CMSC 27200 - Spring 2024 — Lorenzo Orecchia
due 3/25/2024 via Gradescope

H1.

- The assignment is due at Gradescope on 3/25/2024.
- A LaTeX template will be provided for each homework. You are strongly encouraged to type your homework into this template using LaTeX. If you are writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will help facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please list all your collaborators in the appropriate spaces.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.
- Your homework is *resubmittable*. Please refer to the course syllabus on Canvas for a more detailed description of this. For any problem that you have not changed from your last submission, please make sure to indicate this in your submission to help our graders grade faster.

Problem 1 (Syllabus Reading Comprehension) *The following questions all pertain to course policy. None are trick questions, and all can be found in the syllabus on Canvas, or in the policies on the first page of this assignment. For this question, please enter your answer directly as text in Gradescope under HW1 - Problem 1.*

(a) You're having a tough first week back, and you want to turn in this homework assignment four days late. What point penalty will you incur on your submission? What benefits exist for turning your assignment in on time?

(b) You got an N on problem 3, but an E on everything else. You read the feedback, went to OH, and have a better solution now. You want to resubmit it, but you've made no changes to any of your other solutions. How should you format your solution for resubmission in Gradescope?

Collaborators:

Solution: See Gradescope.

Problem 2 (Practice with Big O.) *For this problem, you may want to review the definition of Big O, Big Ω and Big Θ [see DPV 0.3]. Helpful resources can also be found on Canvas. In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$), and* ***give a brief explanation*** *for each answer.*

(a) $f(n) = \sum_{i=1}^{n} i^k$ and $g(n) = n^{k+1}$ for constant $k > 0$.

(b) $f(n) = \sum_{i=1}^{n} i^k$ and $g(n) = n^{k+1}$ for $k \in \Theta(\log n)$.

(c) $f(n) = \binom{n}{5}$ and $g(n) = \sqrt{n}^{3\log_2 8}$

(d) $f(n) = 3^n$ and $g(n) = \sum_{i=1}^{n} 2^i$

(e) $f(n) = \log(n!)$ and $g(n) = n \log n$

(f) $f(m,n) = (n+m)^2$ and $g(m,n) = n^2 + m^2$

Collaborators:

Solution:

(a) $f = O(g)$

$f(n) = 1^k + 2^k + \cdots + n^k$
$g(n) = n^k + n^k + \cdots + n^k$
Therefore $f(n) \leq g(n)$ since the ith term of $g(n)$ is greater or equal to the ith term in $f(n)$.

(b) $f = O(g)$

For some c,
$f(n) = 1^{c\log(n)} + 2^{c\log(n)} + \cdots + n^{c\log(n)}$
$g(n) = n^{c\log(n)} + n^{c\log(n)} + \cdots + n^{c\log(n)}$
Therefore $f(n) \leq g(n)$ since the ith term of $g(n)$ is greater or equal to the ith term in $f(n)$.

(c) $f = \Omega(g)$

$f(n) = \frac{n!}{(n-5)!5!}$
$g(n) = n^{4.5}$
Factorial grows faster than polynomial

(d) $f = \Omega(g)$

$f(n) = 3^n$
$g(n) = 2^1 + 2^2 + \cdots + 2^n$
$f(n)$ grows faster than the highest order term of $g(n)$

(e) $f = O(g)$

$f(n) = \log(1) + \log(2) + \cdots + \log(n)$
$g(n) = \log(n) + \log(n) + \cdots + \log(n)$
ith term of $f(n)$ is less than or equal to ith term of $g(n)$

(f) $f = \Omega(n)$

$f(m,n) = n^2 + 2mn + m^2$
$g(m,n) = n^2 + m^2$

Problem 3 ("Conflicting" Definitions) *Surprisingly, our DPV textbook and the KT textbook provide different definitions for big-O notation. For $f, g : \mathbb{N}_{>0} \to \mathbb{R}_{>0}$, they are:*

- *DPV:* $f(n) = O(g(n)) \iff \exists c \in \mathbb{R}, \forall n \in \mathbb{N}_{>0},\ f(n) \leq c \cdot g(n)$
- *KT:* $f(n) = O(g(n)) \iff \exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}_{>0}, \forall n \geq n_0, \quad f(n) \leq c \cdot g(n)$.

*Write a **formal proof** that the two definitions are equivalent.*

Collaborators:

Solution:

Proof.

By transitivity, DPV is equivalent to KT if:

$$\exists c \in \mathbb{R}, \forall n \in \mathbb{N}_{>0},\ f(n) \leq c \cdot g(n) \iff \exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}_{>0}, \forall n \geq n_0,\ f(n) \leq c \cdot g(n)$$

To prove this, we prove the following:

1. **L.H.S $\implies$ R.H.S**

   If L.H.S holds, then $\exists c \in \mathbb{R}$ such that $f(n) \leq c \cdot g(n)$ for all $n \in \{1, 2, 3, \dots\}$. The R.H.S follows by choosing $n_0 = 1$, in which case it is an equivalent statement to the L.H.S.

2. **L.H.S $\impliedby$ R.H.S**

   If R.H.S holds, then $\exists c \in \mathbb{R}$ and $\exists n_0 \in \{1, 2, 3, \dots\}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

   Note that for $n < n_0$, it is possible $f(n) > c \cdot g(n)$

□

PROBLEM 4 (FASTER FIBONACCI) *This exercise is similar to 0.4 from [DPV]. Recall that the Fibonacci sequence is defined recursively by:*

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}.$$

Is there a faster way to compute the n^{th} Fibonacci number than by fib2 *(DPV, pg. 13)? One idea involves matrices.*

If you are not already familiar with matrix multiplication, you will need to learn just a bit of it here. For this problem, you should familiarize yourself with the following:

- *How to convert a linear system to a matrix equation*
- *How to multiply matrices*
- *Know that matrix multiplication is associative.*

We start by writing the equations $F_1 = F_1$ and $F_2 = F_0 + F_1$ in matrix notation:

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Similarly:

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

and in general:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}.$$

So, in order to compute F_n, it suffices to raise this 2×2 matrix, call it X, to the n^{th} power.

(a) *Show that two 2×2 matrices can be multiplied using 4 additions and 8 multiplications.*

(b) *Give an algorithm to compute X^n using only $O(\log n)$ matrix multiplications. You should provide pseudocode for your algorithm, and give a **formal proof** that it works correctly and only uses $O(\log n)$ matrix multiplications (Hint: Think about computing X^8).*

(c) *We name the algorithm from part (b)* fib3. ***Sketch** a proof that all intermediate results of* fib3 *are $O(n)$ bits long.*

(d) *Let $M(n)$ be the running time of an algorithm for multiplying n-bit numbers, and assume that $M(n) = O(n^2)$ (the school method for multiplication, in Chapter 1 of DPV, achieves this). **Sketch** a proof that the running time of* fib3 *is $O(M(n) log n)$.*

(e) ***Sketch** a proof that the running time of* fib3 *is $O(M(n))$. (Hint: The lengths of the numbers being multiplied get doubled with every squaring.)*

Collaborators:

Solution:

(a) Matrix multiplication is defined as:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

In this expression there are 8 unique multiplication operations and 4 unique addition operations.

(b) Consider the algorithm:

Algorithm 1 Recursive algorithm to compute X^n in logarithmic time

pow(X, n)
Input: $X \in \mathcal{M}_{2\times 2}$, $n \in \mathbb{N}$
Output: $Y \in \mathcal{M}_{2\times 2}$
1: **if** $n = 0$ **then return** I_2
2: **if** $n = 1$ **then return** X
3: $X_{1/2} \leftarrow pow(X, \lfloor \frac{n}{2} \rfloor)$
4: **if** n even **then return** $X_{1/2} \cdot X_{1/2}$
5: **if** n odd **then return** $X_{1/2} \cdot X_{1/2} \cdot X$

Proof.

To prove correctness, use strong induction on n.

- **Base Case**
  For $n = 0$, I_2 is returned.
  For $n = 1$, X is returned.
- **Inductive Case**
  Suppose the algorithm is correct for all $n \in \{0, 1, \ldots, k-1\}$.
  Since $k > 1$, the algorithm computes $X^{\lfloor \frac{k}{2} \rfloor}$ (Line 3). By the inductive hypothesis, this value is correctly computed because $\lfloor \frac{k}{2} \rfloor \leq k - 1$.
  If k is even, $X^{\lfloor \frac{k}{2} \rfloor} = X^{\frac{k}{2}}$ and $X^{\frac{k}{2}} \cdot X^{\frac{k}{2}} = X^k$ is returned (Line 4).
  If k is odd, $X^{\lfloor \frac{k}{2} \rfloor} = X^{\frac{k-1}{2}}$ and $X^{\frac{k-1}{2}} \cdot X^{\frac{k-1}{2}} \cdot X = X^k$ is returned (Line 5).

□

Proof.

We want to prove runtime is $O(\log n)$ with respect to matrix multiplication (matmul).

Note the algorithm is recursive. Each call returns if $n = 0$ or $n = 1$, or recursively calls $pow(X, \lfloor \frac{n}{2} \rfloor)$. The values of n up the call stack thus go as $n, \lfloor \frac{n}{2} \rfloor, \ldots, 1$. Therefore the number of calls goes as $O(\log n)$.

By inspection, the number of matmuls per call is constant and independent of n.

Therefore the whole algorithm is $O(\log n)$ with respect to matmul operations.

□

(c) *Proof.*

All intermediate results of fib3 are 2×2 matrices composed of 4 integers. To prove all such matrices have $O(n)$ bit complexity, use induction on n.

- **Base Case**
  The bit complexity grows proportional to n between $n = 1$, which requires 1 bit, and $n = 2$, which requires 2 bits.

- **Inductive Case**
  Suppose the bit complexities of the results for $n \in \{1, \ldots, k-1\}$ are proportional to n. The result for $n = k$ is computed using the result from $n = \lfloor \frac{k}{2} \rfloor$. The computation requires multiplying 2 $\lfloor \frac{k}{2} \rfloor$-bit integers to get a k-bit integer, in the worst case.
  Therefore the bit complexity is $O(n)$.

□

(d) *Proof.*

Recall from (b) that the number of calls goes as $O(\log n)$ and the number of matmuls per call is constant.

Also recall from (c) that each call must multiply 2 $\lfloor \frac{n}{2} \rfloor$-bit integers, in the worst case. This operation is $O(n^2)$. There are a constant number of these operations each call.

Therefore, the whole algorithm is $O(M(n) \log n)$ with respect to bit operations.

□

(e) *Proof.*

Recall from (c) that each call has bit complexity $O(n)$. Each call performs matmuls that require a constant number of integer multiplications, which are $O(n^2)$. Since n roughly halves going up the call stack, the runtimes thus go as $O(n^2), O(\frac{n^2}{4}), \ldots, O(1)$

Recall from (b) that there are $O(\log n)$ calls, so the whole algorithm is $O(\sum_{i=0}^{\log n} \frac{n^2}{2^{2i}})$. As n grows, the number of addends grows logarithmically. However, the denominators grow exponentially, strongly suppressing the additional terms. Therefore only the first few terms, which go as $O(n^2)$ are relevant. The whole algorithm is thus $O(n^2)$.

□

Problem 5 (Fibonacci Implementation) *For this problem, you will be asked to implement a function that computes the value of the n^{th} Fibonacci number, and compare its runtime to that of the algorithms seen in class. You can find the questions here. Create a copy of the document, implement the required code, turn on link sharing, and include a link to your document here.*

Collaborators:

Solution: colab.research.google.com