Algorithms and Data Structures (ECE 345)

Issued on: Jan 13th, 2023
Due on: Jan 27th, 2023

ASSIGNMENT 1

1. Assignment must be submitted by 5:00 PM EST on the due date through the Quercus submission system as a single PDF file.

2. Assignment must be completed **individually except the programming exercise** for which you can work in group of up to **two** students. Report for the programming exercise must be submitted separately at the same time. Only one report is needed for each group.

3. All pages must be numbered and no more than a single answer for any question.

4. Use LaTeX or Microsoft Word for your assignment writeup. You can find a LaTeX and a Word template for writing your assignment on Quercus.

5. Any problem encountered with submission must be reported to the head TA as soon as possible.

6. Unless otherwise stated, you need to justify the correctness and complexity of algorithms you designed for the problems.

EXERCISE 1 Asymptotics, 10 points

For each of the following functions, decide the asymptotic relationships between $f(n)$ and $g(n)$. i.e., decide if $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. For full marks you must provide proofs that justify your answers.
All logarithms are base 2 unless otherwise stated.

(a) $f(n) = 7n^2 \qquad g(n) = 3n^2$

(b) $f(n) = 4n^7 + 3n^3 + 5 \qquad g(n) = n^4$

(c) $f(n) = \log n + \frac{3}{n} \qquad g(n) = 2\log n$

(d) $f(n) = \frac{\log n}{n} \qquad g(n) = \frac{1}{n}$

(e) $f(n) = 3^n \qquad g(n) = 3^{2n}$

(f) $f(n) = n^k \qquad g(n) = 2^{k\log n}$

(g) $f(n) = \begin{cases} 4^n, & n < 2^{2023} \\ 2^{2023}n^2, & n \geq 2^{2023} \end{cases} \qquad g(n) = \frac{n^2}{2^{2023}}$

(h) $f(n) = 2^{\sqrt{\log n}} \qquad g(n) = (\log n)^{100}$

(i) $f(n) = n^{\log\log n} \qquad g(n) = (\log n)^{\log n}$

(j) $f(n) = 3^{2^n} \qquad g(n) = 2^{2^{n+1}}$

EXERCISE 2 Recurrences, 10 points

Solve the following recurrences by giving *tight* Θ-notation bounds. Assume that $T(n)$ is constant for $n \leq 2$. For full marks you must provide proofs that justify your answers.

(a) $T(n) = 3T(n/4) + n^{0.63}$

(c) $T(n) = 7T(n/3) + n^5$

(b) $T(n) = 2T(n/4) + n\log n$

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

EXERCISE 3 Sorting, 20 points

Consider a black box that takes any set of integers S and an integer k as input. The black box can answer the question "Does there exist a subset of S whose sum is exactly k?" with either *TRUE* or *FALSE* in $O(1)$ time.

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

Given a max-heap represented by array A and an integer key k, describe an algorithm `SET_SECOND_LARGEST(A, k)` that finds the item currently stored in A that has the second-largest key and modifies it to have key k, while preserving the max-heap property on A. Your algorithm should run in $O(\log n)$ time, where $n \geq 2$ is the length of input heap A. You can use any heap operations introduced in class, e.g. `MAX_HEAPIFY`, `HEAP_EXTRACT_MAX`, `HEAP_INCREASE_KEY` and `MAX_HEAP_INSERT`. Justify your answer.

EXERCISE 5 Search Trees, 15 points

Consider a binary search tree (BST) with height h that stores n integers. Given an interval $[a,b]$, describe an algorithm with $O(h+k)$ running time that finds all integers in the range $[a,b]$ in the BST, where k is the number of integers found. Justify your answer.

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:

```
#!/usr/bin/env python2

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 written report, 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.