Assignment 3 - WRITEUP.pdf

Victor Nguyen

January 26, 2022

1 Introduction:

In this writeup, I will be focusing on comparing and contrasting the number of moves/comparisons of certain sorting algorithms. Namely, Insertion sort, Heap sort, Quick sort, and Batcher sort. I also want to look into specific behaviors if we were to use specific arrays over randomized ones.

2 Plots and Analysis:

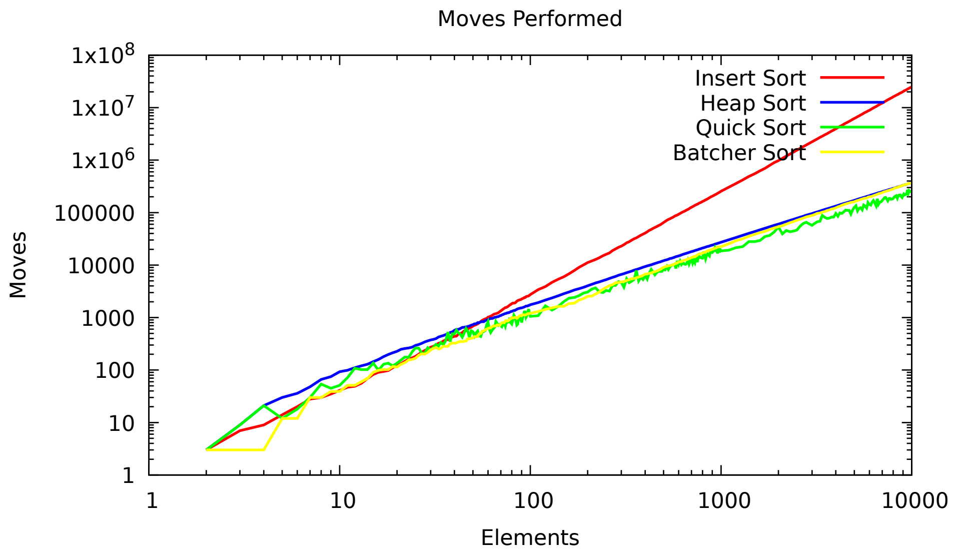

Figure 1: Moves Performed on Randomized Array

In this graph, I wanted to see how many moves each sorting algorithm uses for each n (where n is the length of the array). Surprisingly to me, when the number of elements in the array are below fifty, batcher sort does pretty well compared to the other 3 sorts. This could be due to the nature of the algorithm itself where batcher sort doesn't need to make too many moves for smaller array sizes compared to other algorithms. Of course, when we get to bigger array sizes the algorithms sort of "fall" into this linear growth that makes the algorithm somewhat predictable. There is a notable algorithm that doesn't seem to have a trend though, namely the quick sorting algorithm. It's very "jittery", sometimes doing worse than all of the other algorithms. This could be happening due to the nature of the quick sorting algorithm of divide and conquer. I was curious to see if this was due to the starting seed, so I tried using a different seed.

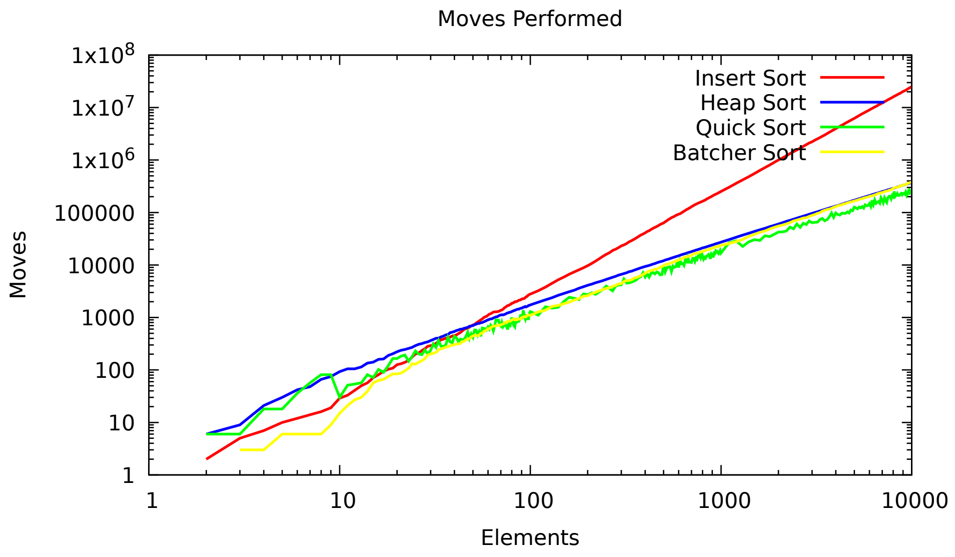

Figure 2: Moves Performed on the random seed 6969

This graph uses a different seed compared to the last graph. This seed that I'm referring to is important because in computers, true random numbers doesn't exist, we can only mimic them through pseudo random number generators. Defining the seed will produce a specific random sequence to fill the array with. For this graph I chose the seed to be 6969. Sadly, the randomness of the numbers doesn't seem to have an affect on the behavior of the sorting algorithms. This graph follows a similar trend as the previous graph. This led me to come up with another hypothesis, which was to test if the order of the numbers mattered. What I mean by this is to see if an array that contained some already ordered numbers would affect the number of moves/comparisons that each algorithm would do. To test this, I will feed each sorting algorithm a sorted array of pseudo random numbers (using quick sort to initially sort the numbers). The result of this can be seen below.

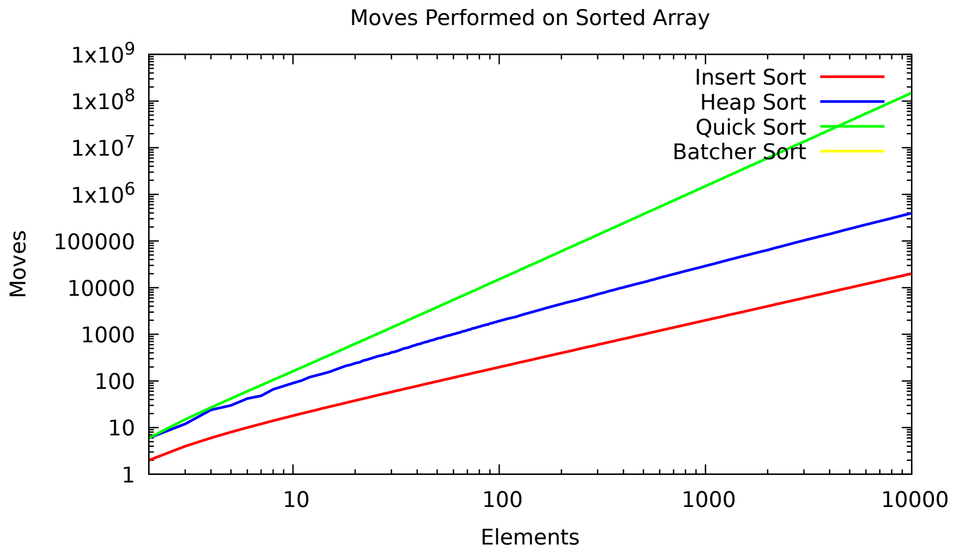

Figure 3: Moves Performed on the Sorted Array

Very interestingly, the quick sort algorithm actually flipped from being the fastest to being the slowest of the four. This tells us some things, for one, quick sort performs worse under conditions if the array is semi sorted. This means that we should rely on another sorting algorithm when it comes to these situations. Another thing to notice about quick sort is that the lines are relatively straight compared to when the array was in a random order. I would assume this is due to the array being already semi organized. This can also be observed for the rest of the algorithms. On the other hand, insertion sort performs very well, which tells us that insertion sort performs well under semi sorted arrays. Heap sort seems like it didn't change much at all, performing somewhere in between. One really weird thing about this graph though, is that batcher sort doesn't even make any moves! What does this mean? Probably means that batcher sort doesn't need to make any moves, which leads me to also check if this is the same case for it's comparisons.

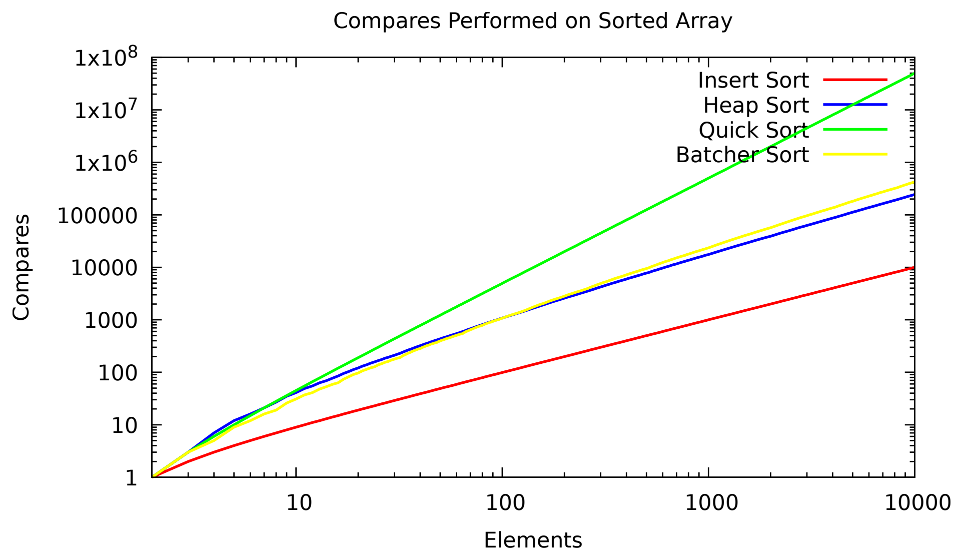

Figure 4: Compares Performed on the Sorted Array

As I expected, batcher sort only made comparisons. It still does better compared to quick sort, but in the end, insertion sort beats everyone. Though, the application of this doesn't seem useful. Insertion sort beats the other sorting algorithms, but whats the point of a sorting algorithm when the array is already sorted? The only logical explanation that I could think of is when the array is ordered and if you just need to insert a new value into the array but in the correct order.

3 Conclusion/What I learned:

1. Each sorting algorithm has its pros and cons. Some algorithms performances can change under certain conditions. Quick and Insertion sort fall under this category. Heap and Batcher sort seem to perform the same regardless of the condition of the array.
2. Consider the state of an array for some project and think whether or not it makes sense to use a specific sorting algorithm for that array.
3. There isn't a tool that can perform the best for every situation. We should test the algorithm against other ones to see if they'll perform better.
4. No matter how good or fast our pc components are, if we're using the wrong sorting algorithm for the job, it'll never beat the optimal sorting algorithm for that job (whatever that algorithm is).