Veet Zaveri
Lab 4 Analysis
605.202 Data Structures
04/27/2022

Testing Machine Specifications:

Device:	Laptop
CPU:	AMD A8-7100 Radeon R5
RAM:	12GB
OS:	Windows 10

File Size used for comparison = 500 numbers per file

Runtimes of the various runs

Following is the average runtime of the both Shell sort and Heap sort on the same size file. This data can be further used to extract the comparison information.

Run	Shell Sort			Heap Sort		
	Normal	**Reversed**	**Random**	**Normal**	**Reversed**	**Random**
1	0.3758	0.4331	0.3987	0.0102	0.0105	0.0104
2	0.4201	0.4085	0.4419	0.0108	0.0119	0.0101
3	0.4429	0.4275	0.4731	0.0101	0.0100	0.0098
4	0.4276	0.4256	0.5174	0.0098	0.0107	0.0091
5	0.3762	0.3873	0.3835	0.0096	0.0097	0.0098
Avg	**0.40852 s**	**0.4164 s**	**0.44292 s**	**0.0101 s**	**0.01056 s**	**0.00984 s**

Order of Data	Shell Sort	Heap Sort
Normal Order (Ascending)	0.40852 sec	0.0101 sec
Reversed Order (Descending)	0.4164 sec	0.01056 sec

Random Order	0.44292 sec	0.00984 sec

The data structures that were used are arrays and trees. Array data structure is used because the initial numbers we need to sort are stored in an array and then after it is sorted it becomes a sorted array. For heap sort we must see the elements of the array as a special type of complete binary tree called heap. A binary tree is a tree in which each member has no more than two offspring. Because each element in a binary tree may only have two children, they are commonly referred to as the left and right child.

Arrays are a collection of things stored in adjacent memory spaces is referred to as an array. The objective is to group together elements of the same category. This makes calculating the position of each element easy by simply adding an offset to a base value, such as the memory address of the array's first member.

Heap sort is a sorting algorithm that uses the Binary Heap data structure to compare items. It's comparable to selection sorting, in which we identify the smallest piece first and place it at the top. The technique is repeated for the remaining items. A complete binary tree is one in which every level is entirely filled, but potentially the final, and all nodes are as far left as allowed.

A Binary Heap is a complete binary tree in which items are stored in such a way that the value of a parent node is bigger or smaller than the values of its two child nodes. The former is known as max heap, whereas the latter is known as min-heap. A binary tree or an array can be used to represent the heap.

Shell sort is a form of insertion sort. We just shift items one place forward in insertion sort. Many moves are required when an element must be pushed far ahead. The purpose of shell sorting is to allow for the interchange of far-flung goods.

I learned a lot about time and space complexity for shell sort, heap sort, and insertion sort and how these can differ depending on the size of array. I also learned more about how shell sort can avoid large shifts if the smaller value is far right and has to be moved to far left. In addition, I learned how since binary heaps are a complete binary tree it can be represented as an array and this array-based representation is space-efficient.

Although there is not much I would do differently next time, the only method I would change is implementing a quicksort algorithm to compare with these algorithms since quicksort may be faster but can hit bad cases more often especially for large sets. With quicksort there is a chance of worse case performance as well.

Time complexity of shell sort is O(n^2) and for heap sort it is O(n*logn). According to output table, as size of array grows for random array values the time complexity of sell sort takes more time than heap sort. For reverse array values, the time complexity of shell sort takes more time

than heap sort for larger array sizes. For in order array values, time complexity of shell sort takes less time than heap sort for larger array sizes. Space complexity for both algorithms is constant since sorting occurs on array of elements and no extra array space is used for sorting.

Time complexity of shell sort is O(n^2) and for insertion sort it is O(n^2). According to output table, for random array values the time complexity of shell sort takes less time than insertion sort. For reverse array values, the time complexity of shell sort takes less time than insertion. For in order array values, time complexity of insertion sort takes less time than shell sort. Space complexity for these two algorithms is constant since sorting is on array of elements and there is no extra array space used for sorting.