CSE 542 – Advanced Data Structures and Algorithms *Jon Turner*

Lab 1

Due 1/29/2013

General notes for labs. The labs are intended to help you get a deeper understanding of the material covered in class. There is typically not a lot of new code required, but you will need to familiarize yourself with the provided codebase. Documentation for the provided code is available on the class website. Copies of the source code will be made available to you through your own personal svn repository. See the link in the left margin of the web site for instructions on accessing the repository.

In each lab you will be adding source code. Sometimes this involves adding new files. Other times, it involves modifying existing code. You are expected to clearly document your code. Each method should have a brief description of what it does, what the parameters are and what result is returned (if any). Any method that involves some non-trivial computation should have additional comments explaining the algorithm. You should write your comments with two purposes in mind:

1. To make it easy for me and the TAs to understand what you have done, so that we can give you appropriate credit. If we can't understand your code, you should not expect to receive full credit.
2. To demonstrate to me and the TAs that you have a complete understanding of the material. When grading an assignment, we're trying to evaluate how well you understand things. If you don't show us that you've mastered the material, you should not expect to receive full credit.

Your source code changes must be submitted online, by doing a commit on your svn repository. This must be done by 6:00 pm on the due date for the lab. At that time, I will be retrieving a copy of all student repositories and those copies will constitute the official submission. Changes made after the deadline don't count.

In addition to the online submissions, you will be provided with a lab report template that includes more detailed instructions and questions for you to answer. You should fill in all required material and include a copy of your completed report in your svn repository. You should also hand in a printed copy of your lab report at the start of class on the due date. You will be asked to include copies of your source code in your lab report. Please format your source code files with no more than 80 characters per line, so that when they are pasted into the report, they can be printed in portrait mode without lines wrapping around. Please fasten the pages of your report securely before you hand it in (preferably with a single staple in the upper left corner).

Lab 1 instructions.

In this lab you will be evaluating the performance of Prim's minimum spanning tree algorithm. You will be using a provided implementation that you will be modifying to compute selected performance statistics. You will also be writing some additional methods to generate the graphs you'll use when evaluating Prim's algorithm. There are several parts to the lab.

A. In this part, you will be making source code modifications that will be used in later steps to measure the performance of Prim's algorithm. In your svn repository, you will find a directory called *grafalgo* that contains various data structures and algorithms that we'll be using over the course of the semester. Before doing anything else, familiarize yourself with the contents of this directory.

   In this part, you will be extending the *d*-heap implementation (the .h file is in the *include* sub-directory, the .cpp file is in *dataStructures/heaps*) to add three counters: *changeKeyCount* should be incremented every time the *changekey* method is called; *siftupCount* should be incremented when the *siftup* method is entered and once more for each loop iteration; *siftdownCount* should be incremented when the *minchild* method is entered and once more for each loop iteration of *minchild*. Make all three counters public, to simplify accessing their values from other parts of the program.

   Verify that the program compiles using the provided *makefiles*. In the top level *grafalgo* directory, type "*make clean*" followed by "*make all*". This should compile all the source code including your changes to *Dheap*. You may need to make some adjustments to the *makefiles* to suit your environment. I normally compile them on a Mac (Mountain Lion), but have also compiled them on a *linux* computer using the g++ compiler. For g++, you may need to add the compiler flag "std=c++0x", since the source code does use some features of the latest C++ standard (now known as C++11). Other compilers may have different ways to specify this.

B. In the lab 1 directory, you will find several .cpp files containing C++ source code. The file *common.c* has several missing pieces. First, you should modify the provided implementation of Prim's algorithm to make the heap parameter *d* a configurable parameter and you should fill in the *PrimStats* structure after the algorithm completes (details in the code). Use the method *Util*::*getTime*() to measure the running time. This method returns the number of microseconds that have elapsed since the first time *getTime* was called, so if you call it twice, say *t0=Util::getTime();...,t1=Util::getTime()*, the difference (*t1–t0*) is the elapsed time between the two calls, in microseconds.

   Next, complete the two methods *badcase* and *worsecase* to return graphs having the characteristics defined in the source code comments. The method *Graph::join()* can be used to create an edge between two vertices and the method *Graph::addEdges()* can be used to add random edges to a graph. See the online documentation and the source code in *include/Graph.h* and *dataStructures/graphs/Graph.cpp* for details. These methods will be used in our evaluation of Prim's algorithm.

   The lab report includes instructions for verifying your code and making some basic observations about the performance of Prim's algorithm.

C. In this part, you will be evaluating the performance of Prim's algorithm on three types of graphs: *random graphs*, graphs produced using *badcase* and graphs produced using *worsecase*. We will be studying how the algorithm performs as the number of vertices grows, while edge density, *m*/*n* remains constant. The provided *script1* will generate all the specific results needed for this section.

D. In this part, you will be running another set of experiments, but in this case, we will hold *n* fixed while increasing *m*. The provided *script2* will generate the results needed for this section.

E. In this part, you will be running a third set of experiments, to evaluate the effect of the heap parameter *d* on the performance of Prim's algorithm. The provided *script3* will generate the results needed for this section.

More detailed instructions can be found in the lab report template in the lab1 directory of your svn repository.