Design and Analysis of Insertion, Merge, and Quick sort

David Shin

California Polytechnic University, Pomona

Computer Science 331

Professor Perez

Abstract

Three different types of algorithms that are part of the divide and conquer strategy were carefully examined. Merge sort was experimentally confirmed to have faster runtime than insertion sort as the number of integers to be sorted grew larger. Conversely, insertion sort was found to perform better than merge sort when sorting relatively small data sets. In order to combine the efficiencies of both algorithms, MergeInsertionSort was implemented. The intersection of both algorithms was experimentally found by taking the average of three runtimes of both insertion and merge sort, then plotted using Excel and finding the intersection of logarithmic trend lines. MergeInsertionSort would first start sorting data using insertion sort, then when the data set became greater than 28 elements, which was experimentally found to be the intersection of both algorithms, sorting the rest of the data set would be passed on to be completed by merge sort. Quick sort was found to be the most efficient compared to all the algorithms.

*Keywords:* merge sort, insertion sort, quick sort, intersection, complexity, average case, worst case, every case, best case, runtime, divide and conquer

Algorithm Discussion

**Insertion Sort:**

Worst Case:

Average Case:

Best Case:

Auxiliary Space:

**Merge Sort:**

Every Case:

Now, we have established the recurrence relation:

After solving the recurrence relation, we get:

Auxiliary Space:

**Quick Sort:** Quick Sort uses tail-recursion, and its runtimes depends on its case.

Best Case: . Proof:

Given the runtime of quicksort, we can verify the best case by using the master theorem (1.1).

(1)

(2)

(3)

Worst Case: . Proof:

Given a base case , we can solve for the worst case using recurrence relations.

(1)

(2)

(3)

(4)

(5)

Average Case: . Proof:

(1.1)

We know that,

Substituting this equation into (1.1), we get

Then, we multiply by n

Applying the above equality to yields

Subtraction of the two previous equalities gives

Then simplifies to

Let

Then we obtain the recurrence,

Which implies,

Auxiliary Space:

Experimental Results

Three experiments were conducted to find the intersection of both merge sort and algorithms. Tables are presented below to represent, n, the number of elements being sorted, and the time in nanoseconds to perform the completed sorting operation. Corresponding scatter plots are presented below, each curve adjusted to follow a logarithmic trend line.

Run # 1 Run # 2

 

Run # 3 QuickSort Random

 

Experimental Results

Conclusion

After analyzing the runtimes of each algorithm, I plotted them on a X Y scatter plot and found that the resulting logarithmic trend line equations of the data sets were identical. Using wolfram alpha, I calculated the intersection of and and found that the resulting intersection is, , which is approximately 28.000. This number was used in MergeInsertionSort to verify that insertion sort would only be used if the number of sorted elements accumulated to be below 28. Resulting runtimes, after this modified algorithm was used reflected a more streamlined and efficient sorting method. Quicksort remained to be the most efficient algorithm even when compared to MergeInsertionSort.

My experience with the project was definitely challenging, especially because it was never a requirement to implement insertion sort, quick sort, or merge sort in my data structures class. I have to say that completing this project made me more knowledgeable about the different types of sorting algorithms and will always resort to quick sort given the choice between the three different algorithms.

References

Khan Academy. (n.d.). Retrieved February 14, 2017, from <https://www.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/analysis-of-insertion-sort>

Neapolitan, R. E. (2015). *Foundations of algorithms* (5th ed.). Burlington, MA: Jones & Bartlett Learning.