22C:16 Homework 9

Due via ICON on Friday, May 4th, 4:59 pm

What to submit: Your submission for this homework will consist of 5 files: (i) `sumDigits.py` for Problem 1, (ii) `recursiveReverse.py` for Problem 2, (iii) `line.py` for Problem 4, (iv) `fraction.py` for Problem 5, and (v) `homework9.pdf` for Problem 3. Each of these Python files should start also with your name, section number and student ID appearing at the top of the file as Python comments. Also make sure that your Python code is appropriately commented. Similarly, your pdf file should also start with your name, section number, and student ID apearing at the top of the file. You will get no credit for this homework if your files are named differently, have a different format, or if your files are missing your information.

1. Write a function called `sumDigits` that takes as a parameter a non-negative integer `n` and returns the sum of the digits of `n`. For example, if `n` were 8971, then the function would return 25 (which is 8 + 9 + 7 + 1). The function should be recursive and should not contain any loops.

   A simple way to solve this problem recursively is to pull out the right most digit (i.e., the one in the one's place), find the sum of the digits in the rest of the number, and then add the digit that was pulled out to the sum.

   Make sure that your solution is well-commented, with the base case(s) and recursive case(s) clearly identified. Submit your solution in a file called `sumDigits.py`.

2. Write a function called `recursiveReverse` that takes a list `L` as a parameter and returns the reversed version of `L`. The function should contain no loops and should do what it is supposed to do by using recursion. The algorithm you should use in this solution is simple: pull out the first element of `L` (i.e., `L[0]`), reverse the rest of the list, and stick the pulled out element at the back of the reversed sub-list.

   Make sure that your solution is well-commented, with the base case(s) and recursive case(s) clearly identified. Submit your solution in a file called `recursiveReverse.py`.

3. This problem asks you to compare the running times of *selection sort* and *merge sort*. Use the Python code posted one the course website for both of these algorithms.

   To perform this experiment, let us start with a positive integer n that will be the size of the list you want sorted. Randomly generate m lists, each of size n, and sort each of these lists first using selection sort and then using merge sort. Compute the average running time of selection sort, with the average taken over m runs. Similarly, compute the average running time of merge sort. My suggestion is to use $m = 20$; but depending on how slow your selection sort function is running, you may have to reduce this down to $m = 10$. For n, my suggestion is to use values 50,000 to 140,000 in increments of 10,000.

   An easy way to generate random (unsorted) sequences of numbers is to use the `shuffle` function from the `random` module.

   After you have finished your experiment, make two plots, one showing the running time of selection sort and the other showing the running time of merge sort. Write a paragraph discussing your results. Make sure your write-up address the following issues: (i) which of the algorithms is faster and how do the relative times change with increasing n, and (ii) do the plots of the running times you see in your experiments correspond to what you have learned about the running times of these algorithms in class.

4. Add a method to the line class called `intersect`. This method would be called as `L1.intersect(L2)`, where `L1` and `L2` are instances of the `line` class. If `L1` and `L2` do

intersect, then this method should return an instance of the `point` class representing the point of intersection of the two lines. If `L1` and `L2` do not intersection, then this method should return `None`.

The problem of finding the intersection of two line segments is a basic operation that graphics programs need to perform extremely fast and accurately. On the course page I have posted a link to a solution on the web that is elegant and fast. You should consider using this, though you should also feel free to come up with your own solutions to the problem.

You should save the updated `line` class in a file called `line.py` and submit this as your solution.

5. Implement a class called `fraction` that we can use to represent fractions such as 2/3, -11/98, 73/17, etc. For example, I would like create an instance `x` of the `fraction` class as follows:

```
x = fraction(10, 21)
```

This would create an instance `x` of the `fraction` class representing the fraction 10/21. I would like you to implement the following methods for the `fraction` class:

(a) The `add` method that could be called as:

```
x.add(y)
```

Here `x` and `y` are instances of the `fraction` class and after the call to `add` the fraction `x` takes on the value of `x + y`.

(b) The `subtract` method that could be called as:

```
x.subtract(y)
```

and it would behave similarly to the `add` method.

(c) The `multiply` method that could be called as:

```
x.multiply(y)
```

and it would behave similarly to the `add` method.

(d) The `divide` method that could be called as:

```
x.divide(y)
```

and it would behave similarly to the `add` method.

(e) The `numerator` method that returns the numerator of the `fraction` instance.

(f) The `denominator` method that returns the denominator of the `fraction` instance.

I want you to take care to ensure that fractions are maintained in their reduced form (e.g., 1/2 rather than 50/100) by the `fraction` class. For example, even if the `fraction` instance `y` is constructed as follows:

```
y = fraction(50, 100)
```

internally `y` should be represented as 1/2. Similarly, after each arithmetic operation, you should ensure that the instance that was operated upon is still in its reduced form.

One way to ensure that `fraction` instances remain in their reduced form is to compute the greatest common divisor (gcd) of the numerator and the denominator and to divide both the numerator and the denominator by this gcd. For this purpose, I want you to implement a function called `gcd` that takes two non-negative integers and returns their greatest common divisor. You will recall that we have studied *Euclid's algorithm* for computing the gcd of two numbers and this is the algorithm you should implement for the `gcd` function.

Suggestion: Each instance of the `fraction` class should have two integer attributes to keep track of the numerator and denominator of the fraction.

You will have to submit the `fraction` class, consisting of the constructor, methods for the four arithmetic operations mentioned above, the `numerator` and `denominator` methods, and the `gcd` function. The class should be saved in a file called `fraction.py`.
