



# 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 naive 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 :

Naive 1

$c = 1$

for  $i = 1$  to  $b$

$c = c * a$

$c = c \bmod m$

return  $c$

Naive 2

$c = 1$

for  $i = 1$  to  $b$

$c = (c * a) \bmod m$

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

$c = \text{fast}(a, y/2, m)$

$c = c * c \% m$

if  $b$  odd

$c = a \% m$

$c = c * \text{call fast}(a, b-1, m) \% m$

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  $O(\log(b))$

The screenshot shows a Sublime Text editor with a C++ file named `Problem1.cpp`. The code implements a fast exponentiation function `fastexponent4` and a `main` function that tests it with various inputs. A terminal window is open, showing the compilation and execution of the program. The terminal output displays the results of the fast exponentiation function for different inputs, showing that the power is calculated correctly and efficiently.

```

35 }
36
37 long long int fastexponent4(long a, long b, int m) {
38     long c = 1;
39
40     while (b > 0)
41     {
42         if (b % 2 == 1)
43             c = (c * a) % m;
44         b = b / 2;
45         a = (a * a) % m;
46     }
47     return c;
48 }
49
50 int main()
51 {
52     long a, b; int m;
53     while(1) {
54         cout << "enter a, b, m : ";
55         cin >> a >> b >> m;
56         printf("Power is %lld\n", fastexponent3(a, b, m));
57     }
58     return 0;
59 }

```

```

Terminal
File Edit View Search Terminal Tabs Help
lil Number Theory $ g++ -o p1 Problem1.cpp
lil Number Theory $ ./p1
enter a, b, m : 1 2 34
Power is 1
enter a, b, m : 3254 435 123
Power is 44
enter a, b, m : 9382 21984 1022
Power is 8
enter a, b, m : 23914 32140 09829
Power is 8049
enter a, b, m : 9824 3249032 320948
Power is 304476
enter a, b, m : 81274 32094 23
Power is 12
enter a, b, m : 8392 123912 12
Power is 4
enter a, b, m : 77 129 23
Power is 4
enter a, b, m : 99 88 77
Power is 22
enter a, b, m : 

```

## 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:

$$O(n * \log_2(\log_2(n)))$$

## Sample Runs :

The screenshot shows a Sublime Text editor window with the file `Primes.java` open. The code in the editor is as follows:

```

32  }
33  }
34  }
35  public static void main(String[] args) {
36      Primes p = new Primes();
37      System.out.println("Random Prime = " + p.primeNum
38  }
39  }

```

Overlaid on the editor is a terminal window titled "Terminal" showing the execution of the program. The terminal output is:

```

lil Number Theory $ java Primes
Random Prime = 1493
lil Number Theory $ java Primes
Random Prime = 2063
lil Number Theory $ java Primes
Random Prime = 401
lil Number Theory $ java Primes
Random Prime = 907
lil Number Theory $ java Primes
Random Prime = 1549
lil Number Theory $ java Primes
Random Prime = 1063
lil Number Theory $ java Primes
Random Prime = 1607
lil Number Theory $ java Primes
Random Prime = 1217
lil Number Theory $ java Primes
Random Prime = 227
lil Number Theory $ java Primes
Random Prime = 2213
lil Number Theory $

```

## 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 geteuclidean(a%b,a,&x,&y)**

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

**\*y = x1;**

**Return gcd**

#### 4) sample runs :

The screenshot shows a Sublime Text editor window titled "Prbelem3.cpp" with the following C++ code:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int geteuclidean (int a, int b, int *x, int *y)
5 {
6     // Base Case
7     if (a == 0)
8     {
9         *x = 0;
10        *y = 1;
11        return b;
12    }
13    int x1, y1; // To store results of recursive call
14    int gcd = geteuclidean (b%a, a, &x1, &y1);
15
16    // Update x and y using results of
17    // recursive call
18    *x = y1 - (b/a) * x1;
19    *y = x1;
20    return gcd;
21 }
22
23 int main()
24 {
25     int x, y, a, b ; cout<<"enter a , b : ";
26     cin >> a >> b;
27     int g = geteuclidean(a, b, &x, &y);
28     cout << "GCD(" << a << ", " << b
29     << ") = " << g << endl;
30     cout << g << " = " << x << "*" << a << " + " << y <<
31     << endl;
32     return 0;
33 }

```

Overlaid on the code is a terminal window titled "Terminal" showing the following sample runs:

```

lil Number Theory $ g++ -o p3 Prbelem3.cpp
lil Number Theory $ ./p3
enter a , b : 22 55
GCD(22, 55) = 11
11 = -2*22 + 1*55
lil Number Theory $ ./p3
enter a , b : 654 125
GCD(654, 125) = 1
1 = -56*654 + 293*125
lil Number Theory $ ./p3
enter a , b : 78545 12358
GCD(78545, 12358) = 1
1 = 5593*78545 + -35548*12358
lil Number Theory $ ./p3
enter a , b : 7855 125
GCD(7855, 125) = 5
5 = 6*7855 + -377*125
lil Number Theory $ ./p3
enter a , b : 77 88
GCD(77, 88) = 11
11 = -1*77 + 1*88
lil Number Theory $

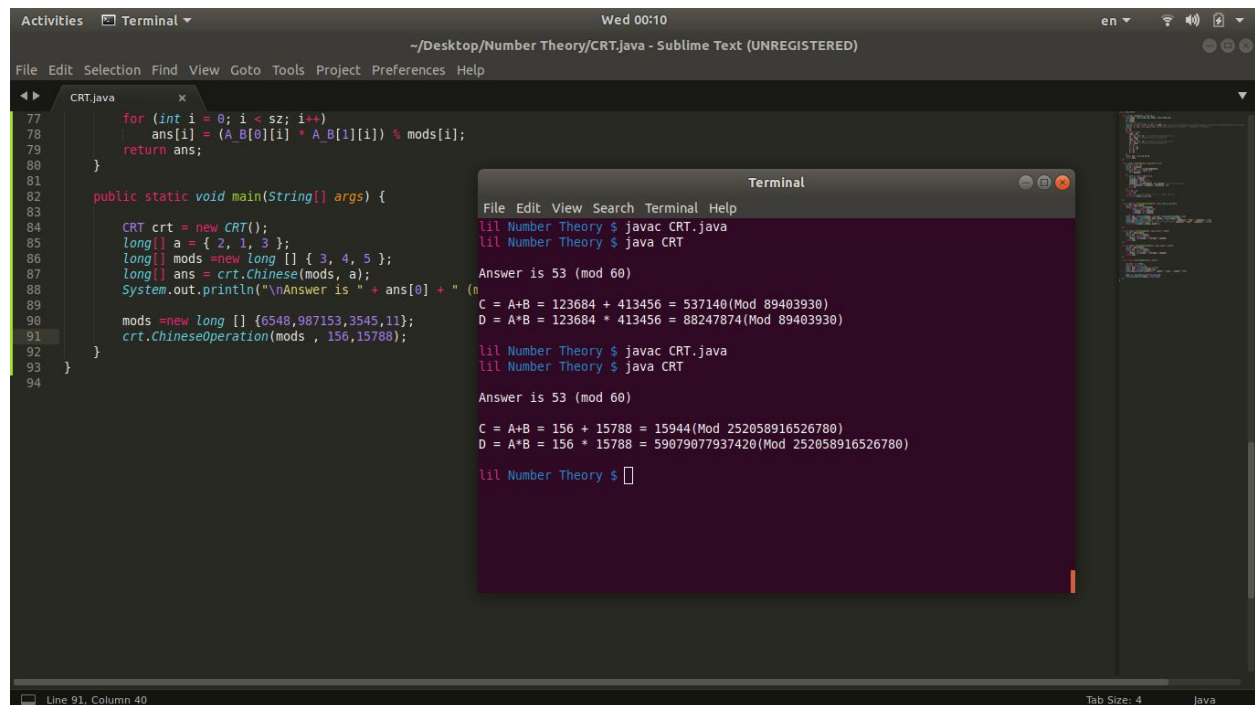
```

# Chinese Remainder Theorem :

## Complexity :

$$O(n * \log_2(\max(m_i, M_k)))$$

## Runs :



The screenshot shows a Sublime Text editor window with the file `CRT.java` open. The code implements the Chinese Remainder Theorem algorithm. A terminal window is overlaid on the editor, showing the output of the program. The output includes the calculation of the modular inverse and the final result.

```
File Edit View Search Terminal Help
lil Number Theory $ javac CRT.java
lil Number Theory $ java CRT

Answer is 53 (mod 60)

C = A+B = 123684 + 413456 = 537140 (Mod 89403930)
D = A*B = 123684 * 413456 = 88247874 (Mod 89403930)

lil Number Theory $ javac CRT.java
lil Number Theory $ java CRT

Answer is 53 (mod 60)

C = A+B = 156 + 15788 = 15944 (Mod 252058916526780)
D = A*B = 156 * 15788 = 59079077937420 (Mod 252058916526780)

lil Number Theory $
```