CS 170: Homework #1

Due on January 26, 2026 at 3:10pm

Zachary Brandt

1 Study Group

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

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."

I have read and understood the course syllabus and policies.

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. $f_8, f_9, f_1, \ldots$

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

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.

(b) Prove correctness using induction.

Solution

(a) Below is a simple recursive algorithm to compute the answer.

```
function STEPS($n$)
    if $n = 0$ then
        return 1
    else if $n < 0$ then
        return 0
    else
        return STEPS($n - 1$) + STEPS($n - 2$)
    end if
end function
```

(b) *Proof.* I'll prove by induction on n that the algorithm correctly computes how many distinct ways there are to climb a ladder with n rungs by climbing either 1 rung or 2 rungs in each step.

Base case: If $n = 0$, there is 1 way to climb the ladder, i.e., do nothing, and STEPS(0) returns 1.

Base case: If $n < 0$, there are 0 ways to climb the ladder, and STEPS(0) returns 0.

Inductive hypothesis: Assume that for all $0 \leq m \leq k$, STEPS(m) outputs the correct result.

Inductive step: For $n = k + 1$, the algorithm returns STEPS(k) + STEPS($k - 1$). By the inductive hypothesis, STEPS(k) correctly counts ways to reach rung k, from which it is possible to take 1 more step, and STEPS($k - 1$) correctly counts ways to reach rung $k - 1$, from which we it is possible to take 2 more steps. Their sum then counts all distinct ways there are to reach rung $k + 1$, i.e., the top, by climbing either 1 rung or 2 rungs in each 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.

(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).

(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

(a) The recurrence equation for the algorithm is $T(n) = T(n-1) + T(n-2) + O(n)$. No it is not bounded in polynomial time because the tree grows approximately exponentially in size with each increase in n.

(b) Below is an efficient algorithm to compute the answer.

```
function STEPS($n$)
    rungs[0] $\leftarrow$ 1
    rungs[1] $\leftarrow$ 1
    for $i \leftarrow 2$ to $n$ do
        rungs[$i$] $\leftarrow$ rungs[$i-1$] + rungs[$i-2$]
    end for
    return rungs[$n$]
end function
```

Let $C(n)$ denote the number of iterations of the for-loop in Steps(n). For $n \leq 1$, the loop does not execute, so $C(n) = 0$. For $n \geq 2$, the loop runs for $i = 2, 3, \ldots, n$, hence $C(n) = n - 1$. Each iteration performs one addition and a constant number of assignments, so costs $O(1)$ time. The initialization of rungs[0] and rungs[1] also costs $O(1)$. Therefore, the total running time is $O(1) + C(n) \cdot O(1) = O(n)$. The space usage is $O(n)$ to store rungs[0..n].

(c) Below is an even more efficient algorithm to compute the answer:

```
function STEPS($n$)
    $A \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
    $A \leftarrow$ MATRIXPOWER($A$, $n+2$)
    return $A[1,1]$
end function
```

The above algorithm produces the correct answer on an arbitrary input of $n \geq 1$. The function leverages matrix multiplication of a 2×2 matrix, A, by computing A^{n+2} and returning the bottom-right entry of the result. This works because A encodes the base cases of the function and computes an update in multiplication

$$\begin{bmatrix} \textsc{Steps}(n+1) & \textsc{Steps}(n) \\ \textsc{Steps}(n) & \textsc{Steps}(n-1) \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \textsc{Steps}(n+2) & \textsc{Steps}(n+1) \\ \textsc{Steps}(n+1) & \textsc{Steps}(n) \end{bmatrix}$$

and on an arbitrary input of $n \geq 1$, computing A^{n+2} will result in a 2×2 matrix with $\textsc{Steps}(n)$ in its bottom-right corner, which is returned.