335-05 — Algorithm Engineering — 3-Sort Race Analysis

Kayla Nguyen
Team: Algorithm

Merge Sort: There are three steps to consider when analysing the runtime of the Merge Sort algorithm. The first step 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 runtime of Merge Sort on an n/2-element subarray plus the runtime for the divide and conquer steps.

Quick Sort: The Quick Sort algorithms runtime 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 runtime 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 runtime 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 runtime for this step will increase with the size of the array. After iterating through the array the lowest value will be swapped to the 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 runtime 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 Quicksort is O(N*logN) and Selection Sort has a Big-O time of O(n^2). The Selection Sort runtime is longer than Merge Sort and Quicksort because the runtime is always longer. Merge and Quicksort are almost always faster than Selection sort. The Big-O time of Merge and Quicksort when compared to each other depends on the input.