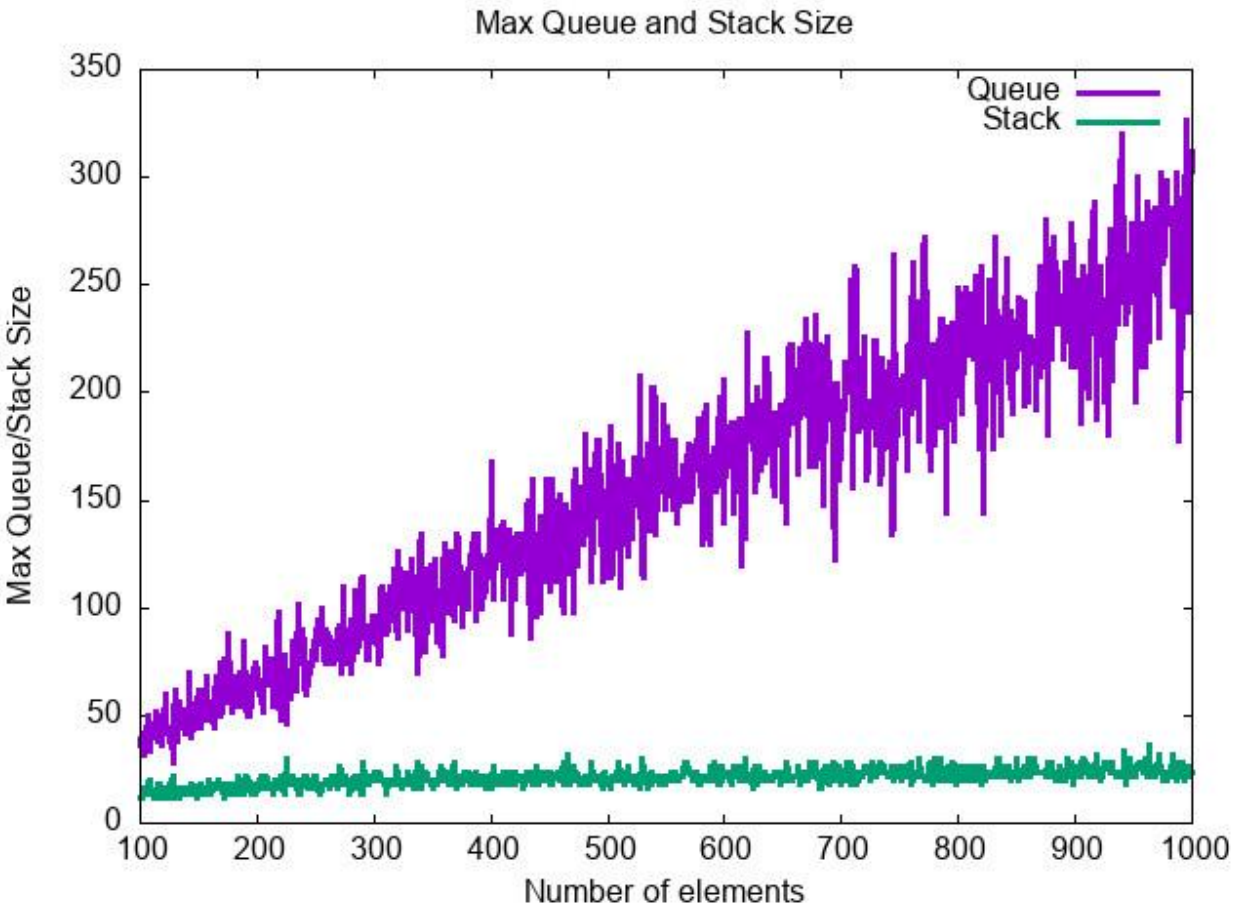
Once I implemented the sorting algorithms and fixed all the bugs I started experimenting with the results. I wrote a bash script which let me easily gather data from all the sorts with different lengths. I plotted the results with gnuplot and found interesting data. The moves plot in the time complexity was one of the plots I made.

I also made a plot of all the comparisons.



This plot is nearly identical to the moves plot where bubble sort is the worst, followed by shell sort, followed by quick sort.

Not only did I collect the moves and comparison data but I also collected data about the max queue size and max stack size.



The plot above shows that as the number of elements increased the max queue size increased. However it was interesting to see that the max stack size remained constant even as the number of elements increased.

Next I collected data about how the sorts performed on average. I performed 1000 different sorts each with a unique seed with 1000 elements. I then calculated the average for the number of moves and sorts. Bubble sort averaged 749883 moves and 498614 compares for an array of 100 elements. Shell sort averaged 66190 moves and 34366 compares. Quick sort averaged 6963 moves and 14421 compares. The average numbers didn't surprise me and looked consistent with the move and compare plots. I also calculated the average for the max stack size and the max queue size which were 22 and 265 respectively.