CMPT 225, Spring 2018, Assignment 1

Due date: January 23, 2018, 5:30 PM (before class)

1. Programming exercise. Translate the pseudocode for insertion sort that we saw in class into a program that is able to query a user for unsigned integers, one at a time, store them in an array, sort the array, and print out the sorted result to standard output. Your code should work correctly on any input of up to 10 integers. You may assume the user will only input actual unsigned integers, not any other type of data. **(35 pts.)**

   Details and suggestions:

   To query the user, use the *cin* object. Get them to enter one number or "E" to end.

   Allocate space for the array ahead of time.

   Warn the user when the number of integers provided reaches 10; accept no more.

   Follow the pseudocode seen in class to implement insertion sort.

   Print out the sorted array, one number per line, using the *cout* object.

   Do not print out anything in between the numbers in the sorted array.

   Design your code modularly, separating the input, output, and sorting.

   Test cases (you must make sure your program works on these):

   When the user inputs 5 your code should output

   5

   When the user inputs 93, 57 your code should output

   57

   93

   When the user inputs 25, 123, 31, 235 your code should output

   25

   31

   123

   235

   When the user inputs 3, 19, 2, 45, 8, 15, 63 your code should output

   2

   3

   8

   15

   19

   45

   63

2. Programming exercise. Extend the code you wrote in the previous exercise to work with unsigned characters as well as unsigned integers. Start by querying the user whether they would like to input integers (by entering "I") or characters (by entering "C"). Then proceed exactly as described above. Your code should work correctly on any input of up to 10 integers or characters. You may assume the user will only input actual unsigned integers or characters, not any other type of data. **(25 pts.)**

   Details and suggestions:

   Start by asking the user to input "I" to sort integers or "C" to sort characters.

   You may assume that the user will only input either "C" or "I" (bonus: check it!).

   Depending on their answer, you will need to use different array types.

   If you designed your code modularly, very little additional code will be required.

   Additional test cases (you must make sure your program works on these if the user enters "C", and all the test cases in the previous exercise if the user enters "I"):

   When the user inputs L your code should output

   L

   When the user inputs q, Q your code should output

   Q

   q

   When the user inputs 2, 1, 3, 5 (these are also characters!) your code should output

   1

   2

   3

   5

   When the user inputs C, i, L, w, B, z, T, your code should output

   B

   C

   L

   T

   i

   w

   z

   Reminder: for exercises 1 and 2, you must submit your source code with the name First_Last_N.cpp, where First is your first name, Last is your last name, and N is the exercise number (in this case 1 and 2, respectively). If your source code does not compile without errors you will receive a grade of 0 for the exercise. You will lose 5 marks for each test case that fails to produce the correct output. The test cases will not be limited to only those listed above, so hard-coding them will not help!

3. This question is to give one an idea of the rate at which functions grow. For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds. Note that $\log n$ means the logarithm in base 2 of n. Some entries have already been filled to get you started. **(5 pts.)**

	1 Second	1 Hour	1 Month	1 Century
$\log n$	$\approx 10^{300000}$			
$\sqrt{n}$				
n				
$n \log n$				
n^2				
2^n	20			

4. Suppose it is known that the running time of an algorithm is $(1/3)n^2 + 6n$, and that the running time of another algorithm for solving the same problem is $111n - 312$. Which one would you prefer, assuming all other factors equal? *Hint:* Your choice may depend on the input size n. **(5 pts.)**

5. Al and Bill are arguing about the performance of their sorting algorithms. Al claims that his $O(N \log N)$-time algorithm is *always* faster than Bill's $O(N^2)$-time algorithm. To settle the issue, they implement and run the two algorithms on many randomly generated data sets. To Al's dismay, they find that if $N < 1000$ the $O(N^2)$-time algorithm actually runs faster, and only when $N \geq 1000$ the $O(N \log N)$-time one is better. Explain why the above scenario is possible. You may give numerical examples. **(10 pts.)**

6. Suppose $f(n) \in O(h(n))$ and $g(n) \in O(h(n))$.

   (a) Is $f(n) + g(n) \in O(h(n))$?

   (b) Is $f(n) \cdot g(n) \in O(h(n))$?

   Prove your answers using the mathematical definition of O seen in class. **(10 pts.)**

7. Show by counterexample that the following statement is false: For any positive constant c, $f(cn) \in O(f(n))$. **(5 pts.)**

8. A sorting algorithm is said to be *stable* if it does not exchange the relative positions of items with equal values. More formally, a sorting algorithm is stable if, for any two elements a and b with equal values, a precedes b *after* the sorting if and only if a precedes b *before* the sorting. Given this definition, is the insertion sort algorithm which was covered in class a stable sorting algorithm? Prove your answer. **(5 pts.)**