First Name Last Name Section #

CSCE 221: Programming Assignment 2 (200 points)

Due October 1 by 11:59 pm

Project Cover Page

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

1. Alexander Ngo, alexanderkngo@tamu.edu

2.Michael Vallejo, mvallejo@tamu.edu

3.Quinn Nguyen, quinnminh_nguyen@tamu.edu

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

1. Michael Vallejo: Selection Sort, Ifstream, Report

2. Alexander Ngo: Bubble Sort, Ifstream, Ofstream, Report

3. Quinn Nguyen: Insertion Sort, Ifstream, Ofstream, Report

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	
Web Material (give URL)	www.stackoverflow.com
Printed Material	
Other Sources	LectureSlides

I certify that I have listed all the sources that I used to develop solutions to the submitted project and code.

Your signature: Alexander Ngo Typed Name: Alexander Ngo Date: 02 October 2017

I certify that I have listed all the sources that I used to develop a solution to the submitted project and code.

Your signature: Michael Vallejo Typed Name: Michael Vallejo Date: 02 October 2017

I certify that I have listed all the sources that I used to develop solution to the submitted project and code.

Your signature: Quinn Nguyen Typed Name: Quinn Nguyen Date: 02 October 2017

Aggie Code of Honor: "An Aggie does not lie, cheat, or steal, or tolerate those who do."

STORY

Consider a situation that you have booked a flight from Houston to London. But due to unfortunate reasons, you reach the airport late. Now in this situation you look up to the board with flight details to check the status of your flight. If the flight board is not ordered in some way and random in nature it will look like this.

Flight Number	Destination	Departure Time	Gate Number
AA223	LAS VEGAS	21:15	A3
BA023	DALLAS	21:00	A3
AA220	LONDON	20:30	B4
VI303	MEXICO	19:00	A7
BA087	LONDON	17:45	A7
AA342	PARIS	16:00	A7
VI309	PRAGUE	13:20	F2
QU607	TORONTO	08:30	F2
AA224	SYDNEY	08:20	A7
AF342	WASHINGTON	07:45	A3

The above table is very tough to read. But say you can see another table which is sorted by Departure Time like this one:

Flight Number	Destination	Departure Time	Gate Number
AF342	WASHINGTON	07:45	A3
AA224	SYDNEY	08:20	A7
QU607	TORONTO	08:30	F2
VI309	PRAGUE	13:20	F2
AA342	PARIS	16:00	A7
BA087	LONDON	17:45	A7
VI303	MEXICO	19:00	A7
AA220	LONDON	20:30	B4
BA023	DALLAS	21:00	A3
AA223	LAS VEGAS	21:15	A3

This project will give you a better understanding of a sorting process.

PROJECT TECHNICALITIES

- Write a C++ program which reads the table of flights into a vector of `Flight` objects (of class `Flight`), and displays the sorted table based on the selected option: the departure time or the destination.

Steps:

1. Read the file with the flight details. Store each flight into a `Flight` class object. Create the vector of `Flight` objects and keep class members as strings, i.e., `flightNum`, `destination`, `departureTime` and `gateNum`.
2. Sort the flights using the selection, insertion, and bubble sort algorithms.

   In the sorted table, the selected sorting option (the departure time or the destination) should be in lexicographical (ascending) order.
3. Display the final sorted data in a table format.

   Remember that positions in the non-sorted columns will also change based on the sorted column. In each iteration of the sorting algorithm, elements change their position in the vector. For example in the selection sort, after the i-th iteration, the (i-1)-st element of the vector is swapped with the smallest element among i-th to n-th elements. The format of the output file is the same as the format of the input file.

Program Implementation Requirements

- Use the provided `flight.h` file to complete the `flight.cpp` file with the required fields.
  - Complete the `compareToDestination()` and `compareToDepartureTime()` functions to compare two `Flight` objects based on the departure time or destination.
- Implement the `selection_sort`, `bubble_sort`, and `insertion_sort` functions defined in the supplemental file `sort.h`.
  - These will take an arbitrary vector of `Flight` objects and sort them. These functions will also take in an argument to switch between sorting by destination or by departure time.
- Divide the problem in three steps: reading from the given input file, sorting the table, and writing the sorted table to a file.
  - You can find information about input and output streams in "*Programming. Principles and Practice Using C++.*" by B. Stroustrup, 2nd edition, Chapters 10.4-11 and 11.4-5.
  - You should use the `getline` function (from `std::string` class) with two arguments to read lines of the `csv` files:

    ```
    istream& getline (istream& is, string& str);
    ```
  - Next, use `getline` function with three arguments, where the third argument is a delimiter character, to parse a line, see this description:

    ```
    istream& getline (istream& is, string& str, char delim);
    ```

which extracts characters from `is` and stores them into `str` until the delimitation character `delim` is found. Use comma (`,`) as a delimiter.

- Insert the code that increases number of comparison `num_cmps++` typically in an `if` or a loop statement
  - This variable will be defined in `sort.h`, but you have to provide the code to manage this variable.
  - Remember that C++ uses the shortcut rule for evaluating boolean expressions. A way to count comparisons accurately is to use comma expressions. For instance,

```
while (i < n && (num_cmps++, a[i] < b)) ...
```

File Format Requirements

- The file will have the same format as given, but will have different data.

  Here is a sample:

```
FLIGHT NUMBER,DESTINATION,DEPARTURE TIME,GATE NUMBER
BC580,McMechen,12:33,Z3
ZK612,Glenrock,05:27,Y8
BF144,South Range,01:33,R4
US141,Tate,17:06,J5
JO745,Edmundson,21:07,V4
```

- Fist line will consist of the column headers.
- Each subsequent line will have a flight details.
- The cvs files will contain details for sets of 10, 100, 1000, and 10000 flights. These files are split into sets with ascending, descending, and random order.
- Flight information format:
  - Flight Number: 2 alphabetic characters followed by a three digit integer.
  - Destination: Name of a city (can be many words)
  - Departure: 24 hour format.
  - Gate Number: 1 alphabetic character followed by a single digit integer.
- Check the input file format for correctness and throw an exceptions if the format is not preserved.
- Write the sorted table to a file and display on a terminal for smaller inputs (10 flights only).

Submission Requirements for Source Code and Report

Turn in an electronic version of your C++ code, your report and testing file to eCampus as a tar file.

1. **C++ code requirements:**

(a) At the top of your source file containing the function `main()` (the first few lines) write a comment containing personal information of a group members, including course number, course section, your first and last name, and email address. If you do not provide this information, points will be deducted. This is the format which is required:

```
/*************************************
Programming Assignment 2 CSCE 221-5xx
Your Last Name(s), First Name(s)
your email(s)
*************************************/
```

(b) Use consistent indentation. Your program should be readable – select variable names wisely and document your code. Get more information about the programming style, read http://www.stroustrup.com/Programming/PPP-style-rev3.pdf

2. **The report requirements.** Return an electronic and hard copy of the report followed by the cover page including answers to provided questions.

(a) A brief description of assignment purpose, its description, the input and output files format, instructions how to run your program.

The purpose of this assignment is to prove that the theoretical number of comparisons, and its Big-O notation is experimentally correct. To do this, we first have to read in the out of order input files that were provided. Then we sorted the entries based on departure time and destination, which generally yielded the same number of comparisons. Because the input values were stored in a vector of structs, we just had to compare a certain entry in the struct, either byDepartureTime or byDestination, then we moved the entire vector entry of the struct if needed. Once the values were sorted, they were outputted to a .csv file in a similar format as the input file, which will allow it to be used by the program written again.
Both the input and output files have the same format as s .csv file. Each line in the file is used by the program as a value to a specific element in the vector. The values on each line are separated by a comma, which will be used by the program as a value in the struct in the vector. Thus, the program must read and write to the input/output files line by line, with each line corresponding to another value in the vector.
To run the program, the values in the PA2.cpp file must be changed to reflect the type of sort that the user wants. In addition, there are two types of ways to sort, byDepartureTime, and byDestination. Thus, there are 6 different combinations of sorts that can be used by the user. All the user would need to do is uncomment the line that contains the type of sort that is required, and run the program.

(b) Program design and the class definitions. Explanation of splitting the program into classes and *a description of C++ object oriented features or generic programming used in this assignment*.

The program was designed to have two different classes: a sorting class, and a flights class. The Flight class is technically designed a s Struct, which just means that all members are public by default. This class has 4 values: flight number, destination, departure time, and gate number. This Flight class is then stored in a vector that was defined in the main file. The sorting class holds our three different sorting algorithms:selection, bubble, and insertion, and wait to be called on in this class. Because the program is split into these different classes, it makes storing and reading values more efficient as it does not conflict with the sorting. The ifstream and ofstream only interacts with the Flight class to ensure that the values are stored in the vector correctly, while the rest of the main file utilizes the sorting class to perform the main operations of this program. The only C++ object oriented features that were used was the output operator overloading, which was used to output the sorted table to the terminal.

(c) C++ object oriented features like input/output operators overloading or constant member functions.

The answer for this can be found in part b above.

(d) **Algorithms.** Briefly describe the features of each of the sorting algorithms.

The bubble sort algorithm has two different cases: one to sort by Destination and one to sort by Departure Time. In both cases, the algorithm is exactly the same, only the data that it reads is different. Bubble sort works by getting the largest value in the array to the very right most, or largest element in the array. In the case of my code, first the code looks at the first element in the array, and compares either the destination or the departure time to the element to the right of it. If the value in in the left element is greater than the one to the right, the values will be swapped. This repeats until the largest value in the array is in the largest element of the array. Once this happens, the code resets and goes back to the first element in the array, and repeats the process. Once this is done, it should throw a True Flag. When a True Flag is seen, the code will break, signifying that the array is now in order. This condition is necessary so that the user knows exactly when the program finished, because the ending condition for the loop is when it performs $\frac{n(n-1)}{2}$ comparisons, which would not be ideal for arrays that were already in order.
The selection sort algorithm has one case to sort by destination and a second case that sorts by departure time. Each case is implemented similarly as the implementation consists of a double for loop that iterates m times through the vector where m is the size of the flights vector and each iteration finds the minimum valued flight (based on either destination or departure time) and moves its index to the current i value, switching it with the flight that was initially there. Essentially, each iteration of the for loop adds one flight to the sorted part of the list by finding the minimum flight left in the unsorted portion and putting it at the end of the sorted portion.
The insertion sort transverse through the arrays. It chooses the current element and compare the previous one by one. If the current element is smaller than the previous element, we swap. We swap until we "insert" the current element into the correct order or it is bigger than previous one.

(e) **Theoretical Analysis.** Provide the theoretical number of comparisons perform on elements of the vector and big-O notation of the sorting algorithms on input of size n in decreasing (dec), random (ran) and increasing (inc) orders and fill the tables below.

# of comps.	best	average	worst
Selection Sort	$\frac{n(n+1)}{2}$	$\frac{n(n+1)}{2}$	$\frac{n(n+1)}{2}$
Insertion Sort	$(n-1)$	$\frac{n(n+1)}{2}$	$\frac{n(n+1)}{2}$
Bubble Sort	$(n-1)$	$\frac{n(n+1)}{2}$	$\frac{n(n+1)}{2}$

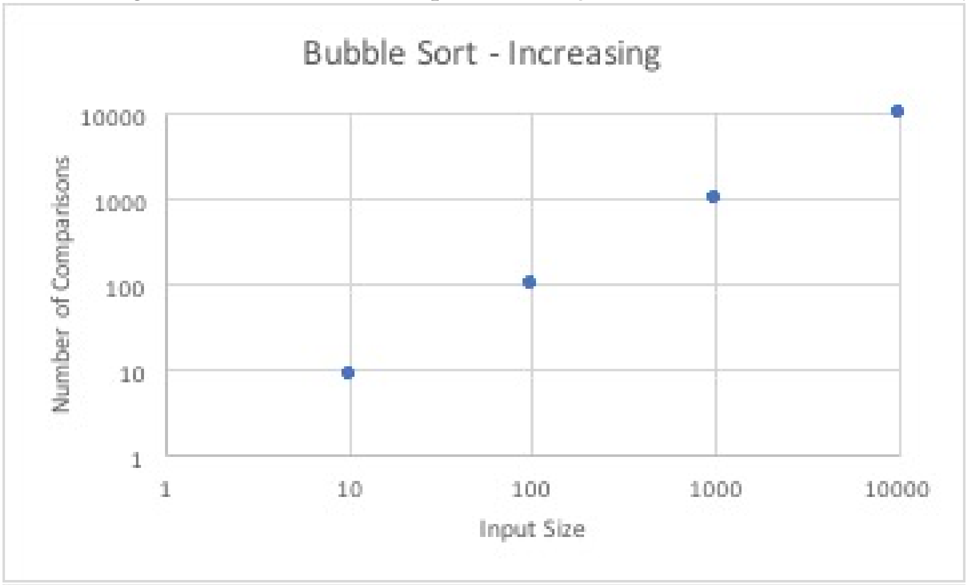
big-O notation	inc	ran	dec
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$

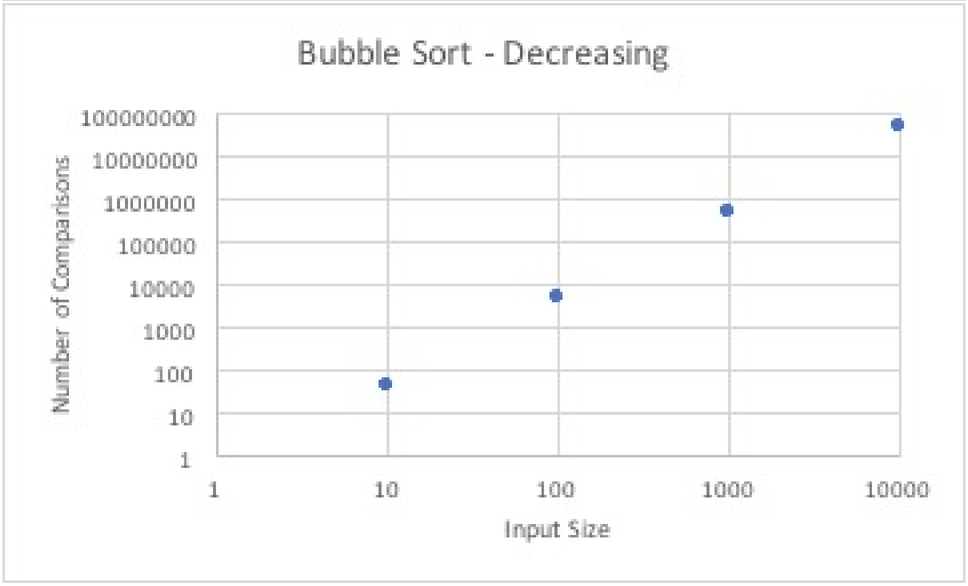
(f) **Experiments and Statistics.** Briefly describe the experiments. Present the number of comparisons (**#COMP**) performed on input data using the following tables.

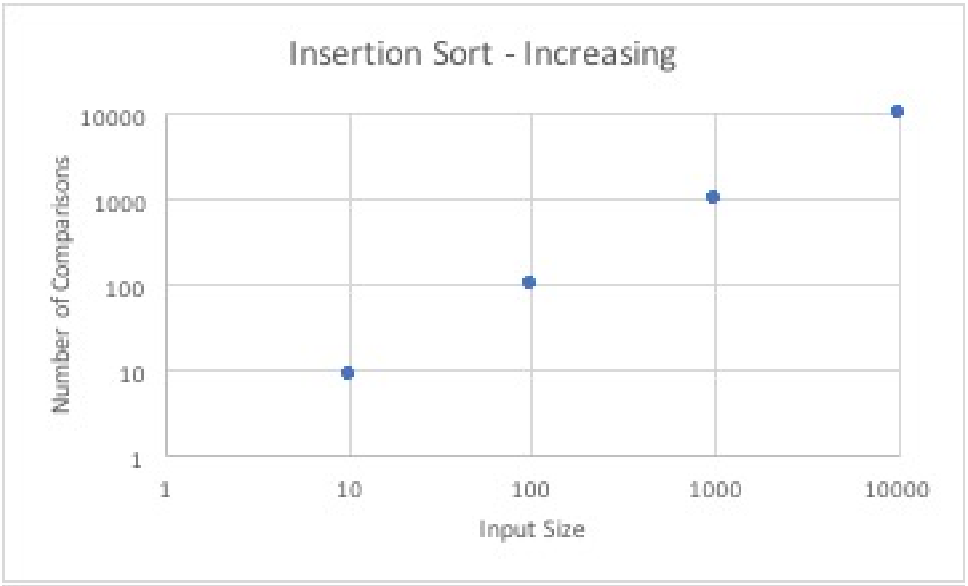
	Selection Sort			Bubble Sort			Insertion Sort		
n	inc	ran	dec	inc	ran	dec	inc	ran	dec
10	45	45	45	9	45	45	9	32	45
10^2	4950	4950	4950	99	4895	4950	99	2911	4950
10^3	499500	499500	499500	999	499175	499500	999	238241	499500
10^4	49995000	49995000	49995000	9999	49993919	49995000	9999	25098813	49995000

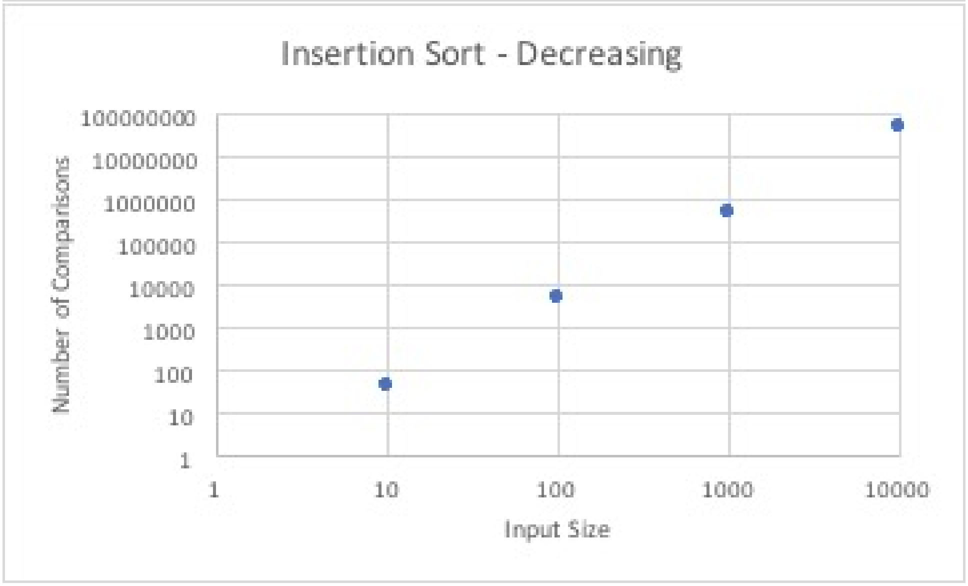
inc: increasing order; dec: decreasing order; ran: random order

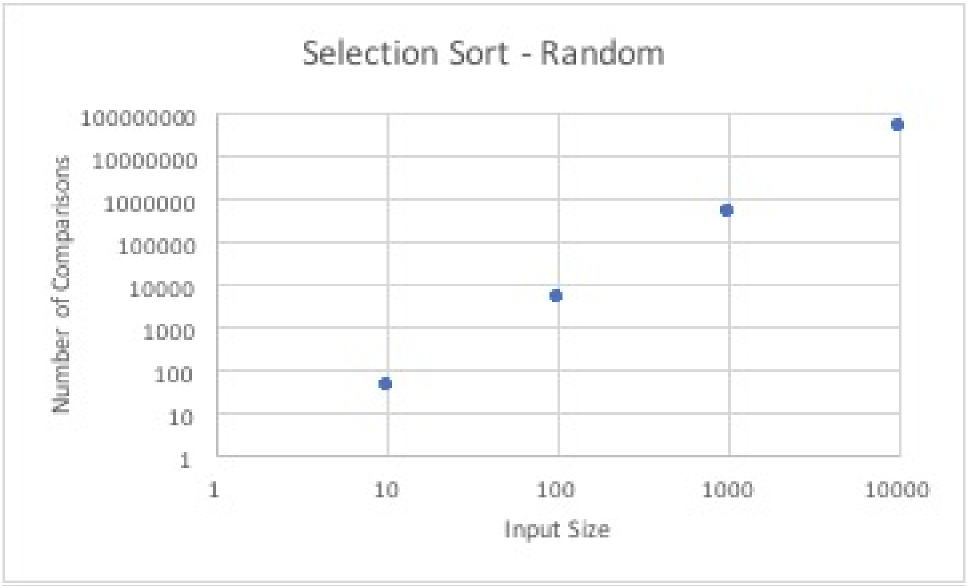
(g) For each sorting algorithm, graph the number of comparisons versus the input size $n = 10^4$, totaling in 9 graphs over the three input cases (inc, ran, dec) versus the input size for the three algorithms.
HINT: To get a better view of the plots, *use logarithmic scales* for both x and y axes.

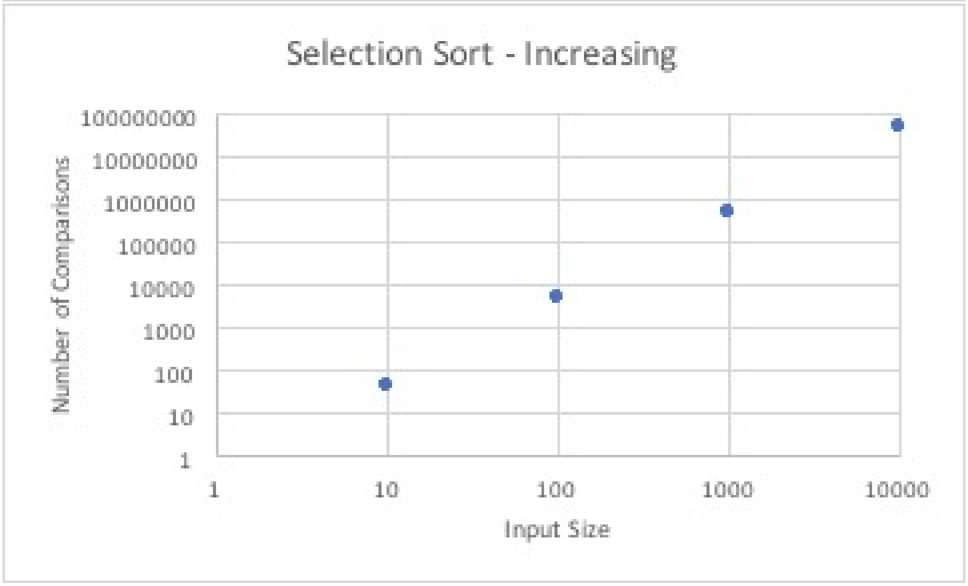


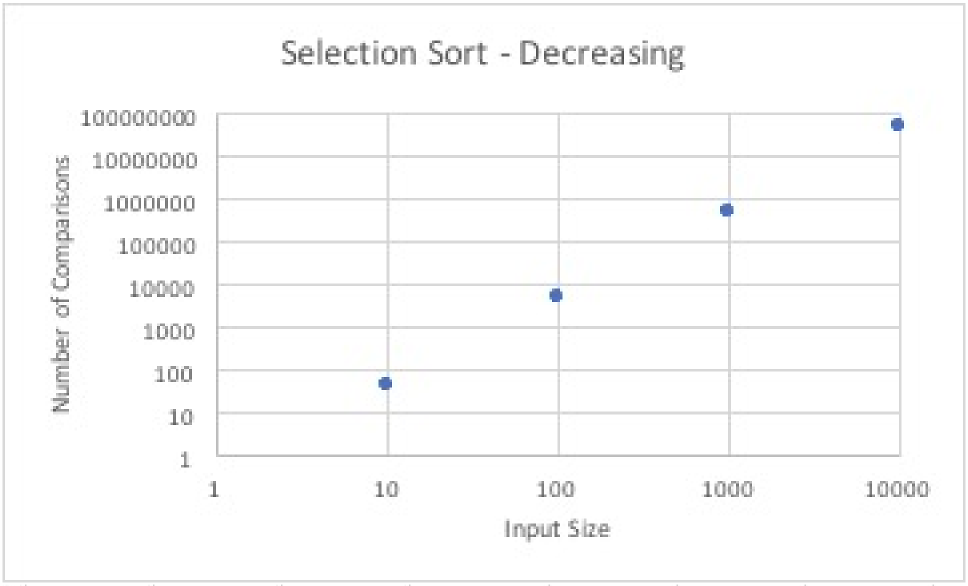


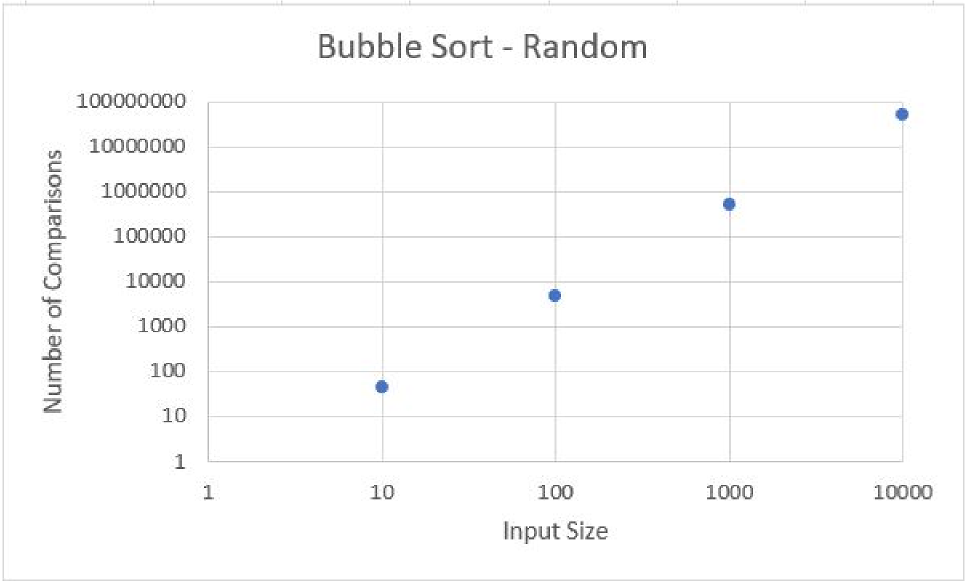

Insertion Sort - Increasing
Number of Comparisons
10000
1000
100
10
1
1
10
100
1000
10000
Input Size

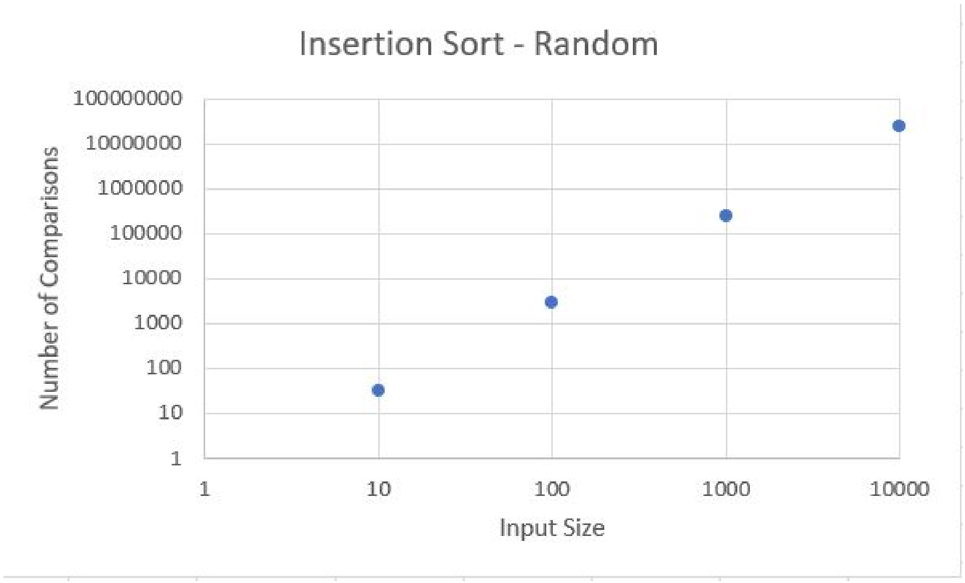

Insertion Sort - Decreasing
Number of Comparisons
100000000
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
Input Size


Selection Sort - Random
Number of Comparisons
100000000
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
Input Size


Selection Sort - Increasing
Number of Comparisons
100000000
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
Input Size

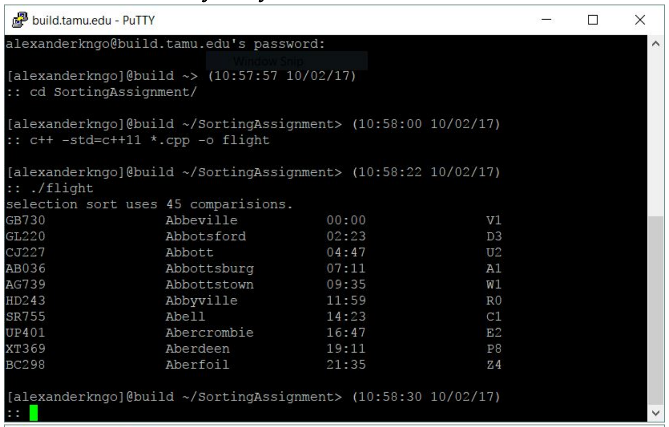

Selection Sort - Decreasing
Number of Comparisons
100000000
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
Input Size


Bubble Sort - Random
Number of Comparisons
100000000
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
Input Size



(h) Testing cases for $n = 10$, supported by screen shots (remember to use white background to save ink)

(i) **Discussion.** Comment on how the experimental results relate to the theoretical analysis and explain any discrepancies you note. Do your computational results match the theoretical analysis you learned from class or the textbook? Justify your answer.



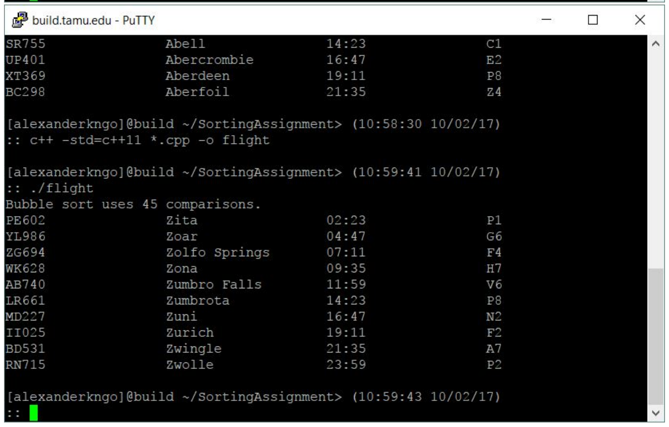
```
alexanderkngo@build.tamu.edu's password:

[alexanderkngo]@build ~> (10:57:57 10/02/17)
:: cd SortingAssignment/

[alexanderkngo]@build ~/SortingAssignment> (10:58:00 10/02/17)
:: c++ -std=c++11 *.cpp -o flight

[alexanderkngo]@build ~/SortingAssignment> (10:58:22 10/02/17)
:: ./flight
selection sort uses 45 comparisions.
GB730               Abbeville           00:00               V1
GL220               Abbotsford          02:23               D3
CJ227               Abbott              04:47               U2
AB036               Abbottsburg         07:11               A1
AG739               Abbottstown         09:35               W1
HD243               Abbyville           11:59               R0
SR755               Abell               14:23               C1
UP401               Abercrombie         16:47               E2
XT369               Aberdeen            19:11               P8
BC298               Aberfoil            21:35               Z4

[alexanderkngo]@build ~/SortingAssignment> (10:58:30 10/02/17)
::
```

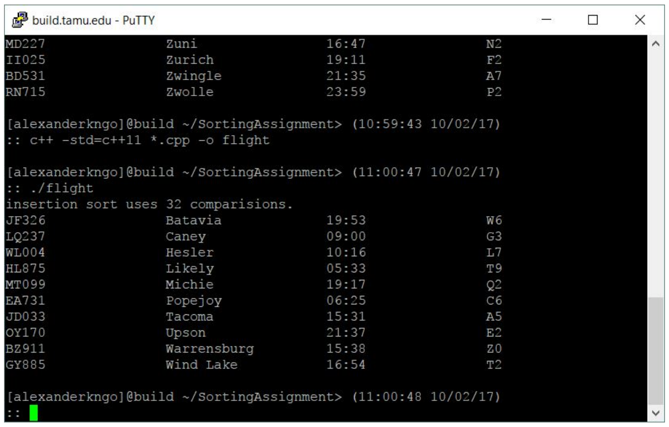


```
SR755               Abell               14:23               C1
UP401               Abercrombie         16:47               E2
XT369               Aberdeen            19:11               P8
BC298               Aberfoil            21:35               Z4

[alexanderkngo]@build ~/SortingAssignment> (10:58:30 10/02/17)
:: c++ -std=c++11 *.cpp -o flight

[alexanderkngo]@build ~/SortingAssignment> (10:59:41 10/02/17)
:: ./flight
Bubble sort uses 45 comparisons.
PE602               Zita                02:23               P1
YL986               Zoar                04:47               G6
ZG694               Zolfo Springs       07:11               F4
WK628               Zona                09:35               H7
AB740               Zumbro Falls        11:59               V6
LR661               Zumbrota            14:23               P8
MD227               Zuni                16:47               N2
II025               Zurich              19:11               F2
BD531               Zwingle             21:35               A7
RN715               Zwolle              23:59               P2

[alexanderkngo]@build ~/SortingAssignment> (10:59:43 10/02/17)
::
```



```
MD227               Zuni                16:47               N2
II025               Zurich              19:11               F2
BD531               Zwingle             21:35               A7
RN715               Zwolle              23:59               P2

[alexanderkngo]@build ~/SortingAssignment> (10:59:43 10/02/17)
:: c++ -std=c++11 *.cpp -o flight

[alexanderkngo]@build ~/SortingAssignment> (11:00:47 10/02/17)
:: ./flight
insertion sort uses 32 comparisions.
JF326               Batavia             19:53               W6
LQ237               Caney               09:00               G3
WL004               Hesler              10:16               L7
HL875               Likely              05:33               T9
MT099               Michie              19:17               Q2
EA731               Popejoy             06:25               C6
JD033               Tacoma              15:31               A5
OY170               Upson               21:37               E2
BZ911               Warrensburg         15:38               Z0
GY885               Wind Lake           16:54               T2

[alexanderkngo]@build ~/SortingAssignment> (11:00:48 10/02/17)
::
```

The experimental results exactly matches that of the theoretical results in the above table for the increasing(best) case and decreasing(worst) case. The random cases closely matches that of the average case, only being off by about a couple hundred or so. Because the theoretical result is only an average case, our experimental results fall into a a good range, which helps to prove the theoretical results, and ultimately prove that the Big-O notation for each sort for each scenario is correct.

(j) **Conclusions.** Give your observations and conclusion. For instance, which sorting algorithm is more suitable for a specific input data? What factors can affect your experimental results?

In the case that the list of flights is already sorted (the ascending files), insertion sort or selection sort are more efficient as they would go through once and find that the list is already sorted, thus making them O(n) instead of O(n^2). Selection sort, however, is the same for best, average, and worst case as all of them have the same number of comparisons and O(n^2). In the depending case, each algorithm results in the same number of comparisons. These sorting algorithms are certainly faster for smaller inputs, but for larger inputs, it would be more efficient to use counting algorithms. In files that are randomly sorted, how random they are can cause variations in experimental results because the run time can be slightly improved or worsened depending on if the random file is more or less sorted.

Turning In

1. Use `tar` program to bundle all your files in one file.
2. Do not turn in the object files and/or executable file.
3. Note: Late projects are penalized according to the weights provided in the syllabus.
4. If your program does not compile on a Linux machine or if there are run-time errors, you will get at most 50% of the total number of points.