CS 124 Homework 1: Spring 2021

Your name:

Collaborators:

No. of late days used on previous psets:
No. of late days used after including this pset:

Homework is due Wednesday at midnight ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignment must be a nonnegative integer at most **two** (college)/**four** (extension school).

Try to make your answers as clear and concise as possible; style will count in your grades. Be sure to read and know the collaboration policy in the course syllabus. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan in your results to turn them in.

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 receive no or little 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 pseudocode will receive little or not credit. Again, try to make your answers clear and concise.

There is a (short) programming problem on this assignment; you DO NOT WORK with others on this problem (e.g., write code together) like you will for the "major" programming assignments. (You may talk about the problem, as you can for other problems.)

1. Suppose you are given a six-sided die that might be biased in an unknown way.

   (a) **(10 points)** Explain how to use rolls of that die to generate unbiased coin flips. Using your scheme, determine the expected number of die rolls until a coin flip is generated, in terms of the (unknown) probabilities p_1, p_2, ..., p_6 that the die roll is 1, 2, ..., 6.

   (b) **(10 points)** Now suppose 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; 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. (You do not need to submit your source code with your assignment.)

   (a) **(10 points)** 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.)

(b) **(4 points)** What's the first Fibonacci number that's at least 2^{31}? (If you're using C longs, this is where you hit integer overflow.)

(c) **(10 points)** 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 value of k such that you can compute the kth Fibonacci number (modulo 65536) in one minute of machine time? (If you reach overflow in the value of k for which you can compute the kth Fibonacci number by one of those methods in less than a minute, instead say how long it took to compute to that overflow.)

3. (a) **(10 points)** Sort the following functions from asymptotically least to greatest: that is, make a series of a statements like "$f_3 = o(f_1)$, $f_1 = \Theta(f_4)$, $f_4 = o(f_5)$, $f_5 = o(f_9)$, ..., $f_8 = \Theta(f_7)$", where each function is either o or Θ of the next. All logs are base 2 unless otherwise specified.

i. $f_1 = (\log n)^{\log n}$
ii. $f_2 = n(\log \log n)^2 / \log n$
iii. $f_3 = \log_3 n$
iv. $f_4 = (\log n)^4$
v. $f_5 = \log_9(n^3)$
vi. $f_6 = 6^{\sqrt[6]{n}}$
vii. $f_7 = (7^n)/(n^{\log n})$
viii. $f_8 = n/(8 \log n)$
ix. $f_9 = (9n)^{\log \log n}$

(b) **(5 points)** Give an example of a function g that would *not* fit into the order above: that is, one for which, for some i, $f_i \notin \Theta(g)$, $f_i = o(g)$, and $g \notin o(f_i)$.

4. In each of the problems below, all functions map positive integers to positive integers.

(a) **(5 points)** Find (with proof) a function f_1 such that $f_1(n+1) \in O(f_1(n))$.

(b) **(10 points)** Prove that there does not exist any function f such that $f(n+1) \in o(f(n))$.

(c) **(5 points)** Find (with proof) a function f_2 such that $f_2(n+1) \notin O(f_2(n))$.

5. Buffy and Willow are facing an evil demon named Stooge, living inside Willow's computer. In an effort to slow the Scooby Gang's computing power to a crawl, the demon has replaced Willow's hand-designed super- fast sorting routine with the following recursive sorting algorithm, known as StoogeSort. For simplicity, we think of Stoogesort as running on a list of distinct numbers. StoogeSort runs in three phases. In the first phase, the first 2/3 of the list is (recursively) sorted. In the second phase, the final 2/3 of the list is (recursively) sorted. Finally, in the third phase, the first 2/3 of the list is (recursively) sorted again. Willow notices some sluggishness in her system, but doesn't notice any errors from the sorting routine.

(a) **(5 points)** We didn't specify what StoogeSort does if the number of items to be sorted is not divisible by 3. Make as small a change as possible to the definition of StoogeSort to define it for those cases in such a way that StoogeSort terminates and correctly sorts.

(b) **(15 points)** Prove rigorously that StoogeSort correctly sorts. (You may not assume all numbers to be sorted are distinct.)

(c) **(5 points)** Give a recurrence describing StoogeSort's running time, and, using that recurrence, give the asymptotic running time of Stoogesort.

6. (a) **(10 points)** Solve the following recurrences exactly, and then prove your solutions are correct. (Hint: Calculate values and guess the form of a solution: then prove that your guess is correct by induction.)

   i. $T(1) = 1$, $T(n) = T(n-1) + 4n - 4$

   ii. $T(1) = 1$, $T(n) = 2T(n-1) + 2n - 1$

   (b) **(10 points)** Give tight asymptotic bounds for $T(n)$ (i.e. $T(n) = \Theta(f(n))$ for some f) in each of the following recurrences.

   i. $T(n) = 4T(n/2) + n^3$

   ii. $T(n) = 17T(n/4) + n^2$

   iii. $T(n) = 9T(n/3) + n^2$

   iv. $T(n) = T(\sqrt{n}) + 1$. (Hint: you may want to change variables somehow.)

7. **(0 points, optional)**[1] InsertionSort is a simple sorting algorithm that works as follows on input $A[0], \ldots, A[n-1]$:

Algorithm 1 InsertionSort

```
Input: A
for i = 1 to n − 1 do
  j = i
  while j > 0 and A[j − 1] > A[j] do
    swap A[j] and A[j − 1]
    j = j - 1
  end while
end for
```


Show that for every function $T(n) \in \Omega(n) \cap O(n^2)$ there is an infinite sequence of inputs $\{A_k\}_{k=1}^{\infty}$ such that A_k is an array of length k, and if $t(n)$ is the running time of InsertionSort on A_n, then $t(n) \in \Theta(T(n))$.

[1] This question will not be used for grades, but try it if you're interested. It may be used for recommendations or TF hiring.