Project Cover Page

This project is a group project. For each group member, please print first and last name and e-mail address.

1. Andrew Schlotzhauer

2. Prem Pokharel

3. Jaiden Gerig

Please write how each member of the group participated in the project.

1. 100%

2. 100%

3. 100%

Please list all sources: web pages, people, books or any printed material, which you used to prepare a report and implementation of algorithms for the project.

Type of sources:	
People	TA, Peer teachers
Web Material (give URL)	http://stackoverflow.com http://courses.cs.tamu.edu/teresa/csce221/csce221-index.html
Printed Material	Data Structures and Algorithms in C++
Other Sources	

I certify that I have listed all the sources that I used to develop solutions to the submitted project report and code.
Your signature Typed Name Andrew Schlotzhauer Date February 12, 2014

I certify that I have listed all the sources that I used to develop a solution to the submitted project and code.
Your signature Typed Name Prem Pokharel Date Februrary 12, 2014

I certify that I have listed all the sources that I used to develop solution to the submitted project and code.
Your signature Typed Name Jaiden Gerig Date February 12, 2014

Programming Assignment #2 Report

February 17, 2014

Part I
Description

1 Description and Purpose

In this programming assignment we had to implement 5 sorting algorithms: selection sort, insertion sort, bubble sort, shell sort, and radix sort in C++. We had to understand how each of these sorting algorithms worked in order to get the running times and number of comparisons. We had to test our sorting algorithms with varying input cases from a file or console, and output to the console window or file. We had to use this collection of data to compare to the known Big-O asymptotic notation.

2 How to run and Input and Output

In order to compile you need to type make all into putty, and then to run it you must type ./sort [-a ALGORITHM] [-f INPUTFILE] [-o OUTPUTFILE] [-h] [-d] [-p] [-t] [-c]. The input is taken from the specified input file or from the console window, and the output is outputted to the specified output file or the console window.

Part II
Explanation of Classes

The Sort class is a base class of all of the other sorting classes, and each sorting class has its own sorting algorithm associated with it. The option class that provides a menu for selecting the algorithms to be used and how the input and output will be organized and used. For object oriented programming, we used inheritance feature, by the sorting class being the superclass and all of the other sorting classes as subclasses. For generic programming, we used the array class to hold the numbers from our input file to be sorted.

Part III
Algorithms

1. Selection Sort - This sort just compares linearly, it selects an element and checks it between each of the elements in the array and puts it where it belongs. There is two for loops used, and 2 if statements, but only one checks between each element, the other just makes sure it is in the correct order.

2. Insertion Sort - This sort removes an element from the array, and places it into the correct position of the already sorted part of the array, until no input elements are left.

3. Bubble Sort - This sort repeatedly compares adjacent elements of an array, so at first the first and second elements are compared and swapped if out of order, then the second and third are compared and so on, and this process continues until the last two elements of the array are compared and swapped if out of order.

4. Shell Sort - This sort breaks up the main array into smaller arrays and sorts the smaller ones, and then decreases the number of smaller arrays and then finally it takes the almost sorted main array and then sorts all of the elements.

5. Radix Sort - This sort is a non-comparison based sorting algorithm, it uses a stable sorting algorithm to sort each number respectively to each digit in each number, so first it sorts each number based on the one's place, then on the ten's, and so on until it is in order.

Part IV
Theoretical Analysis

Complexity	best	average	worst
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Shell Sort	$O(nlogn)$	$nlog^2n$	$O(n^2)$
Radix Sort	$O(n)$	$O(\frac{nk}{d})$	$O(n)$

Complexity	inc	ran	dec
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Shell Sort	$O(nlogn)$	$O(nlog^2n)$	$O(n^2)$
Radix Sort	$O(n)$	$O(n)$	$O(n)$

1. (65 points) **Experiments.**

   (a) Briefly describe the experiments. Present the experimental running times (**RT**) and number of comparisons (**#COMP**) performed on input data using the following tables.

RT	Selection Sort(ms)			Insertion Sort(ms)			Bubble Sort(ms)			Shell Sort(ms)			
n	inc	ran	dec	inc	ran	dec	inc	ran	dec	inc	ran	dec	i
100	.0248	.0283	0.0273	.0011	.0256	.0521	.0267	.0484	.0538	.0045	.0107	.0064	.0
10^3	2.3	2.3	2.3	.01	2.52	4.82	2.48	5.15	5.18	.08	.18	.09	.
10^4	235	230	238	.1	251	507	234	498	515	1.2	2.8	1.4	4
10^5	22050	22780	22710	1	23910	47850	23470	48540	49190	20	30	20	0

#COMP	Selection Sort			Insertion Sort			Bubb	
n	inc	ran	dec	inc	ran	dec	inc	r
100	4950	4950	4950	99	2398	5049	4950	4
10^3	499500	499500	499500	999	256721	500499	499500	499
10^4	49995000	49995000	49995000	9999	24813964	50004999	49995000	4999
10^5	4999950000	4999950000	4999950000	99999	2504598571	5000049999	4999950000	49999

inc: increasing order; dec: decreasing order; ran: random order

   i. For each of the five sort algorithms, graph the running times over the three input cases (inc, ran, dec) versus the input sizes (n); and for each of the first four algorithms graph the numbers of comparisons versus the input sizes, totaling in 9 graphs.
   Excel sheets provided.

   To compare performance of the sorting algorithms you need to have another three graphs to plot the results of all sorts for the running times for *each* of the input cases (inc, ran, dec) separately.

   Hints: To get a better view of the plots, *use logarithmic scales* for both x and y axes.

   To implement the radix sort algorithm that can sort a sequence of unsigned integers (less than $2^{32}-1 \approx 4.3 \cdot 10^9$). Compare its running time with the running time of four comparison-based algorithms. Is your computational results match the theoretical analysis you learned from class or textbook? Justify your answer.

   (b) (5 points) **Discussion.** Relate your experimental results you obtain to the theoretical analysis of the sort algorithms. Comment on how the experimental results relate to the theoretical analysis and explain any discrepancies you note.
   The experimental results were as expected. For example, the insertion sort in the best case took 0.0011 ms whereas in the worst case it took 0.0521 ms. This proofs the $O(n^2)$ for the worst case insertion sort

whereas $O(n)$ for the best case. Comparing the running time for all sorting algorithms for the input size of 10^2, all inputs being in increasing order, selection sort took the longest of 0.0248 ms. This verifies that in selection sort, even if the datas are arranged in order, it requires scanning the entire array. Also, the experimental results of the increasing number of comparisons as the input size increases also verifies the characterstics of the algorithms.

(c) (5 points) **Conclusions.** Give your observations and conclusion. For instance, which sorting algorithm seems to perform better on which case? Do the experimental results agree with the theoretical analysis you learned from class or textbook? What factors can affect your experimental results?
In the case of sorted data, insertion algorithm performed much better than other algorithms. Its running time was least compared to other algorithms. Similarly, in the random case, shell sort algorithm used least amount of comparisons as compared to others. Bubble sort and Insertion sort algorithm used same number of comparisons. In the case of decreaing input shell sort performed much better than other algorithm with least running time and also the number of comparsions were much less than other algorithmsSo, the experimental results agree with the theoretical analysis. Performing the experiments on different machines can vary the result.