CMSC 27200 - Spring 2024 — Lorenzo Orecchia
due 3/29/24 via Gradescope

H2.

- The assignment is due at Gradescope on 3/29/24.
- 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 Consider the recursive version of the gcd algorithm given in class and in the DPV textbook.

1. How much space, i.e., memory allocation, does this algorithm require as stated?
2. Write pseudocode for an iterative version of the algorithm that reduces the space requirement to $O(n)$ bits.

Collaborators:

Solution:

1. The recursive version of the gcd algorithm requires $O(n)$ space, where n is the number of recursive calls the algorithm makes.
2. The algorithm is as follows:

Algorithm 1 Iterative GCD Algorithm

Input: Two numbers a and b with $a > b \geq 0$
Output: The greatest common divisor (gcd) of a and b

```
while b ≠ 0:
    temp = b
    b = a mod b
    a = temp
end while
Return a
```

This algorithm uses a constant amount of space, $O(1)$ space complexity, which is O(n) bits.

Problem 2 Let F_i be the ith Fibonacci number. Consider the execution of the recursive gcd algorithm on inputs (a, b) with $a > b \geq 0$. Show that if $b \leq F_k$ for some k, then the algorithm makes at most k recursive calls.

Collaborators:

Solution: Let's prove this by induction
Proof.

Base Case: For $k = 0$, we have $F_0 = 0$. If $b \leq 0$, then the algorithm will terminate immediately without making any calls. For $k = 1$, we have $F_1 = 1$. If $b \leq F_1 = 1$, so base cases hold.

Inductive Hypothesis: Assume that the statement holds for $k \geq 1$ such that if $b \leq F_k$, then the algorithm makes at most k recursive calls. We need to show it holds for $k + 1$.

1. If $b \leq F_k$, then the algorithm makes at most k recursive calls by our hypothesis.
2. If $F_k < b \leq F_{k+1}$, then the algorithm will make a recursive call with $a = b$ and $b = a \mod b < F_k$. Thus, the algorithm will make at most k recursive calls by our hypothesis. Since the initial call is not counted, the total number of recursive calls is at most $k + 1$.

In both cases, the algorithm makes at most $k + 1$ recursive calls. Therefore, by induction, the statement holds for all $k \geq 0$. □

Problem 3 (Programming: Probabilistic Testing and Primality) Follow this Google Colab template link to implement Fermat primality testing. Do not attempt to edit the linked file. Create your own copy instead by clicking File → Save a copy in Drive. When you have completed it, download it as a Jupyter Notebook and submit it to the appropriate Gradescope assignment.

Collaborators:

Solution: Click Here

Problem 4 Wilson's theorem says that a number N is prime if and only if

$$(N-1)! \equiv -1 (\text{mod } N).$$

In this problem, we will walk through writing a formal proof for this theorem.

(a) We can prove Wilson's Theorem by proving three lemmas:

Lemma 1 If p is prime, only 1 and $p-1$ are their own inverses modulo p.

Lemma 2 If p is prime, $(p-1)! \equiv -1 (\text{mod } p)$.

Lemma 3 ?????

What is the third lemma that needs to be proven for a complete proof of Wilson's Theorem?

(b) Provide a proof of Lemma 1. You may use the fact that if p is prime, then we know every number $1 \leq x < p$ is invertible modulo p.

(c) Provide a proof of Lemma 2 by pairing up multiplicative inverses. You may use the fact that every number $1 \leq x < p$ has a unique inverse modulo p.

(d) Provide a proof of Lemma 3. (Hint: Note that this can only happen if:

$$\gcd(N, (N-1)!) = 1.$$

)

(e) We've just shown that Wilson's theorem is an if-and-only-if condition for primality. Why can't we base a primality test algorithm on Wilson's Theorem?

Collaborators:

Solution:

(a) if N is composite, then $(N-1)! \not\equiv -1 (\text{mod } N)$.

(b) For any prime p, every number $1 \leq x < p$ is invertible modulo p. This implies that for each x in the range $1 \leq x < p$, there exists a unique x^{-1} such that $x \cdot x^{-1} \equiv 1 (\text{mod } p)$. The numbers that are their own inverses are 1 and $p-1$. Only these two numbers satisfy $x \cdot x^{-1} \equiv 1 (\text{mod } p)$ as if other number does, then it implies that p divides either $x-1$ or $x+1$.

(c) For any prime p, according to Lemma 1, each number 2 through $p-2$ is invertable modulo p. Pairing up these inverses with each number in $(p-1)!$, we get that the product of all numbers from 2 through $p-2$ is congruent to 1 modulo p as $x \cdot x^{-1} \equiv 1 (\text{mod } p)$. This implies that $(p-1)! \equiv -1 (\text{mod } p)$.

(d) Assume that N is composite. Then, N has a factor d such that $1 < d < N$. Since $d$divides N, it also divides $(N-1)!$ as the product of $(N-1)!$ contains N. Therefore, $\gcd(N, (N-1)!) \neq 1$. This implies that if N is composite, then $(N-1)! \not\equiv -1 (\text{mod } N)$.

(e) This is because the theorem is computationally inefficient. Calculating $(N-1)!$ modulo N needs to deal with $O(n)$ multiplications and also very large intermediate product, which significantly increases the time complexity that grows extremely rapidly in a way that is much faster than expotential functions. Thus, the primality test is not practical for large N.