# COMP 2119A: Solution for Assignment 1

1.

a) Answer:false.

Assume that $100n^2 + 20n = \Theta(1000n)$, then $\exists c_1, c_2, n_0 > 0$, such that $\forall n \geq n_0$,$c_1(1000n) \leq 100n^2 + 20n \leq c_2(1000n)$.

Then, we have $c_1 \leq n + \frac{1}{50} \leq c_2$.

Take $n = max c_2, n_0$, then $n + \frac{1}{50} > c_2$, so there is a contradiction.

Therefore, $100n^2 + 20n \neq \Theta(1000n)$.

b) Answer: true.

Let $n_0 = 1, c = 99$, then $\forall n \geq n_0, 0 \leq 99n^2 \leq 99n^2$.

Therefore, $99n^2 = O(n^2)$.

c) Answer: true.

Let $n_0 = 1, c = 0.5$, then $\forall n \geq n_0$, we have $n^{0,001} \geq 1$.

So,$0 \leq 0.5n^{2.999} \leq 0.5n^{2.999}n^{0.001} = 0.5n^3$.

Therefore, $0.5n^3 = \Omega(n^{2.999})$.

d) Answer: false.

Take any function $f(n) = O(\log n), g(n) = O(n)$, then $\exists c_1, c_2, n_1, n_2 > 0$, such that

$\forall n \geq n_1, 0 \leq f(n) \leq c_1(\log n)$

$\forall n \geq n_2, 0 \leq g(n) \leq c_2(n)$.

Let $n_3 = max n_1, n_2$, then we have $\forall n \geq n_3, 0 \leq f(n) + g(n) \leq c_1(\log n) + c_2(n)$.

Assume that $O(\log n) + O(n) = \Theta(n)$, then $\exists c_3, c_4 > 0$, such that $\forall n \geq n_3, c_3(n) \leq f(n) + g(n) \leq c_1(\log n) + c_2 n \leq c_4(n)$

$c_3 \leq c_1(\frac{\log n}{n}) + c_2 \leq c_4$

When $\frac{\log n}{n} < \frac{c_3 - c_2}{c_1}$, we have $\frac{c_1 \log n}{n} + c_2 < c_3$. There is a contradiction.

Therefore, $O(\log n) + O(n) \neq \Theta(n)$.

e) Answer: false.

Take any function $f = O(0.0001n)$, then $\exists c_0, n_0 > 0$, such that $\forall n \geq n_0, 0 \leq f(n) \leq c_0(0.0001n)$.

Assume that $O(0.0001n) = O(1)$, then $\exists c_1 > 0$, such that $\forall n \geq n_0, 0 \leq f(n) \leq c_0(0.0001n) \leq c_1$.

It is not true when $n > \frac{c_1}{0.0001c_0}$. There is a contradiction.

Therefore, $O(0.0001n) \neq O(1)$.

f) Answer: true.

Assume $f_i(n) = O(1), i \in (1, 9,000,000)$, then $\exists c_1, ..., c_{9000000}, n_0 > 0$, such that $\forall n \geq n_0, f_i(n) \leq c_i$.

Let $c = c_1 + ... + c_{9000000}$, then we have $\forall n \geq n_0, O(1) + ... + O(1)[9000000 terms] \leq c_1 + c_2 + ... + c_{9000000} = c$.

Therefore, $O(1) + O(1) + ... + O(1)[9000000 terms] = O(1)$.

g) Answer: true.

$f(n)$ and $g(n)$ are positive functions, then $\exists n_0 > 0$, such that $\forall n \geq n_0, f(n) < f(n) + g(n), g(n) < f(n) + g(n)$.

So we have $min f(n) + g(n) < f(n) + g(n)$.

Let $c = 1$, $\forall n \geq n_0$, we have $0 < min f(n) + g(n) < f(n) + g(n) = c(f(n) + g(n))$.

Therefore, $min f(n), g(n) = O(f(n) + g(n))$.

h)False

Let $f(n) = n, g(n) = n^2$

$min\{f(n), g(n)\} = n$ when $n > 1$

if we can find $c_1$ and $c_2$: $c_1(n + n^2) \leq min\{f(n), g(n)\} \leq c_2(n + n^2)$

$c_1(n+n^2) \le n \le c_2(n+n^2)$, it can not be true when $c_1, c_2, n_0 > 0$.
i)True
Assume $f(n) = \Theta(n^2)$, then there exist $c_1, c_2, n_0$ that
$c_1 n^2 \le f(n) \le c_2 n^2$, whenever $n \ge n_0$
$\Rightarrow c_1 \le f(n)/n^2 \le c_2$
$\Rightarrow c_1 \le 6000/n + f(n)/n^2 \le 6000 + c_2$
Let $c_1' = c_1$
$c_1' \le 6000/n + f(n)/n^2$
$\Rightarrow c_1' n^2 \le 6000n + f(n)$
Thus $6000n + \Theta(n^2) = \Omega(n^2)$
j)False
$f(n) = O(n) \Rightarrow f(n) \le c_1 n$
Let $f(n) = 2n$
$2^{2n} \le c_2 2^n \Rightarrow 2^n \le c_2$
cannot find $c_2, n_0$ for all $n \ge n_0$, $2^n \le c_2$ holds.
k)True
Take $n_0 = \lceil 105/c \rceil + 1$
Then for all $n \ge n_0$, $105n < 105 \times 105/c = c \times 105/c \times 105/c < cn^2$
l)False
$log_{200} n < c \times \log_2 n \Rightarrow \log_{200} n / \log_2 n < c$
$\Rightarrow \log 2 / \log 200 < c$
m)False
Take $f(n) = n^2, g(n) = n$
if $\log(f(n)) > c \log(g(n))$
$\log(n^2) > c \log n$
$\Rightarrow c < 2$

2.
(i) $9999^{999}$
(ii) $\log n, \log_{30} n$
(iii) $n^{\varepsilon} (0 < \varepsilon < 0.5)$
(iv) $\sqrt{2}^{\log n}$
(v) $n/(\log \log n^2)$
(vi) $200n, 2^{\log n}$
(vii) $(n + \sqrt{n}) \log n$
(viii) $1.001^n$
(ix) $2^n$
(x) $2^{3n}$

**3.**

We first divide the coins into 3 groups of size 4 for each. In the first weight, we pick two groups randomly. If the two sides are balanced, then the counterfeit coin is in the third group, which can be solved easily in two weights. If not, then it is on either one of the two groups.

We also notice that it can not be solved in one round when there are more than 3 coins.Therefore in the second weight, we should use the previous coins that are safe in the first weight. In particular, we pick 2 coins and 1 coin from each sides, adding coins from the 3rd group to make both sides 3 coins in total, through which we can decide a new group of size 2 or 3, where the counterfeit resides in. By the third weight and results of the first 2 weights, we can solve this problem.Following is the detailed solution.

Without loss of generality, we mark the coins with number 1-12. We define the function $Weight(t_1, t_2)$, whose output is in $L$, $R$ or $E$, meaning that the heavier side is on the left(L) or right(R), or they are balanced(E). And we use $1L$( $1R$) to present the result that: the counterfeit coin is 1, and it is lighter(heavier).

```
  main():
ret1 = Weight((1,2,3,4), (5,6,7,8)) //first weight
if(ret1 == E) //the counterfeit is in (9, 10, 11, 12)
  ret21 = Weight((1,2,3), (9, 10, 11))
  if(ret21 = E)
      ret31 = Weight(1, 12)
      if(ret31 = "L")
          return 12L //12 is lighter
      else
          return 12H //12 is heavier
  else
      ret32 = Weight(9, 10)
      if(ret32 = E and ret21 = L)
          return 11L //11 is lighter
      if(ret32 = E and ret21 = R)
          return 11H //11 is heavier
      if(ret32 = L and ret21 = L)
          return 10L //11 is lighter
      if(ret32 = L and ret21 = R)
          return 9H //11 is heavier
      if(ret32 = R and ret21 = L)
          return 9L //9 is lighter
      if(ret32 = R and ret21 = R)
          return 10H //10 is heavier
if(ret1 == L or R) //the first weight is not balanced
```

```
  ret22 = Weight((1,2, 5),(3, 6, 9))
  if(ret22 == E)
      ret33 = Weight(7, 8)
      if(ret33 == E and ret1 = L)
          return 4H //4 is heavier
      if(ret33 == E and ret1 = R)
          return 4L //4 is lighter
```

```
if(ret_22 == R and ret_1 == R)
    ret_34 = Weight(1,2)
    if(ret_34 == L)
        return 1H //1 is heavier
    if(ret_34 == R)
        return 2H // 2 is heavier
    if(ret_34 == E)
        return 6L// 6 is lighter

if(ret_22 == R and ret_1 == L)
    ret_35 = Weight(1,5)
    if(ret_35 == E)
        return 3H //3 is heavier
    else
        return 5L // 5 is lighter

if(ret_22 == L and ret_1 == L)
    ret_36 = Weight(1, 2)
    if(ret_36 == L)
        return 1H //1 is heavier
    if(ret_36 == R)
        return 2H // 2 is heavier
    if(ret_36 == E)
        return 6L// 6 is lighter

if(ret_22 == L and ret_1 == R)
    ret_37 = Weight(1,5)
    if(ret_37 == E)
        return 3H //3 is heavier
    else
        return 5L // 5 is lighter
```

4.

a)(i) 0101 (ii) 243

b)Change number A to the form of base B (in reversing order). eg. A=10 under binary system (B=2) is 1010, so the algorithm outputs 0101.

c)The worst case happens when B=2. The asymptotically (worst case) tight bound for the running time is $O(\log n)$, where $n$ is the size of A.

For $A = n$, and $2^{k-1} \leq n \leq 2^k$, it is easily to see that the loop will perform $k$ times, and $\log n \leq k \leq \log n + 1$, so we can show that $k = O(\log n)$.

5.

a) Initialize step runs in $c_1 m$ time.

We mainly consider running time in the "for" loop.In the inner "for" loop, it takes constant time.

Therefore, running time in the whole "for" loop is: $\sum_{i=1}^{m} c_i \lfloor m/i \rfloor$

Therefore $T(m) = c_1 m + \sum_{i=1}^{m} c_i \lfloor m/i \rfloor$

Let $c = max\{c_i\}$, we have:

$T(m) = c_1 m + cm + \lfloor m/2 \rfloor + \cdots + 1$

$< c_1 m + cm(1 + 1/2 + 1/2 + 1/4 + 1/4 + 1/4 + 1/4 + \cdots + 1/2^k)$

$< c_1' m \log m$

On the other hand,

$T(m) = c_1 m + m + \lfloor m/2 \rfloor + \cdots + 1$

$> c_1 m + cm(1/2 + 1/4 + 1/8 + \cdots + 1/2^{k+1})$

$> c_2' m \log m$

Therefore, we have $T(m) = \Theta(m \log m)$

Remark:You can also consider:

$T(m) = c_1 m + \sum_{i=1}^{m} c_i \lfloor m/i \rfloor$

Let $c = max\{c_i\}$, we have:

$T(m) \leq c_1 m + c(m + \lfloor m/2 \rfloor + \cdots + 1)$

Since $f(x) = \int_1^m 1/x dx = ln(m)$, and $O(ln(m) = \log\, m)$

We can also have

$T(m) = \Theta(m \log m)$

b) Through observation, we finds that the perfect square numbers will end with 1 since they are only flipped with odd times, while others will be 0. Therefore the idea is to first initialize the array to 0 which takes $O(m)$ time, then traverse the entire array, and set the perfect square number to 1, which also runs in $O(m)$. Following is the detailed algorithm:

Initialize $A[1 \ldots m]$ so that each entry is 0 for$(i = 1, \ldots, m)$

$int s = sqrt(i)$//get floor of square value of $i$

$if(s^2 = i)$

  $return$ 1

$else$

  $return$ 0

6.

$c$ should be constant, but not changed with $n$ In the induction step, The range of $c$ changes with $n$. Therefore we can not find a fixed $c$, and $n_0$, such that for any $n \geq n_0$, $f(n) \leq cn$. Thus the proof is not correct and $f(n) \neq \Omega(n^2)$.

**7.**

a) Initialize $G[1...n]$ to that each entry is 0
for ( $i \in (0, n-1)$ )
  if ( $i == 0$):
    $G[i] = 0$
  else if($i == 1 || i == 2$):
    $G[i] = 1$
  else:
    $G[i] = G[i-1] + G[i-2] + G[i-3]$

Complexity is: $O(n)$
Initialize step runs in $O(n)$
Number of "for" loop is $n$, in each loop, it takes constant time $c_2$, that is $c_2 n = O(n)$
Therefore the program runs in $O(n) + O(n) = O(n)$ time.

b) Supposed that we have a matrix M

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{1}$$

We can find that:

$$\begin{bmatrix} G[n] & G[n-1] & G[n-2] \end{bmatrix} M = \begin{bmatrix} G[n]+G[n-1]+G[n-2] & G[n] & G[n-1] \end{bmatrix}$$

(2)

That is:

$$\begin{bmatrix} G[n] & G[n-1] & G[n-2] \end{bmatrix} M = \begin{bmatrix} G[n+1] & G[n] & G[n-1] \end{bmatrix}$$

(3)

Therefore we have:

$$\begin{bmatrix} G[n] & G[n-1] & G[n-2] \end{bmatrix} = M^{n-2} \cdot \begin{bmatrix} G[2] & G[1] & G[0] \end{bmatrix} \tag{4}$$

To compute $M^{n-2}$, we can apply recursive algorithm since it satisfy the associative law. Following is the detailed algorithm:

//Calculate $M^n$
pow($M$, $n$):
  $if(n \leq 2):$
    $return\ M$
  $elif(n\%2 == 1):$ // n is an odd number
    $return\ M * pow(M, n-1)$
  $else:$ // n is an even number
    $return\ pow^2(M, n/2)$
//main program
Initialize $M$ in equation 1
pow($M$,$n$)

return $M[0][0]$

Complexity: $O(\log n)$
We first analyze the complexity for pow function. Since the basic multiplication of $M * M$ is constant, we have:

$$f(n) = \begin{cases} c_1 & n = 1 \\ f(n/2) + c_2 & n \geq 2 \ \&\& \ n\%2 = 1 \\ f((n-1)/2) + c_2' & n \geq 2 \ \&\& \ n\%2 = 0 \end{cases}$$

Therefore pow function runs in $O(\log n)$
The initialize step in main program also runs in constant time $O(1)$.

Therefore the complexity of the program is $O(1) + O(\log n) = O(\log n)$