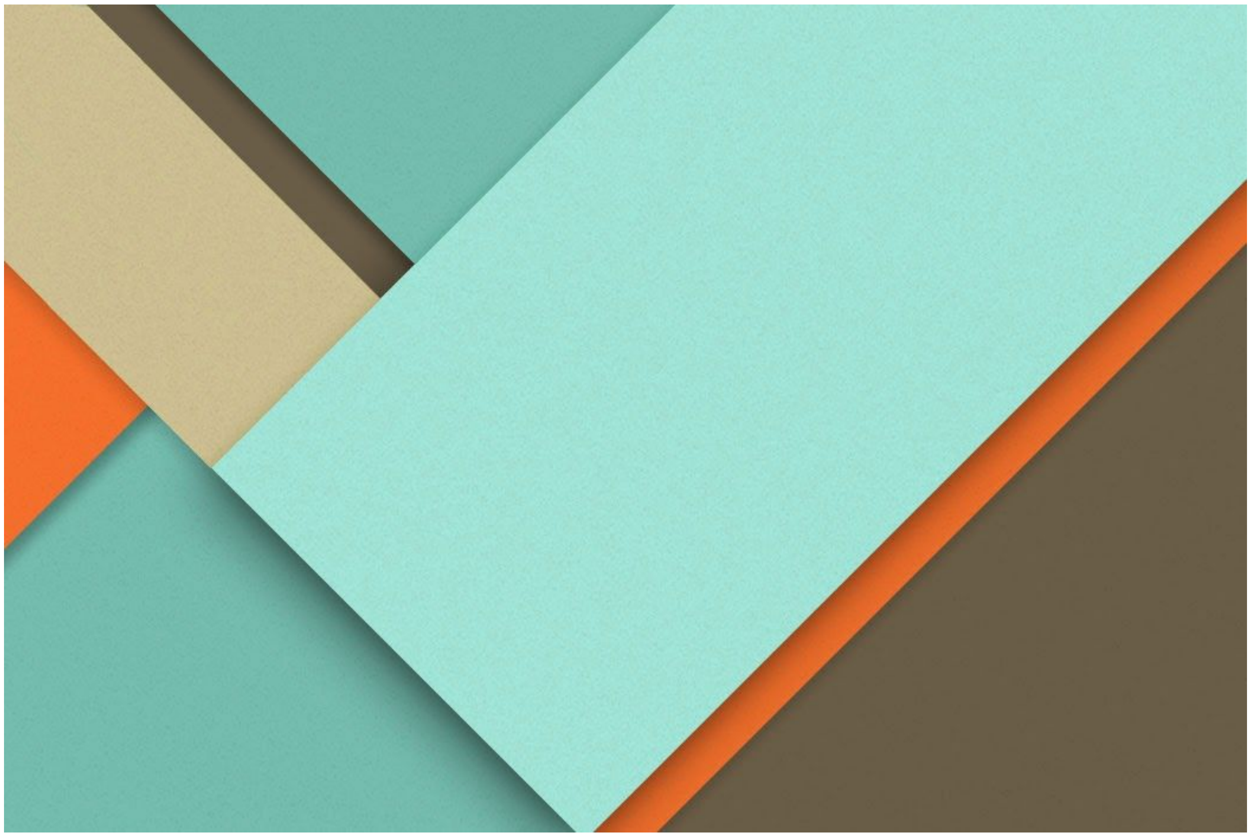


Fast Exponentiation

12.15.2020

Ahmed Gamal Mahmoud Hefny
18010083

Problem Statement

We Are required to implement 4 methods for computing modular Exponentiation and compare the execution time of each method with the increase number of bits representing an integer, Also decide when overflow happens .

Time is compared in the graph below , Also in both Naive 1 / naive 2 time taken is too long while in Fast Exponentiation : iterative is not that long while recursion takes more time that iterative as shown in all graphs and also shown in the sample numbers figure below .

→ Keep in mind that in naive 1 graph time is low because overflow happened at low bit so the function stopped working when overflow happened so the time didn't grow. As i assumed that when overflow happens , i'll stop taking more bits and stop calculating time.

OverFlow Happens in Naive 1 at the 4th bit , Naive 2 at the 16th bit , Fast exponentiation iterative at the 16th bit and Fast exponentiation recursion at the 16th bit

Used Data Structure

1. List.
2. Array.

Pseudo Code

Decimal To Binary(n)

```
out = ''
   list2 = []
   base = 2
   while num != 0
      temp = int(num) / base
      whole = int(temp)
      fl = temp - whole
      num = whole
      m = int(fl * base)
      Add string ( m ) to list 2
```

```
out = list2
return out
```

Naive 1(b,n,m)

```
r = 1
    for i =0 to n
        r = r * b
        if r > 2 ** 31 - 1:
            return -1

    r = r mod m
    return r
```

Naive 2(b,n,m)

```
r = 1
    for i =0 to n
        r =r * b
        if  r > 2 ** 31 - 1
            return -1
        r = r mod m

    return r
```

Exponential Iterative(b,n,m)

```
n2 = DecimalToBinary(n)
    r = 1
    power = b mod m
    k = length of n2
```

```
for i =0 to k
    if n2[i] == '1'
        r = (r * power)
        if r > 2 ** 31 - 1
            return -1
        r = r mod m

    power = (power * power) mod m
return r
```

Exponential Recursion(power,n,m,r)

```
if length of n = 0
    return r
if n[0] == '1'
    r = r * power
    if r > 2 ** 31 - 1
        return -1
    r = r mod m
n = n[1:] to remove first number in the binary
power = (power * power) mod m
return fastExpRecursion(power, n, m,r)
```

Assumptions:

1. Assumed that i'm dealing with signed positive integer numbers
2. Numbers are generated and will be put in the methods and results will be plotted which means user won't put any numbers in UI
3. Number of bits will stop increasing when overflow happens

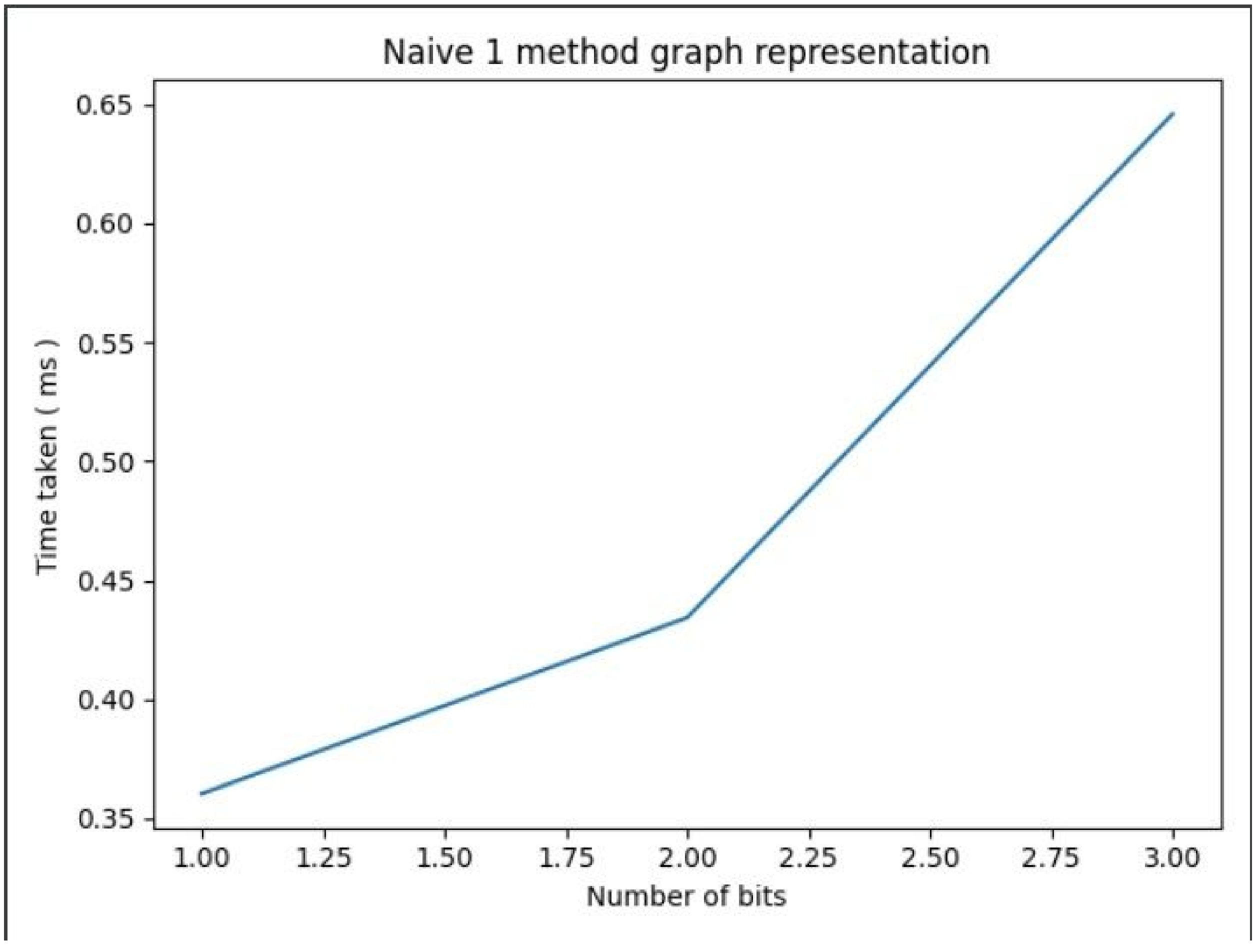
Design Decisions :

In Order to Plot the results :

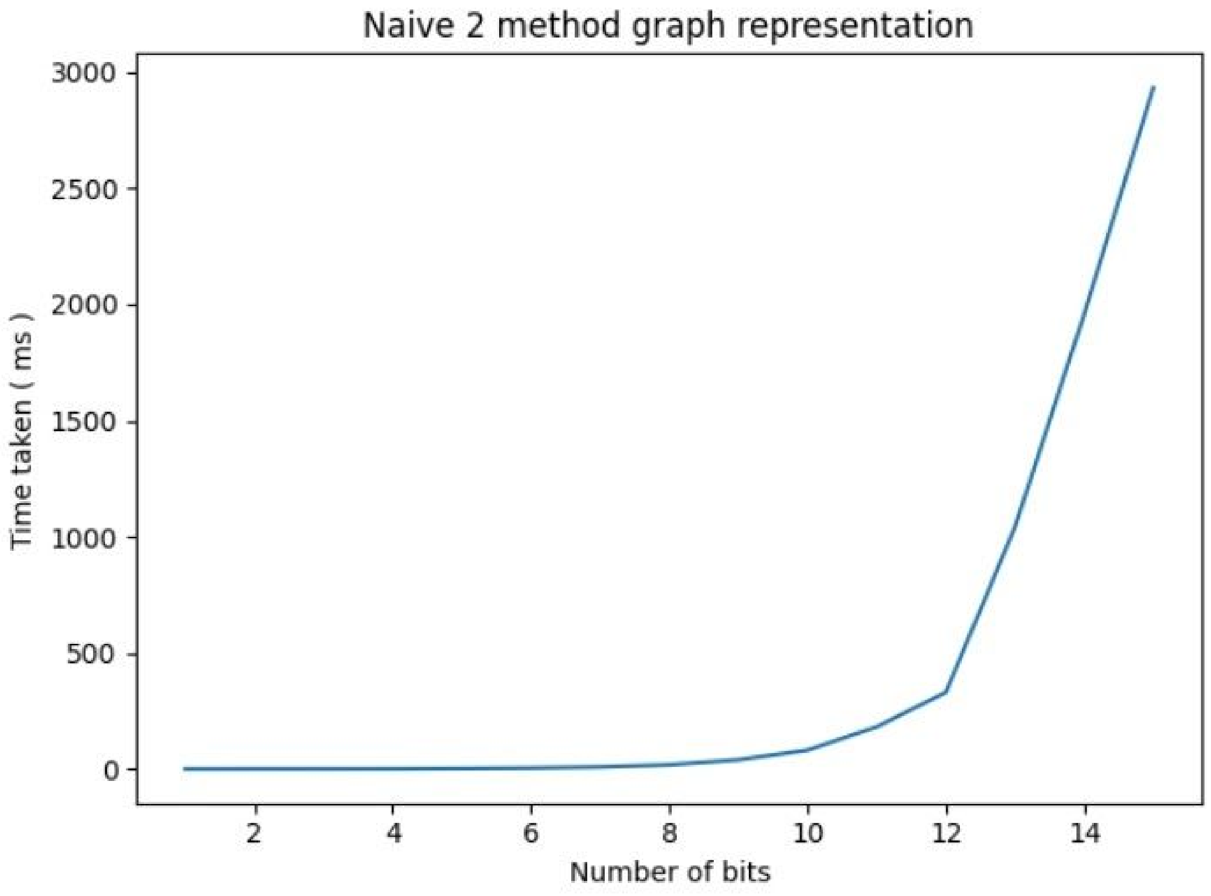
1. B , n, m are randomly generated (for example : 5 bits)
2. In this 5 bit numbers , they are not randomly generated once but 20 times and time taken by the method isn't calculated once But calculated 500 times to get accurate average time
3. IMPORTANT NOTE : Before using FastExponentialRecursion you must do the following :
    i. Call decimaltobinary function to change the *n* to binary
    ii. Initialize *power* variable to power = b mod m
    iii. Then put the power and binary number to the function along with the other parameters
    iv. *r* variable is always initialized to 1
4. decimalToBinary(n) function takes an integer and produces Binary Expansion to this integer BUT in reverse order in order to be used directly in other function according to my algorithm .. Example : 10 = 1010 But my funcion will produce 10 = 0101

Sample Runs

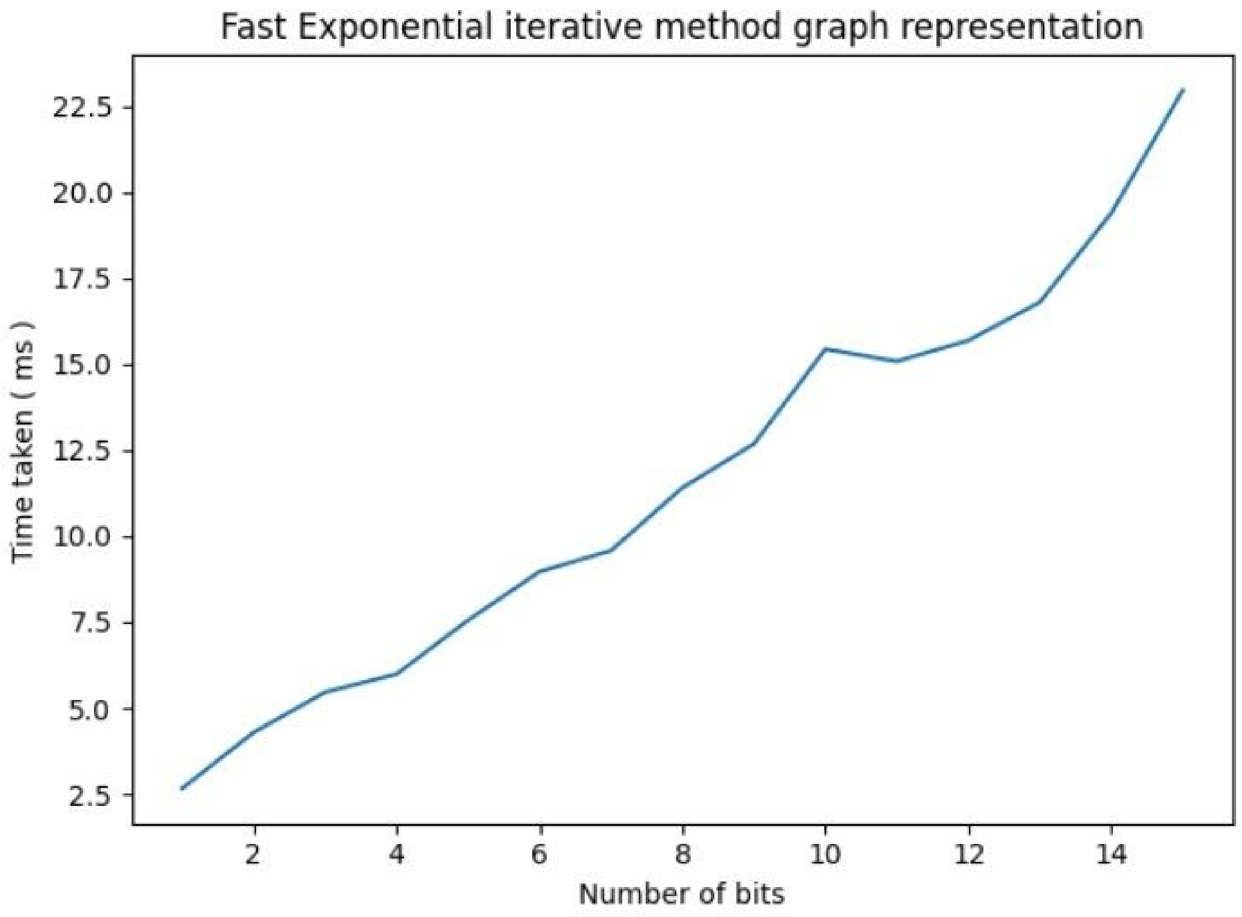
Naive 1 :



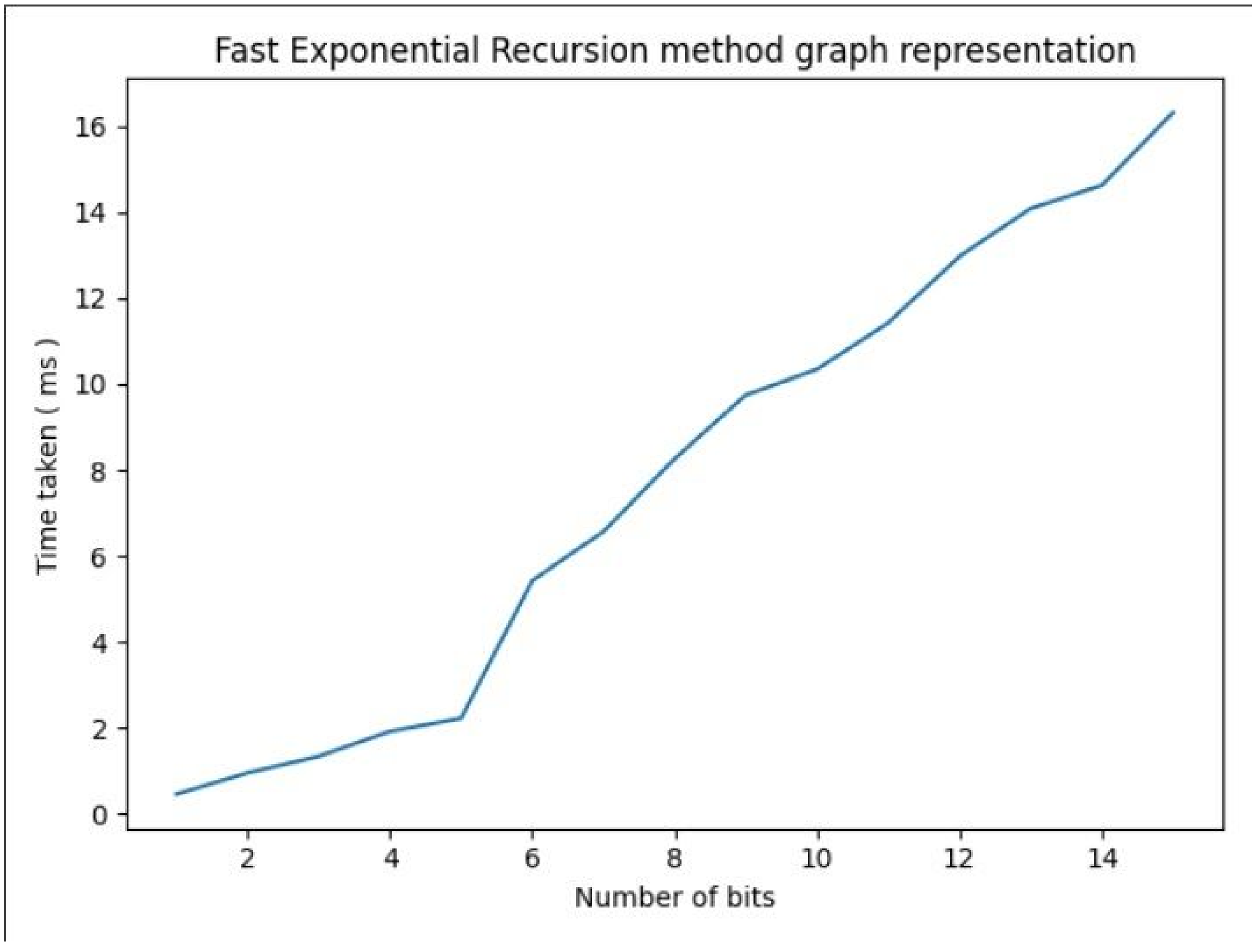
Naive 2 :


Naive 2 method graph representation
Time taken (ms)
Number of bits
3000
2500
2000
1500
1000
500
0
2
4
6
8
10
12
14

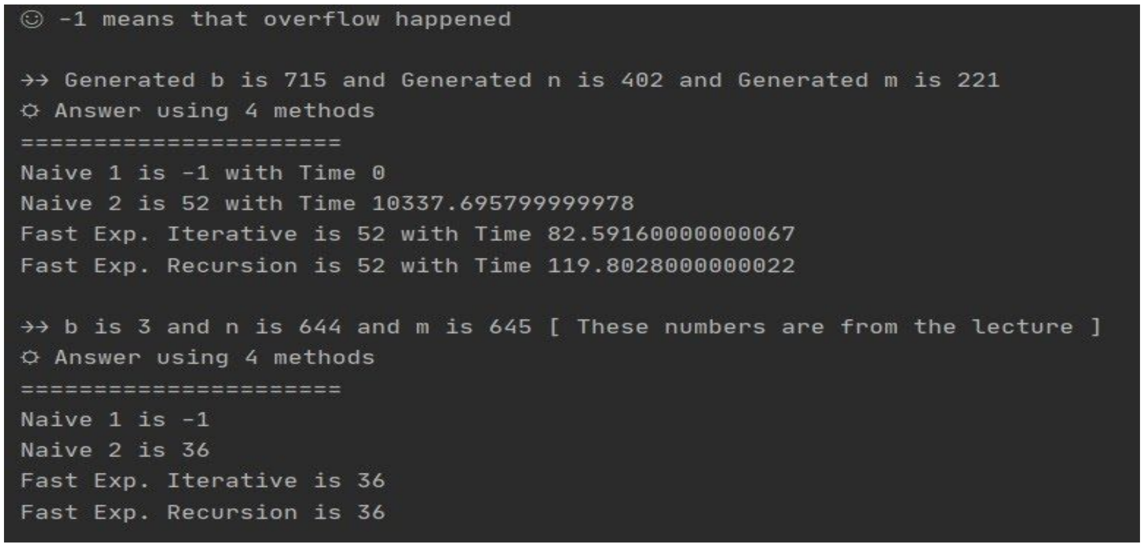
FE iterative :



FE recursion :



Sample numbers :



```
☺ -1 means that overflow happened

→→ Generated b is 715 and Generated n is 402 and Generated m is 221
☼ Answer using 4 methods
======================
Naive 1 is -1 with Time 0
Naive 2 is 52 with Time 10337.695799999978
Fast Exp. Iterative is 52 with Time 82.59160000000067
Fast Exp. Recursion is 52 with Time 119.8028000000022

→→ b is 3 and n is 644 and m is 645 [ These numbers are from the lecture ]
☼ Answer using 4 methods
======================
Naive 1 is -1
Naive 2 is 36
Fast Exp. Iterative is 36
Fast Exp. Recursion is 36
```