Project Cover Page
This project is a group project with up to three students per team. For each group member, please print first and last name and e-mail address.
1. August Mcbride bluejae66@tamu.edu
2. Leilani Horlander-Cruz leilanihc112@tamu.edu
Please write how each member of the group participated in the project.
1. August Mcbride - tested the files, insertion-sort.cpp, selection-sort.cpp, shell-sort.cpp
2. Leilani Horlander-Cruz - answered report questions, radix-sort.cpp, bubble-sort.cpp
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:	
	Benjamin Sullivan
People	
	cplusplus.com
Web Material (give URL)	
Printed Material	
	Powerpoints in class
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 Leilani Horlander-Cruz Typed Name Leilani Horlander-Cruz Date 2-19-17
I certify that I have listed all the sources that I used to develop a solution to the submitted project and code.
Your signature August Mcbride Typed Name August Mcbride Date 2-19-17

CSCE 221 Programming Assignment 2 (200 points)

Program and Reports due February 19th by 11:59pm

- **Report (110 points)**

1. The assignment was to implement five sorting algorithms: selection, insertion, bubble, shell, and radix sort. In order to test the code, varied input cases were used, timing the implementation of sorts was done, recording the number of comparisons performed in the sorts was done, and comparing the computational results with the running time using Big-O asymptotic notation was done. To use the sorts, input data was to be input as one line containing the number of integers to sort, and the following lines each contained one integer out of the total integers to sort. The output data was to contain one integer per line in increasing order, of the input data. The purpose of the assignment was to observe how different algorithms compare to one another in their runtime and implementation. In order to run the program type "./sort -a [algorithm] -f [input file ex. 1_000_decrease.txt] -c -t", you would change inout file accordingly to math the input data you want i.e 100_increase or 10_000_random, etc.
2. Creating different classes allowed for a divide-and-conquer approach to reduce a single large problem into multiple simple sub-problems. We created different .cpp files for each class so changes that needed to made to one sort only needed to be changed in that one class and checked by compiling that one file rather than all of the files at once. These classes are all a part of the main class, Sort. We used polymorphism heavily in this program, which refers to an ability to redefine methods for derived classes.
3. Selection sort: Selection sort divides the list into two parts: a sublist of items already sorted, and a subset of items remaining to be sorted. Initially, the sorted sublist is empty. It then finds the smallest element in the unsorted sublist, exchanging it with the leftmost unsorted element, and moving the sublist boundaries one element to the right.
   Insertion sort: The algorithm builds the final sorted array or list one item at a time, removing one input element each repetition, finding where it goes in the sorted list, and inserts it there. It repeats until no input elements remain.
   Bubble sort: The algorithm steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the incorrect order. This is repeated until no swaps are needed throughout the whole list, which indicates that the list is sorted.
   Shell sort: The sequence is divided into several segments and each segment is sorted using insertion sort. This repeats until the end. This sort uses subsequences rather than sorting the entire sequence as a whole.
   Radix sort: Radix sort sorts data by sorting the least significant digit first, all the way until the most significant digit. The radix sort will need 'd' passes to sort the numbers, where 'd' is the most amount of digits in a number.
4.

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	$O(n\log n)$	Between $O(n(\log n)^2)$ and $O(n^{3/2})$	Depends on gap sequence, but best known is $O(n(\log n)^2)$
Radix Sort	$O(wn)$	$O(wn)$	$O(wn)$

Complexity	inc	ran	dec
Selection Sort	best: $O(n^2)$	average: $O(n^2)$	worst: $O(n^2)$
Insertion Sort	best: $O(n)$	average: $O(n^2)$	worst: $O(n^2)$
Bubble Sort	best: $O(n)$	average: $O(n^2)$	worst: $O(n^2)$
Shell Sort	best: $O(n)$	worst: Between $O(n(\log n)^2)$ and $O(n^{3/2})$	average: $O(n(\log n)^2)$
Radix Sort	best: $O(n)$	average: $O(wn)$	worst: $O(wn)$

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

5.

RT	Selection Sort			Insertion Sort			Bubble Sort		
n	inc	ran	dec	inc	ran	dec	inc	ran	dec
100	0ms	0ms	0ms	0ms	0ms	0ms	0ms	0ms	0ms
10^3	0ms	0ms	0ms	0ms	0ms	10ms	0ms	10ms	10ms
10^4	350ms	350ms	390ms	0ms	390ms	800ms	0ms	730ms	820ms
10^5	35390 ms	35570 ms	40290 ms	0ms	39980 ms	79960 ms	0ms	73790 ms	82120 ms

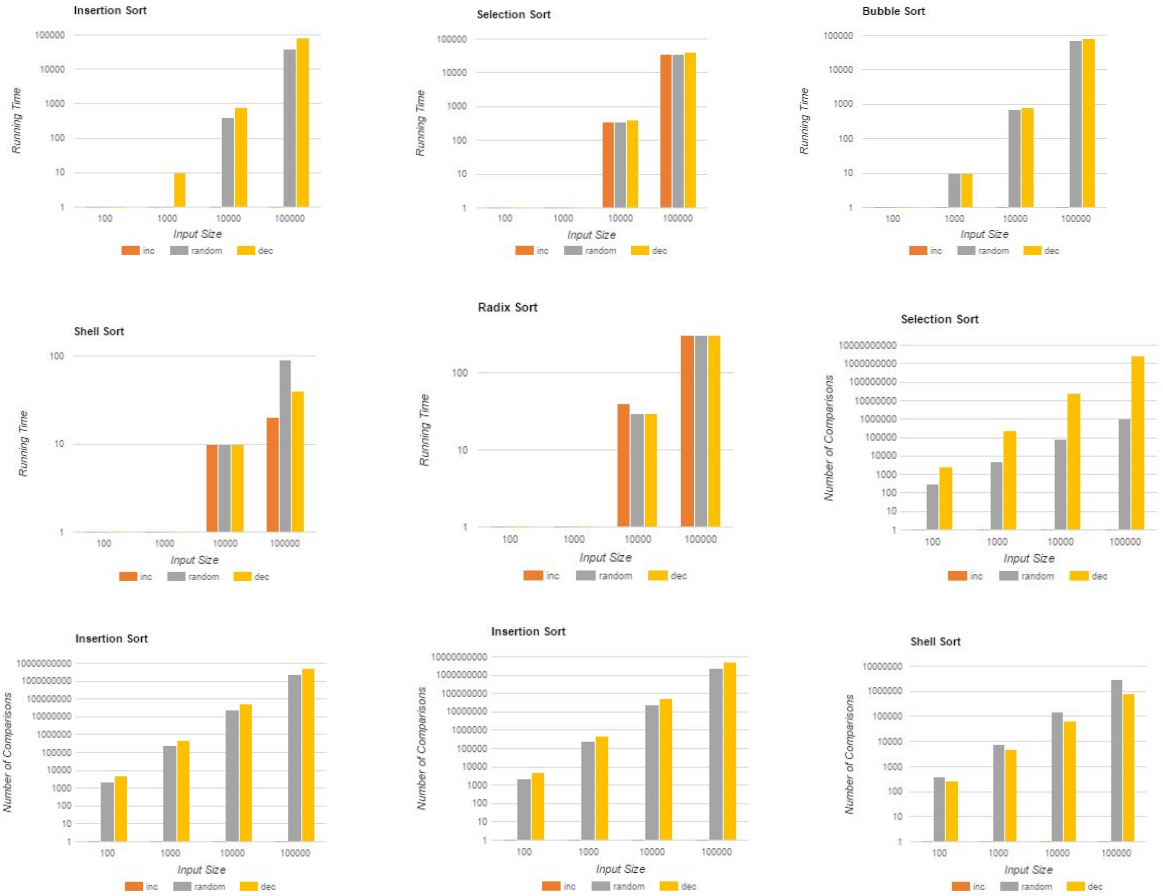
RT	Shell Sort			Radix Sort		
n	inc	ran	dec	inc	ran	dec
100	0ms	0ms	0ms	0ms	0ms	0ms
10^3	0ms	0ms	0ms	0ms	0ms	0ms
10^4	10ms	10ms	10ms	40ms	30ms	30ms
10^5	20ms	90ms	40ms	310ms	310ms	310ms

#COMP	Selection Sort			Insertion Sort		
n	inc	ran	dec	inc	ran	dec
100	0	304	2500	0	2247	4950
10^3	0	5186	250000	0	248834	499500
10^4	0	78174	25000000	0	25136391	49995000
10^5	0	1004344	2499958574	0	2511924549	4999949997

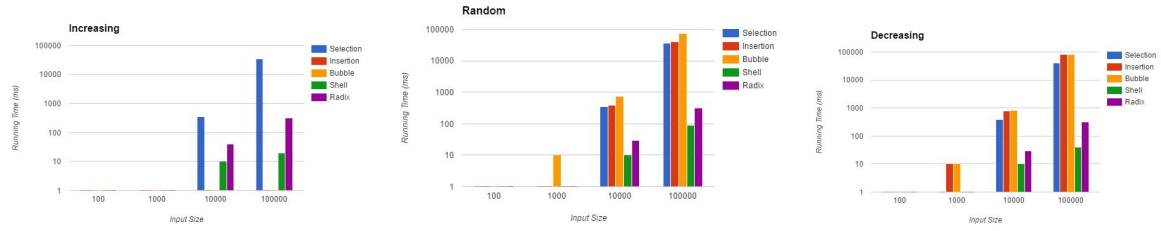
#COMP	Bubble Sort			Shell Sort		
n	inc	ran	dec	inc	ran	dec
100	0	4929	4950	0	397	260
10^3	0	499434	499500	0	7866	4700
10^4	0	49991172	49995000	0	145301	62560
10^5	0	4999807689	4999950000	0	2951871	844559

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

a.



b.



6. Theoretically we noted that the worst case was always when using a already sorted decreasing list. However, the random input data could generate to be equal to or worse than the decreasing input data. Compared to heap sort, merge sort, and quick sort which all run in O(nlogn), radix sort is slower since it runs in O(n).

7. In conclusion, these algorithms all have the best case scenerio of a already sorted increasing list as their input data with the wose case scenerio being a decreasing list. Most of the data follows what was theretically analyzed. The main difference between the algorithms is the runtime of the algorithms on different input data. For example, the radix sort and shell sort have a better runtime on the increasing, decreasing, and random

input of size 10^5. The larger the input data is, the longer the runtime of these algorithms will take

- **(20 points) Use of Git**
    1. (10 points) Host the final working version of your project on your team's GitHub repository. **This repository must be private.**
    2. (10 points) Demonstrate knowledge of branching. Create branches for at least three of the six sorting algorithms. Once the sorting algorithm is working, merge the branch with the master branch but DO NOT delete the branch used to implement the algorithm.