Project Cover Page

This project is a group project with up to four students per team. For each group member, please print first and last name and e-mail address.

1. Isaac Rollo, Isaac_rollo@tamu.edu
2. Syed Abeer Zaidi, abeerz97@tamu.edu
3.
4.

Please write how each member of the group contributed to the project.

1. Coded the insertion sort, shell sort, and reworked the sort.cpp file.
2. Coded the bubble sort, radix sort, selection sort, and reworked the sort.cpp file.
3.
4.

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)	
Printed Material	
Other Sources	

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

1. Your signature	Isaac Rollo	02/17/19
2. Your signature	Syed Abeer Hasan Zaidi	02/17/2019
3. Your signature	Typed Name	Date
4. Your signature	Typed Name	Date

CSCE 221 Programming Assignment 2 (100 points)

Program and Reports due February 17th by 11:59pm

- **Objective**

  In this assignment, you will implement in C++ five sorting algorithms: selection sort, insertion sort, bubble sort, Shell sort, and radix sort. You will test your code using varied input cases, record computational time and the number of comparisons in each sort algorithm, and compare these computational results with the theoretical running time of the algorithms using Big-O asymptotic notation.

- **General Guidelines**

  1. This project can be done in groups of at most four students. Please use the cover sheet at the previous page for your report.
  2. The supplementary program is packed in `221-A2-code.tar` which can be downloaded from eCampus. You need to "untar" the file using the following command on a Linux machine:

     ```
     tar xfv 221-A2-code.tar
     ```

     It will create the directory `221-A2-code`.
  3. Make sure that your code can be compiled using a C++ compiler running on a Linux machine before submission because your programs will be tested on such a machine. Use `Makefile` provided in the directory to compile C++ files by typing the following command:

     ```
     make
     ```

     You may clean your directory with this command:

     ```
     make clean
     ```
  4. When you run your program on a Linux machine, use Ctrl+C (press Ctrl with C together) to stop the program. Do NOT use Ctrl+Z, as it just suspends the program and does not kill the process. We do not want to see the department server down because of this assignment.
  5. Supplementary reading
     (a) Lecture note: Introduction to Analysis of Algorithms
     (b) Lecture note: Sorting in Linear Time
  6. Submission guidelines
     (a) Electronic copy of all your code tested on 15 types of input integer sequences (but do not submit your test files), and reports in LyX/LaTeX and PDF format should be in the directory `221-A2-code`. This command typed in the directory `221-A2-code` will create the tar file (`221-A2-code-submit.tar`) for the submission to eCampus:

     ```
     make tar
     ```

     (b) Your program will be run and tested on TA's input files.

- **Skeleton Code: Description and Implementation**

  1. In this assignment, the sort program reads a sequence of integers either from the screen (standard input) or from a file, and outputs the sorted sequence to the screen (standard output) or to a file. The program can be configured to show total running time and/or total number of comparisons done by the specific sort algorithm.
  2. (*20 points*) Rewrite the provided code using the STL vector instead of the dynamic array `A` used in the code. Check if your new code compiles.
  3. This program does not have a menu but takes arguments from the command line. The code for interface is completed in the skeleton program, so you only have to know how to execute the program using the command line.

The program usage is as follows. *Note that options do not need to be specified in a fixed order. You do not need to list all the options in the command line.* ***Do not enter brackets*** *(see Examples).*

Usage:

```
./sort [-a ALGORITHM] [-f INPUTFILE] [-o OUTPUTFILE] [-h] [-d] [-p]
       [-t] [-T] [-c] [-r]
```

Examples of using the options:

```
./sort -h
./sort -a S -f input.txt -o output.txt -d -t -c -p
./sort -a I -t -c
./sort
```

Description of the options:

```
-a ALGORITHM: Use ALGORITHM to sort.
   ALGORITHM is a single character representing an algorithm:
   S for selection sort
   B for bubble sort
   I for insertion sort
   H for shell sort
   R for radix sort
-f INPUTFILE: Obtain integers from INPUTFILE instead of STDIN
-o OUTPUTFILE: Place output data into OUTPUTFILE instead of STDOUT
-h: Display this help and exit
-d: Display input: unsorted integer sequence
-p: Display output: sorted integer sequence
-t: Display running time of the chosen algorithm in milliseconds
    using clock()
-T: Display running time of the chosen algorithm in milliseconds
    using the chrono class
-c: Display number of comparisons (excluding radix sort)
-r: Provide the number of repeated runs for the sort (default=1)
```

4. **Format of the input data.** The first line of the input contains a number n which is the number of integers to sort. Subsequent n numbers are written one per line which are the numbers to sort. Here is an example of input data where 5 is the number of integers to sort

```
5
7
-8
4
0
-2
```

5. **Format of the output data.** The sorted integers are printed one per line in increasing order. Here is the output corresponding to the above input:

```
-8
-2
0
4
7
```

6. (*25 points*) Your tasks include implementation of the following five sorting algorithms in corresponding cpp files.

   (a) selection sort in `selection-sort.cpp`

   (b) insertion sort in `insertion-sort.cpp`

   (c) bubble sort in `bubble-sort.cpp`

   (d) Shell sort in `shell-sort.cpp`

(e) radix sort in `radix-sort.cpp`

   i. The radix algorithm should sort unsigned integers in the range from 0 to $(2^{16} - 1)$.

   ii. The radix sort can be applied to negative numbers using this input modification: "You can shift input to get non-negative numbers, sort the data, and shift back to the original data"

7. (*10 points*) In order to test the algorithms generate (one by one) input cases of the sizes 10^2, 10^3, 10^4, and 10^5 integers in three different orders

   (a) random order

   (b) increasing order

   (c) decreasing order

   HINT: The standard library `<cstdlib>` provides functions `srand()` and `rand()` to generate random numbers.

8. (*10 points*) Modify code (excluding the radix sort) to count the number of ***comparisons performed on input integers***. The following tips should help you with determining how many comparisons are performed.

   (a) You will measure 3 times for each algorithm (use `-r 3`) on each sequence

   (b) Use a variable (called `num_cmps`) to keep track of the number of comparisons, typically in `if` or loop statements

   (c) Remember that C++ uses the shortcut rule for evaluating boolean expressions. A way to count comparisons accurately is to use the comma expression. For example,

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

   HINT: If you modify `sort.cpp` and run several sorting algorithms subsequently, you have to call `resetNumCmps()` to reset number of comparisons between every two calls to `s->sort()`.

9. To time each sorting algorithm, you use the function `clock()` and the C++ class chrono (see the options `-t` and `-T`). You need to compare these two timings and discuss it in your report.

   (a) Here is the example of how to use the function `clock()`. The timing part for `clock()` is completed in the skeleton program.
   The example of using `clock()`:

   ```
   #include <ctime>
   ...
   clock_t t1, t2;
   t1 = clock(); // start timing
   ...
   /* operations you want to measure the running time */
   ...
   t2 = clock(); // end of timing
   double diff = (double)(t2 - t1)*1000/CLOCKS_PER_SEC;
   cout << "The timing is " << diff << " ms" << endl;
   ```

   (b) You can find information about the class chrono on this website:
   http://www.cplusplus.com/reference/chrono/

- **Report (35 points)**

  Write a report that includes all following elements in your report.

  1. (2 points) A brief description of assignment purpose, assignment description, how to run your programs, what to input and output.

     The purpose of this assignment was to test our knowledge of sorting algorithms and how to implement them using vectors. To run the program just compile and do the output call. The usage information will show up and indicate to the user how to operate the program.

  2. (3 points) Explanation of splitting the program into classes and providing *a description and features of C++ object oriented programming and/or generic programming were used in this assignment.*

This assignment required the use of inheritance in the form of using a parent class sort that holds the common features of all the sort subclasses. This helps reduce the redundancy of writing a new class for each sort type and makes declerations easier.

3. (5 points) **Algorithms.** Briefly describe the features of each of the five sorting algorithms.

   Insertion Sort: Insertion sort sorts the vector using two subvectors, a sorted part and an unsorted part. The sort iterates through the vector expanding the sorted part one by one, when it iterates it takes the new value and compares it to the previous values that have been sorted. When it finds a value that is larger than the new value it will insert the new value in and delete it's old position. Then the sort will iterate to the next value, until the entire vector is sorted.

   Shell Sort: Shell sort divides the vector into smaller subvectors that are then sorted using the the same technique as insertion sort. Once they are all sorted, Shell sort will then divide the entire vector into subvectors again, this time bigger than the previous subvectors, and sort them using insertion sort. It will continue this, until the subvector is the size of the entire vector and the entire vector has been sorted.

   Bubble Sort: Bubble sort finds the maximum value in a vector and pushes it to the back of the vector using a series of comparisons and swaps. If a swap isn't made in one loop of the vector, at any point in the bubble sort, this indicates that the vector is sorted and the bubble sort ends.

   Selection Sort: Selection Sort makes a second vector of equal size to the original and then searches for the smallest value in the original vector. Once it finds that element, it pushes it to the top of the second vector and deletes it from the original vector. No matter how the inputs are arranged, a selection sort algorithm will perform the same number of comparison operations

   Radix Sort: Radix Sort divides a vector into various vectors by the number of digits in the maximum value of the vector. It then uses Counting Sort to sort the vector at each digit, increasing the magnitude of the digit and performing another counting sort call in each loop.

1. (5 points) **Theoretical Analysis.** Theoretically analyze the time complexity of the sorting algorithms with input integers in decreasing, random and increasing orders and fill the second table. Fill in the first table with the time complexity of the sorting algorithms when inputting the best case, average case and worst case. Some of the input orders are exactly the best case, average case and worst case of the sorting algorithms. State what input orders correspond to which cases. You should use big-O asymptotic notation when writing the time complexity (running time).

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)	O(n^2)	O(n^2)
Radix Sort	O(n)	O(nk)	O(n^2)

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

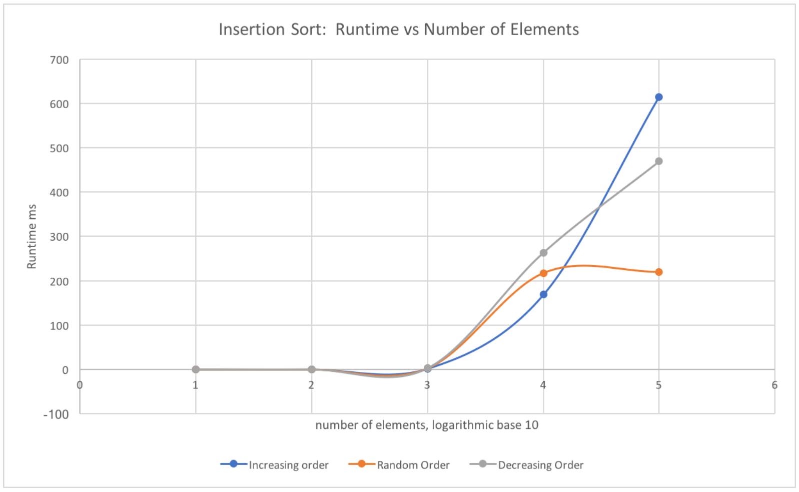
inc: increasing order; dec: decreasing order; ran: random order

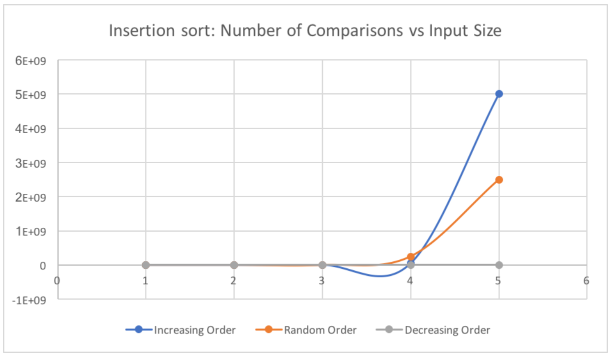
(a) (15 points) **Experiments.**

i. 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			Insertion Sort			
n	inc	ran	dec	inc	ran	dec	inc
100	.0220 ms	.0235 ms	.0230 ms	0.0215 ms	.0285 ms	0.0326 ms	.0030
10^3	1.7068 ms	1.9712 ms	1.9310 ms	1.9185 ms	2.5980 ms	2.9373 ms	.0256
10^4	172.3095 ms	171.8544 ms	169.1161 ms	168.4140 ms	217.1626 ms	263.1523 ms	0.1535
10^5	-111.4981 ms	-7.96561 ms	-134.2975 ms	-410.3598 ms	220.0280 ms	468.9710 ms	1.4802

RT	Shell Sort			Radix Sort		
n	inc	ran	dec	inc	ran	dec
100	.0223 ms	.0289 ms	0.0336 ms	.0237 ms	.02734 ms	.0277 ms
10^3	1.9268 ms	2.3802 ms	2.8838 ms	.2703 ms	.08459 ms	.3674 ms
10^4	166.2680 ms	214.5288 ms	264.2654 ms	3.5687 ms	9.3145 ms	4.2612 ms
10^5	-560.2227 ms	196.9544 ms	-713.6330 ms	37.8410 ms	74.0488 ms	44.7752 ms





#COMP	Selection Sort			Insertion Sort		
n	inc	ran	dec	inc	ran	dec
100	4950	4950	4950	4950	2769	99
10^3	499500	499500	499500	499500	251848	999
10^4	4.9995e+07	49995000	4.9995e+07	4.9995e+07	2.5141e+07	9999
10^5	4.99995e+09	4.99995+e09	4.99995+e09	4.99995e+09	2.49985e+09	99999

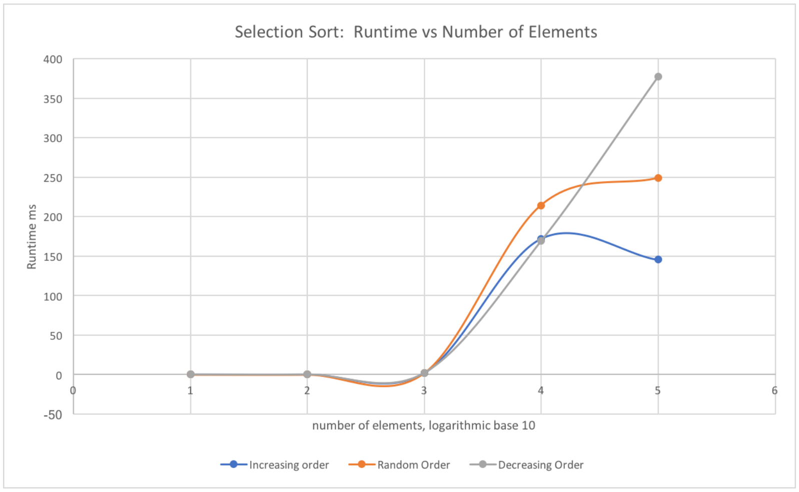
#COMP	Bubble Sort			Shell Sort		
n	inc	ran	dec	inc	ran	dec
100	99	8019	9900	4950	2769	99
10^3	999	948051	999000	499500	251848	999
10^4	9999	9.90301e+07	9.999e+07	4.9995e+07	2.5141e+07	9999
10^5	99999	9.9412e+09	9.9999e+09	4.99995e+09	2.49985e+09	99999

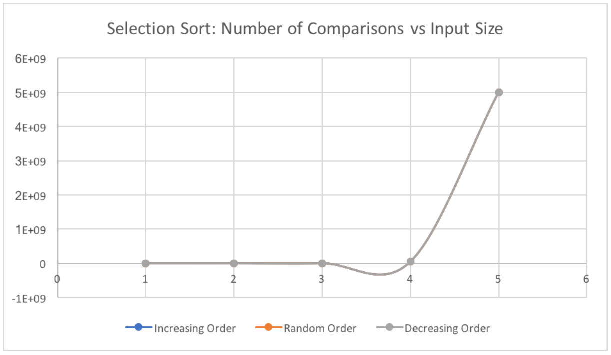
inc: increasing order; dec: decreasing order; ran: random order

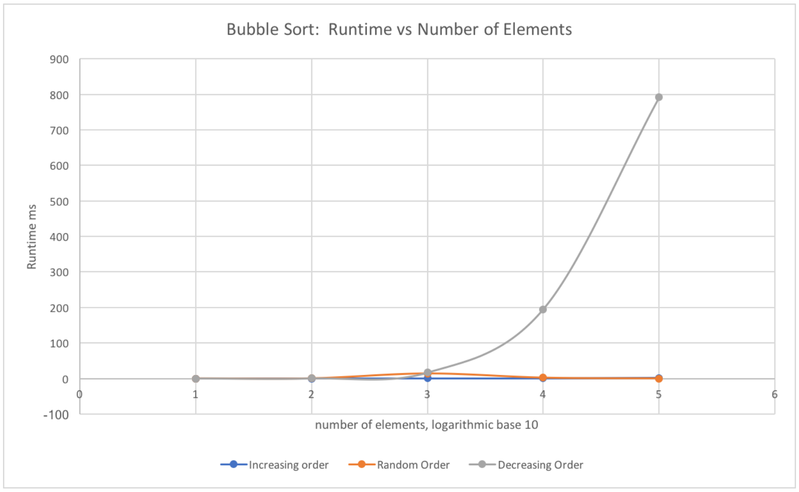
ii. For each of the five sort algorithms, graph the running times over the three input cases (inc, ran, dec) versus the input sizes (n); and for each of the first four algorithms graph the numbers of comparisons versus the input sizes, totaling in 9 graphs.
HINT: To get a better view of the plots, *use logarithmic scales* for both x and y axes.

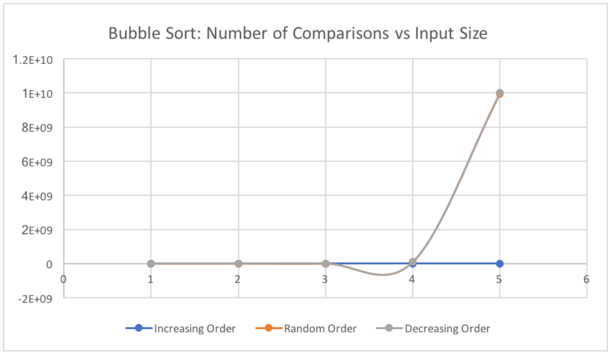
i. To compare performance of the sorting algorithms you need to have another 3 graphs to plot the results of all sorts for the running times for *each* of the input cases (inc, ran, dec) separately. HINT: To get a better view of the plots, *use logarithmic scales* for both x and y axes.

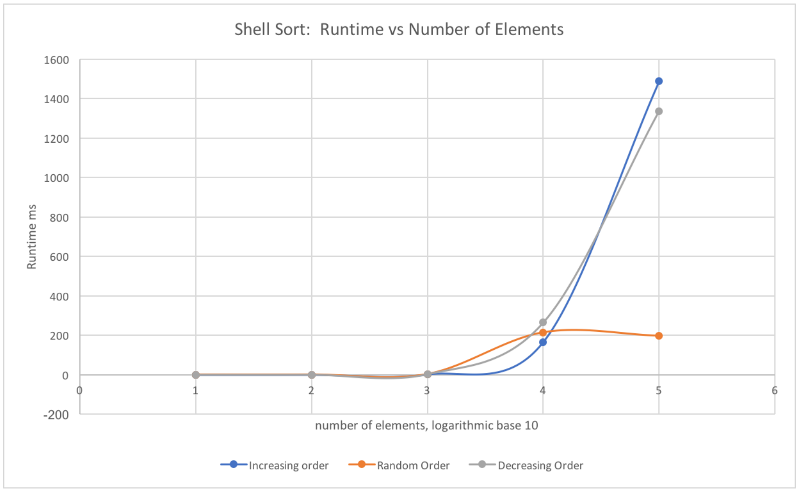
i. (3 points) **Discussion.** Comment on how the experimental results relate to the theoretical analysis and explain any discrepancies you notice. Is your computational results match the theoretical analysis you learned from class or textbook? Justify your answer. Also compare radix sort running time with the running time of four comparison-based algorithms.

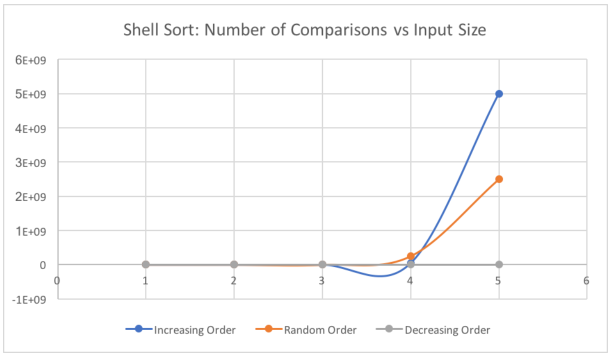

Selection Sort: Runtime vs Number of Elements
Runtime ms
400
350
300
250
200
150
100
50
0
-50
0
1
2
3
4
5
6
number of elements, logarithmic base 10
Increasing order
Random Order
Decreasing Order

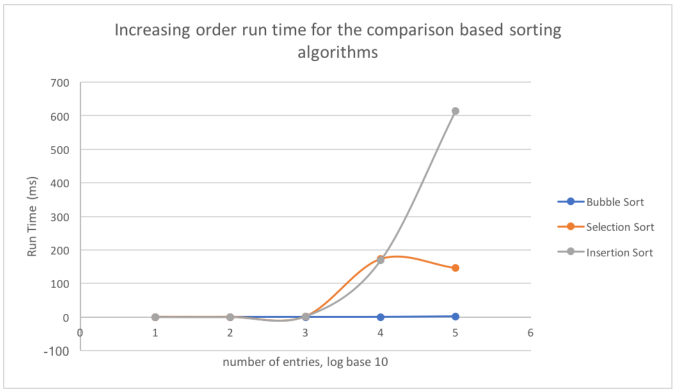

Selection Sort: Number of Comparisons vs Input Size
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
-1E+09
0
1
2
3
4
5
6
Increasing Order
Random Order
Decreasing Order

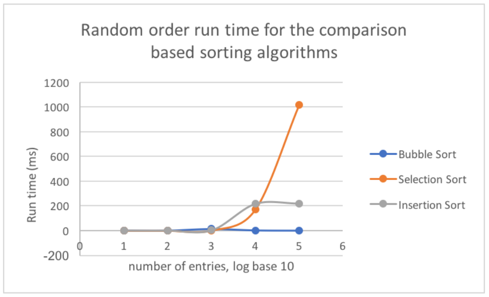

Bubble Sort: Runtime vs Number of Elements
Runtime ms
900
800
700
600
500
400
300
200
100
0
-100
0
1
2
3
4
5
6
number of elements, logarithmic base 10
Increasing order
Random Order
Decreasing Order

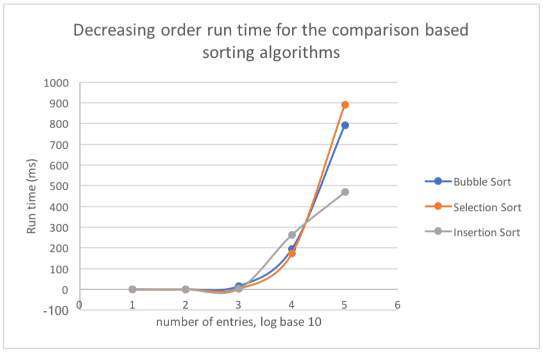

Bubble Sort: Number of Comparisons vs Input Size
1.2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
-2E+09
0
1
2
3
4
5
6
Increasing Order
Random Order
Decreasing Order


Shell Sort: Runtime vs Number of Elements
Runtime ms
number of elements, logarithmic base 10
Increasing order
Random Order
Decreasing Order


Shell Sort: Number of Comparisons vs Input Size
Increasing Order
Random Order
Decreasing Order


Increasing order run time for the comparison based sorting algorithms
Run Time (ms)
number of entries, log base 10
Bubble Sort
Selection Sort
Insertion Sort


Random order run time for the comparison based sorting algorithms
Run time (ms)
number of entries, log base 10
Bubble Sort
Selection Sort
Insertion Sort



The experimental results seem to follow the same pattern as the theoretical analysis and what we learned in class, for the most part. The bubble sort results seem to be random with regards to the runtime, however the number of comparisons performed is in-line with what we expected based on both the theoretical analysis and what we learned in class.

ii. (2 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?
Shell sort and insertion sort worked best with small, unsorted vectors and with vectors sorted in descending order. Bubble sort worked best withsmallers vectors. Radix sort worked best with large unsorted vectors. These results agreed with our predictions based on what we had learned in class. The run-time results were slightly distorted by the machine that they were tested on.