CMPS 261
Fall 2015
Programming Project #3

DUE DATE: Wednesday March 29, @ 23:55

This programming project will be an investigation into various sorting methods and their performance.

INPUT: The program will read data from a file named randomNumbers.data. The file will contain a series of 10,000,000 random numbers in the range 1 to 10,000. There will be multiple numbers per line separated by a space.

Processing:

Step 1:

Create a sort class. The name of the class will be "allSorts". It will include the following methods:

- Perform insertion sort on a dynamic array of integer values.
- Perform heap sort on a dynamic array of integer values. In order to do this, you will need to write your own heap class as well. Refer to tree notes for that information.
- Perform merge sort on a dynamic array of integer values.
- Perform quick sort on a dynamic array of integer values. The Quicksort will use the median-of-three partitioning method.
- A private method that will take a pointer to a dynamically allocated array and return true if the values are in ascending order and false if aren't.

Many of these sorts seem difficult, but there is extensive pseudocode on all of the topics and you can manually run through them to get a good idea of how they work. This is incredibly important for sorting methods. There are also many different visualizations online that should help you understand them.

Step 2: Create the data file. There is a program on the Moodle site that will do this. It's just below the assignment.

STEP 3: Create a dynamically allocated integer array of size 1,000,000.

STEP 4: Read the data file placing each value in the file into the next available element of the array. This means that the values in the array will be in the same order as they are in the input file.

STEP 5: Pass the dynamic array of values to each of the sort methods in the order listed above.(by value so you won't have to re-create the array each time).

Output: For each sorting method:

- Display the name of the algorithm used
- Display the number of comparisons required (increment some counter every time you compare)
- Display the number of swaps required (increment some counter every time you swap)

Other Requirements:

It is your responsibility to thoroughly test your program using your own data. We will compile your program using the Gnu compiler and run it against our own data. You MUST use the file names and formats described in the input section above because that's what we are going to use to test your program.

Remember to include appropriate comments, program structure, and your Certification of Authenticity.

Do not forget about **documentation**, as it does count for a fairly sizeable chunk of the grade. For verification this time you will not have to copy+paste your 10 million random numbers, but do show your exact output from the terminal.

You will submit the cleaned NetBeans project folder as a tar file via the class Moodle site. The name of the tar file will be YOURCLID_proj3