Algorithms and Data Structures (ECE 345)

Issued on: Jan 14th, 2022
Due on: Jan 28th, 2022

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) $f(n) = 5n^2 \qquad g(n) = n^2$

(b) $f(n) = 3n^6 + 4n^2 + 5 \qquad g(n) = n^5$

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

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

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

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

(g) $f(n) = \begin{cases} 4^n, & n < 2^{2022} \\ 2^{2022} n^2, & n \geq 2^{2022} \end{cases} \qquad g(n) = \frac{n^2}{2^{2022}}$

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

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

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

(b) $T(n) = 3T(n/2) + n^{\frac{3}{2}}$

(c) $T(n) = 27T(n/3) + n^3$

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

EXERCISE 3 Sorting, 20 points

Given n cards on the table, each with an integer on it, devise an algorithm to sort them using only one kind of operation `flip(i,j)` on the cards. The operation flips the order from i-th card all through j-th card for $1 \leq i \leq j \leq n$. For example, for the array of cards `[1,4,5,3,2]`, doing `flip(2,5)` will result in `[1,2,3,5,4]`.

(a) Assume that each call of `flip(i,j)` **costs $1**. Now suppose all numbers on the cards are either 1 or 2, describe an algorithm which costs $O(n)$ **amount of money**. Justify your answer.

(b) Assume that the operation of `flip(i,j)` **takes linear time** (i.e., $O(j-i)$ of time). Now suppose the numbers on the cards can take arbitrary integer value, describe an algorithm which takes (expected) $O(n\log^2 n)$ **time**. Justify your answer. (*Hint*: even if you failed to solve part (a), you can take it as a black-box algorithm to use in this part.)

EXERCISE 4 Heaps, 10 points

Given a heap of n numbers stored so that the value at every node is larger than all values in its subtree. Devise an $O(k\log k)$-time algorithm to find the k^{th} largest value in the heap (for any value $1 \leq k \leq n$). **Justify the runtime**. (*Hint*: Keep a separate heap to help you identify the k^{th} largest key.)

EXERCISE 5 Search Trees, 20 points

Let X be a set of n items, each with a key and a priority. For 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.

(a) Given a key value, how would you search for that value in a PK-Tree?

(b) Describe an algorithm to insert a new item into a PK-Tree.

(c) Describe an algorithm to delete an item from a PK-Tree.

(d) Analyze the (expected) runtime of the algorithms above (i.e., searching, insertion and deletion).

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:

```
#!/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 submission.toml file containing information about the submission. An example of this is given alongside this assignment.
4. 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.