Michael Coleman
12/16/2023
CSCI 335

Assignment 2 Report

Algorithm	Size 1000 (in milliseconds)	Size 31K (in milliseconds)	Size 1M (in milliseconds)
Half Selection Sort	2 ms	1974 ms	Input size is too big for selection sort since its greater than 50,000
Std::Sort (Standard Sort)	0 ms	6 ms	194 ms
Merge Sort	0 ms	9 ms	430 ms
In Place Merge	0 ms	8 ms	356 ms
Half Heap	0 ms	4 ms	190 ms
QuickSelect	0 ms	3 ms	90 ms
Worst Case Quick Select	0 ms	5 ms	120 ms

The Half Selection Sort takes O(n^2), but it takes exactly n^2/2 comparisons. The standard sort takes O(nlogn), but it takes exactly n log n comparisons . Merge Sort takes O(n log n), but it takes exactly n*log(n) - (n - 1) comparisons. The In Place Merge Sort takes O (n log n), but it takes exactly n log n comparisons. The Half Heap Sort takes O(nlogn), but it takes exactly nlogn/2 comparisons. The QuickSelect takes O(n), but it takes exactly n comparisons. The worst case of QuickSelect takes O(n^2), but it takes exactly n^2 comparisons.

For the half selection sort, the time saved by ending the sort when you find the median is half of what a normal selection would be. Time is not saved while using std::sort. The merge sort does not stop early once the median is found. If you stopped it

early it could produce the incorrect median. The In Place merge sort does not save time because it does not stop the merging once the median is found. The Half Heap Sort takes half the amount of time as a regular Heap Sort which is O(n log n), so it takes O(n log n /2). QuickSelect is faster than most sorting algorithms since they have a O(n log n) time complexity. Quick Select uses the exact amount of comparisons to get the kth number.

I expected QuickSelect to have the fastest timed result and I was correct. It did better than all of the other sorting algorithms, especially when the input sizes get larger. I expected the Standard Sort function to have an average time result since it has the average time complexity of a sort, O(n log n). This was partially correct because it ran faster than the other sorts that have a time complexity of O(n log n). I expected the In Place Merge Sort to be faster than the regular Merge Sort, which was correct. I thought the Half Heap Sort would be faster than the average sort since it is half of a heap sort which takes O(n log n). This was correct, since it had a faster time than all of the methods except QuickSelect. Finally I expected the Half Selection Sort to be slow since it has a time complexity of O(n^2). I was correct about this since it had the longest timed result out of all of the other methods.

Out of all of my results, I find it interesting that the Half Heap Sort Method was as fast as it was. For medium sized inputs it was almost as fast as the QuickSelect method. I also did not expect the Half Selection Sort to be that slow. I thought that since it was just finding the median and stopping the sort that it would be as fast as some of the other sort functions.