Project 2

William Crutchfield

CMSC 451

10/11/18

Table of Contents

Insertion Sort

The sorting algorithm I chose for Project 1 was Insertion Sort. Insertion sort is a sorting algorithm that builds the final sorted array one element at a time. Insertion sort is efficient for small data sets, but quickly becomes inefficient as the data size grows.

Pseudocode

This section contains the pseudocode for both the iterative and recursive versions of the insertion sort algorithm.

Iterative

The pseudocode for the iterative version of insertion sort is:

```
for i = 1 to list.length do
    x = list[i]
    for j = i-1 to j >= 0 and list[j] > x do
        list[j+1] = list[j]
    end for
    list[j+1] = x
end for
```

Recursive

The pseudocode for the recursive version of insertion sort is:

```
if n > 0 do
    recursive(list, n-1)
    x = list[n]
    for j = n-1 to j >= 0 and list[j] > x do
        list[j+1] = list[j]
    end for
    list[j+1] = x
end if
```

Big-Θ Analysis

This section contains the Big-Θ Analysis for both the iterative and recursive versions of the insertion sort algorithm.

Iterative

When looking at the pseudocode for the iterative version of the insertion sort algorithm, there are two things that stand out. That is, the two `for` loops. These `for` loops determine the Big-Θ for our iterative algorithm. From this alone, we can conclude that the iterative version of our insertion sort algorithm is $\Theta(n^2)$.

Recursive

When looking at the pseudocode for the recursive version of the insertion sort algorithm, there are two things that stand out. That is, the `recursive` call and the `for` loop. These are both call n times, giving us a Big-Θ of $\Theta(n^2)$.

JVM Warm-up

When it came to the issue of benchmarking the program, I needed to ensure that the JVM was properly warmed up before the benchmarking began. I found that the best way to do this was to run “dummy” instances of my program. I ran 250 instances of my program to properly warm-up my code. I found that 250 was a good balance of warm-up time vs efficiency. From the console output, you can see that the execution time of the program averages out to around 0.069 seconds by instance 150. But, before warm-up, the execution time is upwards of 0.1 seconds. This is more than a 30% difference in execution time, showcasing the importance of the JVM Warm-up.

Critical Operations

For the critical operations of the insertion sort algorithm, I felt it was best to count the number of times elements were shifted. For example, if an array `int[] arr = new int[]{0,2,3,1}` is sorted by the insertion sort algorithm `arr = [0,1,2,3]`. The element equal to `1` would need to be shifted 2 times. Giving us a total of 2 critical operations. Due to how the insertion sort algorithm sorts data, the number of shifts would be the best operation to count as the critical operations.

Data Analysis

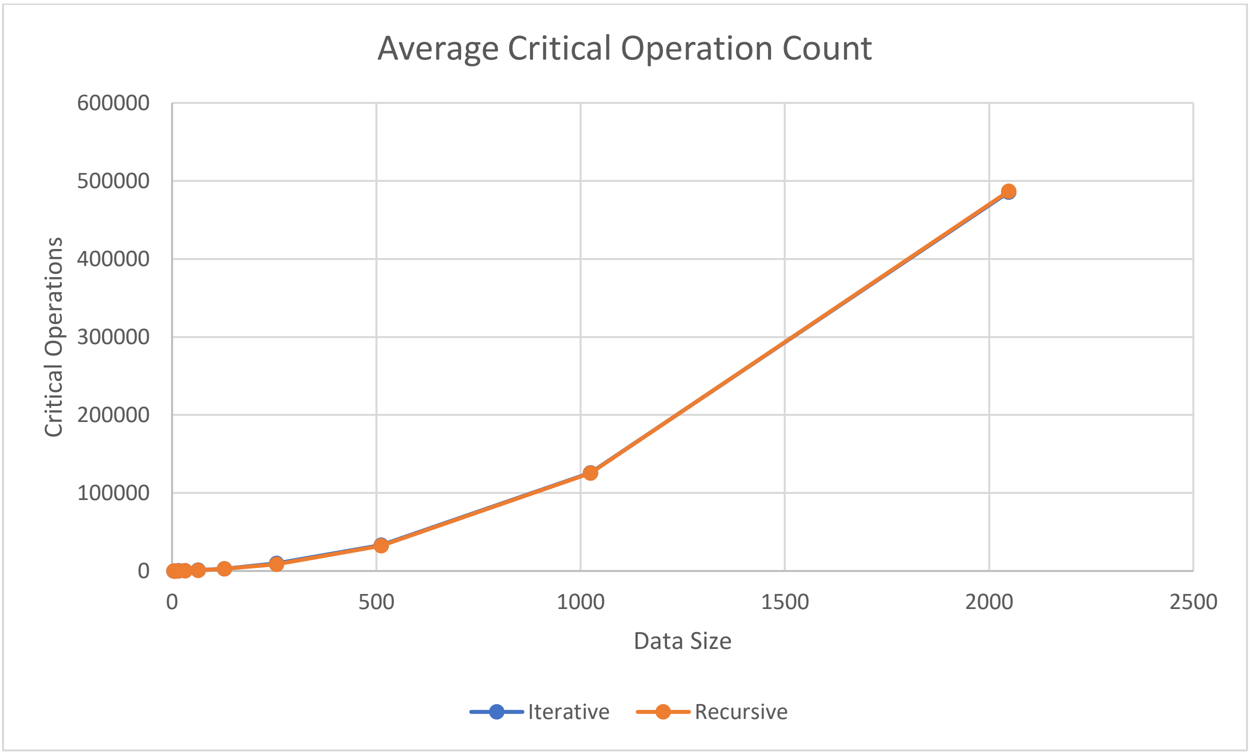
This section contains the analysis of the insertion sort algorithm benchmarking results.

Critical Operations & Execution Times

This section contains the graphs pertaining to the critical operations and execution times of both insertion sort algorithms.

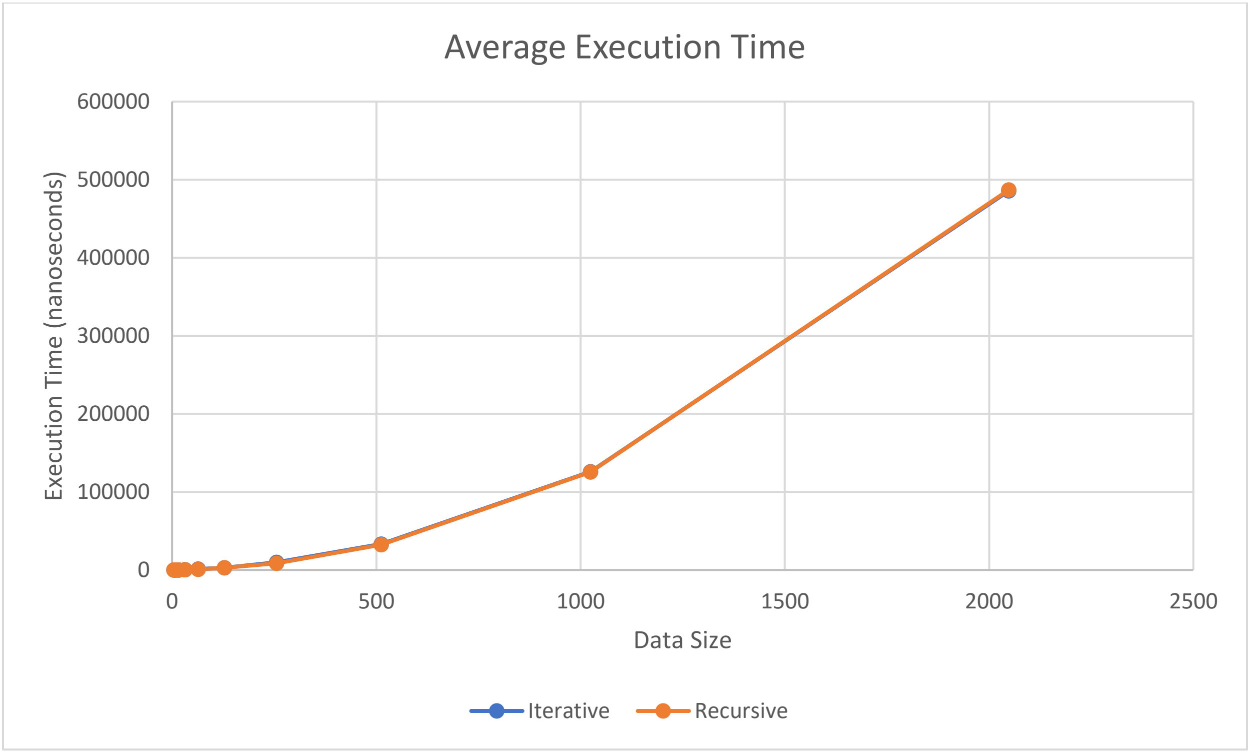
Critical Operations

This graph shows how the Average Critical Operations is the exact same for both the iterative and recursive versions of the insertion sort algorithm.



Execution Times

This graph shows how the Average Execution Time is the very similar for both the iterative and recursive versions of the insertion sort algorithm. However, there is slight variation between the two versions.



Performance

From the data collected on the two versions of the insertion sort algorithm, it is hard to say if one is more efficient than another in terms of critical operation count and execution times. Both the iterative and recursive versions of the algorithm are comparable in terms of performance. The only difference between the two versions of the algorithm would be in terms of memory. The iterative version of the algorithm would be more efficient in this aspect.

Critical Operations vs Execution Time

Looking at the graphs from the “Critical Operations & Execution Times” section, we can determine that the critical operation count and the execution time grow at the same rate. This is true for both versions of the insertion sort algorithm, iterative and recursive.

Coefficient of Variance

The coefficient of variance is a way of understanding the relative standard deviation among a data set. In terms of the insertion sort algorithm, the coefficient of variance shrunk as the data size grew. From the benchmarked data, the insertion sort algorithm had rather stable results. However, very small data sets were very unstable in terms of deviation. This is especially apparent in the execution time data. On data sets of 4 and 8, the coefficient of variance for execution time was 342.56% and 154.30% respectively. While the coefficient of variance for critical operation count was 53.57% and 28.76%. As we can see, the execution time was much more unstable compared to the operation count. However, when the data set was greater than 64, the coefficient of variance was under 10% on average for both execution time and operation count. As the data size grew, the coefficient of variance shrunk. We got coefficient of variance values as low as 1.8% for execution time and 1.56% for operation count. With these results, we can determine that these algorithms are very unstable at low data sets, and very stable at larger data sets.

Big-Θ Analysis

The results from the benchmarking line up with the Big-Θ Analysis of the two versions of the insertion sort algorithm. Both versions of the algorithm had a Big-Θ of $\Theta(n^2)$. It shows as both versions of the algorithm grow at the same rate in terms of critical operation count and execution time.

Conclusion

In conclusion, insertion sort is a very efficient algorithm for sorting small data sets, iteratively and recursively. The iterative and recursive versions of the insertion sort algorithm performed very similarly to each other, but the iterative version is more efficient in terms of memory. However, with small data sets, the performance of the algorithm has a high coefficient of variance. This means that execution times and operation counts can vary up to 342% and 154% respectively. But, with data sizes so small, the deviations shouldn’t dissuade from using insertion sort for these small data sets. Small data sets is where the insertion sort algorithm shines, and is a great sorting algorithm when data sizes are expected to *stay* small.