Introduction – Merge Sort

For this assignment, I opted to benchmark merge sorting. Merge sorting happens when a list of data is split into smaller lists that are broken down to their smallest components, typically a sub list of one element. In here, there would be three types of lists: the list we are working on (active list), the list above it (parent list), and any lists that spawn from it (children lists). Typically, at most two children list belong to a single active list, and the children list compare their elements together to see which of their elements need to be sorted. The act of combining the children lists together is called merging.

Merge sorting can be done in an iterative manner as well as a recursive manner, and as such, both have their own efficiencies. What is shared between them is that, assuming the same list is used for both iterative and recursive approach, the list will be at most N number of elements long, so in the case of any looping, N number of elements will have to be looked at.

When it comes to time, iterative merge sorting is faster than

Pseudo Code – Iterative Approach

Pesudo Code – Recursive Approach

Critical Operations and Analysis

When thinking about any sort of critical operation for the two merge sort algorithms, I felt as though that it would apply to the instances in which any merging is done. What I figured is that the best way to determine this is to look at the endings of each sort where the merging happens. This is where the comparison of elements are taking place. I count each comparison as their own operation.

Comparing My Data with Big-O

Conclusion