Kyle Lugo November 14, 2018

Manifest

heapsort.h - A code file containing an implementation of the heapsort method of sorting.

mergesort.h - A code file containing an implementation of the mergesort method of sorting.

quicksort.h - A code file containing an implementation of the quicksort method of sorting.

insertion.h - A code file containing an implementation of the insertionsort method of sorting.

sorting.cpp - The main code file. Calls the other code file's functions for the purpose of analyzing their runtime performance when sorting integers.

Project3.pdf - The report for the project. Contains a brief summary of the project and an analysis on sorting algorithm runtimes.

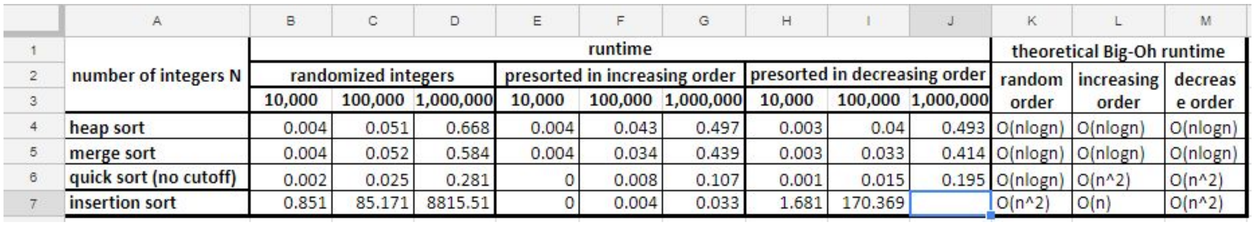
Project Overview

This project is designed for the purpose of analyzing the performance of four different sorting algorithms on three different data sets three times, with a different sample size each time. The first data set contained pseudo random numbers that were generated by the program. The second data set contained already sorted numbers from least to greatest. The third data set contained sorted numbers from greatest to least. All three data sets were used on each algorithm three times. The first time, the data sets contained 10,000 numbers, the second time they contained 100,000 numbers and the third time they contained 1,000,000 numbers.

The four algorithms used were heapsort, mergesort, quicksort and insertionsort. Each of these algorithms had their own coding implementations inside header files with corresponding names. These files were called by the main file sorting.cpp, which would pass data sets to each algorithm to sort while measuring the time in seconds that the algorithm takes to complete the sort. These runtimes were printed into the console and then documented.

Analysis

In regards to each algorithm, different runtimes were obtained for each of the three different data sets. With the heapsort, similar runtimes were found between all three data sets when a sample size of 10,000 was being used. As the sample size increased, the runtime was practically the same between the sorted ascending and descending data sets, with the random data set taking a bit longer due to its random nature.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	number of integers N	runtime									theoretical Big-Oh runtime		
2		randomized integers			presorted in increasing order			presorted in decreasing order			random order	increasing order	decrease order
3		10,000	100,000	1,000,000	10,000	100,000	1,000,000	10,000	100,000	1,000,000			
4	heap sort	0.004	0.051	0.668	0.004	0.043	0.497	0.003	0.04	0.493	O(nlogn)	O(nlogn)	O(nlogn)
5	merge sort	0.004	0.052	0.584	0.004	0.034	0.439	0.003	0.033	0.414	O(nlogn)	O(nlogn)	O(nlogn)
6	quick sort (no cutoff)	0.002	0.025	0.281	0	0.008	0.107	0.001	0.015	0.195	O(nlogn)	O(n^2)	O(n^2)
7	insertion sort	0.851	85.171	8815.51	0	0.004	0.033	1.681	170.369		O(n^2)	O(n)	O(n^2)