COS 126 Midterm 1 Programming Exam Spring 2015

This part of your midterm exam is like a mini-programming assignment. You will create four programs, compile them, and run them on your laptop. Debug your code as needed. This exam is open book, open browser—but only our course website and booksite! Of course, no internal or external communication is permitted (e.g., talking, email, IM, texting, cell phones) during the exam. You may use code from your assignments or code found on the COS126 website. When you are done, submit your program via the course website using the submit link for Programming Exam 1 on the Assignments page.

Grading. Your program will be graded on correctness, clarity (including comments), design, and efficiency. You will lose a substantial number of points if your program does not compile or does not have the proper API, or if it crashes on typical inputs.

Even though you will electronically submit your code, *you must turn in this paper* so that we have a record that you took the exam and signed the Honor Code. *Print your name, login ID, and precept number on this page* (now), and write out and sign the Honor Code pledge before turning in this paper. *Note*: It is a violation of the Honor Code to discuss this midterm exam question with anyone until after everyone in the class has taken the exam. You have 90 minutes to complete the test.

"I pledge my honor that I have not violated the Honor Code during this examination."

____

____

[*copy the pledge onto these lines*]

____

[*signature*]

NAME: ________________________

LOGIN: ________________________

PRECEPT: ________________________

Part 0	/12
Part 1	/8
Part 2	/5
Part 3	/5
TOTAL	/30

The *aliquot sum* of a positive integer is the sum of its positive divisors other than itself (these divisors are called its *proper divisors*). For example, the aliquot sum of 12 is 16 because 1 + 2 + 3 + 4 + 6 = 16, and the aliquot sum of 25 is 6 because 1 + 5 = 6. In this exam you will write code that explores some interesting properties of aliquot sums.

Part 0: Aliquot.java (12 points)

First, write a program to compute and print the aliquot sum of an integer entered on the command line. Since no proper divisor of a positive integer n is bigger than n/2, just add up all the proper divisors up to n/2. (There are faster ways to do this, but using them will not earn any extra points.)

Note that the aliquot sum of the integer 1 is 0 (no proper divisors other than itself), and the aliquot sum of 0 we will arbitrarily say is also 0.

In Aliquot.java, you will write a `main()` to compute the aliquot sum of an integer n, entered on the command line.

The behavior of your program should exactly match the following examples:

```
% java-introcs Aliquot 12
The Aliquot sum of 12 is 16

% java-introcs Aliquot 25
The Aliquot sum of 25 is 6

% java-introcs Aliquot 13
The Aliquot sum of 13 is 1
```

There is no need to use arrays for this program, nor for any part of this exam. (We will deduct points if you do.)

Part 1: Perfect.java (8 points)

In Perfect.java, first copy the code you wrote in Part 0 and change it into a static method that returns the answer:

```
// returns aliquot sum of n
public static int aliquot(int n)
```

Note that this method, unlike the main() method in your Aliquot.java, does not read from the command line, and does not print anything.

Then write two other methods to check for special properties of the aliquot sum. A *perfect number* is one whose aliquot sum is equal to itself. The smallest such number is 6, since 6 = 1 + 2 + 3. Write a method, isPerfect(), to determine this:

```
// returns true if n is a perfect number, returns false otherwise
public static boolean isPerfect(int n)
```

Amicable numbers are two different numbers, each of which is the aliquot sum of the other. The smallest such pair is 220 and 284: the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, and 110, which sum to 284; the proper divisors of 284 are 1, 2, 4, 71,and 142, which sum to 220. Write another method to determine whether a number is a member of such a pair:

```
// returns true if n is an amicable number, returns false otherwise
public static boolean isAmicable(int n)
```

Note that a perfect number in not in an amicable pair with itself!

You may write a main() method to test your other methods, but you may also using DrJava's Interactions Pane to test. You can do so by typing, for example: `Perfect.isPerfect(6)` or `Perfect.isAmicable(220)`. The result in both cases should be `true`. Test with other inputs as well. Any main() you write in Perfect.java will not be graded.

Part 2: Cuddly.java (5 points)

An *aliquot sequence* is a sequence of numbers each of which is the aliquot sum of its predecessor. For example, the aliquot sequence starting at 10 is 10, 8, 7, 1, 0.

Some aliquot sequences repeat; perfect numbers and amicable numbers are examples. If an aliquot sequence starting at some integer n eventually cycles back to n, we will refer to the sequence as an *aliquot cycle* and to the number of elements in the cycle as the *period* of the cycle. Thus, perfect numbers have period 1 and amicable numbers have period 2.

Sociable numbers are numbers in an aliquot cycle whose period is greater than 2. The first such sociable group is 12496, 14288, 15472, 14536, 14264.

For this part of the exam, first write a method to determine the period (if any) of the aliquot sequence starting at n:

```
// returns period of aliquot sequence starting at n if it exists
// returns -1 otherwise
public static int aPeriod(int n)
```

If the sequence starting at n reaches 0, return -1. Also return -1 if the sequence doesn't come back to n within 30 steps. Avoid the magic constant by including this line at the top of Sequence.java:

```
private static final int MAX = 30;
```

Write another method to determine whether a number is a member of a sociable group (with period bounded by MAX):

```
public static boolean isSociable(int n)
```

Numbers that are perfect, or amicable, or sociable are together called *cuddly numbers*. Write a method to decide whether a number is cuddly (subject to looking only MAX steps ahead):

```
public static boolean isCuddly(int n)
```

Do not copy your aliquot() method from Perfect.java. Instead, use it by calling Perfect.aliquot().

You may write a main() method to test your other methods, but you may also using DrJava's Interactions Pane to test them. Any main() you write in Cuddly.java will not be graded.

Part 3: Cycle.java (5 points)

Write a program, **Cycle.java**, that prints information about the aliquot sequence of each number in StdIn. Your program will have a main() method that reads integers from StdIn and prints results to StdOut. It will use the methods you wrote in Perfect.java and Cuddly.java to determine whether each integer is perfect, amicable, sociable, or not cuddly (under the MAX assumption). If the integer is amicable or sociable, Cycle.java will print the number's aliquot cycle.

First, write a method that prints an aliquot cycle starting at n:

```
// prints aliquot cycle starting at n
// method assumes that argument n is cuddly
public static void printCycle(int n)
```

For example, Cycle.printCycle(284) should print: 284 220. Note that you must call printCycle() *only* on numbers known to be cuddly!

Your main() method in Cycle.java will read ints from StdIn and for each one, print to StdOut information about its cuddliness, using *exactly* the output format given below.

If the file `inputs.txt` contains these integers:

```
5
6
10
95
284
14288
276
```

then your main method should behave as follows.

```
% java-introcs Cycle < inputs.txt
5 is not cuddly
6 is perfect
10 is not cuddly
95 is not cuddly
284 is amicable: 284 220
14288 is sociable: 14288 15472 14536 14264 12496
276 is not cuddly
```

Once again, do not copy methods from your Perfect.java or Cuddly.java into this program. Instead, just use them by calling Perfect.aliquot(), Cuddly.isCuddly(), and so on.

AFTERWORD

Many numbers have aliquot sequences that end at 0. Some have sequences that eventually reach a perfect number; for example, the aliquot sequence starting at 95 is 95, 25, 6, 6, 6, . . . Other starting numbers have sequences that eventually reach an amicable pair or a sociable group.

A 150-year-old conjecture is that *all* positive integers have aliquot sequences that end in one of these ways. This has not yet been proved, however: there may exist numbers whose sequences contain no cycles and never reach 0! The smallest number whose aliquot fate remains unknown at the time of this exam is 276. (Its aliquot sequence has been computed out to the 1793rd element, which contains 185 digits!)