Design and Analysis of Algorithms
University of California, Irvine
Prof. Aparna Chandramowlishwaran

September 28, 2016
EECS 215

Homework 1

Due on **Wednesday, October 12, 2016 by 9pm**

Reading: Chapters 1-4

Mark the top of each submission with your name, the course number, the date and the names of any students with whom you collaborated.

You will often be called upon to give an algorithm to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. Atleast one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm. Remember, your goal is to communicate. Full credit will be given *only* to correct solutions which are described clearly. Convoluted and obtuse descriptions will receive low marks.

(6 Points) Asymptotic Notation

For each group of functions, sort the functions in increasing order of asymptotic (big-O) complexity:

1. *(2 Points)* **Group 1**

$$f_1(n) = n^{0.999999} \lg n$$
$$f_2(n) = 10000000n$$
$$f_3(n) = 1.000001^n$$
$$f_4(n) = n^2$$

2. *(2 Points)* **Group 2**

$$f_1(n) = 2^{2^{1000000}}$$
$$f_2(n) = 2^{100000n}$$
$$f_3(n) = n \lg n$$
$$f_4(n) = n\sqrt{n}$$

3. *(2 Points)* **Group 3**

$$f_1(n) = n^{\sqrt{n}}$$
$$f_2(n) = 2^n$$
$$f_3(n) = n^{10}.2^{n/2}$$
$$f_4(n) = \sum_{i=1}^{n}(i+1)$$

(20 Points) **Recurrences**

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 10$. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 2T(n/3) + n \lg n$

(b) $T(n) = 3T(n/5) + \lg^2 n$

(c) $T(n) = T(n/2) + 2^n$

(d) $T(n) = T(\sqrt{n}) + \Theta(\lg \lg n)$

(e) $T(n) = 10T(n/3) + 17n^{1.2}$

(f) $T(n) = 7T(n/2) + n^3$

(g) $T(n) = T(n/2 + \sqrt{n}) + \sqrt{6}046$

(h) $T(n) = T(n-2) + \lg n$

(i) $T(n) = T(n/5) + T(4n/5) + \Theta(n)$

(j) $T(n) = \sqrt{n}T(\sqrt{n}) + 100n$

(19 Points) Sorting

1. *(9 Points)* Do Exercise 2.3-5 and 2.3-6 on page 39 in CLRS.
2. *(10 Points)* You will implement the binary search version of the insertion sort algorithm in C/C++.

Getting the scaffolding code

We will use the `git` distributed version control system. The baseline code for this assignment is available at this URL: `https://github.com/EECS215/hw1.git`

To get a local copy of the repository for your work, you need to use git to clone it. To do that, run the following command .

```
$ git clone https://github.com/EECS221/hw1.git
```

If it works, you will see some output similar to the following:

```
Cloning into 'hw1'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 11 (delta 0), reused 11 (delta 0), pack-reused 0
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
```

There will be a new directory called **hw1**.

Compiling and running your code

We have provided a small program, broken up into modules (separate C/C++ files and headers), that performs sorting. For this homework, you will make all of your changes to just one file as directed below. We have also provided a `Makefile` for compiling your program. To use it, just run `make`. It will direct you with the right flags to produce the executable. For example, `make insertion-sort` will produce an executable program called `insertion-sort` along with an output that looks something like the following:

```
# Add the changed file
$ git add README

# Commit -- saves file with a message ("-m" option)
$ git commit -m "Added README"
```

Run `insertion-sort` on an array of size `100` as follows:

```
$ make mergesort-omp
g++  -O3 -g -o driver.o -c driver.cc
g++  -O3 -g -o sort.o -c sort.cc
g++  -O3 -g -o parallel-mergesort.o -c parallel-mergesort.cc
g++ -O3 -g -o mergesort-omp driver.o sort.o parallel-mergesort.o
```

C/C++ style guidelines

Code that adheres to a consistent style is easier to read and debug. Google provides a style guide for C++ which you may find useful: `http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml`.

Part of your grade on assignments is based on the readability of your code.

Insertion Sort

Although we've given you a lot of code, to create an insertion sort using binary search, you just need to focus on editing `insertion-sort.cc`. Right now, it is mostly empty. In this file, implement the insertion sort algorithm.

Submission

- When you've written up answers to all of the above questions, turn in your write-up and tarball of your code by uploading it to eee.uci.edu dropbox. LATE HOMEWORKS WILL NOT BE ACCEPTED.
- You may work in **teams of two**. All team members may submit identical code. Be sure to indicate your assignment partner in your submission.