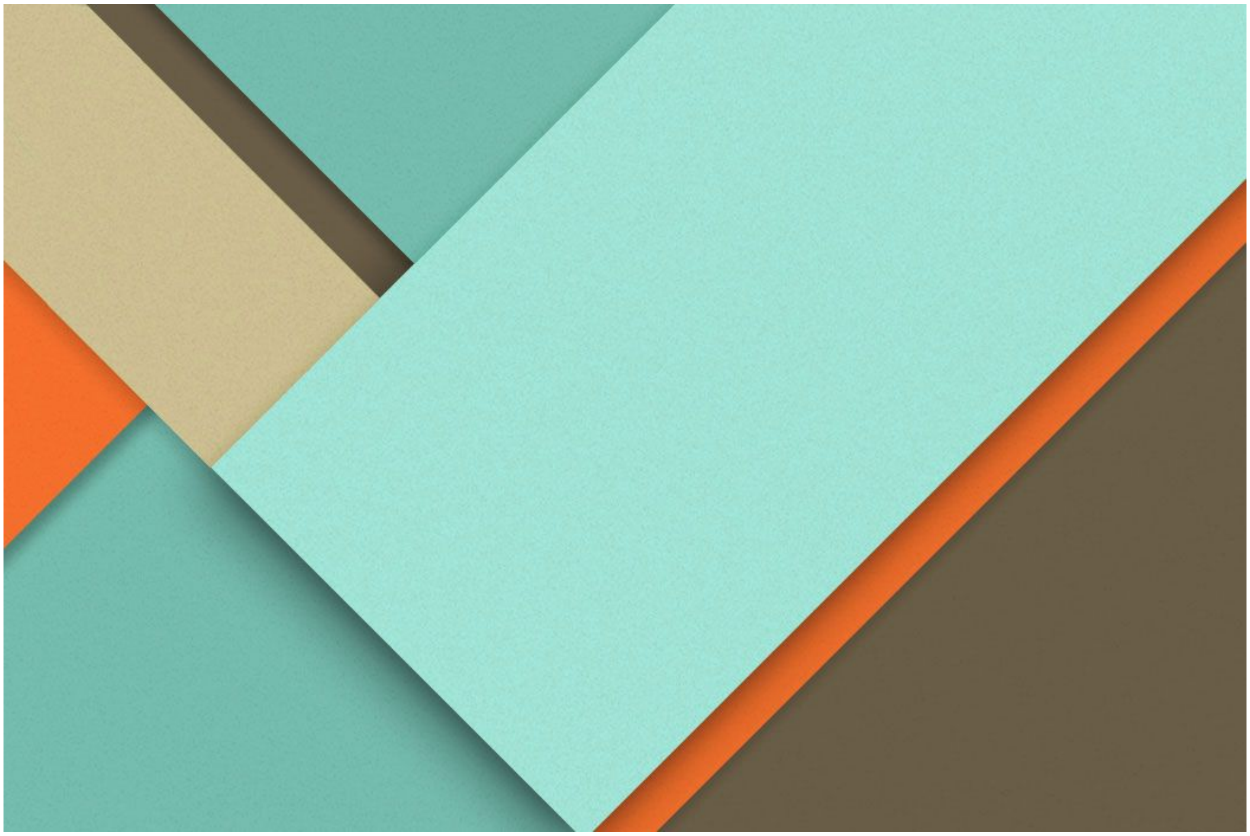


Number Theory

01.12.2019

Mahmoud kamal, Makram William
Faculty of Engineering, Alex Uni. CS dp.

Fast Exponentiation

1) problem statement :

• Implement the following procedures and compare the execution time of each with the increase of number of bits representing an integer. Also report on when the procedure breaks (overflow).

• Implement it in 4 versions. The following two na ̈ıve versions, in addition to, fast exponentiation in iterative and recursive versions.

2) Used data structure :

We only use int variables to get c

3) pseudo code :

Na ̈ıve 1

c =1

for i = 1 to b

c = c * a

c= c mod m

return c

Na ̈ıve 2

c= 1

for i = 1 to b

c = (c * a) mod m

return c

fast iterative

c=1

while b>0

if b is odd then

c = c *a % m

b = b/2

a = a*a % m

fast recursive

if a equal 0 return 0

if b equal 0 return 1

if b even then

c= fast (a, y/2,m)

c = c*c %m

if b odd

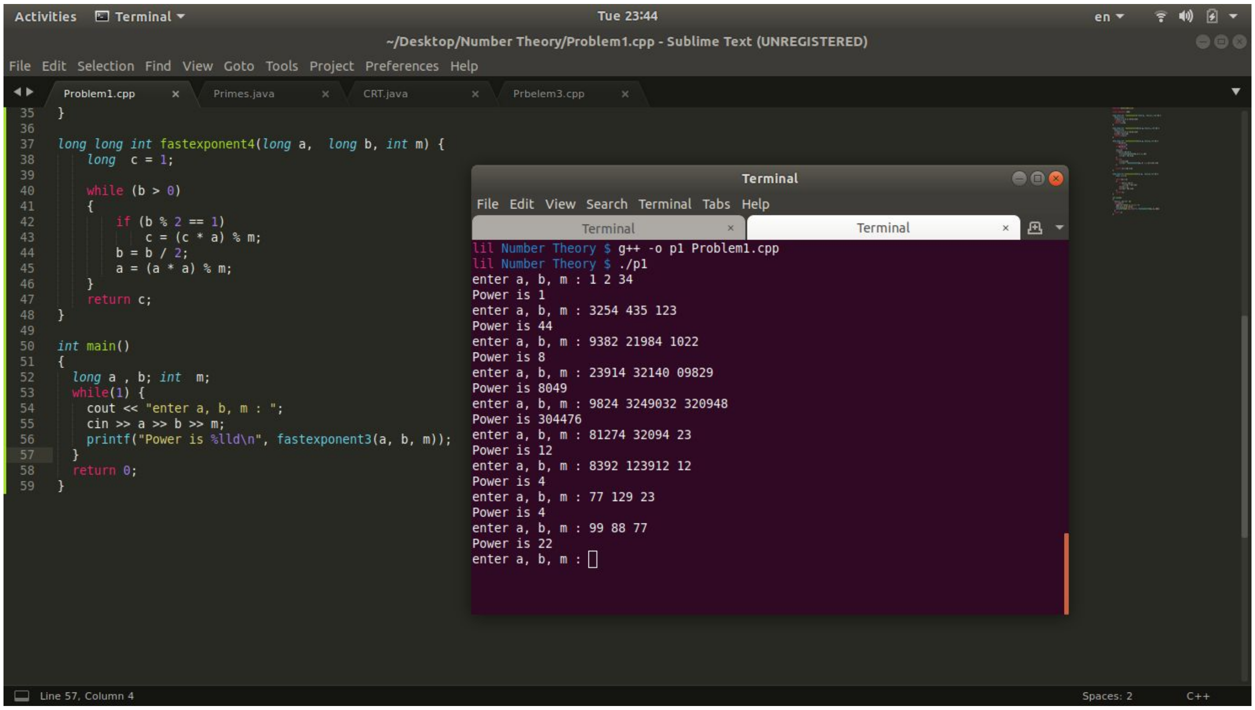
c= a %m

c = c* call fast(a,b-1,m) %m

return (c+m)%m

4) sample runs :

we can notice that time complicity for the first 2 naïve is O(b), but 3,4 are O(log(b))



Primes Generator :

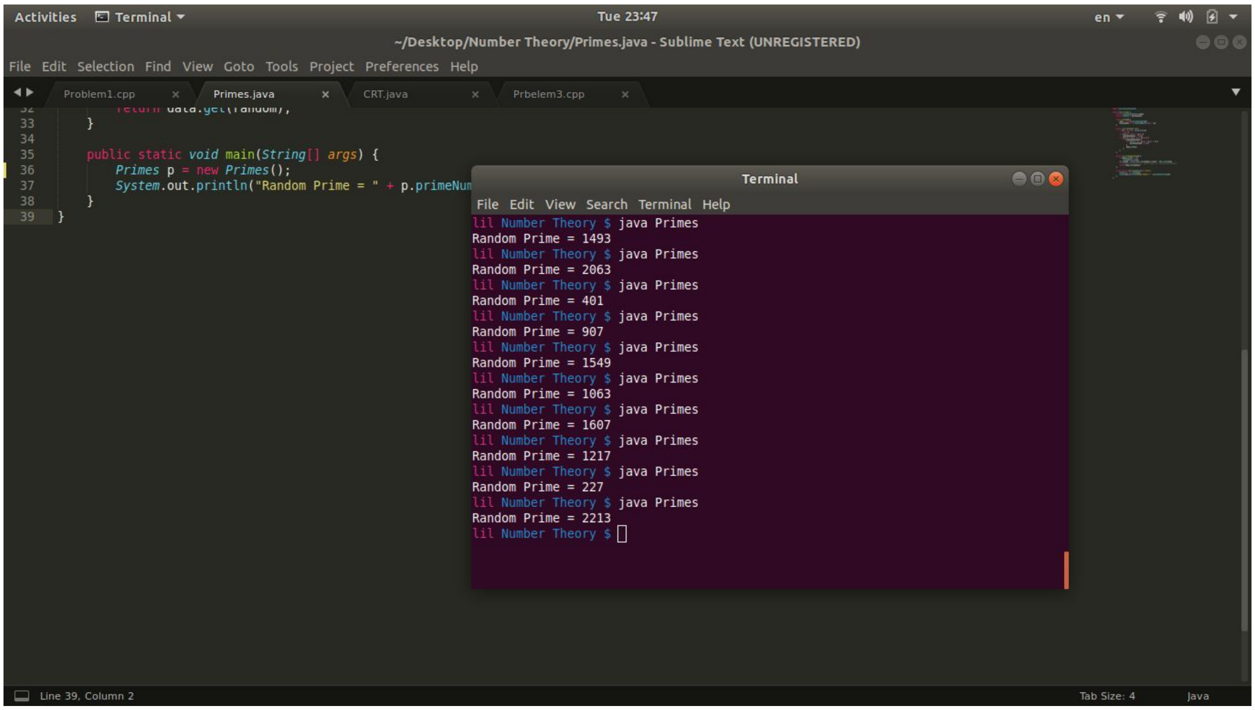
Problem Statement :

This is a prime number generation procedure and show its execution time in terms of the number of bits representing an integer.

Complexity:

O(n * log2 (log2 (n)))

Sample Runs :



Ecluid's Algorithm :

1) problem statement :

- **Input: a, b**
- **Output: d = gcd(a,b) and s, t such that d = s.a + t.b**

2) data structure:

We only use int variables to get d,s,t

3) pseudo code :

getEuclidean (a,b, *x,*y)

if a equals 0 then

***x=0, y=1, return b**

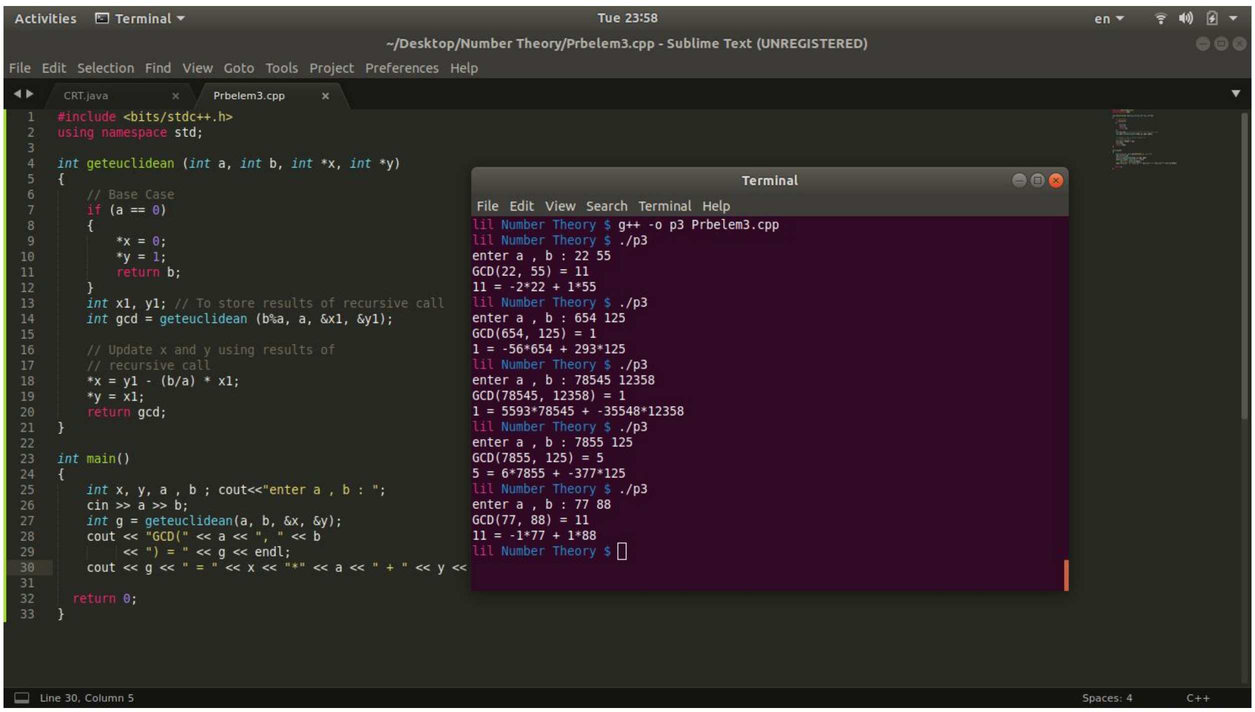
D= Call geteculidean(a%b,a,&x,&y)

```
*x = y1 - (b/a) * x1;

  *y = x1;

Return gcd
```

4) sample runs :



Chinese Remainder Theorem :

Complexity :

O(n * log2 (max (mi, Mk)))

Runs :