Discrete Assignment 2

Name1 : Ali Muhammad Ali Hamed
ID: 42

Name 2 : Muhammad Zidan
ID: 57

Problem Statement 1

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.

Used Data structure : long integers

Pseudo code 1

```
c =1
for i = 1 to b
c = c * a
c= c mod m
return c
```

This one complexity is o(b) and when the number gets bigger it takes much time and memory to execute

Pseudo code 2

```
c= 1
for i = 1 to b
c = (c * a) mod m
return c
```

This one like the above but the difference is that i make mod before i save the the number to C so that make the memory smaller and can take bigger numbers than the above.

Fast Exp.

Pseudo code 3

Result =1
 a= a%m

```
While b >0
{       if b&1 ==1
                Result = (result * a) %m
        b=b>>1
        a=(x*x)%m
} return Result
```

Pseudo code 4

```
    if (A == 0)
        return 0;
    if (B == 0)
        return 1;
    long y;
    if (B % 2 == 0)
    {
        y = Recursive(A, B / 2, M);
        y = (y * y) % M;
    }
    else
    {
        y = A % M;
        y = (y * Recursive(A, B - 1, M) % M) % M;
    }
    return ((y + M) % M);
}
```

The last two methods have complexity of O(Log2(n: max number of bits of(b,a))) and they are faster than the previous two.

```
Choose from 1 , 2 , 3 And 4
1.Fast Exponentiation
2.Extended Euclidean Algorithm
3.Chinese Remainder Theorem
4.Prime Number Generation
1
Enter the first number
45
Enter the second number
8
Enter the mod number
7
2 (naive_1)
2 (naive_2)
2 (Fast using Iterator)
2 (Fast using Recursive)
```

Problem Statement 2

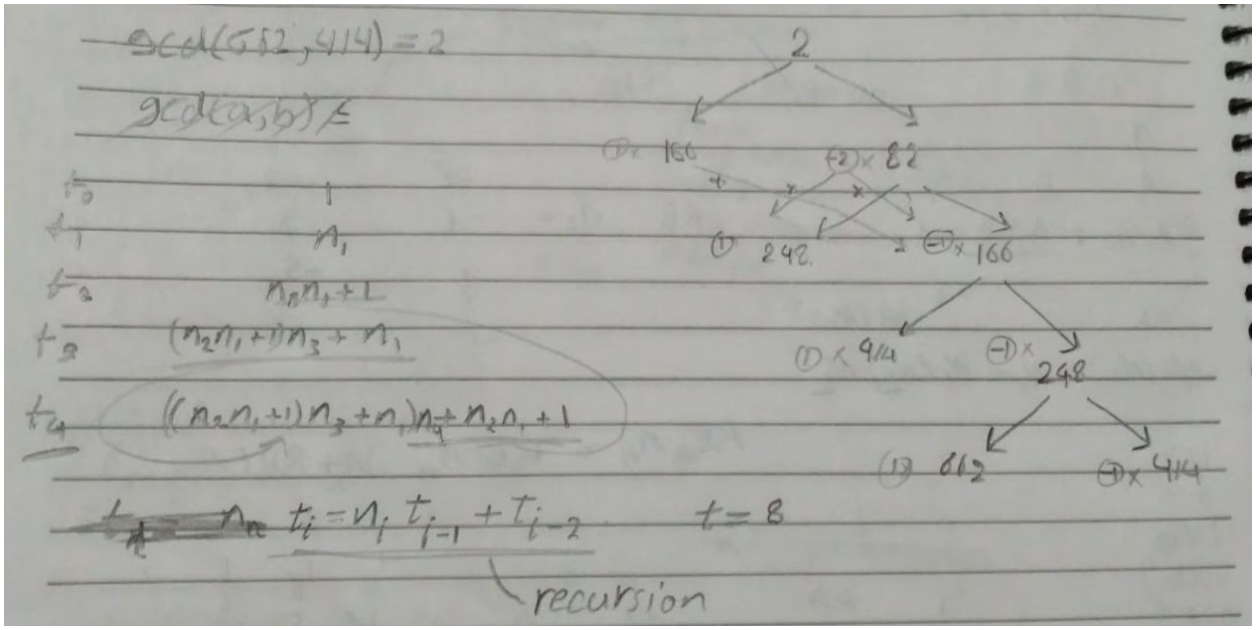
We need to construct a method that when we enter two numbers (say a, b) to it at gives us there GCD, using Euclidean theorem. And also it gives us another two numbers (s, t) so that sa + tb = gcd(a, b).

Used data structures:

- An arraylist to store some numbers that we use later in the algorithm.

Algorithms used:

I used recursion to first construct the main gcd method, and then to store numbers, and used it to find t. Then I get s.



مساء النور

```
a = 662
b = 414
d = 2
d is the Greatest Common Divisor of a and b.
s = -5, t = 8
s and t are two numbers in which for a given other two numbers a, b
 this relation applies: sa + tb = d, where d is the gcd for both a and b.
 You can find yourself that 2 = -5 × 662 + 8 × 414.
```

Pseudocode to find tn:
If i == 0 then return 1;
If i == 1 then return n[0]
Return t_i - 1 * n[i - 1] + t_i - 2

Problem Statement 3

We need to get the result of multiplying integers by using the CRT representation for the numbers instead of their regular way of representing.
Used data structures:
Arrays of type long to store the values.

```java
public static long inverse (long number, int m){
    long inverse = 0;
    number %= m;
    for (int i = 0; i < m; i++){
        if((number * i) % m == 1){
            inverse = i;
            break;
        }
    }
    return inverse;
}
```

```
A = 159
B = 951
m number = 3
M = {1999
2003
2017
}
Addition:
(Regular Addition) A + B = 1110
(Addition using CRT) A + B = 1110
Using the regular way, addition took 500 milliseconds, and using CRT addition took 173100.
In this situation Regular Way is better by 172600 milliseconds.
Multiplication:
(Regular Addition) A × B = 151209
(Addition using CRT) A × B = 151209
Using the regular way, addition took 300 milliseconds, and using CRT addition took 171400.
In this situation Regular Way is better by 171100 milliseconds.
```

Problem Statement 4

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

Decisions : i used sieveOfEratosthenes to generate the Primes

Data Structure : boolean array

Pseudo code:

```
boolean prime[] = new boolean[n+1];
    for(int i=0;i<n;i++)
        prime[i] = true;

    for(int p = 2; p*p <=n; p++)
    {
        // If prime[p] is not changed, then it is a prime
        if(prime[p] == true)
        {
            // Update all multiples of p
            for(int i = p*p; i <= n; i += p)
                prime[i] = false;
        }
    }
```

The boolean array index is the number and the value of the index is true when the number is prime and false otherwise.

```
Choose from 1 , 2 , 3 And 4
1.Fast Exponentiation
2.Extended Euclidean Algorithm
3.Chinese Remainder Theorem
4.Prime Number Generation
4
Enter a number
55
false
Choose from 1 , 2 , 3 And 4
1.Fast Exponentiation
2.Extended Euclidean Algorithm
3.Chinese Remainder Theorem
4.Prime Number Generation
4
Enter a number
101
true
Choose from 1 , 2 , 3 And 4
1.Fast Exponentiation
2.Extended Euclidean Algorithm
3.Chinese Remainder Theorem
4.Prime Number Generation
```