Homework #1
RELEASE DATE: 03/09/2020
DUE DATE: 03/24/2020, **noon** on Gradescope/CEIBA

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

This homework set comes with 200 points and 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

1.1 More from the Class

(1) (10%) The five properties of algorithm that were discussed in our class was actually proposed by Professor Donald Knuth. Try to find who Professor Donald Knuth is, and write down a short paragraph ($\leq$ 15 sentences) to introduce him in a way that a usual high school student would worship him.

(2) (10%) The following getMinIndex has been introduced in the class. Prove that when there are multiple minimum elements within the array, the one with the smallest index would be returned.

```
getMinIndex(integer array arr, integer len)
minpos ← 0
for i ← 1 to len − 1 do
  if arr[i] < arr[minpos] then
    minpos ← i
  end if
end for
return minpos
```

By proving, we mean that you should list your "claim" related to the goal of this problem first, and then illustrate a rigorous mathematical argument to justify the claim.

1.2 Sequential Search for Greatest Common Divisor

The greatest common divisor $gcd(a, b)$ between two positive integers a and b is defined as the largest positive integer that divides both a and b without a remainder. Mathematically speaking, for any integer $k > gcd(a, b)$, $(a \bmod k) \neq 0$ or $(b \bmod k) \neq 0$. In the following problems, we translate several properties of the *gcd* function to corresponding algorithms. In particular, we will pay some respect to Euclid's algorithm, which is arguably one of the earliest algorithms ever.

(1) (10%) The following GCD-By-Def algorithm simply translates from the definition of *gcd*. Note that GCD-By-Def is not an algorithm yet because the **for** loop does not end, which violates the

finiteness property of an algorithm. Provide an upper bound for i in the **for** loop that completes this algorithm. Prove that using this upper bound would work. Note that the tightness of your upper bound will be a grading criteria—the tighter, the better!

```
GCD-By-Def(positive integer a, positive integer b)
ans ← 1
for i ← 2 to ··· do
  if a mod i = 0 and b mod i = 0 then
    ans ← i
  end if
end for
return ans
```

(2) (10%) Dr. Speedup is not satisfied with the performance of GCD-By-Def, and thus designed another algorithm called GCD-By-Reverse-Search below. Prove that the algorithm is correct for positive integers a, b using the mathematical definition of *gcd*.

```
GCD-By-Reverse-Search(positive integer a, positive integer b)
for i ← min(a, b) to 1 do
  if a mod i = 0 and b mod i = 0 then
    return i
  end if
end for
```

(3) (10%) Assume that $a > b$. What is the minimum number of iterations that GCD-By-Reverse-Search needs (for a fixed a)? When does the situation happen? *(The situation is usually called the best case.)*

(4) (10%) Assume that $a > b$. What is the maximum number of iterations that GCD-By-Reverse-Search needs (for a fixed a)? When does the situation happen? *(The situation is usually called the worst case.)*

1.3 Binary Algorithm for GCD

Well, as a CSIE student, you shall not be satisfied with the efficiency of the algorithms mentioned above. Let's study another faster algorithm. The algorithm, called Binary Algorithm for GCD. "Binary", you say. Yes, binary is a magic word in computer science. Unlike the previous two algorithms, the Binary Algorithm needs only subtraction and division by 2 to work.

```
GCD-By-Binary(positive integer a, positive integer b)
n ← min(a, b), m ← max(a, b), ans ← 1
while n ≠ 0 and m ≠ 0 do
  if n is even and m is even then
    ans ← ans × 2
    n ← n/2
    m ← m/2
  else if n is even and m is odd then
    n ← n/2
  else if n is odd and m is even then
    m ← m/2
  end if
  if n > m then
    swap(n, m)
  end if
  m ← (m − n)
end while
```

return $n \times ans$

(1) (10%) Write down the values of n, m and ans in the end of each iteration when $a = 42$ and $b = 14$.

(2) (10%) Prove that GCD-By-Binary satisfies the *finiteness* property. That is, the **while** only runs for a finite number of iterations for any given pair of positive integers (a, b).

(3) (10%) Assume that k is a positive common divisor of a and b. Prove that

$$k \cdot gcd(a/k, b/k) = gcd(a, b).$$

You can see that the Binary Algorithm takes $k = 2$ somewhere.

(4) (10%) Assume that $a > b$. Please prove that $gcd(a, b) = gcd(a - b, b)$. You can see that the Binary Algorithm uses this somewhere.

(5) (Bonus 10%) Assume that $a > b$. What is the maximum number of iterations that GCD-By-Binary needs (for a fixed a)? Provide the tighest upper bound on the maximum number that you can think of, and prove your answer.

1.4 Euclid's Algorithm for Greatest Common Divisor

Now, let's take a look at Euclid's algorithm. Euclid's algorithm, as you probably learned in high school, is as follows.

GCD-By-Euclid(positive integer a, positive integer b)
$n \leftarrow \min(a, b)$, $m \leftarrow \max(a, b)$
while $m \bmod n \neq 0$ **do**
  $tmp \leftarrow n$
  $n \leftarrow m \bmod n$
  $m \leftarrow tmp$
end while
return n

(1) (10%) Write down the values of m, n and tmp in the end of each iteration when $a = 21$ and $b = 13$.

(2) (10%) Prove that GCD-By-Euclid satisfies the *finiteness* property. That is, the **while** only runs for a finite number of iterations for any given pair of positive integers (a, b).

(3) (10%) Assume that it takes T iterations of **while** to run GCD-By-Euclid(a, b) for some positive integers a and b. How many iterations does it take to run GCD-By-Euclid($2a$, $2b$)? Formally prove your result.

(4) (Bonus 10%) Prove that $gcd(a, b) = gcd(b, a \bmod b)$. You can see that Euclid's algorithm uses this property somewhere.

1.5 Comparison of GCD

In all the implementations below, your code should be able to work correctly for any positive integers a and b.

(1) (30%) Implement GCD-By-Def, GCD-By-Reverse-Search, GCD-By-Binary and GCD-By-Euclid in one single file `hw1_5.c` or `hw1_5.cpp`. The code should take two 4-byte integers a and b from the standard input per line until reaching $a = 0$. Then, the code should output the *gcd* of a and b computed by each algorithm and the **number of iterations** the algorithm takes.

Sample Input

```
3 2
4 3
0
```

Sample Output *(Note that the AAA, BBB, ⋯ should be filled by the actual numbers.)*

```
Case (3, 2): GCD-By-Def = 1, taking AAA iterations
Case (3, 2): GCD-By-Reverse-Search = 1, taking BBB iterations
Case (3, 2): GCD-By-Binary = 1, taking CCC iterations
Case (3, 2): GCD-By-Euclid = 1, taking EEE iterations
Case (4, 3): ...
```

(2) (10%) Consider $a = 32460$ and $b \in \{32000, 32001, 32002, \cdots, 32460\}$. Plot b versus the number of iterations (of **for** or **while**) that GCD-By-Def, GCD-By-Reverse-Search, GCD-By-Binary, GCD-By-Euclid takes to compute $gcd(a, b)$ as curves on the same figure. Briefly state your findings.

1.6 GCD of Big Integers

(1) (30%) Sometimes one `int` variable cannot represent the integer we want. Even if we use an `unsigned` 4-byte integer, the maximum possible value is only $2^{32} - 1$. So we may need a data *structure*, such as an integer *array* to represent larger values. We will call it BigInt. For instance, you can use an integer array where each element represents one (decimal) digit, like representing 1722 by

```
int digits[10]={2,2,7,1};
```

There are other choices, of course. Please implement a (non-negative) BigInt data structure of your choice and one of the GCD algorithms introduced above for two non-negative BigInt in one single file `hw1_6.c` or `hw1_6.cpp`

Input: Two positive integers A and B separated by one space. Number of decimal digits of A and B will be no more than 256.

Output: One integer that represents $gcd(A, B)$.

Sample Input

```
98765432198765432198765432 123456789123456789123456789
```

Sample Output

```
9000000009000000009
```

Time Limit: 1 second for each test case.

Hint from TAs: You are encouraged to use C++ and implement a BigInt class with operator overloading like -, <, *, and / to make it easier to reuse your programs in the previous part. Also, because implementing division or modulo operations may be difficult, the TAs recommend GCD-By-Binary, not GCD-BY-Euclid.

Submission File

Please submit your written part of the homework to Gradescope before the deadline. Also, you need to upload your coding part as a single ZIP compressed file to CEIBA before the deadline. The zip file should be like `b86506054.zip`, where the file name should be changed to your own school ID. The ZIP file should contain the following items:

- `hw1_5.c` or `hw1_5.cpp`
- `hw1_6.c` or `hw1_6.cpp`
- an optional `README`, anything you want the TAs to read before grading your code

The TAs will use the `Makefile` provided on the course website to test your code. **Please make sure that your code can be compiled and run with the Makefile on CSIE R217 linux machines. Otherwise your program "fails" its most basic test and can result in ZERO!**