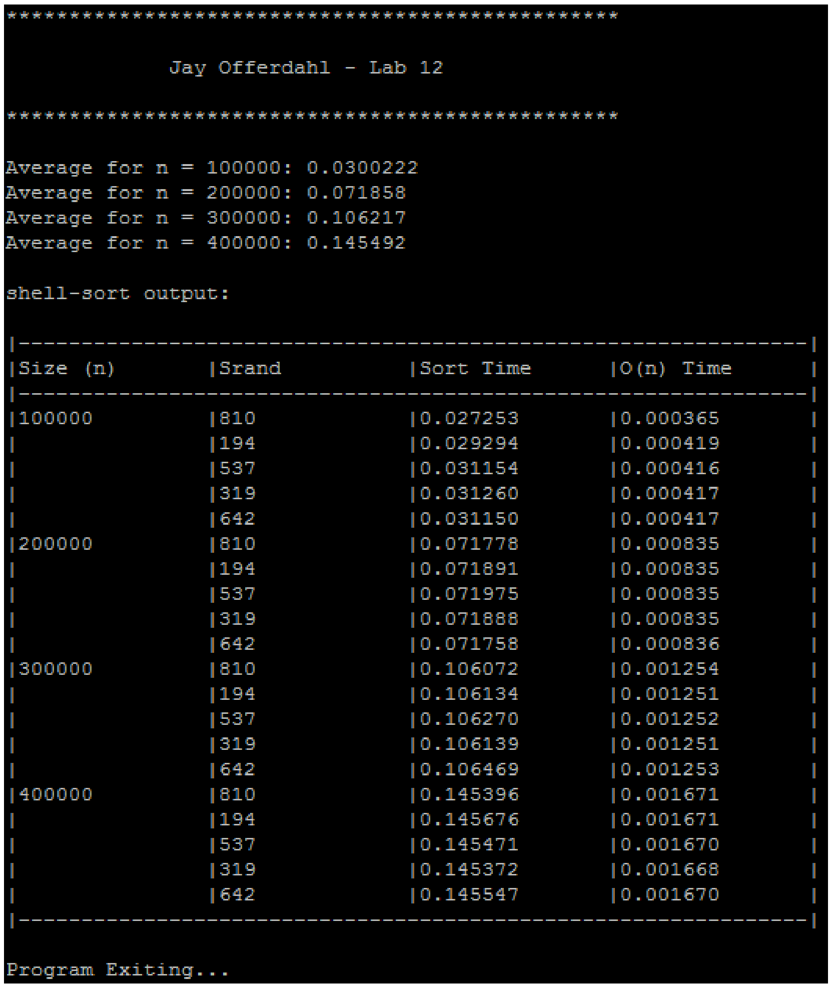
Jay Offerdahl
EECS 560
Lab 12 Report

Lab 12: Shell Sort

Looking at the data for this lab, we can clearly see that the trend is very close to linear. Knowing that shell sort has an average case time of O(nlg(n)), it's most definitely possible for this curve to resemble an nlg(n) curve. However, with this small of a test set, it's hard to see any amount of curving. Not to mention the O(nlg(n)) curve is extremely straight. To combat this issue, I tracked the amount of time my program took to do an O(n) operation. For this operation, I simply timed my "checkSorted()" method, which loops through the whole list to make sure all values are in ascending order. When I plotted this data onto the graph, it showed a much straighter line. Even though this line appears flat, it is linear (simply judging off of the values).

After comparing my shell sorts average outputs with an O(n) operation, I can confidently say that shell sort has complexity O(nlg(n)). Again, while the actual curve of the nlg(n) curve may only be slightly visible here, it the slope is much higher than a linear slope. On another note, if we traced the left edge of the blue line to its intersection on the x-axis, it would have a positive value of x, meaning it would then take negative time to sort data amounts less than this. With this, it's easy to conclude this is an O(nlg(n)) curve.



```
*************************************************

             Jay Offerdahl - Lab 12

*************************************************

Average for n = 100000: 0.0300222
Average for n = 200000: 0.071858
Average for n = 300000: 0.106217
Average for n = 400000: 0.145492

shell-sort output:

|----------------------------------------------------------------|
|Size (n)        |Srand           |Sort Time       |O(n) Time       |
|----------------------------------------------------------------|
|100000          |810             |0.027253        |0.000365        |
|                |194             |0.029294        |0.000419        |
|                |537             |0.031154        |0.000416        |
|                |319             |0.031260        |0.000417        |
|                |642             |0.031150        |0.000417        |
|200000          |810             |0.071778        |0.000835        |
|                |194             |0.071891        |0.000835        |
|                |537             |0.071975        |0.000835        |
|                |319             |0.071888        |0.000835        |
|                |642             |0.071758        |0.000836        |
|300000          |810             |0.106072        |0.001254        |
|                |194             |0.106134        |0.001251        |
|                |537             |0.106270        |0.001252        |
|                |319             |0.106139        |0.001251        |
|                |642             |0.106469        |0.001253        |
|400000          |810             |0.145396        |0.001671        |
|                |194             |0.145676        |0.001671        |
|                |537             |0.145471        |0.001670        |
|                |319             |0.145372        |0.001668        |
|                |642             |0.145547        |0.001670        |
|----------------------------------------------------------------|

Program Exiting...
```

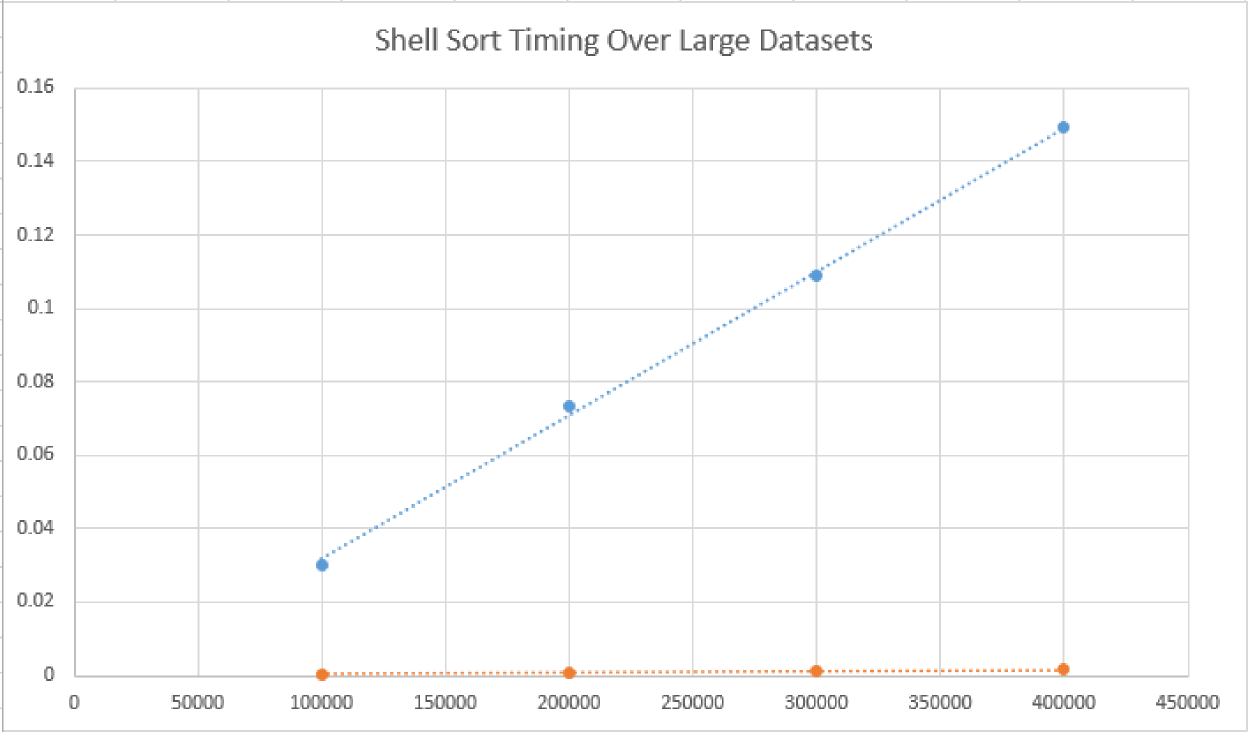
Jay Offerdahl
EECS 560
Lab 12 Report

Results (Shell Sort and a linear operation "checkSorted()")

Data Results - Shell Sort			
100000	200000	300000	400000
0.027935	0.07348	0.108698	0.149076
0.027914	0.073548	0.108902	0.149382
0.031048	0.073582	0.108905	0.14918
0.031989	0.073567	0.108767	0.149236
0.031956	0.073358	0.109116	0.149285
0.030168	0.073507	0.108878	0.149232

O(n) time			
100000	200000	300000	400000
0.000362	0.000836	0.001251	0.00167
0.00042	0.000835	0.001251	0.00167
0.000419	0.000836	0.001254	0.00167
0.00042	0.000835	0.001254	0.00167
0.000416	0.000836	0.001252	0.001668
0.000407	0.000836	0.001252	0.00167

Averages		
Data Size	Time	O(n) Time
100000	0.030168	0.000407
200000	0.073507	0.000836
300000	0.108878	0.001252
400000	0.149232	0.00167



Jay Offerdahl
EECS 560
Lab 12 Report

Code:

```
//***************************************************************************
//
//			Author:	Jay Offerdahl
//			Class:	EECS 560 (Data Structures)
//			Lab:	Tues. 11a - 12:50p
//			Lab #:	12
//
//***************************************************************************

#include <string.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <stdlib.h>
#include "Timer.cpp"

void printTable(int, int, std::string[][4]);
void fillArr(int, int*, int*, int);
void shellSort(int*, int, std::string);
bool checkSorted(int*, int);

int main(int argc, char* argv[])
{
	std::cout << "************************************************\n\n";
	std::cout << "		Jay Offerdahl - Lab 12		\n\n";
	std::cout << "************************************************\n\n";

	if(argc  < 2) {
		std::cout << "Invalid usage: Please specify an input file: ./lab [filename]\n";
		return 1;
	}
	std::ifstream file(argv[1]);

	if(!file) {
		std::cout << "There was an error opening the input file...\n";
		return 1;
	}

	// Initialize arrays and their current sizes
	int *arr = new int[400001]();
	int arrSize = 0, row, temp;
	int dataSizes[] = { 100000, 200000, 300000, 400000 };
	int seeds[] = { 810, 194, 537, 319, 642 };
```

```
double duration;
char buffer[256];
std::string line;
std::string output[21][4];

output[0][0] = "Size (n)";
output[0][1] = "Srand";
output[0][2] = "Sort Time";
output[0][3] = "O(n) Time";

Timer tim;
double average = 0.0;

for(int i = 0; i < 4; i++) {
        int currentSize = dataSizes[i];
        getline(file, line);

        // Fix the input to not have any commas
        temp = line.find_first_of(", ", temp + 1);

        while(temp != std::string::npos) {
                line[temp] = ' ';
                temp = line.find_first_of(", ", temp + 1);
        }

        output[i * 5 + 1][0] = std::to_string(currentSize);

        // Do five tests for this data size
        for(int j = 1; j <= 5; j++) {
                arrSize = 0;
                row = i * 5 + j;

                // Fill the array with random numbers from -3n to 3n
                fillArr(seeds[j - 1], arr, &arrSize, currentSize);
                output[row][1] = std::to_string(seeds[j - 1]);

                // Sort the array using shell sort
                tim.start();
                shellSort(arr, arrSize, line);
                duration = tim.stop();

                sprintf(buffer, "%5.6f", duration);
                output[row][2] = buffer;

                average += duration;

                tim.start();
                // Assert this array is sorted
                if(!checkSorted(arr, arrSize)) {
                    std::cout << "The array was not sorted after shell sort...\n";
                    return 1;
                }
                duration = tim.stop();

                sprintf(buffer, "%5.6f", duration);
                output[row][3] = buffer;
            }
            std::cout << "Average for n = " << currentSize << ": " << average / 5 << "\n";
            average = 0.0;
        }
        // Print out the output
        printTable(21, 4, output);

        std::cout << "\nProgram Exiting...\n\n";
        delete arr;
        return 0;
}

/**
 * Sorts the input array using shellsort
 * @param arr     - the input array to be sorted
 * @param arrSize - the size of the input array
 * @param stream  - a stream holding the increments to use with shellsort
 */
void shellSort(int* arr, int arrSize, std::string scheme) {
        int inc, temp, j;

        std::stringstream stream(scheme);
        stream >> inc;

        while(stream) {
            // Get the next value of increment
            for(int i = inc; i <= arrSize; i++) {
                temp = arr[i];
                j = i - inc;
                while(j > 0 && arr[j] > temp) {
                    arr[j + inc] = arr[j];
                    j -= inc;
                }
                arr[j + inc] = temp;
            }
            stream >> inc;
        }
}
```

```
/**
 * Determines if the input array is sorted or not
 * @param  arr  - the input array to check
 * @param  size - the size of the input array
 * @return      - return true if sorted in ascending order
 */
bool checkSorted(int* arr, int size) {
	for(int i = 1; i < size; i++) {
		if(arr[i] > arr[i + 1])
			return false;
	}
	return true;
}

/**
 * Prints out the input two dimensional array
 * @param rows - the number of rows in the array
 * @param cols - the number of columns in the array
 * @param arr  - the input array to pring
 */
void printTable(int rows, int cols, std::string arr[][4]) {
	std::cout << "\nshell-sort output:\n";
	std::cout << "\n|------------------------------------------------------------|\n";
	for(int i = 0; i < rows; i++) {
		if(i == 1)
			std::cout << "|------------------------------------------------------------|\n";
		for(int j = 0; j < cols; j++) {
			if(j == 0)
				std::cout << "|";
			std::cout << arr[i][j];
			if(arr[i][j].length() >= 8)
				std::cout << "\t|";
			else
				std::cout << "\t\t|";
		}
		std::cout << "\n";
	}
	std::cout << "|------------------------------------------------------------|\n";
}

/**
 * Fills the input array with randomly generated numbers, srands with seed
 * @param seed     - the input value to seed the RNG with
 * @param arr      - the input array to fill
 * @param arrSize  - the size of the input array
 * @param fillSize - the number of values to put into the array
 * @note		- modifies the arrSize variable to reflect correct size
 */
void fillArr(int seed, int* arr, int* arrSize, int fillSize) {
	int val;
	srand(seed);

	// Randomly generate the values in the array
	for(int i = 1; i <= fillSize; i++) {
		val = rand() % (fillSize * 6) - fillSize * 3;

		arr[(*arrSize)++] = val;
	}
}
```