Assigned: Monday, 23 March 2015, **Due: Wednesday, 1 April, 11:59pm**

Three Sorts

1. Overview

This assignment will test your ability to write three different sorting algorithms in C++. We have discussed a number of different sorts, and you will have the freedom to choose from the myriad of different sorting algorithms that have been developed.

2. Implementation Specifics

You need to write three programs that are able to sort integers that are input through the "standard input" (stdin) stream. In other words, numbers can either be typed into the command shell, or they can be read in from a text file of integers (see below for an example of the latter). The numbers will go into an array in the following `struct`:

```
struct arrayWithLength{
        int length;
        int *arr;
};
```

As you can see, the struct gives you the length of the integer array as well as a pointer to the array itself.

Output from the sorted array will be printed to standard out (stdout) using the `printArray()` function, which you need to write.

3. The Three Sorts

The following is a breakdown of the sorting programs you can write:

A) One of your sorts must be **selection sort**. The sample code you will be given has a SelectionSort class that you should use to write the code, but the algorithm is in our class notes or online (see Section 6 for a list of references)

B) One of your sorts must be from the following list:

shell sort, merge sort, quicksort, insertion sort

C) Your final sort can be from the previous list, *or* it can be any other reasonable sorting algorithm, except for **bubble sort** or **heap sort** (only because we just wrote heap sort in lab). "Reasonable" is defined as a sort that can be completed for all of our test inputs in three minutes or less. That precludes sorts such as Bogosort (see: http://en.wikipedia.org/wiki/Bogosort), sleep sort (see: http://stackoverflow.com/questions/6474318/what-is-the-time-complexity-of-the-sleep-sort), or other sorts that, while in principal could sort, are not feasible as general purpose sorts. Ideas for out-of-the-ordinary (or non-comparison) sorts are **radix sort, tim sort, bucket sort, library sort, quick3**.

4. Describing Your Sorts, and Finding Sorting Algorithms

In your `ReadMe.md` file, you need to **describe the algorithm for each sort in your own words**. This is especially important if you chose an out-of-the ordinary sort. **Include a discussion of the worst case and best case complexity, and justify your response.** If you include an average complexity, please provide a reference or a proof. You should have a section in your `readme.txt` file dedicated to each sort. You should also have a list of references where you found the algorithm for your sorts.

You may use offline or online resources to find the algorithm for your sorts. **DO NOT** copy C++ code from the internet! Almost all sorts have very detailed algorithms online, written in both C++ and other languages. We can't stop you from looking at code samples of the algorithms, but you are on your honor to translate only algorithms and not specific code into C++. In other words, you are free to look at algorithms (indeed, you must), but you can't copy specific code.

5. Other Comments (IMPORTANT!)

- The `Makefile` (provided) will create three executable files:
  - selectionSort
  - sort2
  - sort3

- See the Makefile itself for details on the names for the files you need to produce.

- Testing for this assignment will include timed tests as well as functionality tests, with varying input sizes and element orderings. If your sorts do not finish within the three minute time-limit for our input data, we will consider them to be incorrect. You should test your code on $O(n^2)$ sorting algorithms on input sizes up to 100,000 elements. You will have test input files that have that many input integers. We will only test on positive integers (including 0).

- In order to test your sorts, you can either type (or paste) a list of integers (separated by newlines) into the terminal (followed by ctrl-D), or you can input them directly from test files, as follows:

```
./selectionSort < randList100000.txt
```

- As always, direct any specific questions to the Piazza class site.

6. References

Feel free to use any of the following sites to discover sorting algorithms:

- http://en.wikipedia.org/wiki/Sorting_algorithm
- http://www.sorting-algorithms.com
- http://betterexplained.com/articles/sorting-algorithms/

7. Low Level Details

Getting the files

There are two ways to get the files for this assignment. The first is by copying the original files from the class folder. The second is to use "git" to pull the files from the GitHub cloud server.

Method 1: copy files from the class folder

First ssh to the homework server and, and make a directory called sorting

```
ssh -X your_cs_username@homework.cs.tufts.edu
mkdir hw5
```

change into that directory

```
cd hw5
```

At the command prompt, enter (don't forget the period):

```
cp /comp/15/public_html/assignments/hw5/files/* .
```

Method 2: pull from GitHub:

At the command prompt, enter

"`git clone https://github.com/Tufts-COMP15/2015s_HW5.git`"

change into the directory that git created:

"`cd hw5`"

Using Eclipse (assuming you have followed the steps above to get the files):

NOTE: You will need to handle having more than one main() function in your three mainX.cpp files — the best way to handle the issue is to rename your other main() functions main1(), main2() while you are testing your third sort.

1. See online video https://www.youtube.com/watch?v=BbnLZ2oqrD0
2. If in the lab, simply open Eclipse, and then start following from step 6 below.
3. If at home, ensure that you have installed (Mac) XQuartz, or (Windows) Cygwin (see here: https://www.youtube.com/watch?v=BbnLZ2oqrD0) or (Win) putty and XMing (http://sourceforge.net/projects/xming/).
4. ssh to the homework server using the -X argument:
   ssh -X your_cs_username@homework.cs.tufts.edu
5. start eclipse by typing "eclipse" (no quotes)

6. Use the following steps to set up your project (you'll get used to the steps quickly, steps A & B only need to be done once at the beginning of the course):
   A. Default location for your workspace is fine
   B. Click on "Workbench", then click "Window->Open Perspective->Other", and choose C++
   (note: if C++ is not listed, close Eclipse and re-open by typing Eclipse)
   C. Window->Preferences
      General->Editors->Text Editors
      Displayed tab width (8) (recommended)
      Show print margin: 80 col
      Show line numbers

C/C++ -> Code Style
Select K&R [built-in] and then “Edit”, then rename to “K&R 8-tab”
Change Tab size to 8 (recommended)
Click “Apply” then “Ok”

D. File->New C++ Project
Fill in project name (no spaces!) IMPORTANT: the name CANNOT be exactly the same as your folder name. I suggest to simply put name_project (with the _project) for all project names (e.g., hw5_project)
Use default location should be checked.
Select Empty Project->Linux GCC
Click Next
Click "Advanced Settings"
1. On the left, under C/C++ General->Paths and Symbols (left hand side)
(you may have to click on the triangle next to C/C++ General)
Select "[Debug]" at the top for Configuration
Source Location Tab
Click "Link Folder"
Check "Link to folder in the file system"
Browse to find your folder.
(should be something like h/your_username/hw5)
Click OK
Click Apply

2. C++ Build->Settings (may have to click on the triangle)
For Configuration (at the top), select [All configurations]
GCC C++ Compiler: Command should be "clang++" (no quotes)
GCC C Compiler: Command: clang
GCC C++ Linker: Command: clang++
Click "Apply"
Click OK

Click Finish

E. Click on the white triangle next to your project name.
The folder you linked to should be listed. Click on its triangle.
The files in that folder should be listed.

F. Test the build by clicking on the hammer in the icon bar.
You should see some compiling messages in the Console window at the bottom. (Things like, "Building file...; clang++, etc.)

G. The program will compile, but will have many warnings. You need to write the functions — double-click on List_linked_list.h to see the header file and List_linked_list.cpp to see the code.

Compiling and running:

1. To make all three of your sorts, type the following: **`make`**
2. To make an individual sort, enter “**`make selectionSort`**” (or **`sort2`** or **`sort3`**)

Providing:
At the command prompt, enter “**`make provide`**” and press enter or return.