CSCE 221 Cover Page

Programming Assignment #3

First Name: Anthony Last Name: Shao UIN: 725004846

**Any assignment turned in without a fully completed cover page will receive ZERO POINTS.**

Please list all below all sources (people, books, webpages, etc.) consulted regarding this assignment:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CSCE 221 Students | Other People | Printed Material | Web Material (URL) | Other |
|  |  |  | Stack Overflow | CSCE 221 PowerPoint Slides |
|  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Recall that University Regulations, Section 42, define scholastic dishonesty to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Today’s Date: 4/25/2018

Printed Name (in lieu of a signature): **Anthony Shao**

**Introduction**

The objective of this assignment is to implement three ways of sorting numbers (in ascending order). First, is to implement selection sort. Second, is to implement heap sort. Third is to implement merge sort. All of the sorting algorithms are implemented using an array data structure.

**Theoretical Analysis**

The first implementation is selection sort. This sorting algorithm for an array of n elements takes O(n2) time cost because it iterates through the entire array to find the smallest item; then it swaps with the first item of the array. The two steps are repeated with the remaining n – 1 items in the array. The number of comparisons made is n(n - 1), which is O(n2) time cost. Time complexity is O(n2) for all cases of the best (sorted), average (random), and worst (reverse).

The second implementation is heap sort. This sorting algorithm for an array of n elements takes O(n log n) time cost. The complete binary tree maps the binary tree structure into the array indices. In other words, it first builds a heap data structure using a max heap, which takes O(n) time cost to insert n items. Then, the root node (first index), already the largest element is removed and replaced with the last item in the array (array size decreases). After each removal, the heap itself is rearranged (also called heapify) so that the heap property is maintained. Because building max heap takes O(n) time cost and runs one time, and heapifying per removal takes O(log n) time cost and is called n times, the overall time complexity for heap sort is O(n + n log n) or O(n log n). Time complexity is O(n log n) for all cases of best, average, and worst.

The third implementation is merge sort. This sorting algorithm for an array of n elements takes O(n log n) time cost. The array is first divided into subarrays each with one item. Then each partitioned array is repeatedly compared, sorted, and merged into new sorted subarrays until there is one entire array. Time complexity is O(n log n) for all cases of best, average, and worst.

**Experimental Setup**

Regarding machine specification, I used a Lenovo Yoga Pad 2 with Windows 10 running Microsoft’s Visual Studio 2017 and Window’s Command Prompt to implement, code, and run the three types of sorting algorithms for this assignment. Another platform I used to run this programming assignment, after encountering a weird timer error with Command Prompt, was Texas A&M’s build.tamu.edu server from my laptop using PuTTY and WinSCP.

Inputs into the arrays were generated using a random number generator. I first created an array with varying sizes. The sizes varied by increments of 10,000 (from 10,000 to 100,000). This was to test for long runtimes. Then, I generated random numbers between 0 and 1,000,000 for a larger range and variation into an input array. If the user chose to have the test inputs sorted, the input array was rearranged so that the numbers are in ascending order. If the user chose to have the test inputs reverse of sorted, the input array was rearranged so that the numbers are in descending order. Otherwise, the input array was left as is (for random test input). All the sorting algorithms were implemented using an array data structure due to its easy use and handling of arrays. Each experiment (type of test input and sorting algorithm) was tested at least 20 times.

**Experimental Results**

(Note: Selection sort uses the left-side y-axis range while both heap sort and merge sort use the right-side y-axis range. Otherwise, if both use the same y-axis range, then the heap sort and merge sort graphs would seem to have disappeared.)

The graph shown is the average times of sorting algorithms with sorted, reversed, and random test inputs. In the long-run, the best performing sorting algorithm is merge sort. Then, the next best performing sorting algorithm is heap sort. And last place is selection sort because. The selection sort algorithm’s performance appears to have a quadratic trend. This closely matches selection sort’s theoretical complexity of O(n2). Both the heap sort and merge sort algorithms’ performances appear to have a linear trend in the long run. This closely matches both heap sort and merge sort’s theoretical complexities of O(n log n). The input does indeed affect the performance; for selection sort, because the time complexity is quadratic, the larger the input, the much longer the time; for heap sort and merge sort, the complexity is far faster and change in performance is far less noticeable. These two sorts finished sorting less than 0.1 seconds for 100000 elements whereas selection sort finished sorting ~17 seconds. Overall, the theoretical analysis of all the sorting algorithms strongly agrees with the experimental results.

Shown above, as for sorted test inputs and its reverse, the trend appears to be identical. Shown below, for random test inputs, both heap sort and merge sort took a longer time than their sorted and reverse test inputs.