

EECS 340: Assignment 1

Shaochen (Henry) ZHONG, `sxz517`

Zhitao (Robert) CHEN, `zxc325`

Due and submitted on 01/27/2020
EECS 340, Dr. Koyuturk

1 Problem 1

1.1 (a)

With $\text{gcd}(a, b)$ giving the greatest common divisor of (a, b) , the loop invariant is:

$$\begin{aligned} \text{gcd}(a, b) &= \text{gcd}(x, y) \\ \text{for } a, b, x, y &\in \mathbb{Z}^+ \end{aligned} \tag{1}$$

1.2 (b)

1.2.1 Initialization

The loop invariant is held true during initialization of the `while` loop as we defined $x = a$ and $y = b$ in the algorithm, thus *Equation 1* is perserved.

1.2.2 Maintenance

Assume we had x, y in the previous iteration, for this iteration we shall have x', y' . Without loss of generality, assume $x > y$, we must have the following according to the algorithm:

$$\begin{aligned} x' &= x - y \\ y' &= x \end{aligned}$$

Let d for $d \in \mathbb{Z}^+$ being the greatest common divisor of x and y , a.k.a $d = \text{gcd}(x, y)$. As d being a divisor of both x and y , we may therefore have $x = kd$ and $y = jd$ for $k, j \in \mathbb{Z}^+$. Then we may infer:

$$x - y = dk - dj = d(k - j) \tag{2}$$

From *Equation 2* and the known fact that $y = jd$, we may say that d is also a common divisor of $x - y$ and y . By the defination of gcd , this means the upper bound of d cannot be

greater than $\gcd(x - y, y)$. Thus we may conclude:

$$\begin{aligned} \gcd(x, y) = d &\leq \gcd(x - y, y) \\ \Rightarrow \gcd(x, y) &\leq \gcd(x - y, y) \end{aligned} \quad (3)$$

Now similarly, let e for $e \in \mathbb{Z}^+$ being the greatest common divisor of $x - y$ and y . We may therefore have $x - y = le$ and $y = me$ for $l, m \in \mathbb{Z}^+$. Then we may infer:

$$x = (x - y) + y = le + me = e(l + m) \quad (4)$$

From *Equation 4* and the known fact that $y = me$, we may say that e is also a common divisor of x and y . By the definition of \gcd , this means the upper bound of e cannot be greater than $\gcd(x, y)$. Thus we may conclude:

$$\begin{aligned} \gcd(x - y, y) = e &\leq \gcd(x, y) \\ \Rightarrow \gcd(x - y, y) &\leq \gcd(x, y) \end{aligned} \quad (5)$$

By observing *Equation 3* and *Equation 5*, we may conclude $\gcd(x, y) = \gcd(x - y, y)$. As we have discovered $x' = x - y$ and $y' = y$ earlier in this section, this means $\gcd(x, y) = \gcd(x - y, y) = \gcd(x', y')$. Thus, the loop invariant is held true during the maintenance of the **while** loop.

1.3 (c)

The **while** loop always terminates as the condition $x = y$ will eventually be reached. Due to the fact that we have x and y for $x, y \in \mathbb{Z}^+$; without loss of generality, we assume $x > y$, therefore we must have $x' = x - y$ and for $x' \in \mathbb{Z}^+$.

As we have only a finite amount of $x', x'', x''' \dots$ to decrease in the bound of $\mathbb{Z}^+ \in [0, x]$, the decremental calculation of $x_{k+1} = x_k - y_k$ ¹ will eventually reach a condition where $x = y$ due to the “well ordering” nature of the natural numbers. Thus, the **while** loop always terminates.

1.4 (d)

In **Section 1.2**, we have proven that $\gcd(x, y) = \gcd(x - y, y)$ assuming $x > y$ for $x, y \in \mathbb{Z}^+$. Following the principle of induction, we can generalize it as:

$$\begin{aligned} \gcd(x_k, y_k) &= \gcd(x_{k+1}, y_{k+1}) \\ \text{for } x', y' &\in \mathbb{Z}^+, \text{ while} \\ x_{k+1} &= x_k - y_k, y_{k+1} = y_k & \text{if } x_k > y_k \\ y_{k+1} &= y_k - x_k, x_{k+1} = x_k & \text{if } y_k > x_k \end{aligned} \quad (6)$$

¹ for k being the numbers of iteration went through in the **while** loop.

As we have proven in **Section 1.2**, that the **while** loop within the algorithm must terminate. Combined such finding with *Equation 6*, we must also have a:

$$\begin{aligned} \gcd(a, b) &= \gcd(a_k, b_k) = \gcd(a_{k+1}, b_{k+1}) = \dots \\ &\dots = \gcd(a_{k+n}, b_{k+n}) = \gcd(a_{k+n}, a_{k+n}) \end{aligned} \tag{7}$$

As we known by calculation that $\gcd(a_{k+n}, a_{k+n}) = a_{k+n}$, thus, a_{k+n} will be the greatest common divisor of (a, b) . As the induction performed in *Equation 7* according to constraints defined in *Equation 6* is an exact mathematical mimic of the iterations of the given Euclidean algorithm, we may conclude that **Euclidean(a, b)** returns $\gcd(a, b)$.

2 Problem 2

As suggested by Dr. Koyuturk, this is a kindly reminder to the grader that this problem is answered with the assumption of the first index of an array being [0]. If you'd prefer [1], please let us know through Canvas comment section and we will adjust accordingly in our future submission.

2.1 (a)

Algorithm 1 TwoSum(A, B, n, x) with two pointers

```

1: procedure
2:    $j \leftarrow n - 1$ 
3:    $i \leftarrow 0$ 
4:   while  $i < n$  and  $j \geq 0$  do
5:     if  $A[i] + B[j] < x$  then
6:        $i \leftarrow i + 1$ 
7:     else if  $A[i] + B[j] > x$  then
8:        $j \leftarrow j - 1$ 
9:     else
10:      return  $i, j$ 
11:  return False

```

2.2 (b)

The following is graphical demonstration of TwoSum(A, B, 5, 12):

A					B				
0(i)	1	2	3	4	0	1	2	3	4(j)
1	3	5	7	9	2	4	6	7	10
0	1(i)	2	3	4	0	1	2	3	4(j)
1	3	5	7	9	2	4	6	7	10
0	1(i)	2	3	4	0	1	2	3(j)	4
1	3	5	7	9	2	4	6	7	10
0	1	2(i)	3	4	0	1	2	3(j)	4
1	3	5	7	9	2	4	6	7	10

Thus we have $A[i] + B[j] = 12$ for (i, j) being $(2, 3)$.

2.3 (c)

2.3.1 Loop invariant

To enter the **while** loop with indexes i, j , it is impossible to have any combination of (a, b) where $a \in \{A[0], A[1], \dots, A[i-1]\}$ and $b \in \{B[j+1], B[j+2], \dots, B[n-1]\}$ to be $a + b = x$.

2.3.2 Initialization

The loop invariant is held true during the initialization, since for $i = 0$ and $j = n - 1$, we must have $a = \emptyset$ and $b = \emptyset$. Thus, it is impossible to have a combination of (a, b) to be $a + b = x$.

2.3.3 Maintenance

Due to the sorted nature of array A, B , we must have:

$$\underbrace{A[0] \leq A[1] \leq \dots \leq A[i-1]}_a \leq A[i] \Rightarrow a \leq A[i] \quad (8)$$

$$\underbrace{B[n-1] \geq \dots \geq B[j+2] \geq B[j+1]}_b \geq B[j] \Rightarrow b \geq B[j] \quad (9)$$

Assume we get $A[i] + B[j] = k$ within an iteration of the **while** loop, we either have $k < x$ or $k > x$. In both cases, we may confidently say that there will be no $a + b = k$; as the index of a must be smaller than i and the index of b must be greater than j . Thus, if there is any $a + b = x$, such loop would have been terminated immediately and we should not be able to reach to indexes i, j in the **while** loop. Therefore, as we are now at the i, j iteration of the **while** loop, this suggests there is no (a, b) combination where $a + b = x$. By this, the loop invariant is held true for all cases within the **while** loop.

Then, to proceed the loop, we may infer the following from *Equation 8* and *9*:

$$\text{if } k < x : a + B[j] \leq A[i] + B[j] = k \Rightarrow a + B[j] < x \quad (10)$$

$$\text{if } k > x : A[i] + b \geq A[i] + B[j] = k \Rightarrow A[i] + b > x \quad (11)$$

As there is no (a, b) combination available to form $a + b = x$, we must increase the value of index i in the case of $k < x$ to achieve a hopefully greater k' in the next iteration; and similarly, to decrease the value of index j to achieve a hopefully lesser k' in the case of $k > x$. Thus, the conditional statements in the pseudocode in **Section 2.1** are justified, and the algorithm is proven to be correct.

2.4 (d)

The worst case of this algorithm is when it returns **False**. In such case, we will have i and j traveling the entire bound of $\mathbb{Z}^+ \in [0, n)$. The loop will execute $2n$ times; thus the run time is $\Theta(n)$.

3 Problem 3

3.1 Describe the process using a loop: Pseudocode

3.2 Define loop invariant and termination

At the start of the k^{th} iteration of the **while** loop, we must have

$$l + p = m + n - (k - 1) \quad (12)$$

$$p \bmod 2 = n \bmod 2 \quad (13)$$

When the **while** loop terminates at $l + p = 1$, there can only be two cases:

$$l = 1, p = 0 \quad (n \bmod 2 = 0) \quad (14)$$

$$l = 0, p = 1 \quad (n \bmod 2 = 1) \quad (15)$$

Thus, we may infer that if n is odd, a *Pisidian* shall remain. Otherwise, if n is even, a *Lydian* shall remain.

Algorithm 2 Kepler442bArena(m, n)

```
1: procedure
2:    $l \leftarrow m$ 
3:    $p \leftarrow n$ 
4:   while  $(l + p) > 1$  do
5:     Let two individuals fight each other.
6:     if Both individuals are Lydians then
7:        $l \leftarrow l - 1$ 
8:     else if Both individuals are Pisidians then
9:        $p \leftarrow p - 2$ 
10:       $l \leftarrow l + 1$ 
11:    else
12:       $l \leftarrow l - 1$ 
13:  return  $l, p$ 
```

3.3 Proof of loop invariant

3.3.1 Initialization

At the beginning it is known that $k = 1$ and $b = n$, thus we may have:

$$l + p = m + n - (k - 1) = l + p = m + n - (1 - 1) \Rightarrow l + p = m + n \quad (16)$$

$$p \bmod 2 = n \bmod 2 \quad (17)$$

As the above two *Equations* satisfy the assumption of the loop invariant, we may say that the scenario of $k = 1$ is held true.

3.3.2 Maintenance

Case 1 As it is known that $l + p = (l - 1) + p$ and $k = k + 1$, we may conclude that $l + p + k$ must remain the same.

Case 2 As it is known that $l + p = (l + 1) + (p - 2)$ and $k = k + 1$, we may conclude that $l + p + k$ must remain the same.

Case 3 As it is known that $l + p = (l - 1) + p$ and $k = k + 1$, we may conclude that $l + p + k$ must remain the same.

As all three cases satisfy the loop invariant of the algorithm, the induction is therefore proven to be valid.