Basic Number theory Algorithm

1. Binary exponenetial

Raising a to the power of n is naively represented as multiplying by a n1 times: an=aa...a. However, for large an or n, this method is impractical.

The objective behind binary exponentiation is to divide the work using the exponent's binary representation.

Since the number n has exactly $[\log_2 n] + [\log_2 n\rfloor + 1$ digits in base 2, we only need to perform O(logn)multiplications, if we know the powers a1,a2,a4,a8,...,a[logn]

The final complexity of this algorithm is O(logn): we have to compute logn powers of a, and then have to do at most logn multiplications to get the final answer from them.

2. GCD

Euclid's algorithm is based on **Euclid's rule** which states

If x and y are positive integers with x ≥ y, then gcd(x, y) = gcd(x mod y, y).

Proof

We can easily notice that $x\ mod\ y$ can also be seen as

$$x\%y = x - ky$$

where k is any non-negative integer constant such that $x - ky$ is least possible positive integer.

From this we can easily conclude that it is enough to show simpler rule $\gcd(x, y) = \gcd(x - y, y)$.

This rule allows to get a very good recursive algorithm where we decrease the value of (x,y) by replacing it with (y,x mod y) until we arrive at our base case where one of the value will be 0 and we know gcd(x,0) = x.

Pseudocode

```
function Euclid(x, y)
Input: Two integers x and y with x≥ y≥ 0
Output: gcd(x, y)
if y= 0: return x
return Euclid(y,x mod y)
```

Time Analysis

The time complexity of the algorithm depends upon how fast the values of parameter decrease as it is directly directly proportional to no of recursive steps.

Lemma *If x ≥ y, then x mod y < x/2*

Proof:

Notice that we can divide this into 2 cases

Case 1 *y <=x/2* : We know by the property of division that x mod y < y and in this case y <= x/2 . Hence x mod y < x/2.

Case 2 *y>x\2* : In this case since *2y* > *x*, our **x mod y** will just be **x-y** and since x/2 <y implies *x-y* < *x/2*. Hence x mod y < x/2

From this lemma we can say that after any 2 consecutive recursive steps values of both input parameter are at least halved (length of each input parameter is decreased by 1-bit).

Hence for n-bit numbers it will have at max 2n recursive calls. Each call will have division operation of quadratic order. Final complexity = 2n * O(n^2) = O(n^3)

Extended Euclid Algorithm

Problem that this algorithms solves:

`Input`: Two positive integers a and b with $a \geq b \geq 0$

`Output`: Integers x, y, d such that $d = \gcd(a, b)$ and $ax + by = d$

Lemma: *If d divides both a and b, and d = ax + by for some integers x and y, then necessarily d = gcd(a, b)*

Proof:

Since we know d divides both a and b it is a common divisor and hence $d \leq \gcd(a, b)$. And since gcd(a,b) also divides both a and b it should also divide ax + by = d, which implies gcd(a, b) ≤ d. Putting these together, d = gcd(a, b).

The coefficients x and y are just a small extension to Euclid’s algorithm

```
function extended-Euclid(a, b)
Input: Two positive integers a and b with a ≥ b ≥ 0
Output: Integers x, y, d such that d = gcd(a, b) and ax + by = d
if b = 0: return (1, 0, a)
(x0, y0, d) = extended-Euclid(b, a mod b)
return (y, x0 - floor(a/b)y0, d)
```

Proof of Correctness of recursive version of bezout's coefficient

Lemma *For any positive integers a and b, the extended Euclid algorithm returns integers x, y, and d such that gcd(a, b) = d = ax + by.*

$$\gcd(b, a \mod b) = bx_0 + (a \mod b)y_0$$

We know the value of modulo operator is calculated by $a \mod b = (a - \lfloor a/b \rfloor b)$

$$d = \gcd(a, b) = \gcd(b, a \mod b) = bx + (a \mod b)$$

$$= bx_0 + (a - \lfloor a/b \rfloor b)y_0 = ay_0 + b(x_0 - \lfloor a/b \rfloor by_0)$$

Hence d = ax + by with x = y and y = $x_0 - \lfloor a/b \rfloor by_0$,