- "*Tell me, and I forget. Teach me, and I may remember. Involve me, and I learn.*" — attributed to Benjamin Franklin.
- These exercises are my attempt at getting you involved in the material that we saw in the lectures. You are *highly encouraged* to solve these problems and get your solutions verified by the TAs.
- You are not *required* to submit your solutions to these problems. You won't get any credit just for solving these problems.
- You *may* take the help of any resource (e.g., your classmates, friends, the TAs, the internet) for figuring out these problems.
- Continuous evaluation for this course will be in the form of in-class quizzes. The questions in these quizzes will be (loosely) based on these exercises (and others to come). If you have learned how to solve these exercises, you should have no trouble doing well in the quizzes. Otherwise, otherwise.

The first few questions are based on the following procedure that we saw in class.

DIVIDE(x, y)

```
1  if x = 0
2        q = 0
3        r = 0
4        return (q, r)
5  z = ⌊x/2⌋
6  (q, r) = DIVIDE(z, y)
7  q = 2 × q
8  r = 2 × r
9  if x is odd
10       r = r + 1
11 if r ≥ y
12       r = r − y
13       q = q + 1
14 return (q, r)
```

1. Verify (by "running" the code by hand) that this pseudocode works correctly for a handful of (valid) inputs x, y of your choice.

2. Translate this pseudocode to Python. Generate a large number of valid (x, y) pairs. Run your program on these inputs (x, y). Programmatically verify (how? ...) that the outputs match the expected (q, r) values.

3. Complete the argument of correctness of DIVIDE(x, y) that we did in class, using induction on the *value* of x.

4. Write an argument of correctness of DIVIDE(x, y) using induction on the *number of bits* in the binary representation of x.

> The next few questions are based on the following procedure that we saw in class.

```
EUCLID(x, y)
1  if y = 0
2        return x
3  (q, r) = DIVIDE(x, y)
4  gcd = EUCLID(y, r)
5  return gcd
```

5. Verify (by "running" the code by hand) that this pseudocode works correctly for a handful of (valid) inputs x, y of your choice.

6. Translate this pseudocode to Python. Generate a large number of valid (x, y) pairs. Run your program on these inputs (x, y). Programmatically verify (how? ...) that the outputs match the expected values.

7. Recollect and write down the argument of correctness of EUCLID(x, y) that we did in class.

8.

   (a) Write pseudocode for a recursive function ODDOREVEN(x) that takes a positive integer x as argument and returns whether x is even or odd. Use the following logic for this function:

      1. Return the correct answer when $x \in \{1, 2\}$
      2. If $x > 2$ then: Compute the answer for $x - 2$. Use this answer to construct the correct answer for x, and return it.

   (b) Implement your pseudocode in Python and test it on a few hundred inputs.

   (c) **Prove** using induction (on what?) that your pseudocode is correct.

9.

(a) Write pseudocode for a recursive function OddOrEven(x) that takes a positive integer x as argument and returns whether x is even or odd. Use the following (different) logic for this function:

1. Return the correct answer when $x \in \{1, 2\}$
2. If $x > 2$ then: Compute the answer for $x - 1$. Use this answer to construct the correct answer for x, and return it.

(b) Implement your pseudocode in Python and test it on a few hundred inputs.

(c) **Prove** using induction (on what?) that your pseudocode is correct.

10. Consider the following set of operations to compute a number x:

1. Initialize: $x = 1$
2. Do either one of the following steps, a finite number of times:
   (a) $x = (x \times 2)$
   (b) $x = (x + 3)$

Note that the two types of steps can be *interleaved*: you can opt to multiply in one step, and add in the next, and vice versa.

Here are some easily verified examples of numbers x that can be obtained in this way:

- $x = 1$
- $x = 2$
- $x = 4$

Here are some less obvious examples. All of these were obtained using **ten** update operations:

- $x = 262$
- $x = 646$
- $x = 800$
- $x = 61$
- $x = 44$

(a) Write the pseudocode for a **recursive** function IsValid(y) that takes a positive integer y as argument and returns:

- **True** if y can be obtained by starting with $x = 1$ and applying some finite sequence of operations as described above, and

- **False** otherwise. That is, the function should return **False** if, starting with $x = 1$, *no possible finite sequence of these operations, howsoever long,* will result in the number y.

(b) Prove using induction that your pseudocode is correct.

11. The sequence of *Fibonacci numbers* is defined as follows:

$$f_1 := 1,\ f_2 := 2,\ f_{n+2} := f_{n+1} + f_n,\ \forall n \in \mathbb{N}$$

Given natural numbers a, b with $a < b$, give a *recursive* algorithm to find whether a, b are *consecutive* Fibonacci numbers; that is, whether a, b are of the form f_i, f_{i+1}, respectively, for some $i \in \mathbb{N}$. Prove using induction that your algorithm is correct.

12. A *graph* G is defined as an ordered pair (V, E) where V is a finite set of *vertices* and $E \subseteq \binom{V}{2}$ is a set of *edges*. For more information, visit Graph. A graph is *connected* if there is a path between every pair of vertices. For more information, visit Connectivity. Given a connected graph $G := (V, E)$ and a vertex $v \in V$, consider the following algorithm:

```
DFS(G, v)
1  Label v as Discovered
2  for all edges {v, w}
3      if w is not labeled Discovered
4          DFS(G, w)
```

Prove that

(a) The algorithm DFS terminates.

(b) Upon termination, every vertex in the graph G is labeled as Discovered.

13. Write the pseudocode for a **recursive** function IsPalindrome(string, start, end) that checks if the given string is a palindrome within the index range start to end. Prove using induction that your pseudocode is correct.

14. Write the pseudocode for a **recursive** function FindLargest(numbers) that finds the largest number in the list numbers given as its argument. Prove using induction that your pseudocode is correct.

The remaining questions are links to programming problems from LeetCode. Solve each of them using a **recursive** algorithm implemented **in Python3**. Test your program using LeetCode's testing mechanism.

15. Power of three
16. Reverse linked list
17. Palindrome linked list
18. Merging sorted lists
19. Predict the winner