CPSC 335-05 Project 3: Sorting Race

Algorithm Running Time Analysis

William Covington

This paper analyzes the running time of Merge Sort, Quick Sort, and Selection Sort for Project 3 in CSUF 335-05 for the Fall 2019 semester.

Merge Sort:

The Merge Sort algorithm's running time can be analyzed by breaking down the divide-and-conquer algorithm into 3 parts. The first part is the arrays division. This takes constant runtime regardless of array size because it only computes the midpoint of the array. The next step is recursively dividing the subarrays until they are single elements. The running time in this step will increase with the input size because more characters in the array will require more recursive calls to break the array into single elements. The final step of the array is to merge the elements back together. This step involves comparing two elements and swapping the positions if necessary. After the comparison all the sorted elements are stored back in the array. Overall the runtime of Merge Sort is twice the sum of the running time of Merge Sort on an n/2-element subarray plus the running time for the divide and conquer steps.

Quick Sort:

The Quick Sort algorithms running time can be analyzed by breaking down the divide-and-conquer algorithm into three parts. The first step is selecting a pivot point. This takes constant running time because any element can be used as a pivot. The next step is moving through the array and shifting the elements to the left or right of the pivot if necessary. An element comparison is done for each element of the array during this step. The number of swaps will vary with the array input. The final step is to swap the pivot. This only takes constant time because the pivot and it's new location is already known.

Selection Sort:

The Selection Sort algorithm's running time can be analyzed by breaking it down into two parts. The first part is finding the lowest value in the array. This involves iterating through each element and comparing it with the lowest known value. The running time for this step will increase with the size of the array. After iterating through the array the lowest value will be swapped to furthest unsorted value on the left. This operation only takes constant time because the location of the lowest element and the left-most unsorted element are known. The running time of each pass will decrease with each iteration because the number of elements that need to be compared decreases.

Big O Runtime:

The Big O runtime of each Merge Sort and Quick Sort is O(N*logN) and Selection Sort has a Big O time of O(n^2). Selection sort run time is longer than Merge Sort and Quick Sort because the running time is always longer. Merge and Quick sort are almost always faster than Selection sort. The Big O time of Merge and Quick sort when compared to each other depend on the input.