**SORTING ALGORITHM ANALYSIS**

**OVERVIEW**

Big O notation is a way to demonstrate how well an algorithm performs as its input size grows. It can be written as O(n), where n is the number of inputs and O is the order of complexity. As more values are added to an algorithm, the order of time and space complexity will change. Since n is the number of inputs, when n is scaled up, the faster Big O grows after each step, the worse the algorithm performs. Big O notation represents the worst-case time complexity, Big Ω represents the best-case time complexity, and Big 𝜃 represents the average time complexity.

**ALGORITHM COMPLEXITIES**

Bubble sort has a Big O value of O(n2) with n-1 comparisons on the 1st pass, n-2 in the 2nd, n-3 in the 3rd and so one. So, the total number of comparisons = . Big O notation is simplified to the fastest growing, worst-case scenario. In this case, it would be O(n2) since .

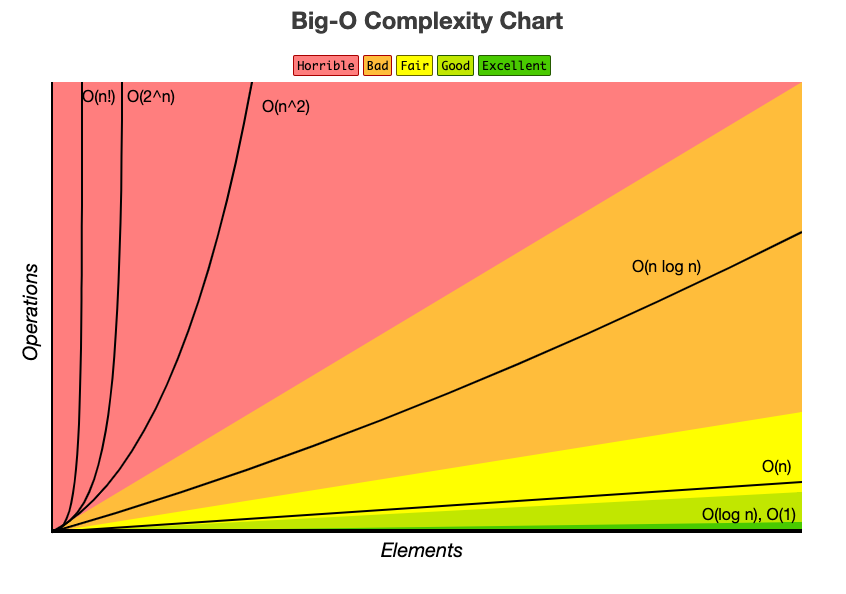
Selection sort also has a Big O value of O(n2). The algorithm iterates through the array using a for loop. Hence, for every element, a nested for loop is used to find the smallest element in the remaining part of the array. Since every element is visited for every iteration, a Big O value of O(n2) is established.

Insertion sort has a Big O value of O(n2) because a nested while loop inside the for loop can cause the runtime of the algorithm to compound as each item is referenced to the input twice, similar to the selection sort case above.

Merge sort has a Big O value of O(n \* log n). In merge sort, since the original array is being halved, it can be represented using a logarithmic function, log n, and the number of steps in merger sort can be represented by log n +1 (at most). Also, to merge the subarrays originally made from n elements, a run time of O(n) is required. Therefore, the total time for this sorting algorithm is n(log n + 1), which would produce a time complexity of O(n \* log n) since the low order term +1 can be discarded.

Quick sort has a Big O value of O(n2). A divide-and-conquer method partitions the array into sub arrays. Based on the information provided above in merge sort, this would have a Big O value of O(n\*log n). However, the algorithm then uses recursion to sort those arrays independently. Recursion has a Big O value of n2. Since Big O notation is simplified to the fastest growing, worst-case scenario, the Big O value for quicksort would be simplified to O(n2).

**FIGURES**



**Figure 1:** Displays the graphs of different Big O runtime scenarios and ranks them based on desirability.

*Chart, line chart

Description automatically generated*

**Figure 2:** Compilation of Data Recorded for each Sort, Array Size, and Data Type on a linear scale, with a second-degree polynomial trendline.