Assignment 2

Number theory

Name: Muhammed Essam Khamis

ID: 67.

Exercise 1:

Problem statement:

Implementation of a subroutine that takes three positive integer arguments (*a*, *b*, *n*) and returns the value of (a^b mod *n*), where the arguments are represented by about 100 decimal digits. And a Comparison between execution time of that subroutine implementation using the three different approaches discussed at the lecture. Draw a 2D line chart for the three implementations, where the x-axis represents the integer size and the y-axis represents the execution time.

Used data structure:

In this exercise we used bigInteger class which allow us to save and make mathematical operations on number that has a big size which will not fit in any small data type.

Algorithms used:

We implemented this exercise with three methods:

1-First of them takes the power and divide it by 2 then recurrence again until it reaches the base case which the power is equal to zero then the method returns one because any number with power of zero is equal to one after the method return we check the power if it is odd or even, if it's even then the result will be multiplication of the returned value with itself else the result

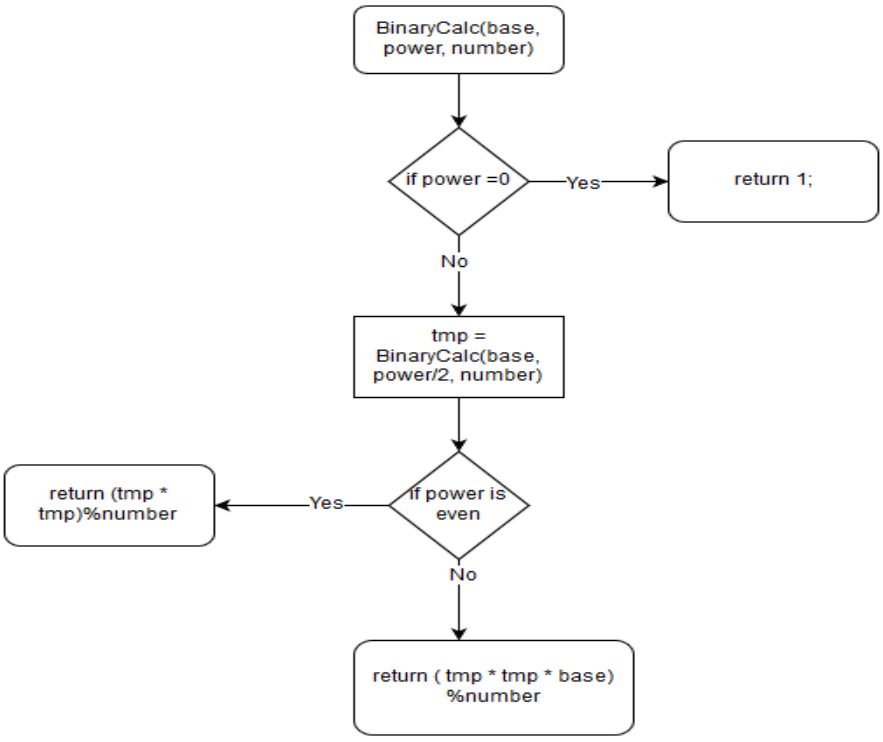
will be the multiplication of the returned value with itself and with the base. Then we take the mod of the multiplication.

2- second method was to multiply the base by itself with number equal the number of power, then we take the mod of the result

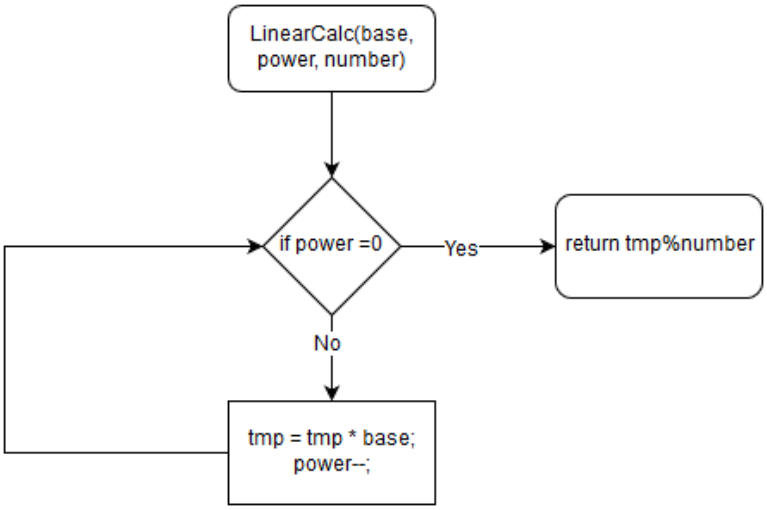
3- third method was to multiply the base with itself then take the mod of the multiplication this differ from the second method because we wait until the loop finishes then we take the mod, but in this method we take the mod inside the loop.

Flow charts:

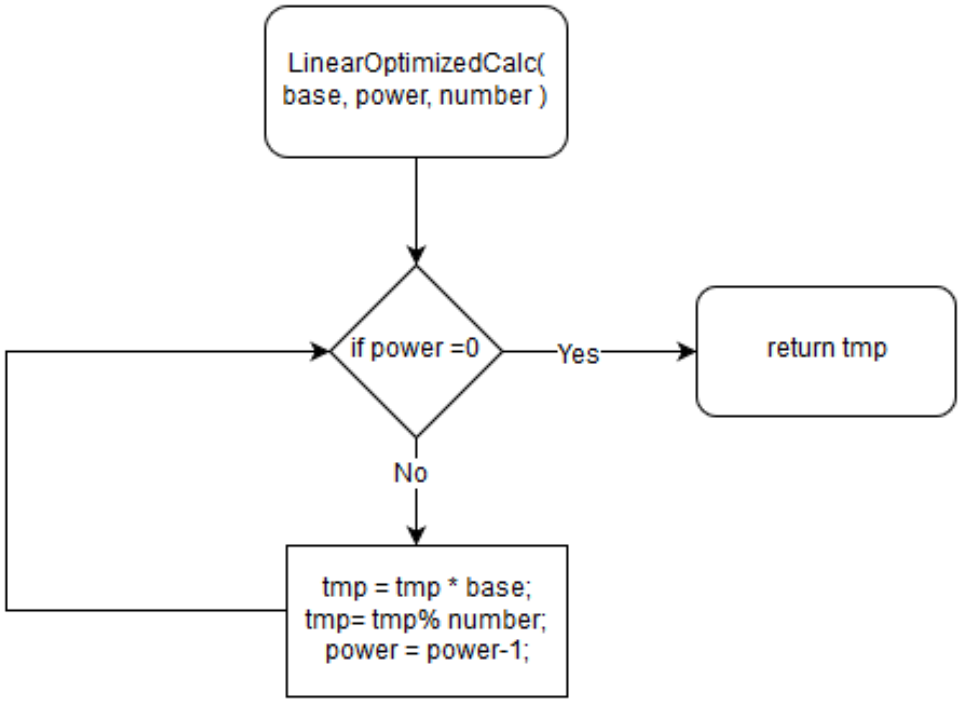
1-



2-


LinearCalc(base, power, number)
if power =0
Yes
return tmp%number
No
tmp = tmp * base;
power--;

3-


LinearOptimizedCalc(base, power, number)
if power =0
Yes
return tmp
No
tmp = tmp * base;
tmp= tmp% number;
power = power-1;

Sample Runs:

```
Enter A:
2
Enter B:
1000
Enter N:
9
Method 1 : 1ms
Method 2 : 5ms
Method 3 : 5ms
```

```
Enter A:
2
Enter B:
10000
Enter N:
9
Method 1 : 2ms
Method 2 : 26ms
Method 3 : 16ms
```

Exercise 2:

Problem statement:

Implementation of the extended Euclidean algorithm that finds the multiplicative inverse of *a* mod *n*, where *a* and *n* are positive integers that are relatively prime (i.e. gcd(*a*, *n*) = 1). The multiplicative inverse *b* is a positive integer that is uniquely determined such that (*ab*) mod *n* = 1.

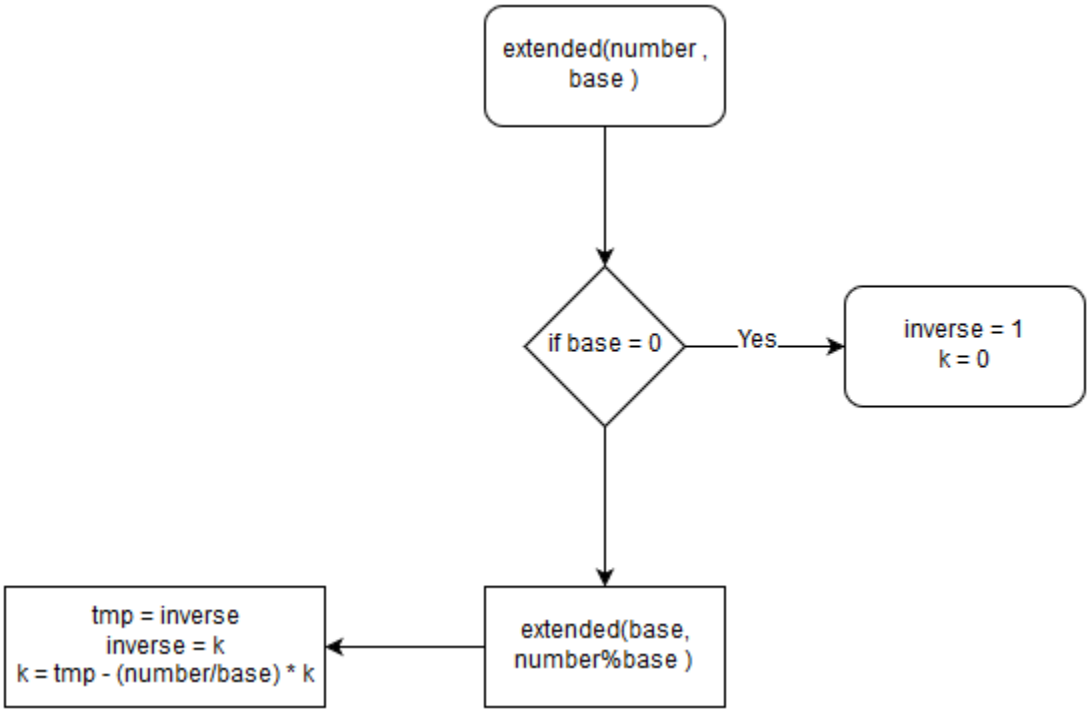
Used Data structure:

In this exercise we didn't use any external classes we only used arrays and simple data types.

Algorithm used:

Since ab mod n = 1 then ab = k*n +1 which k is a regular integer, from that we can say that ab-k*n = 1 and since the gcd between a and n is one the we can get the value of b using extended Euclidean algorithm.

Flow charts:



Sample runs:

```
Set N :
50
Set A
19
Result:
29
```

```
Set N :
541
Set A
21
Result:
438
```

Exercise 3:

Problem statement:

Implementation the unique mapping of Chinese Remainder Theorem, Let $M = m1 \times m2 \times \cdots \times mk$, such that for every $i \neq j$, gcd(mi,mj)=1 (i.e. relatively prime) There is a bijection $A \leftrightarrow (a1, a2, \ldots, ak)$ where $A \in ZM$ and the k-tuple $(a1, a2, \ldots, ak) \in Zm1 \times Zm2 \times \cdots \times Zmk$, The mapping from A to $(a1, a2, \ldots, ak)$ is such that $ai = A \bmod mi$ For the reverse mapping from $(a1, a2, \ldots, ak)$ to A ,$A = \Sigma ki\ (aiMiMi^{-1} \bmod M)$, where $Mi = mi \times m2 \times \cdots \times mi-1 \times mi+1 \times \cdots \times mk$, Mi^{-1} = *multiplicative inverse of Mi mod mi.*

Used Data structure:

We didn't use a complex data structure we only used arrays and simple data types.

Algorithm used:

We take the array of the small (m) which is used to convert any number less than their multiplication to an array has the same size of the array of small (m), then we take the map of A and map of B then convert map to its number as explained in problem statement. Then sum them and take the

mod of M, which give use the same result if we sum the maps and convert them to number.

Pseudo-code:

1- Take the map of M.
2- Take the map of A.
3- Take the map of B.
4- Convert map of A to a number.
5- Convert map B to a number.
6- Sum those two numbers.
7- Take the mod of the sum.
8- Sum map of A and B.
9- Take the mod to each element in the result map.
10- Convert result map to number.
11- Compare the time of the two methods.

Sample Runs:

```
Enter # of elements to be in the map :
3
Enter #1 :
3
Enter #2 :
5
Enter #3 :
8
Enter Elements of A :
Enter #1 :
1
Enter #2 :
0
Enter #3 :
4
Enter Elements of B :
Enter #1 :
0
Enter #2 :
2
Enter #3 :
7
Method 1 : 67 with time : 167336 ns
Method 2 : 67 with time : 48193 ns
```