CS124–Spring, 2020 Homework 1 Out: January 27, 2020
Due: Feb. 7, 2020

Homework will be due at midnight on the given day. Please don't try turning it in at 11:59 and then have it fail and then have to e-mail us; it's annoying. Pretend it's due at 11:45. Really. Try to make your answers as clear and concise as possible; style will count in your overall mark. Be sure to read and know the collaboration policy in the course syllabus. Be sure to check the back of the page; problems occasionally show up there too!

Assigments are expected to be turned in electronically in pdf format. If you do assignments by hand, you will need to scan in your results to turn them in. Instructions for how to electronically submit assignments will be given in class and on the class website.

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will recieve little or no credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of psuedocode will receive little or not credit. Again, try to make your answers as clear and concise as possible.

1. Suppose you are given a six-sided die, that might be biased in an unknown way. Explain how to use die rolls to generate unbiased coin flips, and determine the expected number of die rolls until a coin flip is generated. Now suppose instead you want to generate unbiased die rolls (from a six-sided die) given your potentially biased die. Explain how to do this, and again determine the expected number of biased die rolls until an unbiased die roll is generated. For both problems, you need not give the most efficient solution – simple, non-recursive are all that we are looking for – however, your solution should be reasonable, and exceptional solutions will receive exceptional scores.

2. On a platform of your choice, implement the three different methods for computing the Fibonacci numbers (recursive, iterative, and matrix) discussed in lecture. Use integer variables. How fast does each method appear to be? Give precise timings if possible. (This is deliberately open-ended; give what you feel is a reasonable answer. You will need to figure out how to time processes on the system you are using, if you do not already know.) Can you determine the first Fibonacci number where you reach integer overflow? (If your platform does not have integer overflow – lucky you! – you might see how far each process gets after 10 seconds.)

   Since you should reach integer overflow with the faster methods quite quickly, modify your programs so that they return the Fibonacci numbers modulo $65536 = 2^{16}$. (In other words, make all of your arithmetic modulo 2^{16} – this will avoid overflow! You must do this regardless of whether or not your system overflows.) For each method, what is the largest Fibonacci number you can compute in ten seconds of machine time?

   Submit your source code with your assignment. Please give a reasonable English explanation of your experience with your program(s).

3. Indicate for each pair of expressions (A, B) in the table below the relationship between A and B. Your answer should be in the form of a table with a "yes" or "no" written in each box. For example, if A is $O(B)$, then you should put a "yes" in the first box.

A	B	O	o	Ω	ω	Θ
$\log n$	$\log(n^2)$					
$\log(n!)$	$\log(n^n)$					
$\sqrt[3]{n}$	$(\log n)^6$					
$n^2 2^n$	3^n					
$(n^2)!$	n^n					
$\frac{n^2}{\log n}$	$n \log(n^2)$					
$(\log n)^{\log n}$	$\frac{n}{\log(n)}$					
$100n + \log n$	$(\log n)^3 + n$					

4. For all of the problems below, when asked to give an example, you should give a function mapping positive integers to positive integers. (No cheating with 0's!)

   - Find (with proof) a function f_1 such that $f_1(2n)$ is $O(f_1(n))$.
   - Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.
   - Prove that if $f(n)$ is $O(g(n))$, and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.
   - Give a proof or a counterexample: if f is not $O(g)$, then g is $O(f)$.
   - Give a proof or a counterexample: if f is $o(g)$, then f is $O(g)$.

5. We found that a recurrence describing the number of comparison operations for a mergesort is $T(n) = 2T(n/2) + n - 1$ in the case where n is a power of 2. (Here $T(1) = 0$ and $T(2) = 1$.) We can generalize to when n is not a power of 2 with the recurrence

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n - 1.$$

   Exactly solve for this more general form of $T(n)$, and prove your solution is true by induction. Hint: plot the first several values of $T(n)$, graphically. What do you find? You might find the following concept useful in your solution: what is $2^{\lceil \log_2 n \rceil}$?