Matthew Pacual-Mead
Prof. German
12/18/20
CPSC-350

Assignment 6: Sorting Algorithms

Time Differences

Prior to timing each of the sorting algorithms, I had a general idea of what I thought the results would be. Since the average case for Bubble sort and Selection sort is quadratic (O(n^2)), I expected that these two algorithms would be the slowest. I also expected Insertion sort to be similar to these times as well, as its worst case time complexity is quadratic as well. Since my test data was randomized, I did not expect the input data to be partially sorted. As a result, I assumed that Insertion sort would not be able to achieve its average case of a linear time complexity. I thought that my best performing algorithm would be Merge Sort and Quicksort, as these time complexities of the algorithms for their average cases were O(n*log(n)). After running my program, the time for each algorithm was as follows:

- Bubble Sort: 73.3756 seconds
- Selection Sort: 18.7812 seconds
- Insertion Sort: 10.8616 seconds
- Quick Sort: 17.1176 seconds
- Merge Sort: 0.0495681 seconds

Analysis of Time Differences

From the five sorting algorithms, the best performing algorithm was Merge sort, which was able to sort a list of 100,000 double numbers in less than a second. The next shortest time among the five sorting algorithms was Insertion sort, sorting the data in only 11 seconds. This was a bit surprising as I had expected Quicksort to be one of the top performers for sorting the data due to its linearithmic runtime. However, this could have been due to Insertion Sort ability to perform at linear time if our data was partially sorted. As I had predicted the algorithms that took the longest to sort were Bubble Sort (73.38 seconds) and Selection sort(18.7812). Due to their quadratic time complexities, it was easy to tell that these algorithms would take the longest to sort the data. But what was surprising was that Selection sort was almost able to sort the list as fast as Quicksort. This was extremely odd considering that the time complexity for Quicksort is O(n*log(n)) compared to Selection sorts time complexity of O(n^2)).

Tradeoffs of the Sorting Algorithms

As we know, each of these sorting algorithms have their own benefits and drawbacks. For example, the Bubble sort algorithm was the easiest sorting algorithm to implement of the five. It both took the least amount of time to code for and has a very straightforward concept. But as we learned from its performance, this is the slowest sorting algorithm. As a result, I would only recommend using this Bubble sort if you are unable to access the resources needed to implement the other sorting algorithms. Selection sort was another algorithm that was fairly easy to implement but was still somewhat slow. Selection sort was able to outperform Bubble sort, as this algorithm requires far fewer swaps when sorting. However, like Bubble sort, Selection sort is a viable option for sorting large amounts of data, and I would recommend opting for another sorting algorithm. The benefits of Insertion sort is that the average case for this algorithm is linear if the data is partially sorted. In cases like these, this Insertion has the fastest time complexity of all five sorting algorithms. However, its drawback is that its worst case run time is quadratic which is fairly slow. The benefits of Quick sort is that it is able to sort data both quickly and efficiently. Quick sort has a run time of O(n*log(n)) and is able to maintain a space complexity of only O(1). The biggest drawback with Quick sort is that its performance can degrade to a time complexity of O(n^2) when the algorithm partitions unequally sized parts. Cases like these are extremely rare, but can happen. The advantages of Merge sort is that both its average case and worst case runtimes are O(n*log(n)), which is very good. However, the key disadvantage of this algorithm is that this algorithm requires O(N) additional space for the temporary array of merged elements. As a result, this algorithm would not be ideal in situations where memory use is limited.

Impact of Programming Language

Using C++ was very helpful for this assignment because it contains a lot of features that were crucial to implementing these sorting algorithms. For example, pointers in C++ allow users to dynamically allocate memory which is essential for creating arrays containing random numbers. Without pointers, it would be much harder to create this program because arrays require a fixed size at runtime.

Shortcoming of Empirical Analysis
Overall, this assignment was very fun because it allowed me to carry out my own empirical analysis for the first time. Empirical analysis is a very obvious approach to testing algorithms, but it is not always the optimal solution. For example, Empirical Analysis can be very cost efficient because it requires the implementation of the algorithms which costs time and money. In relation to this assignment, running the code with the large input of data took fairly long. Furthermore, it is very dependent on many variables including the platform/hardware and the compiler/linkers.