**Discrete Math**

**Functions**

1. **Problem Statement :**

Implementation to different function related to Discrete math :

1. Fast Modular Expontiation
2. Chinese Reminder Theorem (CRT) Applications
3. Prime Number Generator
4. Extended Euclidean Theorem
5. **Data Structure Used:**

Array of long integers .

1. **Algorithms Used :**
2. Fast Modular Expontiation :
3. Iterative :foo (a,b)

Complexity : O(Log2(n: max number of bits of(b,a)))

while b > 0

if b mod 2

ans = (ans \* a) mod m;

b--;

else

a = (a\*a)%m;

b /= 2;

return ans

1. Recursive :foo (a,b)

Complexity : O(Log2(n: max number of bits of(b,a)))

if b == 0

return 1;

if b mod 2

return (a \* foo((a \* a) mod m, (b - 1) / 2)) mod m;

else

return foo((a \* a) mod m, b / 2)mod m;

1. Naïve\_1 :

Complexity : O(b) , has a big chance in getting memory overflow if c\*a > int/long size range

c =1

for i = 1 to b

c = c \* a

c= c mod m

return c

1. Naïve\_2 :

Complexity : O(b) but avoids memory overflow for some range and consumes more time in computing (mod m)

c= 1

for i = 1 to b

c = (c \* a) mod m

return c

1. Extended Euclidean Theorem : foo(a,b)

Complexity : O(Log2(n : max number of bits (a,b)))

[This](https://www.youtube.com/watch?v=-uFc7-wOplM) was translated to a code

q = a / b

r = a % b

d = r;

a = b;

b = r;

while r != 0

s = s2;

s2 = s1 - s2 \* q;

s1 = s;

t = t2;

t2 = t1 - t2 \* q;

t1 = t;

d = r;

r = a % b;

q = a / b;

a = b;

b = r;

return d , s2 , t2;

1. Chinese Reminder Theorem : foo(a[] , m[])

Complexity : O(n \* Log2(max number of bits (mi,Mk))

// arrays represents functions of form X = ai (mod mi)

M =

For ( i =0 , i <n )

M­k = M/m­i

Yi = Mk-1 % m­i  // Modular inverse to Mk

x += (ai \* M­k \* Yi)

return x%M

1. Prime Number Generator : foo()

Complexity : O(n \* log2(log2(n)))

bool b[] = false ;

for (int i = 2; i \* i <= n; i++) //Assuming no primes before 2

if (!b[i])

primes.add(i);

for (int j = i \* i; j <= n; j += i)

b[j] = true;

return primes[random(primes.size)];

1. **Assumptions :**
2. In Extended Euclidian Theorem , S will always be multiplied to max(a,b) , T will always be multiplied to min(a,b).
3. Only the CRTOperation function will print on its own , the rest you should print the outcome on yourself in the main

N.B : Already made some print statements in the main

1. **Design decision :**
2. In CRT , I use Extended Euclidean Theorem to get inverse to ( Mi % mi ) using the fact

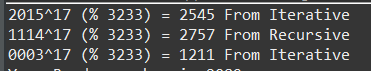
If gcd(a,b) =1 , then [ 1 = a\*S + b\*T ]where T would be the inverse of (b-1 )% a

1. **Sample Runs :**
2. **Modular Expontiation :**

Test was taken from Sheet 4.6 Ex. 25

(not Sure we should use 0003 or 0300 in the last one ,

but the answer is correct XD )

****

1. **Extended Euclidean Theorem :**

Test was taken from Sheet 4.3 Ex. 42

EGCD(356 , 252)



From Our code :

From an Online Calculator : [Link](https://planetcalc.com/3298/) to try yourself



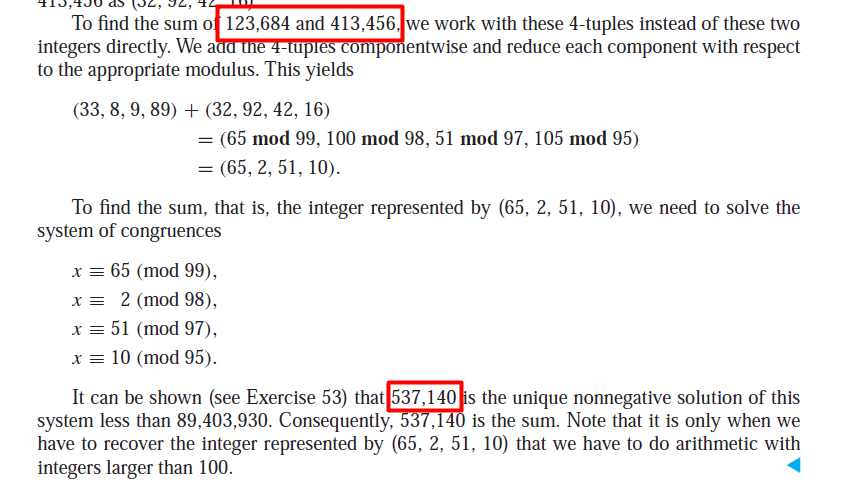
1. **Chinese Reminder Theorem :**

Test was taken from Book Section 4.4 pg 280

(At least the sum , The multiply operation was proved Correct by calculating it by Win Calculator )







1. **Prime Generator :**





They’re Randomly Generated I swear :D