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:

(a) There's no penalty for submitting an assignment four days late. The benefits of submitting on time include

1. Keeping up the pace with the class.
2. Receiving feedback on time.

(b) As each question is separated on Gradescope, we could just submit the changed solution to the specific question.

Extra space for your solution

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) $f = O(g)$: This is because $g(n) = n^{k+1}$ always grows faster than $f(n) = \sum_{i=1}^{n} i^k$ for $k > 1$

(b) $f = O(g)$: $k \in \Theta(\log n)$, g(n) still grows faster than $f(n)$.

(c) $f = \Omega(g)$: $\binom{n}{5}$ is $\frac{n!}{5!(n-5)!}$, which approximate to $\frac{n^5}{5!}$. $\sqrt{n}^{3\log_2 8}$ can be written as $n^{\frac{9}{2}}$. As n^5 grows faster than $n^{\frac{9}{2}}$ when n approaches infinity, $f = O(g)$.

(d) $f = \Omega(g)$: The sum of 2^i from $i = 0$ to $i = n$ is $2^{n+1} - 1$. This exponent with a base 2 is smaller than 3^n. Therefore, f is lower bounded by g.

(e) $f = O(g)$: $f(n) = \log(n!)$ can be written as $\log(n)\log(n-1)\ldots\log(1)$, which has n items in between. This always grows slower than $n\log(n)$.

(f)
 (1) $f = O(g)$ if m or n < 0: $f(m,n) = (n+m)^2$ can be written as $n^2 + m^2 + 2mn$. As $2mn < 0$, we have $f(m,n)$ smaller than $n^2 + m^2$.
 (2) $f = \Omega(g)$ otherwise: In this case, $2mn$ is positive, so it's an addition to $m^2 + n^2$. Therefore we have $f = \Omega(g)$.

Extra space for your solution

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

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

Collaborators:

Solution: if$f(n) = O(g(n))$ by the DPV definition, then we have a constant c such that $f(n) \le c \cdot g(n)$ for all n. This directly satisfies the KT definition, where we could simply choose $n_0 = 1$. Therefore, according to DPV, this also meets the requirement for KT. On the other way around, we assume $f(n) = O(g(n))$ by the KT definition. This means there exists a c and n_0 such that$f(n) \le c \cdot g(n)$ for all $n \ge n_0$. As the KT definition restricts the value of n to be greater or equal to n_0, which is a subset of the condition of n in the DPV definition that requires the inequality to hold for all n. Therefore, KT inherently satisfies the DPV definition as well. Both ways show they're equal, concluding our proof.

Extra space for your solution

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 $O(M(n) logn)$.*

*(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) To show that two 2×2 matrices can be multiplied using 4 additions and 8 multiplications, consider two matrices $A = \begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix}$ and $B = \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix}$. The product AB is given by:

$$AB = \begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix} = \begin{pmatrix} a_1a_2 + b_1c_2 & a_1b_2 + b_1d_2 \\ c_1a_2 + d_1c_2 & c_1b_2 + d_1d_2 \end{pmatrix}$$

(b) The algorithm to compute X^n using only $O(\log n)$ matrix multiplications is the attached Algorithm 1.
Proof:

Base Case:

- $n = 0$: By definition, any non-zero square matrix raised to the power of 0 is the identity matrix.
- $n = 1$: X is itself.
  Base cases use $O(\log n) = O(\log 1) = 0$ multiplications ($\log 0$ is invalid, we ignore this trivial case).

Inductive Step: Assume the algorithm correctly computes X^k for all integer $0 \leq k < n$, and assume the algorithm performs $O(\log k)$ multiplications for $k < n$.

- If n is even, then by our hypothesis, the algorithm can compute X^n as $X^n = X^{2m} = X^m \cdot X^m$, where m is a value of k. Therefore, squaring it will give us the correct value of X^n. Moreover, by inductive hypothesis, X^m requires $\log m = \log \frac{n}{2})$ multiplications, so need one additional multiplication to get n. Therefore, the number of multiplications is $O(\log \frac{n}{2}) + 1$, which can be simplified to $O(\log(n))$.
- if n is odd, then $n - 1$ is even, and by hypothesis, the algorithm can compute $X^{(n-1)/2}$, we call it X^m, correctly as $n - 1 \in k$. Then we have $X^n = X^{2m+1} = X \cdot X^m \cdot X^m$. X^m is correct in the hypothesis, then squaring it and multiplying it by X gives the correct X^n. Moreover, by inductive hypothesis, X^m requires $\log m = \log \frac{n}{2}$ multiplications, we need to square it and multiply by X, so we have $O(\log(\frac{n}{2}) + 2) = O(\log(n))$ multiplications.

  As both cases are proven right, this concludes our proof.

Algorithm 1 fib3

```
    Input: One squared matrix $X$ and a number $n$
    Output: A squared matrix $X^n$
if $n == 0$ then return $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
else if $n == 1$ then return X
else
    $Y = X^{\frac{n}{2}}$
    if $n \mod 2 == 0$ then
        return $Y * Y$
    else
        return $X * Y * Y$
    end if
```

(c) **Proof:** The nth Fibonacci number never exceeds 2^n. Therefore, the number of bits required to represent it is at most n bits. In the above algorithm, intermediate results are obtained by matrix multiplications involving matrices whose elements are Fibonacci numbers. Since the largest Fibonacci number involved in calculating nth Fibonacci number is F_n, which requires $O(n)$ bits, all intermediate matrix elements are also $O(n)$ bits.

(d) For each iteration, the running time is $M(n)$, and we have $O(\log n)$ iterations. Therefore, we have $O(M(\text{n}) \log n)$ running time.

(e) As each squaring doubles the bits needed to represent the number, and we need $O(\log(n))$ squarings, we have $O(1) + O(2^2) + O(2^4) + \cdots + O(2^{2\log(n)})$, which finally converges to $O(n^2)$. This is equivalent to $O(M(n))$.

Extra space for your solution

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: Click Here

-