Assignment-1

Design and Analysis of algorithm

Trishan Mondal

§ Problem 1

Let's rewrite the algorithm for `extended_Euclid`. [1]

```
def extended_Euclid(a,b):
    """
      a,b are non-negative integers
      The function returns(u,v,d)such that d=gcd(a,b)
     and d=ua+vb
    """
if b==0: return(1,0,a)
   (u,v,d)=extended_Euclid(b,a%b)
return(v,u-v*(a//b),d)
```

(a) The problem, asked to prove is wrong. Just for counter-example take $199, 3$ we can see that $\gcd(199, 3) = 1$ but then $199u+3v = 1$ if $|v| \leq 3$ it is not possible to find such u, v. Rather we can prove, $|u_i| \leq \frac{b_i}{d}, |v_i| \leq \frac{a_i}{d}$. For the base case we will take $(t-1)$-th step. It is given that, $(u_t, v_t) = (1, 0)$, so $(u_{t-1}, v_{t-1}) = (0, 1)$ corresponding $a_{t-1} = dq_t$ and $b_{t-1} = d$, it is satisfying the hypothesis. Let, the hypothesis is true for v_i, u_i. We have $a_i = b_{i-1}$ and $a_{i-1} = qa_i + b_i$ and hence,

$$|u_{i-1}| = |v_i| \leq \frac{a_i}{d} = \frac{b_{i-1}}{d}$$

We have, $a_{i-1} = q_i a_i + b_i$, this will give us,

$$\begin{aligned} a_{i-1} &= q_i a_i + b_i \\ &\geq dq_i|v_i| + d|u_i| \\ &\geq d|u_i - q_i v_i| \\ &= d|v_{i-1}| \end{aligned}$$

Induction step is complete and hence we are done.

(b) To get the total bit operation needed for `extended_Euclid` we need to find out the worst case, which can be found from the following computations.

$$\begin{aligned} \begin{pmatrix} a_{i-1} \\ b_{i-1} \end{pmatrix} &= \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \\ \begin{pmatrix} a_0 \\ b_0 \end{pmatrix} &= \begin{pmatrix} q_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_2 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} q_t & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_t \\ b_t \end{pmatrix} \\ &\geq \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_t \\ b_t \end{pmatrix} \\ &= \begin{pmatrix} F_t & F_{t-1} \\ F_{t-1} & F_{t-2} \end{pmatrix} \begin{pmatrix} d \\ 0 \end{pmatrix} \end{aligned}$$

[1]** Every algorithm written in this assignment is not syntax of any specific language they are just an algorithm.

From the above calculation we get, $a_o \geq F_t d$, in other words $F_t \leq \frac{a_0}{d}$. The following computations will help us to approximate t.

$$
\begin{aligned}
& F_t \leq \frac{a_0}{d} \\
& \Rightarrow \left(\frac{1+\sqrt{5}}{2}\right)^t - \left(\frac{1-\sqrt{5}}{2}\right)^t \leq \frac{a_0}{d} \\
& \Rightarrow \left(\frac{1+\sqrt{5}}{2}\right)^t \leq \frac{a_0}{d} + 1 \\
& \Rightarrow t \leq \log_{\left(\frac{1+\sqrt{5}}{2}\right)} \left(\frac{a_0}{d} + 1\right)
\end{aligned}
$$

Since, a_0 and d are n-bit so, $t \sim O(n)$ will give us the worst case. We will calculate the number of bit operation for this case. Recall that, $q_i = \lfloor \frac{a_{i-1}}{b_{i-1}} \rfloor$, $a_i = b_{i-1}$ and $b_i = a_{i-1} - q_i b_{i-1}$. The a_{i-1}, b_{i-1} are $O(n)$-bit number, to determine a_i and b_i we need $\sim O(n^2)$ bit operations ($O(n^2)$ for multiplication and $O(n)$ for addition). For the worst case we took $a_0 = F_n d$ and $b_0 = F_{n-1} d$, so the `extended_Euclid` will be called $\sim n$ times. So the total bit operation needed for this case is $O(n^3)$. ■

§ Problem 2

Let's re-write the given algorithm.

```
def modified_Euclid(a,b):
""" a,b are non-negative integers.
 The function returns d such that d=gcd(a,b)
"""
if b==0: return a
  r=a%b
if r<b/2:
   return modified_Euclid(b,r)
else:
   return modified_Euclid(b,b-r)
```

(a) Let, $a > b > 0$ be the given integers. Let, $a = bq + r$ where, $0 \leq r \leq b - 1$, then we have to show $\gcd(a, b) = \gcd(b, r)$. Let, $d = \gcd(a, b)$, since d divides both a and b it must divide r, in other words $d \mid \gcd(b, r)$. Also, $\gcd(b, r) \mid r, b$ hence $\gcd(b, r) \mid a$, which gives $\gcd(b, r) \mid \gcd(a, b) = d$ and hence $d = \gcd(b, r)$.

We remain to show that, $d = \gcd(b, b - r)$. We have shown, $\gcd(b, r) = d$, we will show that, $\gcd(b, r) = \gcd(b, b-r)$. By the similar argument we have, $\gcd(b, r) \mid \gcd(b, b-r)$ and also $\gcd(b, b-r) \mid \gcd(b, r)$, which gives us $\gcd(b, b - r) = \gcd(b, r)$. The above calculation shows us if the algorithm terminates it will return $\gcd(a, b)$. For any input $a > b > 0$, the algorithm calls `modified_Euclid(b,r)` if $r < \frac{b}{2}$ and calls `modified_Euclid(b,b-r)` otherwise. Here, b is decreasing by a factor of half at each calling. So it will terminate to 0 at some time. Hence, the algorithm returns $\gcd(a, b)$.

(b) We will do the calculation for two separate cases. One for t is even and other is t is odd. We know, $F_{t+1} = F_t + F_{t-1}$ and $F_t = F_{t-1} + F_{t-2} \leq 2F_{t-1}$, so, modified_Euclid($F_{t+1}, F_t$) will call modified_Euclid(F_t, F_{t-2}) as, $F_t - F_{t-1} = F_{t-2}$. Since, $F_t = 2F_{t-2} + F_{t-3}$ and $F_t \leq 2F_{t-3}$ the algorithm will call modified_Euclid(F_{t-2}, F_{t-4}) and from here modified_Euclid(F_{t-2k}, F_{t-2k-2}) will be called at k-th step. At the end the code will reach to $(F_2, F_0) = (1, 0)$ if t is even. So, the total number of tile the function is called is, $1 + \left(\frac{t}{2} - 1\right)$, which is $\frac{t}{2}$.

If t is odd then the code will reach at the pair (F_3, F_1) and then it will call `modified_Euclid(1,0)`, so we need to call the function $\frac{t-1}{2} + 1 = \frac{t+1}{2}$ times. ■

§ Problem 3

The following `perf.pow(N)` will return 1 or 0 according to N is perfect power or not.

```
1     def perf.pow(N):
2      for E in range (2,bit(N)) #This loops runs from 2 to bit(N) which is ~ log N = n
3       a = 1; b = bit(N) // E; A = pow(2,b) #we are taking  A = 2^b
4       while A-a > 1
5         Q = (a+A)//2 # using the algorithm of bisection to closely approximate Q
6         x = modular_expo(Q,E,N+1)
7           if x == N return 1
8           if x< N return  a = Q
9           else return A = Q
10    return 0
11
```

Description of The algorithm. There is two loop we used in the algorithm. The for loop at line 2 is running E from 2 to $n = \lceil \log_2 N \rceil$ then we are setting $A = 2^{\lfloor \frac{n}{E} \rfloor}$, which is $\lfloor \frac{n}{E} \rfloor$-bit number. Now we are using bisection method to get integer closer to the solution of $x^E - N = 0$. This is the reason we are defining $Q = \lfloor \frac{a+A}{2} \rfloor$, it is not hard to see that, $Q^E \leq N$ hence, if $Q^E \equiv N \pmod{N+1}$ it will be actually N otherwise we are returning $A = Q$,$a = Q$ according to $x > N$ or $x < N$ and finally if for all the range we can't get any Q then we are returning 0.

Correctness of the algorithm. Let, A_i, a_i be the A and B at the i-th step, then observe,

$$|A_i - a_i| = \frac{1}{2}|A_{i-1} - a_{i-1}| = \cdots = \frac{1}{2^i}|A_0 - a_0| < \frac{N}{2^i}$$

so the algorithm terminates after finite step. If N is not a perfect power then there do not exist any Q, E such that $Q^E = N$. So line 7 will fail at each iteration step and after the loop is complete the algorithm will `return = 0`.

If $N = Q^E$ for some $Q, E \geq 2$, E will be less than or equal to $\lfloor \log_2 N \rfloor$. We know for the bisection method $Q \in [a_i, A_i]$ for every $i \geq 1$. If $Q_{i-1} = Q$ then line 7 will return 1 and otherwise $Q_{i-1}^E < N$ for every E in the range $2, \cdots, \lceil \log_2 N \rceil$. Then $Q \in (Q_{i-1}, A_{i-1}) = (a_i, A_i)$ so,

$$|Q - Q_i| \leq |A_i - a_i| = \frac{1}{2^i}|A_0 - a_0| < \frac{N}{2^i}$$

but for $i = \lceil \log_2 N \rceil$, $|Q - Q_i| < 1$ and hence $Q_i = Q$ at the $\lceil \log_2 N \rceil$-th step. For suitable E, the algorithm will return $Q^E = N$ and hence, the algorithm will return 1. Thus our algorithm is correct.

Time complexity. Since N is n bit number, $\lceil \log_2 N \rceil \sim n$. Inside the while loop, line 6 will take $\sim O(n^3)$ time to calculate modular exponent. (line 4) At each step $A - a$ is reducing by factor of $\frac{1}{2}$, the while loop will run $\lfloor \frac{n}{E} \rfloor$ time for each E and E will run from 2 to n. (line 3)The division $\lfloor \frac{n}{E} \rfloor$ and calculation of A will take at most $O(n^3)$ time. So the total time complexity is,

$$\begin{aligned} &\sum_{E=2}^{n} O(n^3) \lfloor \frac{n}{E} \rfloor + O(n^3) \\ &\sim O(n^4) \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}\right) \\ &\sim O(n^4 \log n) \end{aligned}$$

§ Problem 4

We will define a function `b_l(x,y,l)`, which will give us ℓ-th decimal number of the binary expansion of $\frac{x}{y}$ whenever, $x < y$.

```
def b_l(x,y,l) :
 """
 whenever x < y is this function will
 return l-th decimal term of the binary expansion  of x/y
 """
a = modular_expo(2,l-1,y) # this function calculates 2^(l-1) modulo y
r = a*x % y
return (2*r)//y #this is floor of 2r/y

```

The algorithm is taking input x, y, ℓ and give us b_ℓ (as required for the problem). From the following calculates we can confirm that our algorithm is correct.

$$\frac{x}{y} = \sum_{k\geq 1} b_k 2^{-k}$$

$$\frac{2^{\ell-1}x}{y} = \sum_{i=0}^{\ell-1} b_{\ell-1-i}2^i + \sum_{k\geq 1} b_{\ell-1+k}2^{-k} \cdots (1)$$

since, $2^{\ell-1}x \equiv r \pmod{y}$, we have, $2^{\ell-1}x = qy + r$ which gives, $\frac{x}{y} = q + \frac{r}{y}$ by comparing with (1) we get, $\sum_{k\geq 1} b_{\ell-1+k}2^{-k} - \frac{r}{y}$ is an integer since both the term is strictly less than 1, only possibility is

$$\frac{r}{y} = \sum_{k\geq 1} b_{\ell-1+k}2^{-k}$$

which means, $\frac{2r}{y} = b_\ell + \sum_{k\geq 1} b_{\ell+k}2^{-k}$ and hence $\lfloor \frac{2r}{y} \rfloor = b_\ell$. Now we will calculate time complexity of the algorithm.

Time complexity. The function `modular_expo(a,b,c)` has time complexity $O(n^3)$ where a, b, c are n-bit numbers. Modular multiplication $ax \pmod{b}$ has time complexity $O(n^2)$, where a, b, x are n-bit numbers. The operation `a//b` has time complexity $O(n^2)$, here a, b are n-bit number. So, the described algorithm has $O(n^3)$ complexity. ■