Computer Science & Engineering Department
IIT Kharagpur
Computational Number Theory: CS60094
Lecture II

Instructor: Goutam Biswas *Spring Semester 2014-2015*

1 Basic Properties of Integers I

We have already informally introduced some of the following basic concepts. In most of these case we shall omit the proofs.

1.0.1 Division Theorem

If $a, b \in \mathbb{Z}$ and b is positive, then there are two unique integers q (*quotient*) and r (*remainder*), such that $a = bq+r$, where $0 \leq r < b$. The value of $q = \lfloor a/b \rfloor$ and $r = a \mod b = a - b \times \lfloor a/b \rfloor$.
Example 1. If $a = 14$ and $b = 5$, then $14 = 5 \times 2 + 4$, so $q = 2$ and $r = 4$. If $a = -14$ and $b = 5$, then $-14 = 5 \times -3 + 1$, so $q = -3$ and $r = 1$.

This can be generalised as follows: If $a, b \in \mathbb{Z}$, b is positive, and x is a real number, then there are two unique integers q (*quotient*) and r (*remainder*), such that $a = bq + (\lceil x \rceil + r)$, where $0 \leq r < b$. In fact this is equivalent to saying that $a - \lceil x \rceil = bq + r$, $0 \leq r < b$.

We give a proof of the generalised form. It is clear that there are b integers in the interval $[x, x+b)$. They are $\lceil x \rceil, \cdots, \lceil x+b \rceil - 1 = \lceil x \rceil + (b-1)$.
Let $c = a - \lceil x \rceil$. We define the set of non-negative integres

$$S = \{c - by : \ y \in \mathbb{Z} \text{ and } c - by \geq 0\}.$$

We claim that S is a non-empty: if $c \geq 0$, $c - b \cdot 0 \in S$; otherwise $c - b \cdot c > 0$ as $b > 0$ and $-c$ is positive, so $c - b \cdot c \in S$.
By the *well-ordering principle* there is a smallest element r of S. Let $r = c - bq$, where $q \in \mathbb{Z}$. So $a - \lceil x \rceil = c = bq + r$ i.e. $a = bq + \lceil x \rceil + r$ where $r \geq 0$.
We claim that $r < b$, otherwise $0 \leq r - b = c - b(q+1)$. So $r - b \in S$, but that is impossible as $r - b$ is less than r. So we have

$$a = bq + (\lceil x \rceil + r), \ 0 \leq r < b.$$

Clearly $\lceil x \rceil + r \in [x, x+r)$.

1.0.2 Binary Relation Divides ('|')

We define a binary relation '|' on the set of integers, $a|b$. Examples are $3|12$, but $5 \not| 12$.

For an integer n, let $D(n)$ be the set of positive divisors of n. It is clear that $D(n) = D(-n)$. If $m, n \in \mathbb{Z}$, then $D(m) \cap D(n)$ is the set of *common divisors* of m and n. If both m and n are not zero, then $D(m) \cap D(n)$ is finite and has a largest element known as the *greatest common divisor* of m and n. It is written as $gcd(m, n)$ or at times (m, n). We have the following proposition.

1.0.3 Bezout's Identity

Proposition 1. Let m, n be integers so that both are not zero. $gcd(m, n)$ is the smallest positive integer that can be written in the form $mx + ny$, where $x, y \in \mathbb{Z}$. This is known as *Bezout's Identity*.
Proof: We define the set of positive integres

$$S = \{mx + ny : \ x, y \in \mathbb{Z} \text{ and } mx + ny > 0\}.$$

The set S is non-empty. We assume without any loss of generality that $m \neq 0$. If $m < 0$, then take $x = -1$ and $y = 0$; otherwise take $x = 1$ and $y = 0$. By the well ordering principle there is a smallest element $d \in S$. Let $d = mx_0 + ny_0$. We claim that $d|m$ and $d|n$. If $d \not| m$, then by the division theorem $m = dq + r$, where q and r are integers and $0 < r < d$. But then $r = m - dq = m - (mx_0 + ny_0)q = m(1 - x_0q) - ny_0q$. Clearly $r \in S$. But that is not possible as d is the smallest element of S and $r < d$. So $r = 0$ i.e. $d|m$. Similarly we can prove that $d|n$. So the smallest element of S is in $D(m) \cap D(n)$, a common divisor of m, n.

Let $c \in D(m) \cap D(n)$, any common divisor of m and n such that $m = cm'$ and $n = cn'$. We have $d = mx_0 + ny_0 = c(m'x_0 + n'y_0)$. So $c|d$ i.e. $gcd(m, n) = d$. □

From the previous proof we further note that every element of

$$S = \{mx + ny : \ x, y \in \mathbb{Z} \text{ and } mx + ny > 0\}$$

is a multiple of d. If $1 \in S$, then m and n are relatively prime. In other words if m, n are relatively prime, then there are integers x, y such that $mx + ny = 1$.

Following proposition gives an algebraic view of this identity.
Definition 1: Let R be a non-empty set equipped with two binary operations '$+$' (addition - need not be usual) and '$\cdot$' (multiplication - again need not be usual), where $(R, +, 0)$ is an *abelian* or *commutative group* with 0 as the identity element of '$+$', and $(R, \cdot, 1)$ is a commutative monoid such that the operation '$\cdot$' is distributed over '$+$'. We call the algebraic structure $(R, +, 0, \cdot, 1)$, a *commutative ring with identity*.

Let I be a subgroup of $(R, +, 0)$ such that for all $a \in I$ and for all $r \in R$, $a \cdot r \in I$. I is called an *ideal* of R.

The set of integers $\mathbb{Z}$ is an example of a *commutative ring with identity* with the usual addition and multiplication operations. We shall characterise $gcd(m, n) = d$ through the ideals of $\mathbb{Z}$.

It is not difficult to show that for all $a \in \mathbb{Z}$, the set $a\mathbb{Z} = \{ax : \ x \in \mathbb{Z}\}$ is an ideal of $\mathbb{Z}$.
Proposition 2. Let m, n be integers and $gcd(m, n) = d$, then $d\mathbb{Z} = m\mathbb{Z} + n\mathbb{Z}$, We define $gcd(0, 0) = 0$.
Proof: We use the following facts about an ideal over $\mathbb{Z}$.

1. If I and J are two ideals, then so is $I + J$.

2. Every ideal I over $\mathbb{Z}$ is of the form $a\mathbb{Z}$. Following is a proof of this fact.

   Proof: If $I = \{0\}$, then $d = 0$. Take $I \neq \{0\}$. Consider

   $$P = \{a \in I : \ a > 0\}.$$

   The set is non-empty as both x and $-x$ are in I and one of them will be positive. By the *well ordering principle* there is a least element $a \in P$. From the definition of an ideal, for all $x \in \mathbb{Z}$, $a \cdot x \in I$. So $a\mathbb{Z} \subseteq I$.

Let $b \in I$, we claim that $a|b$; otherwise, $b = aq + r$, $0 < r < a$. So, $r = b - aq$ and r must belong to P. But this is a contradiction as $r < a$. So $b = ac$ for some $c \in \mathbb{Z}$, and $I \subseteq a\mathbb{Z}$.

Now we come to the final proof. We already know that $m\mathbb{Z} + n\mathbb{Z} = d\mathbb{Z}$ for some $d \in \mathbb{Z}$. If $m = 0 = n$, then $d = 0$. So we assume that both m and n are not zero. It is clear that $m = m \cdot 1 + n \cdot 0$ and $n = m \cdot 0 + n \cdot 1$ are in $d\mathbb{Z}$. So d divides both m and n, hence $d \in D(m) \cap D(m)$.

Again $d \in d\mathbb{Z} = m\mathbb{Z} + n\mathbb{Z}$, so $d = mx_0 + ny_0$. If $c \in D(m) \cap D(n)$, then $m = cm_0$ and $n = cn_0$ i.e. $d = c(m_0x_0 + n_0y_0)$, so $c|d$. □

1.1 GCD Algorithms

An efficient algorithm to find the *greates common divisor (gcd)* or *highest common factor (hcf)* is known for more than two-thousand years. It is the well known a Euclid's algorithm, taught in the school.

1.1.1 Euclid's Algorithm

Let s and l be two non-negative integers so that $l \geq s \geq 0$.

$$gcd(l, s) = \begin{cases} l & \text{if } s = 0, \\ gcd(s, l \bmod s) & \text{otherwise.} \end{cases}$$

If $s > 0$, we know that $l = qs + r$, where $q = \lfloor \frac{l}{s} \rfloor$ and $r = l \bmod s$, so that $0 \leq r < s$. An integer d is a divisor of s and l if and only if it is a divisor of s and r. So $D(l) \cap D(s) = D(s) \cap D(r)$ i.e. $d = gcd(s, l)$ if and only if $d = gcd(r, s)$.

The Euclidean algorithm simplifies the problem by reducing the arguments. And non-negative arguments cannot be reduced indefinitely, so the process terminates.

Let l, s be integers so that $l \geq s \geq 0$. The Euclid's algorithm generates two finite sequences of non-negative integers, $r_0, r_1, \cdots, r_{n+1}$, the sequence of remainders; and $q_1, q_2, \cdots, q_n$, the sequence of quotients. We set $r_0 = l, r_1 = s$.
If $n > 0$ and $r_n = gcd(l, s)$, the computation sequence is as follows:

$$\begin{aligned} r_0 &= r_1q_1 + r_2, \text{ so that } 0 < r_2 < r_1, \\ r_1 &= r_2q_2 + r_3, \ 0 < r_3 < r_2, \\ &\vdots \\ r_{i-1} &= r_iq_i + r_{i+1}, \ 0 < r_{i+1} < r_i, \\ &\vdots \\ r_{n-1} &= r_nq_n + (r_{n+1} = 0) \end{aligned}$$

We have $r_{i-1} = r_iq_i + r_{i+1}$, and $gcd(r_{i-1}, r_i) = gcd(r_i, r_{i+1})$. So,

$$gcd(l, s) = gcd(s, r_2) = \cdots = gcd(r_{n-1}, r_n) = gcd(r_n, 0) = r_n.$$

The algorithm correctly computes the GCD.
Proposition 3. The Number of steps of computation, n, is $O(\log s)$.

Proof: We have $r_{i-1} = r_i q_i + r_{i+1}$, $r_{i+1} < r_i$, for all $i = 1, 2, \cdots, n$. So, $r_{i-1} = r_i q_i + r_{i+1} \geq r_i + r_{i+1} > 2r_{i+1}$. This implies that $r_{i+1} < \frac{r_{i-1}}{2}$.
In every two steps of computation, the length of the remainder is reduced by at least 1-bit. The length of s is $\lceil \log_2 s \rceil$ bits. So the number of steps (n) required to make the remainder zero is less than $2 \log_2 s$, i.e. $n = O(\log s)$. □
Proposition 4. Let $l > s > 0$, and $n \geq 1$ be the number of division steps required for gcd computation. Then $l \geq F_{n+2}$ and $s \geq F_{n+1}$, where F_n is the n^{th} Fibonacci number[1].
Proof: The proof is by induction on the number of steps n.
Basis: ($n = 1$) - if there is one step of computation, then $s \geq 1 = F_2$, but then $l > s$, so $l \geq 2 = F_3$.
We assume that the claim is true for $n - 1$ steps. We prove that the claim is true if there are n steps. After the first step, $l = sq_1 + r$, the computation takes $n - 1$ steps starting with $r_1 = s$ and $r_2 = r$. By our hypothesis $s = r_1 \geq F_{n+1}$ and $r = r_2 \geq F_n$.
Now $l = sq_1 + r_2 =\geq s + r_2 \geq F_{n+1} + F_n = F_{n+2}$. □
Theorem 5. (*Lamé*) If $l \geq s \geq 0$ and $s < F_{k+1}$, then $gcd(l, s)$ can be computed in less than k steps.

This is a direct consequence of the previous proposition.

Proposition 6. Let l, s be integers so that $l \geq s > 0$. If the number of steps to compute $gcd(l, s)$ is n, then $n \leq \frac{\log s}{\log \Phi} + 1$, where $\Phi = \frac{1+\sqrt{5}}{2}$ is the *golden ratio* and its approximate value is 1.618.
Proof: $s > 0$, so $n \geq 1$. If $n = 1$, the statement is true.
If $n > 1$, we claim that $r_{n-i} \geq \Phi^i$, for $i = 0, 1, \cdots, n$. That is $r_0 \geq \Phi^n$, $r_1 \geq \Phi^{n-1}, \cdots, r_{n-1} \geq \Phi^1 = \Phi, r_n \geq \Phi^0 = 1$.
Case $i = 0, 1$*:* r_n is not zero, so $r_n \geq 1$. $r_n < r_{n-1}$, so $r_{n-1} \geq r_n + 1 \geq 2 > \Phi^1$.
Case $n \geq i > 1$*:* We use induction:

$$r_{n-i} = r_{n-i+1} q_{n-i+1} + r_{n-i+2} \geq r_{n-(i-1)} + r_{n-(i-2)} \geq \Phi^{i-1} + \Phi^{i-2} = \Phi^{i-2}(1 + \Phi) = \Phi^i.$$

Note that $1 + \Phi = 1 + \frac{1+\sqrt{5}}{2} = \left(\frac{1+\sqrt{5}}{2}\right)^2 = \Phi^2$. So $s = r_1 = r_{n-(n-1)} \geq \Phi^{n-1}$. Taking logarithm we get $\log s \geq (n-1) \log \Phi$, implies that $n \leq \frac{\log s}{\log \Phi} + 1$. □

Number of steps in Euclid's algorithm is not to be confused with the time complexity of the algorithm. Let l, s be two k-bit numbers.

Crude Analysis: Number of divisions are $O(\log s) = O(k)$. Each division involves k or fewer bit numbers, and it takes $O(k^2)$ basic steps. So the time complexity is $O(k^3)$.

But we can do a better analysis. At every stage we compute the quotient q_i and remainder r_{i+1}, $i = 1, \cdots, n$ ($r_{i-1} = q_i r_i + r_{i+1}$). The time complexity for division is $O(\log q_i \times \log r_i)$. So the running time is,

$$\begin{aligned} T &= \sum_{i=1}^{n} (\log r_i \times \log q_i), \\ &\leq \log s \sum_{i=1}^{n} \log q_i, \\ &\leq \log s \sum_{i=1}^{n} (\log r_{i-1} - \log r_i + 1), \text{ number of quotient bits,} \end{aligned}$$

[1]The sequence of F_n, the Fibonacci sequence is $0, 1, 1, 2, 3, 5, 8, 13, 21, \cdots$.

$$\begin{aligned}
&\leq \log s(\log r_0 - \log r_n + n),\\
&\leq \log s(\log l + \log s),\ n = O(\log s),\\
&= O(\log s \log l).
\end{aligned}$$

So the time complexity is quadratic, not cubic.

1.2 Extended Euclid's Algorithm

We already have proved that the $gcd(a, b) = d$ can be written as $d = ax + by$, where x, y are integers. The *Bezout's coefficients* x and y can be computed along with the gcd. The corresponding algorithm is known as the *extended Euclid's algorithm.*

Let $l \geq s \geq 0$, the extended gcd algorithm generates four finite sequences of integers, $r_0, r_1, \cdots, r_{n+1}$ ($r_0 = l, r_1 = s$) is the sequence of remainders, $q_1, q_2, \cdots, q_n$ is the sequence of quotients, $x_0, x_1, \cdots, x_n$ and $y_0, y_1, \cdots, y_n$ are two sequences of coefficients. Similar to the Euclid's algorithm remainders and quotients can be computed as usual. The sequence of coefficients satisfy the following property.

$$r_i = lx_i + sy_i, \text{ for all } i = 0, \cdots, n.$$

So the initial values of the coefficients are as follows:

$$\begin{aligned}
x_0 \leftarrow 1, \quad y_0 \leftarrow 0,\\
x_1 \leftarrow 0, \quad y_1 \leftarrow 1,
\end{aligned}$$

satisfying $r_0 = lx_0 + sy_0 = l \cdot 1 + s \cdot 0$ and $s = r_1 = lx_1 + sy_1 = l \cdot 0 + s \cdot 1$.

The rules for computation of the next set of coefficients is simple to derive.

$$\begin{aligned}
r_{i-1} &= r_i q_i + r_{i+1},\\
lx_{i-1} + sy_{i-1} &= (lx_i + sy_i)q_i + (lx_{i+1} + sy_{i+1}),\\
lx_{i+1} + sy_{i+1} &= l(x_{i-1} - x_i q_i) + s(y_{i-1} - y_i q_i).
\end{aligned}$$

So we have

$$x_{i+1} \leftarrow x_{i-1} - x_i q_i,\ y_{i+1} \leftarrow y_{i-1} - y_i q_i, \text{ for } i \leftarrow 1, 2, \cdots, n.$$

Using this fact the extended algorithm looks as follows:

```
extGCD(l, s)
    $r_0 \leftarrow l$, $r_1 \leftarrow s$
    $x_0 \leftarrow 1$, $x_1 \leftarrow 0$
    $y_0 \leftarrow 0$, $y_1 \leftarrow 1$
    while $r_1 > 0$ do
        $q \leftarrow \lfloor r_0/r_1 \rfloor$
        $rt \leftarrow r_0 \mod r_1$
        $r_0 \leftarrow r_1$
        $r_1 \leftarrow rt$
        $xt \leftarrow x_0 - x_1 \times q$, $yt \leftarrow y_0 - y_1 \times q$
        $x_0 \leftarrow x_1, x_1 \leftarrow xt$
        $y_0 \leftarrow y_1, y_1 \leftarrow yt$
    return $(r_0, x_0, y_0)$
```

The time complexity of the algorithm is similar to the original algorithm.

Starting from the input pair $(l, s) = (r_0, r_1)$, we calculate a sequence of pairs, $(r_1, r_2), (r_2, r_3), \cdots, (r_n, r_{n+1} = 0)$. In the 2×2 matrix notation we may write for $i = 1, \cdots, n$,

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix}$$

If we expand the right hand side we get

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_{i-1} \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} \begin{pmatrix} l = r_0 \\ s = r_1 \end{pmatrix}$$

If we define M_i for $i = 0, \cdots, n$,

$$M_i = \begin{cases} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \text{if } i = 0, \\ \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} M_{i-1} & \text{if } i > 0. \end{cases}$$

We ge the values of the coefficients.

$$M_i = \begin{pmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{pmatrix}$$

and we have

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = M_i \begin{pmatrix} l \\ s \end{pmatrix}$$

The determinant of each 2×2 matrix is -1. So the determinant of M_i, $det(M_i)$, is $(-1)^i$.

Example 2. We consider $l = 40902$ and $s = 24140$. Following table shows the values of different variables at the beginning of each iteration.

i	r_i	q_i	x_i	y_i	$lx_i + sy_i$
0	40902		1	0	40920
1	24140	1	0	1	24140
2	16762	1	1	-1	16762
3	7378	2	-1	2	7378
4	2006	3	3	-5	2006
5	1360	1	-10	17	1360
6	646	2	13	-22	646
7	68	9	-36	61	68
8	34	2	337	-571	34

So we have $gcd(40920, 24140) = 34 = 40920 \times 337 + 24140 \times (-571)$.

1.3 Binary GCD Algorithm

In a computer subtraction and division by 2 are less costly than ordinary division and extraction of remainder. Division by 2 can be performed by right-shift operation on data. We note the following properties when l and s are positive integers.

- If both l and s are even, $gcd(l, s) = 2 \times gcd(l/2, s/2)$.
- If l is odd and s is even, then $gcd(l, s) = gcd(l, s/2)$.
- Similarly, if l is even and s is odd, then $gcd(l, s) = gcd(l/2, s)$.
- If both l and s are odd, then $gcd(l, s) = gcd((l - s)/2, s)$.

Using these facts Josef Stein proposed the *binary gcd* algorithm in 1967[2]. The algorithm is as follows. We assume that l and s are non-negative integers.

```
binGCD($l, s$)
    $e \leftarrow 0$
    while $2|l \wedge 2|s$ do
        $l \leftarrow l/2,\ s \leftarrow s/2,\ e \leftarrow e + 1$
    repeat
        while $2|l$ do $l \leftarrow l/2$
        while $2|s$ do $s \leftarrow s/2$
        if $s < l$ then $l \leftrightarrow s$
        $s \leftarrow s - l$
    until $s = 0$
    return $2^e \times l$
```

We claim that after two executation of the '`repeat-until`' loop, the number of bits of the larger argument is reduced by one. If the larger argument execute one of the inner '`while`' loops, one bit is reduced. Otherwise, after the subtraction we get the larger argument to be even which in the next iteration will be divided by 2.
So the time complexity is $O(l^2)$ where l is the length of the larger input.

We can use the *binary GCD algorithm* to compute the Bezout's coefficients. Following is an algorithm where both s, l are positive.

```
extBinGCD(l, s)
    $e \leftarrow 0$
    while $2|l \wedge 2|s$ do
        $l \leftarrow l/2,\ s \leftarrow s/2,\ e \leftarrow e + 1$
    $x_0 \leftarrow 1,\ y_0 \leftarrow 0,\ x_1 \leftarrow 0,\ y_1 \leftarrow 1$
    $L \leftarrow l,\ S \leftarrow s$
    repeat
        while $2|l$ do
            $l \leftarrow l/2$
            if $2|x_0 \wedge 2|y_0$ then
                $x_0 \leftarrow x_0/2,\ y_0 \leftarrow y_0/2$
            else $x_0 \leftarrow (x_0 + S)/2,\ y_0 \leftarrow (y_0 - L)/2$
        while $2|s$ do
            $s \leftarrow s/2$
            if $2|x_1 \wedge 2|y_1$ then
                $x_1 \leftarrow x_1/2,\ y_1 \leftarrow y_1/2$
```

[2]It seems that it was known to China in the first century CE.

```
16           else $x_1 \leftarrow (x_1 + S)/2,\ y_1 \leftarrow (y_1 - L)/2$
17       if $s < l$ then
18           $l \leftrightarrow s,\ x_0 \leftrightarrow x_1,\ y_0 \leftrightarrow y_1$
19       $s \leftarrow s - l,\ x_1 \leftarrow x_1 - x_0,\ y_1 \leftarrow y_1 - y_0$
20   until $s = 0$
21   $d \rightarrow 2^e \times l$
22   return $(d, x_0, y_0)$
```

If $l = 2^k \cdot l'$ and $s = 2^k \cdot s'$, the $gcd(l, s) = 2^k{\cdot}gcd(l', s')$. If $gcd(l', s') = xl' + ys'$, then $gcd(l, s) = 2^k gcd(l', s') = 2^k \cdot (xl' + ys') = xl + ys$. This justifies line:1-3 of the algorithm.

If none of l and s are even and $s < l$, then we perform line:18-19, where $gcd(l, s) = gcd(s, l)$ after exchange and $gcd(s, l) = gcd(s - l, l)$. Here $l = x_0L + y_0S$ and $s = x_1L + y_1S$, $s - l = (x_1 - x_0)L + (y_1 - y_0)S$. So the line: 19.

When the control reaches line:6 for the first time, at least one of s and l is *odd*, so one of L and S is also *odd*. When the control reaches line:17, both l and s are *odd*. When the computation finishes line:18-19, s is even and l is *odd*. So in the `repeat-until`-loop, in every iteration only one `while`-loop will be executed. In fact except for the first iteration, the second `while`-loop will always be executed as the value of s is *even*.

The computation of inner '`while loop`' is as follows. Let us assume that in some iteration we have $s = x_1L + y_1S$ and $s = 2s'$ (even). There are two possibilities.
Case I: when both x_1 and y_1 are even, we have $s' = \frac{x_1}{2}L + \frac{y_1}{2}S$. So the modified values of (x_1, y_1) are $(x_1/2, y_1/2)$.
Case II: Let us assume, without any loss of generality, that x_1 is *odd* and y_1 is *even*. According to our assumption, $s = x_1L + y_1S$ is even. So L must be even. But then and S cannot be even as all common even factors of S and L are taken out. In this case both $(x + S)$ and $(y - L)$ are even and we may write $s' = (x + S)L/2 + (y - L)S/2$.

The analysis of worst case time complexity is as usual. It is $O(l^2)$, where l is the length of the larger argument.

1.3.1 Fundamental Theorem of Arithmetic

The theorem says that every integer $n > 1$ can be expressed as unique (up to reordering) product of finite number of primes.

$$n = p_1^{e_1} p_2^{e_2} \cdot \cdots \cdot p_k^{e_k},$$

where p_i, $i = 1, \cdots, k$ are distinct primes and e_i's are positive integers. Clearly negative integers also have unique factorisation with a (-1) factor.

For each prime p we define a map $\nu_p : \mathbb{Z} \setminus \{0\} \rightarrow \mathbb{N}_0$[3] such that for each prime p and integer $n > 0$, $\nu_p(n) = e_p$, where $\nu_p(1) = 0$ and for all $n > 1$, $n = p^{e_p}m$, such that $p \not| m$. It is clear that $p \not| n$, then $e = 0$. Using this functional notation we may express any integer other than zero as

$$n = \pm \prod_p p^{\nu_p(n)},$$

where the product is over all primes

This function can be used to state several interesting properties. Let a, b be non-zero integers, then

[3] $\mathbb{Z} \setminus \{0\} = \{\cdots, -2, -1, 1, 2, \cdots\}$, $\mathbb{N}_0 = \{0, 1, 2, 3, \cdots\}$.

1. $\nu_p(a \cdot b) = \nu_p(a) + \nu_p(b)$,
2. $a|b$ if and only if $\nu_p(a) \leq \nu_p(b)$,
3. $gcd(a, b) = \prod_p p^{min(\nu_p(a),\nu_p(b))}$,
4. $lcm(a, b) = \prod_p p^{max(\nu_p(a),\nu_p(b))}$,

References

[AB] *Computational Number Theory* by Abhijit Das, (will be published from CRC Press).

[MD] *Primality Testing in Polynomial Time From Randomized Algorithms to "PRIMES is in P"*, bMartin Dietzfelbinger, LNCS 3000 (Tutorial), Pub. Springer, 2004, ISBN 3540403442.

[VS] *A Computational Introduction to Number Theory and Algebra* by Victor Shoup, 2nd ed., Pub. Cambridge University Press, 2009, ISBN 978-0-521-51644-0.