1 Basic Information

For each lab, you must check in the code before class on the day when the lab is due. The code will be written in Java.

The coding portion of this course will be managed accordingly:

- all code will be graded by an auto-grader
- all code will distributed and submitted exclusively via svn
- you will need to use the command line to check out, complete, and submit your code

This document will explain a little bit more about each of these aspects of the course. However, rather than giving you an isolated, abstract tutorial on using the command line, the rest of this document will simply be written in the context of using the command line.

2 SVN

We will use subversion (see also http://subversion.apache.org) to distribute assignments. Each student in CSE 241 will be given their own svn repository, which is accessible only by you, the instructor, the TAs, and a few technical support folks in the department. As is standard in the CSE Department, your repository name will be of the form Lastname-YYYYYY, a combination of your last name and your 6-digit student ID number. If your last name contains multiple words, spaces, or punctuation, please check with a TA to make sure that your Lastname is what you think it is.

When you are prompted for a username and password in order to access this account, you should provide your wustlkey username and associated password, not your cec username and password!

The most recent checked in version of your repository can always be viewed (by those with permission!) on the web at the URL: https://shell.cec.wustl.edu:8443/cse241_sp14/svn/WUSTL_key/, where WUSTL_key is your key to the WUSTL system.

This is the version of your code and assignment that the instructor and TAs will see when they grade your assignment. After you submit your homework via svn, it is a very good idea to check out this URL and make sure that everything looks up to date. It is your responsibility to make sure that you submit your homework both punctually and successfully. One of the two is not sufficient.

2.1 SVN Checkout

The first step is to check out your subversion directory. To achieve this you need to use the `svn checkout` command (or `svn co`, for short) in the following steps:
1. Create a directory (we’ll call it cse241) on the machine where you want to check out your repository. This machine should have Java installed if you want to compile and run your code.
   
   yourhome> mkdir cse241
   yourhome> ·

2. Enter said directory
   
   yourhome> cd cse241
   yourhome/cse241> ·

3. checkout your repository
   
   yourhome/cse241> svn co
   https://shell.cec.wustl.edu:8443/cse241_sp14/svn/WUSTL_key
   (no space between the above two lines)

   This will create a directory inside cse241 called WUSTL_key. If you step into this directory:

   yourhome/cse241> cd WUSTL_key
   yourhome/cse241/WUSTL_key> ·

   and look at what’s inside:

   yourhome/cse241/WUSTL_key> ls
   lab0
   yourhome/cse241/WUSTL_key > ·

   you will see one more directory: lab0. lab0 contains the files needed to complete the coding portion of the first assignment.

   Aside: to return to the directory 1 higher up, simply type:

   yourhome/cse241/WUSTL_key> cd ..
   yourhome/cse241> ·

2.2 SVN Commit

   It’s a good idea to commit your code at regular intervals. Svn can be a backup for you as well as a way to turn in your assignment. For example, when you finish coding up a distinct portion of the assignment, you might commit a working version of your code with the comment:

   yourhome/cse241/WUSTL_key/lab0> svn commit -m ’’working’’

   When you are completely finished, make sure to commit again:

   yourhome/cse241/WUSTL_key/lab0> svn commit m ’’finished lab1’’

2.3 SVN Add

   A good svn user should understand that if you create any new files inside an svn managed directory, they will not be automatically added to your svn repository. This is a good thing, since it keeps your repo from being filled with executables, .o, and ~ files that you might generate while running and editing your code. However, if you ever create a new file that you want to submit, you will need to explicitly add and commit
it to your repository. We mention this capability for completeness only. In this course, it is *not* be expected that you will be creating new files in order to finish assignments. Unless you are told explicitly otherwise, you should really not need to use this command for this course.

### 2.4 SVN Update

In the beginning of the semester, your repo will contain only lab1. As the semester progresses, we will add new homework directories to your repo. In order to see the new directories that we push to your repositories, you will need to perform an `svn update` as follows:

```bash
yourhome/cse241/WUSTL_key> ls
lab0
yourhome/cse241/WUSTL_key> svn update
yourhome/cse241/WUSTL_key> ls
lab0 lab1
yourhome/cse241> .
```

### 3 Completing The Assignment

We will provide all of the skeleton code for the lab. Generally, you will have to just implement the main algorithms, and all of the input and output routines will be handled by the provided code. Individual lab assignments will tell you which files to change and which methods to implement. Please do not change any of the method signatures. The individual lab assignments will tell you which files should be changed. You shouldn’t have to change or add any other files in order to complete the assignment. Therefore, if you are adding any useful methods, please add them only to the specified files.

#### 3.1 Compiling, Running and Testing

If you are working on a Windows machine, please follow the instructions provided on Piazza to set your classpath appropriately. If you are working on a Mac or any of the ccc machines, everything should just work as long as you have java installed on your machine. Within each lab’s directory, you will see three shell scripts in order to compile and run your code. In order to compile your code, type:

```bash
yourhome/cse241/WUSTL_key/lab0 > ./compile.sh
```

If your compile is successful, you will see something like this:

```
Compiling...
-----------
Compiled files:
src/Lab0.java
src/Foo.java
src/TestLab0.java
```

If your compile is unsuccessful, it will show your the error. You should fix it and try to compile again. Once your code compile correctly, you should try to test it using the provided tests by typing:

```bash
yourhome/cse241/WUSTL_key/lab0 > ./test.sh
```
The screen should show you whether you passed or failed the tests. You can also run your code by providing arguments by typing:

\texttt{yourhome/cse241/WUSTL_key/lab0} > \texttt{./run.sh <args>}

where args represents space separated arguments taken by Lab0.java file.

Also feel free to write additional tests to test your code.

### 3.2 The Autograder

Your code will be graded by an auto-grader. You will be explicitly told what methods to complete/modify. Don’t touch, mess with, rename, or in any way modify any of the other parts of the code you have been given without checking with an instructor first. Your code will also be tested on new inputs that you will not be given access to.

**Things not to do:**

- **Rename methods:** if we ask you to complete the method \texttt{qsort()} and you rename it to \texttt{quicksort()}, our code will look for a method called \texttt{qsort()}, not find it, and you will not receive credit for your work.

- **Change the signature of methods:** if \texttt{qsort()} returns a vector, and you change \texttt{qsort()} so that it returns an array, our code will look for a method called \texttt{qsort()} that returns a vector, not find it, and you will not receive credit for your work.

- **Modify helper code:** We have given you a lot of helper code for each assignment. If you modify any of the helper code (for example, by adding a new method to a class we give you and then calling this new method in the method we ask you to complete) when the auto-grader tests your code, the new method will not exist in its version of the class we gave you, your code will not compile, and you will not receive credit for your work.

- **Litter your code with print statements:** Yes, a few print statements can be a very helpful thing, while you are coding and debugging. But if your submitted code prints a line of text for each stage of a sorting algorithm, and we test your code on an input size of ten million, your code will run so slowly that the the auto-grader will assume it does not terminate, kill it, and you will not receive credit for the assignment. So please remove these print statements before submitting the code.

**Important!** Ten million print statements will really annoy any instructor or TA who decides to run your code. It can also bring down the machine that the auto-grader is running on. True Story: in another CSE class that will go un-named, one student who will also go un-named forgot to remove debug print statements and when the code was run, it created an output file on the order of terabytes in size, which took down the entire file system and brought the machine to a screeching halt. Let’s just say... this story didn’t end well for the student.

Please think about the ways in which you may be affecting our ability to automatically grade your code and when in doubt, ask ahead whether a change you are making is acceptable.
4 Using Eclipse

You can choose to use eclipse instead of the command line to compile and run your code. However, we strongly recommend the command line. In particular, the instructor and the TAs WILL NOT help with debugging your eclipse issues. In addition, eclipse occasionally messes up the directory structure of the code which breaks the autograder. It is your responsibility to figure out and fix these issues. If your code does not work with the autograder, you will not get credit for the lab.

If you do not have eclipse and subversive installed, then please go to http://www.cse.wustl.edu/~bracy/class/cse131/, look at the “tutorials” tab, and then “eclipse installation” and follow the instructions. If you took CSE131 or CSE132, you should already have this infrastructure set up. One of your TAs, Daniel, has graciously uploaded an eclipse guide on Piazza, which you can feel free to use to check out your code, etc.

5 Grading

Your code will be autograded for technical correctness. Therefore, please make sure of the following before you submit:

- The code compiles and runs without errors.
- You have not changed the names of any provided functions or classes within the code.
- You have removed all debugging print statements that you may have added.

You may commit your code as many times as you like. If you commit a change, then we will automatically run tests on your code. When you next update, the result file will be updated to tell you whether you passed or failed these tests. You can use this facility to submit your code early and test it. The last submission before the submission deadline will be used for your grade for your lab. Therefore, if you have a working version, and then later introduce errors in your code, make sure that you revert to the previous version before the submission deadline.

We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else’s code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don’t try. We trust you all to submit your own work only; please don’t let us down. If you do, we will pursue the strongest consequences available to us.