Assignment 3
Sorting: Putting your affairs in order

WRITEUP

Zack Traczyk
CSE13S - Spring 2021

Due: April 25th at 11:59 pm

1 Time Complexity

In this assignment, four sorting algorithms were implemented: Bubble Sort, Shell Sort, Quicksort with a stack, and Quicksort with a queue. Each algorithm has a different balance of memory usage, comparisons, and swamps resulting in different runtime and Big-O analyses. Since the length of the array will be discussed often, this value will be referred to as n.

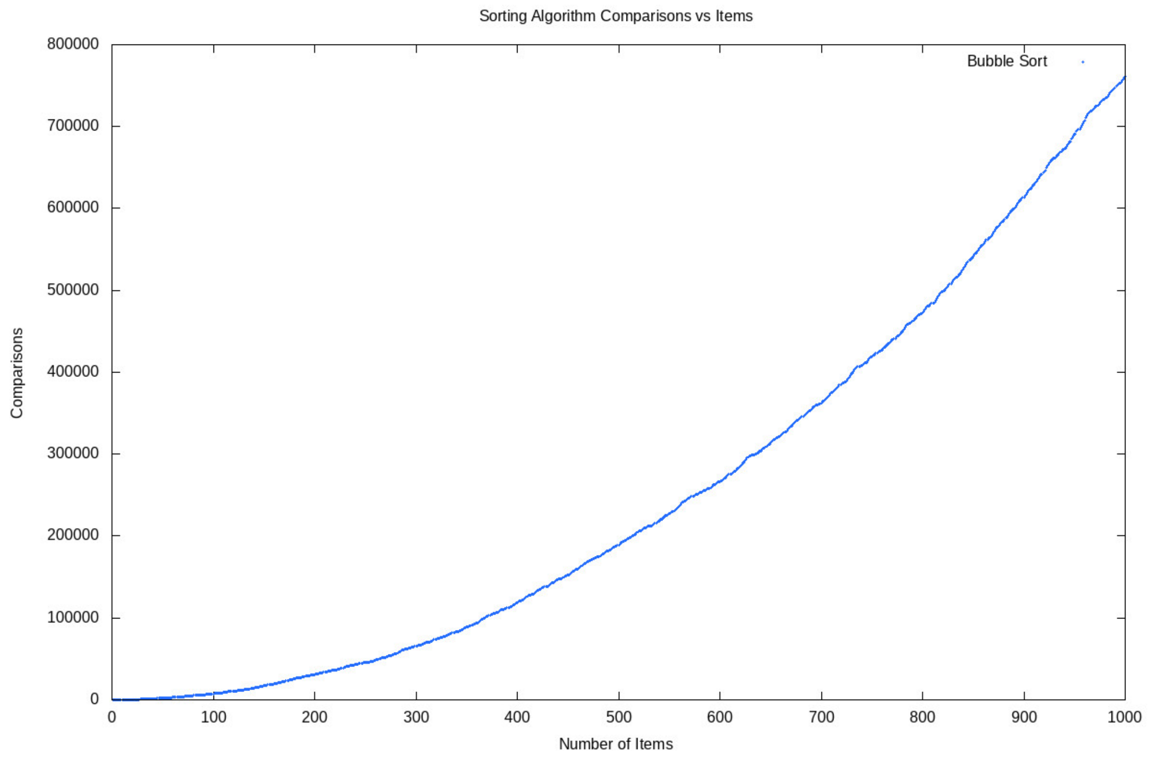
1.1 Bubble Sort

Bubble Sort is relatively naive sort, simply switching bordering values until the array is sorted. Unsurprisingly, this approach is not efficient for larger arrays. Bubble Sort had the worst time complexity out of all the sorts implemented. Figure 1 displays the relationship between the amount of comparisons the algorithm performs for n values. Bubble Sort's comparisons increase exponentially as n resulting in a time complexity of $O(n^2)$.

A reversed array requires the most amount of comparisons in bubble sort. Since the items bubble to the top,

The corresponding C value for bubble sort is

Figure 1: Bubble Sort

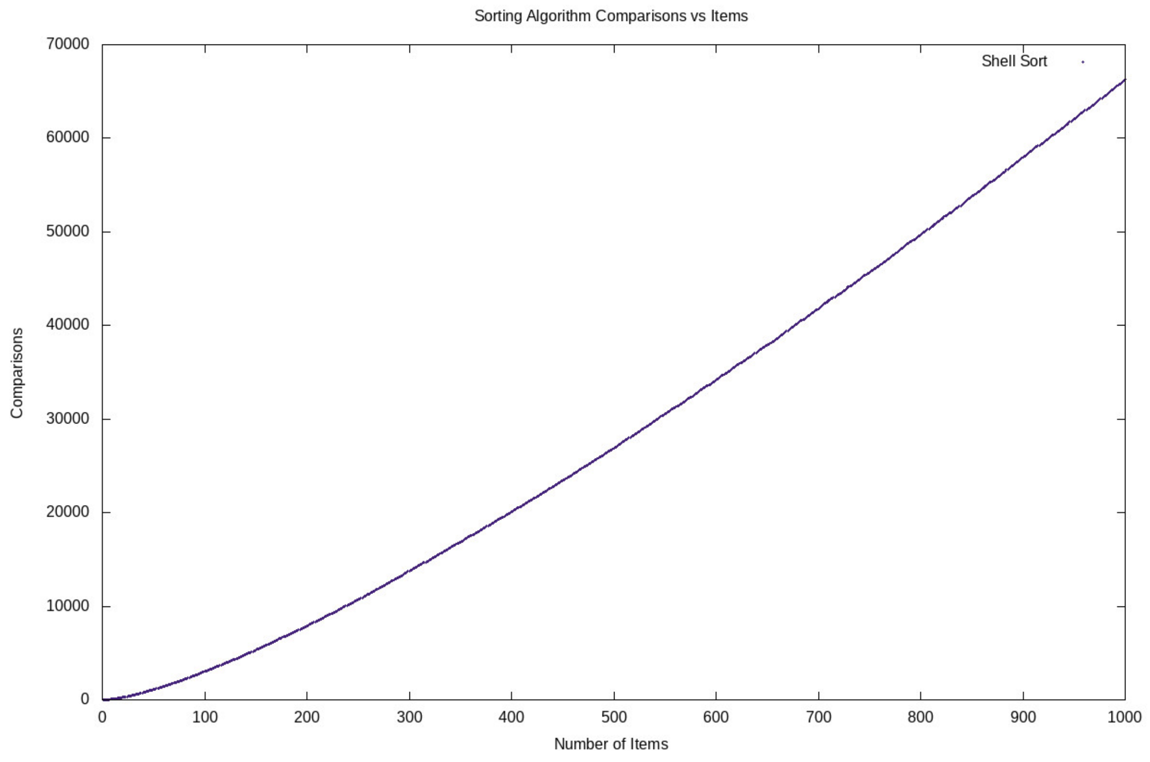


1.2 Shell Sort

Shell sort preformed much better than Bubble Sort. Just by looking at figure 2 the curve is obviously less slopped than figure 1. A generic shell sort algorithm's worst case scenario is $O(n^2)$. However this is because the gap size is 1, equivalent to bubble sort. This implementation uses a Pratt sequence of gaps. This results in a time complexity of $O(nlogn)$.

Reverse

Figure 2: Shell Sort



1.3 Quicksort

Quick sort preformed the best by far. Figure 3. Figure 4.

Figure 3: Quicksort (Stack)

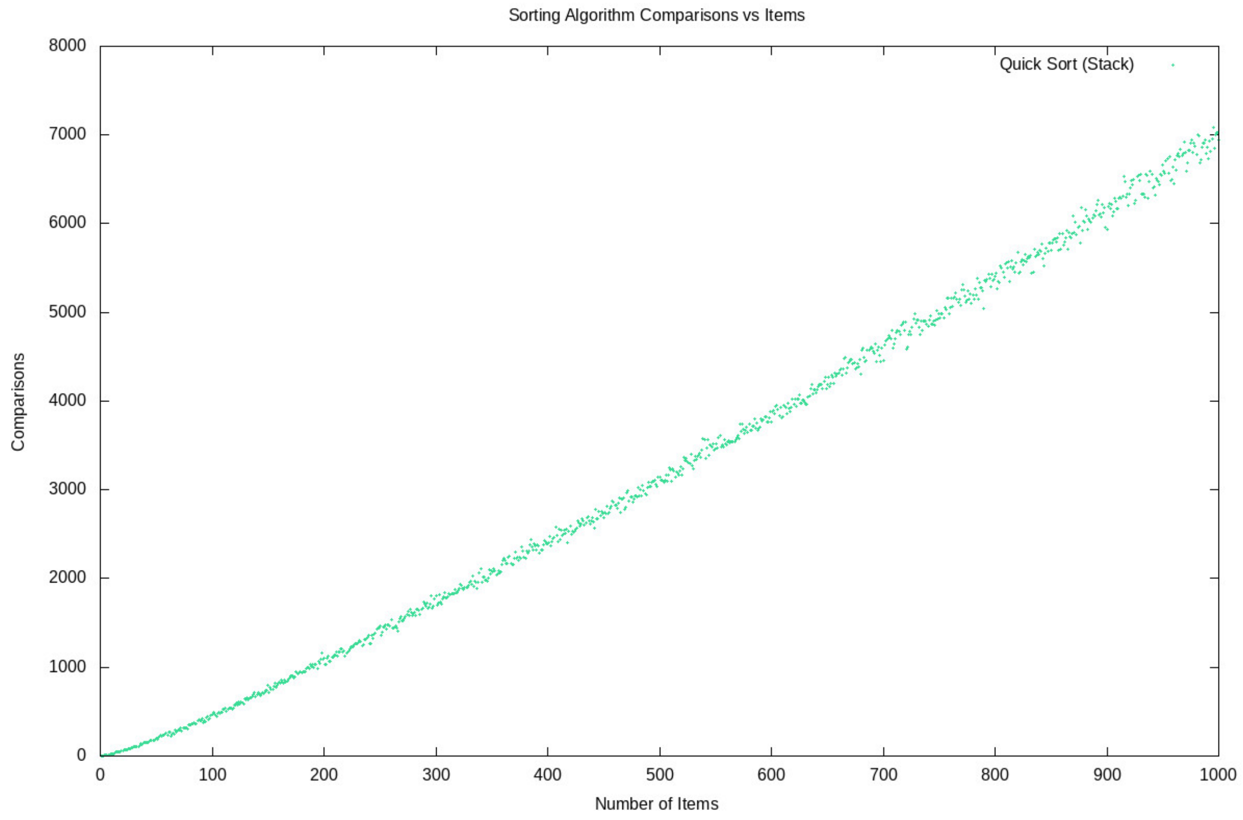
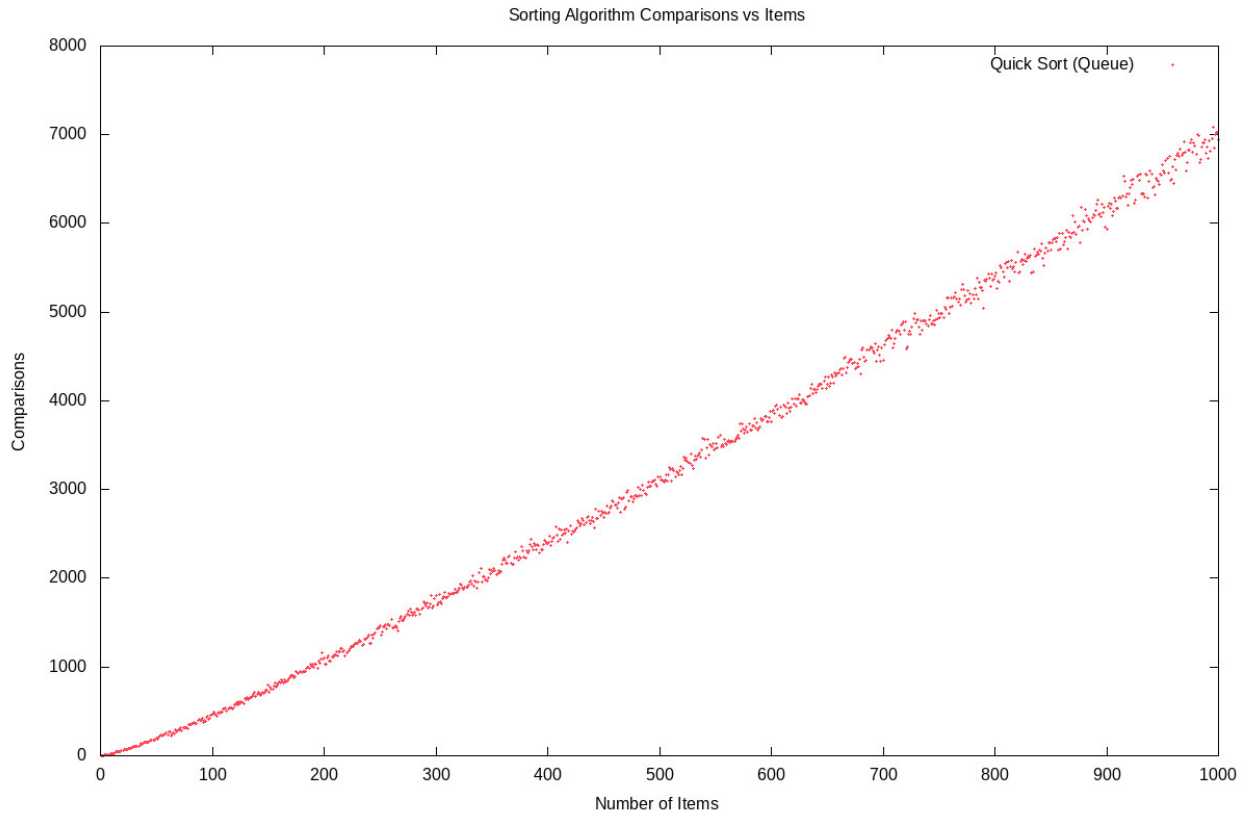


Figure 4: Quicksort (Queue)

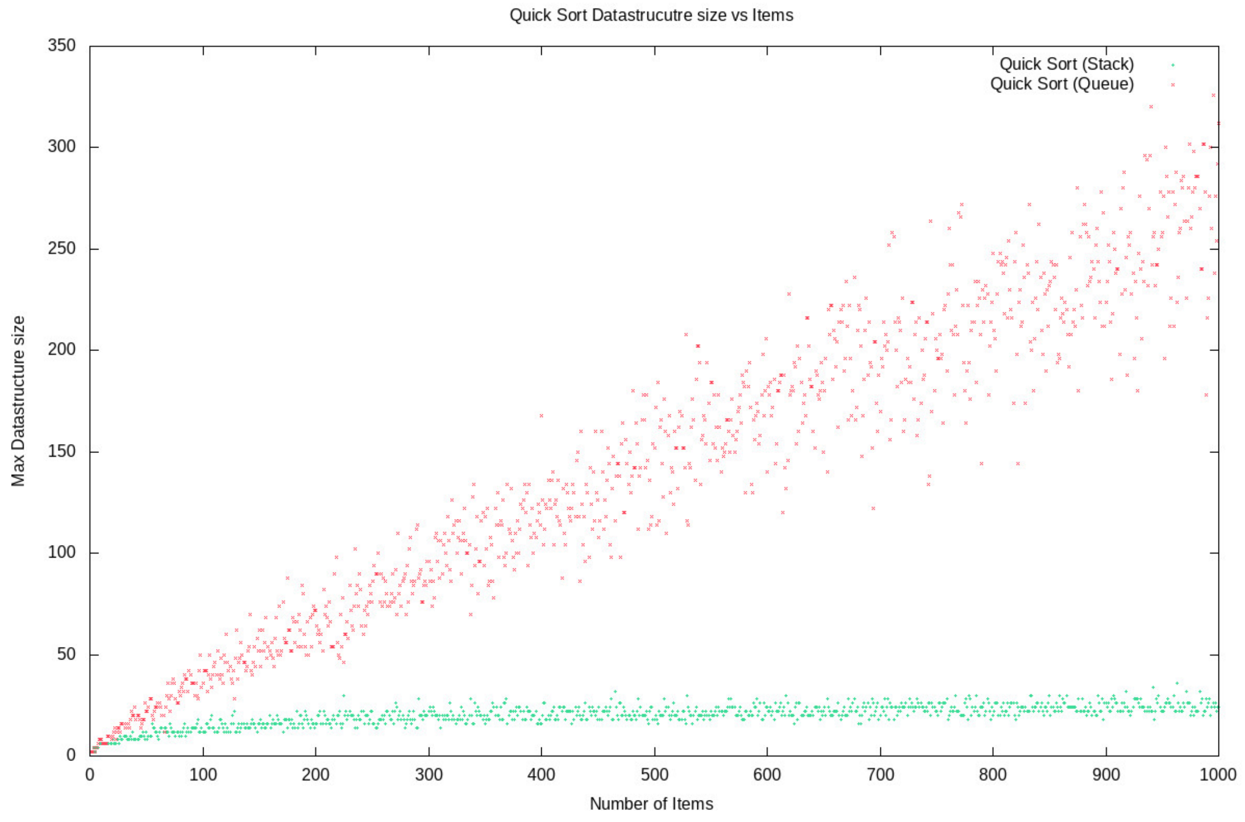


[h]

There is a linear relationship between the size of the array that is being sorted and the size of the resulting queue/ stack. This is because the queue/ stack is initialized as the $2n$.

When looking at size figure 5.

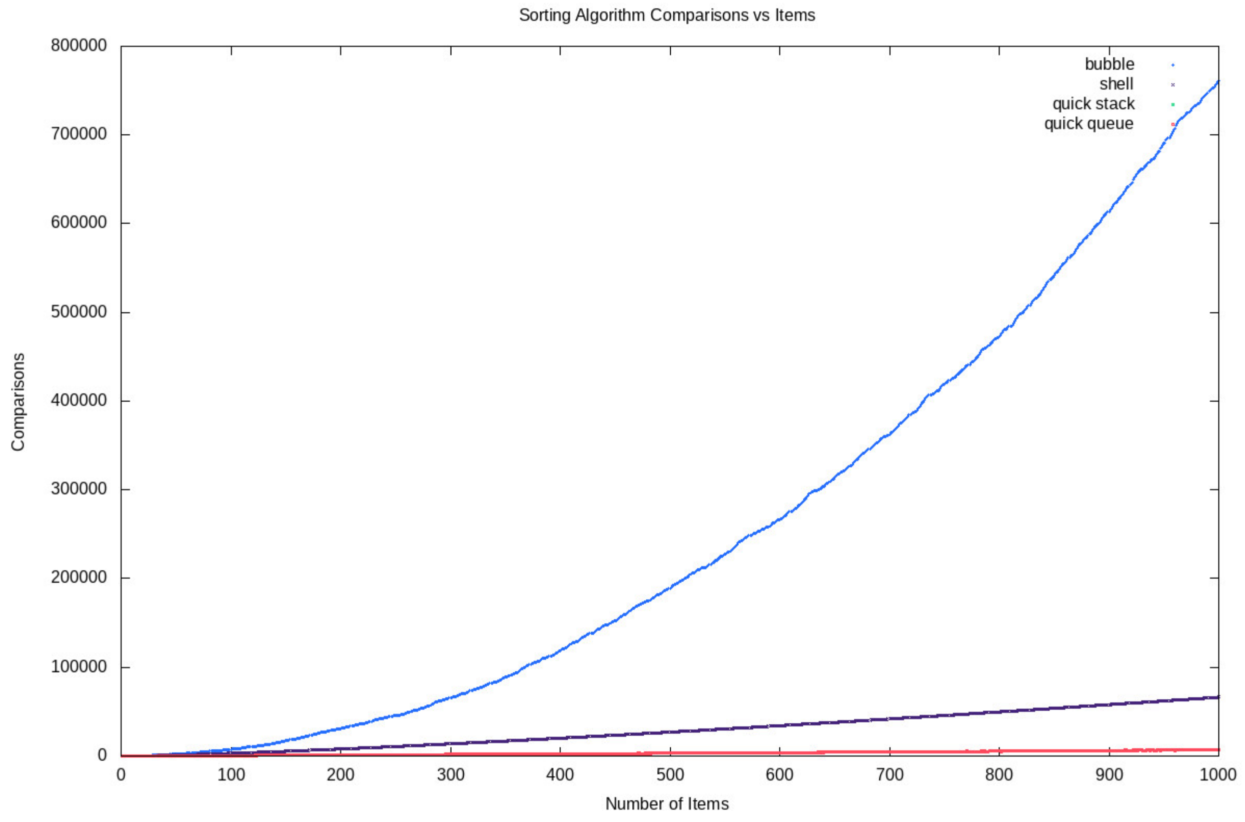
Figure 5: Quicksort Data Structure Size Comparison



1.4 Overall

As I stated, overall figure 6.

Figure 6: Overall Comparison



2 What I Learned

Picking the right algorithm is crucial for accomplishing fast code. Furthermore, right does not necessarily mean smallest Big-O or even the fastest. It is about considering memory usage, and size of array.