CS 170 Homework 1

Due **Monday 1/26/2026, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Course Policies

Please read through the course policies below:

1. The exams will be at the following times. We do not plan on offering alternate exams, so please mark your calendars:
   (a) **Midterm 1**: Thursday, 2/26/2026, 7:00 PM - 9:00 PM
   (b) **Midterm 2**: Thursday, 4/9/2026, 8:00 PM - 10:00 PM
   (c) **Final**: Friday, 5/15/2026, 7:00 PM - 10:00 PM
2. Homework is due Mondays at 10:00pm, with a late deadline at 11:59pm to protect against technical issues. There is no penalty for submitting before the late deadline, but absolutely no submissions will be accepted after 11:59pm.
3. Your lowest homework will be dropped at the end of the semester. By filling out the mid-semester survey and end-of-semester course evaluation, you have the opportunity for two additional drops.
4. Each homework will have one question for you to attempt on your own, labeled "solo question." You will get detailed feedback on your answer to help callibrate you on your learning. On the remaining questions you may collaborate with your fellow students and with AI tools, but you must always list your collaborators and write up the solutions in your own words. (Details in the policies linked below.)
5. The primary source of communication for CS170 will be through Edstem.
6. **Syllabus and Policies:** `https://cs170.org/policies/`
7. **Homework Guidelines:** `https://cs170.org/resources/homework-guidelines/`
8. **Regrade Etiquette:** `https://cs170.org/resources/regrade-etiquette/`
9. **Forum Etiquette:** `https://cs170.org/resources/ed-etiquette/`

Copy and sign the following sentence on your homework submission.

"I have read and understood the course syllabus and policies."

Solution: I have read and understood the course syllabus and policies. -Alan Turing

3 Asymptotic Complexity Comparisons

Order the following functions so that for all i, j, if f_i comes before f_j in the order then $f_i = O(f_j)$. Do not justify your answers.

- $f_1(n) = 3^n$
- $f_2(n) = n^{\frac{1}{3}}$
- $f_3(n) = 12$
- $f_4(n) = 2^{\log_2 n}$
- $f_5(n) = \sqrt{n}$
- $f_6(n) = 2^n$
- $f_7(n) = \log_2 n$
- $f_8(n) = 2^{\sqrt{n}}$
- $f_9(n) = n^3$
- $f_{10}(n) = \log_3 n$
- $f_{11}(n) = \log^2 n$

As an answer you may just write the functions as a list, e.g. $f_8, f_9, f_1, \ldots$

Solution: $f_3, \{f_7, f_{10}\}, f_{11}, f_2, f_5, f_4, f_9, f_8, f_6, f_1$

4 Counting Steps (Solo Question)

You can climb a ladder with n rungs by climbing either 1 rung or 2 rungs in each step. How many distinct ways are there to climb to the top?

(a) Give a simple recursive algorithm to compute the answer.

Solution: On input n, the desired algorithm $A(n)$ first checks if $n = 0$ or $n = 1$, in which case it outputs 1. Otherwise, for $n \geq 2$, $A(n)$ recursively calls $A(n-1), A(n-2)$, and outputs $A(n) = A(n-1) + A(n-2)$.

(b) Prove correctness using induction.

Solution: The proof is by strong induction. For the base case, a ladder with $n = 0$ or 1 rung by definition has $A(0) = A(1) = 1$ way to climb (take no steps, or take one 1-rung step, respectively).

For the inductive step when $n \geq 2$, assume by the inductive hypothesis that $A(n-1), A(n-2)$ output the correct values for inputs $n-1, n-2$ respectively. To climb an n-rung ladder, the first step either climbs 1 or 2 rungs. If the first step climbs 1 (resp. 2) rungs, there are $A(n-1)$ (resp. $A(n-2)$) ways to climb the remaining rungs. Hence the total number of ways to climb n rungs is $A(n) = A(n-1) + A(n-2)$. Thus $A(n)$ outputs the correct value, completing the inductive step.

5 Counting Steps Efficiently

(a) Analyze the running time of your algorithm from the previous question. Is it bounded by a polynomial in n? Assume for simplicity that two integers can be added in one timestep.

Solution: Let $T(n)$ denote the running time (i.e. number of integer operations) to compute $f(n)$. Then $T(0) = T(1) = 1$, and for $n \geq 2$, by definition

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

Therefore

$$T(n) \geq T(n-1) + T(n-2) \geq (T(n-2) + T(n-3)) + T(n-2) \geq 2T(n-2).$$

Expanding this recurrence $T(n) \geq 2T(n-2) \geq 2^2 T(n-4) \geq \cdots$ gives that $T(n) \geq 2^{n/2}$. (You can similarly show that $T(n) \leq 2^n$.)

(b) Give an efficient algorithm to compute the answer, and show the running time is bounded by $O(n)$ (polynomial in n is also acceptable).

Solution: The algorithm defines a length-$(n+1)$ array a and initializes $a[0] = a[1] = 1$. It then loops through $i = 2, \ldots, n$, and sets $a[i] = a[i-1] + a[i-2]$. By definition this algorithm performs n integer operations, and $a[n]$ equals the value $A(n)$ from the previous question (as both satisfy the same recursive relation and base cases).

(c) (Extra credit) Give an even more efficient algorithm to compute the answer, and analyze its running time, which should be sub-polynomial, again assuming each integer arithmetic operation takes 1 step. *Hint: can you compute the answer by multiplying* 2×2 *matrices?*

Is this running time a fair assessment of how the algorithm will perform in practice?

Solution: The recurrence relation $f(n) = f(n-1) + f(n-2)$ can be expressed in matrix notation as

$$\begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n-1) \\ f(n-2) \end{pmatrix}.$$

Let $M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. Expanding the RHS above with repeated applications of this formula gives

$$\begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix} = M \begin{pmatrix} f(n-1) \\ f(n-2) \end{pmatrix} = M^2 \begin{pmatrix} f(n-2) \\ f(n-3) \end{pmatrix} = \cdots = M^{n-1} \begin{pmatrix} f(1) \\ f(0) \end{pmatrix} = M^{n-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

We can use repeated squaring to compute M^{n-1} using $O(\log n)$ matrix multiplications. Specifically, letting $\ell = \lfloor \log_2(n-1) \rfloor$, we successively compute $M, M^2, M^4 = (M^2)^2, \ldots, M^{2^\ell} = (M^{2^{\ell-1}})^2$. Then M^{n-1} is a product of some subset of these matrices M^{2^i}. Each squaring operation performs a 2×2 matrix multiplication, which uses $O(1)$ integer arithmetic operations, for a total running time of $O(\log n)$ integer operations.

However, our assumption that integer arithmetic operations take constant time is unrealistic, as $f(n) = 2^{\Theta(n)}$ requires $\Theta(n)$ bits to store. Hence even outputting the result $f(n)$ takes $\Omega(n)$ time in practice.

