CS 124 Homework 1: Spring 2024

Collaborators:

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

Homework is due Wednesday Jan 31 at 11:59pm ET. You are allowed up to **twelve** late days, but the number of late days you take on each assignment must be a nonnegative integer at most **two**.

Try to make your answers as clear and concise as possible; style may count in your grades. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan your papers to turn them in.

Collaboration Policy: You may collaborate on this (and all problem sets) only with other students currently enrolled in the class, and of course you may talk to the Teaching Staff or use Ed. You may also consult the recommended books for the course and course notes linked from the timetable. You may not use Generative AI or large language models, or search the web for solutions, or post the questions on chat forums. Furthermore, you must follow the "one-hour rule" on collaboration. You may not write anything that you will submit within one hour of collaborating with other students or using notes from such sources. That is, whatever you submit must first have been in your head alone, or notes produced by you alone, for an hour. Subject to that, you can collaborate with other students, e.g. in brainstorming and thinking through approaches to problem-solving.

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 no credit. Again, try to make your answers clear and concise.

There is a (short) programming problem on this assignment; you **should *NOT* code** with others on this problem like you will for the "major" programming assignments later in the course. (You may talk about the problem, as you can for other problems.)

Problems

Please note that Question 4 will use concepts covered on Monday, 1/29 (as well as in sections). Students are advised not to begin this question until after they learn it in a formal setting.

1. In class we saw a simple proof that Euclid's algorithm for n-bit integers terminates in $O(n)$ steps. In this problem, you will provide a tighter bound on the constant factor in $O(n)$.

   Given positive integers A, B satisfying $A \geq B$, let $(A', B') = (B, A \bmod B)$, denote the result of one step of Euclid's algorithm.

   (a) (**5 points**) Prove that $A' + B' \leq \frac{2}{3}(A + B)$. Conclude that starting from (A, B), Euclid's algorithm terminates after at most $\log_{3/2}(A + B)$ steps.

Proof goes here

(b) (**3 points**) In lieu of $A+B$, consider a more general function $g(A,B) = A+\beta B$ for $\beta > 0$. Let $A = QB+R$, where R is the remainder and Q is the quotient when dividing A by B. Give an exact expression for $L' = g(A',B')$ solely in terms of B, R, β, and give a lower bound L for $g(A,B)$ solely in terms of B, R, β. Briefly justify your answer

(c) (**5 points**) Define a choice of β for which the ratio L'/L in the previous question is always the same regardless of B, R. Prove your answer. (Note: β and this ratio will be irrational numbers)

(d) (**5 points**) Use this choice of β to prove an improved upper bound on the number of steps of Euclid's algorithm starting from (A,B).

(e) (**2 points**) Construct an increasing sequence of inputs (A,B) for which the bound in the previous question is asymptotically tight. Informally justify why the sequence you constructed is tight in 1-2 sentences.

(Note; Part (e) above may be attempted independently before attempting other parts and the answer may give a hint to solving Parts (b)-(d).)

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) **(15 points)** How fast does each method appear to be? (This is deliberately open-ended; part of the problem is to decide what constitutes a reasonable answer.) Include precise timings if possible—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) **(15 points)** Since you should reach "integer overflow" with the faster methods quite quickly, modify your programs so that they return the Fibonacci numbers modulo 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 k^{th} Fibonacci number (modulo 65536) in one minute of machine time? If that value of k would be too big to handle (e.g. if you'd get integer overflow on k itself) but you can still calculate F_k quickly, you may report the largest value k_{max} of k you can handle and the amount of time the calculation of $F_{k_{\text{max}}}$ takes.

3. (a) **(10 points)** Make all true statements of the form $f_i \in o(f_j)$, $f_i \in O(f_j)$, $f_i \in \omega(f_j)$, and $f_i \in \Omega(f_j)$ that hold for $i \le j$, where $i, j \in \{1,2,3,4,5\}$ for the following functions. No proof is necessary. All logs are base 2 unless otherwise specified.

      i. $f_1 = (\log n)^{\log n}$

      ii. $f_2 = 2^{\sqrt{\log n}}$

      iii. $f_3 = 2^{(2^{\sqrt[3]{\log\log\log n}})}$

      iv. $f_4 = n^{\log\log n}$

      v. $f_5 = (\log\log n)^n$

(b) **(5 points)** Give an example of a function $f_6 : \mathbb{N} \to \mathbb{R}^+$ for which *none* of the four statements $f_i \in o(f_6)$, $f_i \in O(f_6)$, $f_i \in \omega(f_6)$, and $f_i \in \Omega(f_6)$ is true for any $i \in \{1,2,3,4,5\}$.

4. (a) Solve the following recurrences exactly, and then prove your solutions are correct:

   i. **(5 points)** $T(1) = 1$, $T(n) = T(n-1) + n^2 - n$

   ii. **(5 points)** $T(1) = 1$, $T(n) = 3T(n-1) - n + 1$

   (Hint: Calculate values and guess the form of a solution. Then prove that your guess is correct by induction.)

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

   i. **(3 points)** $T(n) = 9T(\lfloor n/3 \rfloor) + n^2 + 3n$

   ii. **(7 points)** $T(n) = 4T(\lfloor \sqrt{n} \rfloor) + \log n$ (Hint: it may help to apply a change of variable)

5. One of the simplest algorithms for sorting is BubbleSort — see code below.

Algorithm 1 BubbleSort

```
Input: A[0], ..., A[n-1]
for i = 0 to n-1 do
  for j = 0 to n-2 do
    if A[j] > A[j+1] then
      Swap A[j] and A[j+1]
    end if
  end for
end for
```

In this problem we will study the behavior of a twisted version of BubbleSort, described below.

Algorithm 2 TwistedBubbleSort

```
Input: A[0], ..., A[n-1]
for i = 0 to n-1 do
  for j = 0 to n-1 do
    if A[i] < A[j] then
      Swap A[i] and A[j]
    end if
  end for
end for
```

Your task is to prove that TwistedBubbleSort also correctly sorts every array. (While not necessary, you may assume for simplicity that the elements of A are all distinct.)

(a) (**2 points**) Explain in plain English why TwistedBubbleSort is different from BubbleSort. I.e., describe at least one difference in the swaps made by the two algorithms.

(b) (**5 points**) Prove that after the i-th iteration of the outer loop of TwistedBubbleSort, the largest element of A is at the i-th index.

(c) (**10 points**) Prove that after the i-th iteration of the outer loop of TwistedBubbleSort, indices 0 to i of the array are sorted.

6. **(0 points, optional)**[1] InsertionSort is a simple sorting algorithm that works as follows on input $A[0]$, $\ldots$, $A[n-1]$.

Algorithm 3 InsertionSort

```
Input: A[0],...,A[n-1]
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.