CSc 144–002 — Discrete Mathematics for Computer Science I — Fall 2023 (McCann)
https://cs.arizona.edu/classes/cs144/fall23-002/

Homework #6

(50 points)

Due Date: November 10th, 2023, at the beginning of class

Directions

1. **This is an INDIVIDUAL assignment; do your own work! Submitting answers created by computers or by other people is NOT doing your own work.**
2. Start early! Getting help is much easier n days before the due date/time than it will be n hours before. Help is available from the class staff via piazza.com and our office hours.
3. Write complete answers to each of the following questions, in accordance with the given directions. Create your solutions as a PDF document such that each answer is clearly separated from neighboring answers, to help the TAs easily read them. Show your work, when appropriate, for possible partial credit.
4. When your PDF is ready to be turned in, do so on gradescope.com. Be sure to assign pages to problems after you upload your PDF. Need help? See "Submitting an Assignment" on https://help.gradescope.com/.
5. **Solutions submitted more than five minutes late will cost you a late day. Submissions more than 24 hours late are worth no points.**

Topic: Proof of Biconditionals

Reminders: To prove a biconditional, you have to do two proofs, one of the "if" and one of the "only if." You may use different proof techniques for each.

1. (8 points) Prove this conjecture: n is even iff $4 \mid n^2$.
2. (10 points) Prove this conjecture: $7 \mid i$ iff $7 \mid (2i)$, where $i \in \mathbb{Z}$. *Hint:* One of the two proofs is very easy, but the other is more challenging. Remember that an odd times an odd is always odd, but an even times either an odd or an even is always even.

Topic: Integers

3. (8 points) For each of the following congruences, (i) list the smallest five values of a that satisfy the congruence, and (ii) create a function on the domain $\mathbb{Z}^*$ whose range is all values of a that satisfy the congruence.
   (a) $a \equiv 3 \pmod{7}, a \in \mathbb{Z}^*$
   (b) $a \equiv 7 \pmod{2}, a \in \mathbb{Z}^*$
4. (4 points) Which, if any, of the following sets of integers are pairwise relatively prime? For those that are not, explain why they are not.
   (a) 31, 32, 33
   (b) 27, 56, 175
5. (4 points) For each of the following pairs of values or expressions, what is their GCD and their LCM?
   (a) $2^2 \cdot 3^3 \cdot 5^2,\ \ 2^2 \cdot 3^2 \cdot 5^3$
   (b) $231,\ 650$

(Continued on the back . . .)

Program: Testing Primality

6. (16 points) We learned in class that we can test an integer n to see if it is prime by testing it against values up to and including $\sqrt{n}$. There's another theorem that might help even more: All prime integers ≥ 5 are either one less than, or one more than, a multiple of six. Note that not all such integers are prime. For example, 24 is a multiple of 6. 23 is prime, but 25 is not.

   (The proof is easy: All integers have one of these six forms: $6x, 6x+1, 6x+2, 6x+3, 6x+4$, and $6x+5$. Numbers from $6x, 6x+2$, and $6x+4$ are all even, and $3 \mid (6x+3)$. That leaves $6x+1$ and $6x+5$ (which is the same form as $6x-1$) as the only two forms that could be prime.)

   Would using this result speed up primality testing significantly? Or maybe the 'brute force' technique of just dividing the integer by all values from 2 up to itself is already fast enough? One way to find out is to code them up and check their speed.

   Write a complete Python 3 or Java 16 program (your choice!) that implements three different primality tests, runs all three on an integer provided by the user, and reports how long it took each of them to test that integer. The three primality approaches that you need to code are:

   (a) **Brute Force (2 Up To n):** Given the integer n, this method checks each integer from two though $n-1$ to see if it divides n evenly. If a divisor is found, it immediately returns false. Otherwise, it returns true.

   (b) **Square Root (2 Through $\sqrt{n}$):** This is the same as the brute force approach, except that we only check from two through $\sqrt{n}$.

   (c) $6x \pm 1$ **(2 Through $\sqrt{n}$):** This is the same as the second approach, but after eliminating those $6x+y$ forms that cannot be prime, checks the remaining values that are one more than or one less than multiples of six.

   Code each as a separate Python 3 function (or Java 16 method), and have your main program call each in turn.

   Measuring the time each approach requires depends on the language. In Python 3, import the counter (`from time import perf_counter`), then record the time both before and after you call the approach to be measured. The difference is the elapsed time in secounds. Java 16's method of choice is `System.nanoTime()`, which reports the current time in billionths of a second, so a division is needed to convert to seconds. For example, with Python on the left and Java on the right:

```
start = perf_counter()          start = System.nanoTime()
answer = method1(n)             answer = method1(n)
stop = perf_counter()           stop = System.nanoTime()
seconds = stop - start          seconds = (stop - start) / 1000000000.0;
```

   Input: The user is to provide n, the integer whose primality is to be tested. You may prompt the user for n, or get n from the command line, or both.

   Output: Your program is to report the time required by each method to test the user's input value. Formatting of the numeric values is optional. This output format is suggested but is not required:

```
Brute Force took 8.546489319 seconds to find that 1000000007 is prime.
Square Root took 0.001104095 seconds to find that 1000000007 is prime.
   6x +/- 1 took 3.20028E-4 seconds to find that 1000000007 is prime.
```

   Hand In: To 'answer' this question in your PDF, include the following items within the PDF:

   (a) Your complete source code, including what you feel is adequate documentation.

   (b) Your program's output when run on the following set of inputs: 2, 4, 5, 1000000007, and 1761059681.