Number theory Assignment

By : Bassem Mohamed

ID : 13

CODE DISCRIPTION

There are 2 class MainWindow and MathMethods.

MainWindow contains GUI code.

MathMethods contains the main code of the 3 the problems.

Question one

First method (straight forward method):

This method depends on Java's java.math.BigInteger class has a modPow().

This method is fastest one.

Seconded method(simple loop):

This methods depends on equivalence of these 2 equations:

$$c \equiv (a \cdot b) \pmod{m}$$
$$c \equiv (a \cdot (b \pmod{m})) \pmod{m}$$

originally the base is multiplied b times to itself, this algorithm reduce the base after each multiplication by the mod. It's time is O (b)

The algorithim:

```
public static BigInteger modularExploop(BigInteger a,BigInteger
b,BigInteger n){
```

```
            BigInteger i = new BigInteger("0");
            BigInteger c=new BigInteger("1");
            for (; i.compareTo(b) < 0; i = i.add(BigInteger.ONE)) {
                  c=(c.multiply(a)).mod(n);
            }
            return c;
      }
```

Third method(Right to left binary method)

This method the more efficient than the last one. it's time is O (log b).

exponent can be written as:

$$e = \sum_{i=0}^{n-1} a_i 2^i$$

The value b^e can then be written as:

$$b^e = b^{\left(\sum_{i=0}^{n-1} a_i 2^i\right)} = \prod_{i=0}^{n-1} \left(b^{2^i}\right)^{a_i}$$

The solution c is therefore:

$$c \equiv \prod_{i=0}^{n-1} \left(b^{2^i}\right)^{a_i} \pmod{m}$$

The algorithm:

```
public static BigInteger modularExpBinary(BigInteger a,BigInteger
b,BigInteger n){
            BigInteger c=new BigInteger("1");
          while (b.compareTo(BigInteger.ZERO) ==1){
              if (b.mod(new BigInteger("2")).equals(new
BigInteger("1"))){
                  c=(c.multiply(a)).mod(n);
              }
              b=b.shiftRight(1);
              a=(a.multiply(a)).mod(n);
          }
          return c;
```

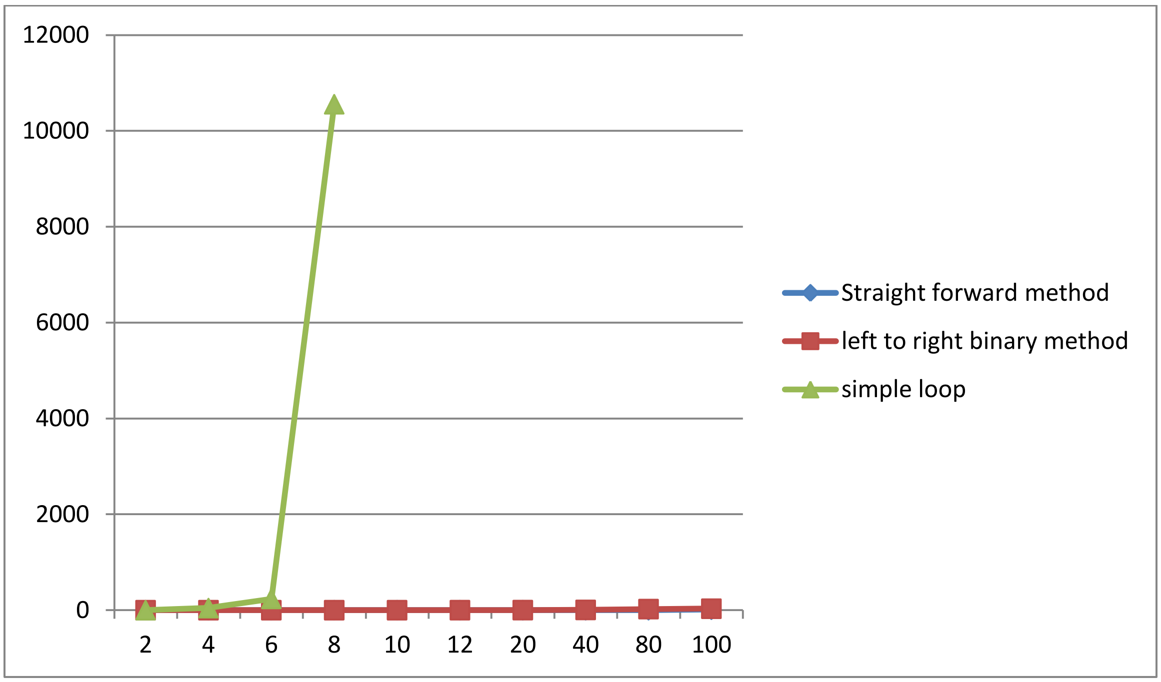
```
}
```

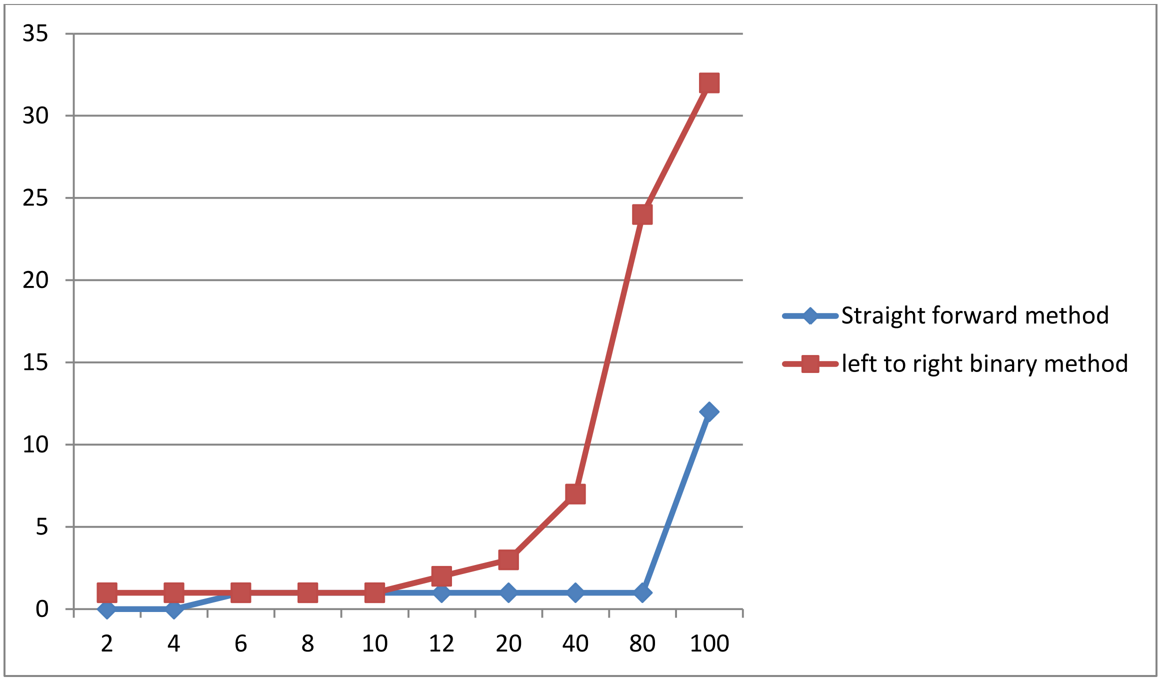
Fourth method (Euler method)

This method depends of the reduction of the exponent using Euler totient function.

The following charts for the execution time of the first three methods

y-axis is time in millisecond and x-axis is number of decimal digits.





Question two

In this question I used complete search to find the inverse.

Using this equation (ab) mod n = 1, I searched a value of b that satisfy the equation.

The code:

```
public static long getModularInverse(int a , int m){

      long i = 1;
      while ((i * m + 1) % a != 0){
            i++;
      }
      return ((i * m + 1) / a);
}
```

Question three

I used the algorithm given in the section

③ Chinese Remainder Theorem (CRT),

$x \equiv a_1 \pmod{m_1}$

$x \equiv a_2 \pmod{m_2}$

$x \equiv a_3 \pmod{m_3}$

$\vdots \qquad \vdots$

* Input: m_i's and a_i's
  Output: x which satisfies the set of equations
* Condition: every pair of m_i's is relatively prime.

No.
Date.

Steps:

$M = m_1 * m_2 * m_3 \cdots$

$M_1 = \frac{M}{m_1} \Rightarrow Y_1 = \text{inverse}(M_1) \bmod m_1$

$M_2 = \frac{M}{m_2} \Rightarrow Y_2 = \text{inverse}(M_2) \bmod m_2$

$M_3 = \frac{M}{m_3} \Rightarrow Y_3 = \text{inverse}(M_3) \bmod m_3$

$\vdots$

$\therefore x = (\Sigma a_i Y_i M_i) \bmod M$

The code:

```
public static  int CRT(int[] a , int [] m){

      int bigM=1;
      int[] mArr = new int[m.length];
      int[] yArr = new int[m.length];
      int x=0;
      for (int i = 0; i < m.length; i++) {
            bigM=m[i]*bigM;
      }

      for (int i = 0; i < mArr.length; i++) {
            mArr[i]=bigM/m[i];
            yArr[i]=(int) getModularInverse(mArr[i], m[i]);
            x=(x+(a[i]*yArr[i]*mArr[i]))%bigM;
      }

      return x;
}
```

sources: Wikipedia and tutorial notes.