Algorithms and Data Structures (ECE 345)

Issued on: Jan 12th, 2019
Due on: Jan 29th, 2019

ASSIGNMENT 1

1. Assignment must be submitted by **6:00 PM**.
2. All submissions must include an attribution table and all pages must be numbered.
3. All submissions must be made in groups of up to two students.
4. A physical copy must be delivered to the course dropbox. Instructions for submitting the code component can be found on the course syllabus under "Assignments and Exam Schedule".
5. No more than a single answer for any question. The report for the last question must also be handed in with the physical copy of the assignment.
6. Any problem encountered with submission, either physical or electronic, must be reported to the head TA as soon as possible.

EXERCISE 1 Asymptotics, 10 points

Sort the following 10 functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. Do as much as you can, you will receive partial credit even if you do not sort all of them. For full marks you must provide proofs that *justify your answers*.
All logarithms are base 2 unless otherwise stated.

$$\begin{array}{ccccc} n & n^{1+(1/\log n)} & \log n! & \log\log n! & e^n \\ \sum_{i=2}^{n}\left(\frac{1}{i-1}-\frac{1}{i+1}\right)+2 & n^{\log\log n} & \left(1-\log\frac{1}{1-1/n}\right)^n & \log^{\log n} n & n\log^2 n \end{array}$$

To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = O(g(n))$, $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$, and $f(n) \gg g(n)$ to mean $f(n) = \Omega(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted either as $n^3 \gg \binom{n}{2} \equiv n^2 \gg n$ or as $n^3 \gg n^2 \equiv \binom{n}{2} \gg n$.

EXERCISE 2 Recurrences, 10 points

Using the **Master Theorem**, solve the following recurrences by giving *tight* Θ-notation bounds. For full marks all constants needed for the proofs must be defined, either as a constant or as an inequality.

(a) $T(n) = 2T(n/4) + n^{0.51}$

(b) $T(n) = \sqrt{2}T(n/2) + \log n$ (*Hint* use derivatives if it helps)

(c) $T(n) = 3T(n/2) + n$

(d) $T(n) = 4T(n/2) + n^2$

EXERCISE 3 Sorting, 20 points

There exists a black box that takes as input any set of integers S and an integer k. The black box will instantly answer the question "Does there exist a subset of S whose sum is exactly k?" with either $TRUE$ or $FALSE$.

Given a set of n integers T and a target sum t, devise an algorithm that uses the black box $O(n)$ times to return an actual subset of T that adds up to t, or determines that there is no such subset. For full marks provide your algorithm, in pseudo code or clear point form, and argue the algorithm's correctness.

EXERCISE 4 Heaps, 15 points

Consider a binary heap consisting of n elements. Prove that there are exactly $\lceil \frac{n}{2} \rceil$ leaves in the heap. Be careful with the use of floor or ceil functions.

EXERCISE 5 Search Trees, 15 points

Let X be a set of n items, each with a key and a priority. For the sake of simplicity, assume that no two keys or priorities are identical. A **PK-Tree** for X is a rooted binary search tree whose nodes are the items in X such that:

(i) $key[x] < key[y]$ if x is a left descendant of y or y is a right descendant of x, and

(ii) $priority[x] < priority[y]$ if x is a descendant of y.

1. Given a key value, how would you search for that value in a PK-Tree?
2. Give an algorithm to insert a new item into a PK-Tree.
3. Give an algorithm to delete an item from a PK-Tree.
   Hint: Use rotations to restore the priority structure of the tree after an insertion or deletion.

EXERCISE 6 Programming Exercise, 30 points

In this coding exercise you will implement various sorting algorithms and you will explore two important aspects of algorithm design: the practical role of the constants in asymptotic analysis, and how the theoretical asymptotic bounds translate into runtime limitations for various algorithms.

This exercise entails several steps outlined below. Please abide to these, as any deviation may result in losing marks.

You are to:

1. Pick any **two** sorting algorithms that are covered in lecture, and **one** sorting algorithm that is **not** covered in lecture (it can still be from CLRS as long as we do not explicitly cover it in lecture), such that **at least two** of these algorithms have different worst case time complexities. For example, if two of them are $O(n^2)$, then the third has to be anything but $O(n^2)$. This also means that you could pick all three algorithms to have different worst case time complexities. For example, $O(n^2)$, $O(n \log n)$, $O(n^3)$.
2. Implement all these three sorting algorithms in the **same programming language of your choice**.

Deliverable must be a .tar.gz or .tgz file containing:

1. Your source code, including a full build environment. Please remove any generated executables before submission. The sources must be buildable on the `unNNN.eecg` machines in our undergrad Linux workstation labs. Our `unNNN.eecg` machines support Debian Jessie Linux. Preinstalled are Java (openjdk), Python, and gcc.

2. A Makefile that generates executables when ‘make‘ is invoked from the root directory of the submission such that:

   (a) Three executables (`sort1`, `sort2`, and `sort3`) are generated, each corresponding to one of the algorithms you implemented. The executables could be compiled binaries or an executable script with the appropriate shebang. For example, for a python script:

```
1 #!/usr/bin/env python2
2
3 print("Hello World")
```

   (b) Each executable must be callable like ‘sort1 inputfile‘, ‘sort2 inputfile‘, or ‘sort3 inputfile‘.

   (c) Each executable accepts a file of CSV (comma seprated values) data as input, where the integer sort key is the first entry, with an arbitrary number of CSV values following. For example each row in the input file will look like the following: `sort key, value 1, value 2, ...`. You cannot make any assumptions about the number of values in each row of the file.

   (d) Each of them sorts the input based on the sort key (the first column in the example above) and prints the result to standard output, in the same format as the input.

3. A submission.yml file containing information about the submission. An example of this is given alongside this assignment.

4. A written report as part of your Assignment 1 submission, where you address the following points:

   (a) Experimentally identify the **runtime complexity** and identify any **constants** that determine the performance of your algorithms. You will need to run your experiments for **various input sizes** to obtain good approximations for the growth functions that characterize your algorithms' runtimes, but also to obtain a good estimate for the constants.

   (b) Provide one graph that demonstrates how you determined the constants for each of your implementations.

   (c) Provide a table that lists row-wise the algorithm implemented, its growth function, and its constant.

   (d) Pick any **one** of your algorithms and realize an optimization of your choice; **re-run** the experiments and report the outcome in your graph and table as well.

   (e) What do you conclude about the performance of the four implementations (three different algorithms, one with additional optimization)?

   (f) Diligently list all sources that you used and discuss the optimization you applied.

Some Comments:

1. We recommend that you use Java, Python or C/C++ (gcc).

2. A sample CSV input file will be provided to you.

3. Submission instructions for the programming exercise will be provided separately.