CS-473: Algorithms I Spring 2018

Assignment 1: Getting Started

Instructor: Mehmet Koyutürk

Due Date and Instructions

Please return hard-copies at the beginning of the class meeting (8:40) on **Friday, February 9, 2018**.

- Hand-writing is accepted but the course personnel reserves the right to reject grading an assignment if the hand-writing is not legible.
- Assignments written in LaTeX will receive 5 bonus points.
- You can submit the assignment as a team of 2 students, however no pair of students are allowed to work together on more than 3 assignments.
- You are also allowed to talk to other students and the course personnel about the solutions, but you must write the answers yourself. Answers copied from other students or resources will be detected, and appropriate action will be taken.

Problem 1

The greatest common divisor (GCD)of two integers a and b is defined as the largest integer that can divide both a and b without a remainder. For example, the GCD of 30 and 54 is 6, whereas the GCD of 7 and 5 is 1. The following procedure was developed by Euclid to compute the greatest common divisor of two positive integers a and b. In this exercise, we will prove the correctness of this algorithm.

```
procedure EUCLIDIAN(a, b)
1  x ← a
2  y ← b
3  while x ≠ y do
4      if x > y then
5          x ← x − y
6      else
7          y ← y − x
8  return x
```

(a) [10 pts] State the loop invariant for the while loop in this procedure.

(b) [20 pts] Prove the loop invariant.

(c) [5 pts] Using the termination property of your loop invariant, prove that procedure EUCLIDIAN computes and returns the greatest common divisor of a and b.

Problem 2

Let A and B be two arrays, each consisting of n numbers sorted in increasing order. We are also given a number x and would like to find and return i and j such that $A[i] + B[j] = x$. If no such pair exists, we would like to return `FALSE`. We would like to develop an algorithm to solve this problem in linear time.

(a) [15 pts] Using pseudo-code, describe an algorithm that correctly solves this problem in linear time.

(b) [5 pts] Explain graphically how your algorithm works.

(c) [10 pts] Using loop invariants, prove that your algorithm is correct.

(d) [5 pts] Show that the worst-case runtime of your algorithm is $\Theta(n)$.

Problem 3

Using your favorite programming or scripting language, implement Insertion-Sort and Merge-Sort. Then, using identical platforms for the two algorithms, answer the following questions:

(a) [20 pts] Consider instance sizes $n = 2^4, 2^8, 2^{12}, 2^{16}, 2^{20}$. Generate 10 random instances (such that either the numbers in the array are randomly generated, or the ordering of the numbers is randomized). Plot the runtime of each algorithm as a function of n (you need to use error bars to visualize the distribution of runtimes for different random instances). At what value of n does Merge-Sort become more efficient than Insertion-Sort? Which algorithm has a more variable runtime for different instances that have the same size?

(b) [10 pts] For each value of n considered above, generate an instance that represents the worst case for Insertion-Sort. Repeat the same analysis (in this case, you do not need error bars since you have a single instance and both algorithms are deterministic, but running the algorithms multiple times and reporting averages can be more reliable to account for other factors). What is the critical value of n in this case?