Arman Ashrafian

CS 1D

Extra Credit Assignment

Big O Analysis

**Part 1: QuickSort**

The time complexity of the quick sort algorithm depends greatly on the chosen pivot. From the experiment, the algorithm ran the slowest when picking the first element in the reverse ordered array or the already ordered array. The run time improved drastically when picking the middle element in these arrays, or by just picking a random pivot. The randomly ordered array performed the same for every test since picking the first, middle, or random element for the pivot would always yield a random pivot. The worst case occurs when the partition process picks the greatest or smallest element as the pivot which gives a running time of Ө(n2). The best case for a quicksort occurs when the middle value is chosen for the pivot. This gives a logarithmic running time for the recursive call and the partition function still runs in linear time giving a run time of Ө(nlogn). The average case for the quicksort O(nlogn) since good pivots will usually be chosen as shown by the randomly ordered array experiment. One of the best ways to choose a pivot is to get the first, last, and middle elements and choose the median of these values.

**Part 2: Heap Sort & Merge Sort**

**Heap Sort:** The time complexity of the heap sort depends of the heapify function and the buildHeap function. Heapify runs in O(logn) since it is essentially a binary tree, and buildHeap is runs in linear time O(n). Therefore, the total time complexity of the Heap Sort algorithm in O(nlogn).

**Merge Sort:** The time complexity of the merge sort algorithm is O(nlogn) for all three cases (worst, average, best). This is because the merge sort always divides the array in two halves and takes linear time to merge two halves.