# 6.875 Number Theory Lecture Notes

Vinod Vaikuntanathan

September 22, 2021

This evolving set of notes will serve as lecture notes for the number-theoretic portion of 6.875.

## 1 Groups

An Abelian group $G = (S, \circ)$ is a set $S$ together with a binary operation $\circ : S \times S \to S$ which satisfies

- **Identity:** There is an element $\mathsf{Id} \in S$ such that for all $a \in S$, $a \circ \mathsf{Id} = \mathsf{Id} \circ a = a$.
- **Inverse:** For every $a \in S$, there is an element $b \in S$ such that $a \circ b = b \circ a = \mathsf{Id}$.
- **Associativity:** For every $a, b, c \in S$, $a \circ (b \circ c) = (a \circ b) \circ c$.
- **Commutativity:** For every $a, b \in S$, $a \circ b = b \circ a$.

Notice that we defined Abelian (or commutative) groups by default. Those will be the only type of groups that we see in this class. The definition of general groups is the same as the above, except without the commutativity property. Also, most of the time in this class, the groups we deal with will be finite, namely $S$ will be a finite set. An occasional exception is the additive group $\mathbb{Z}$ of all integers.

**Some notations.**

- We will sometimes abuse notation and say that $g \in G$ when we really mean that $g \in S$.
- We will denote the multiplicative shorthand for iterated group operations by default. That is, for $g \in S$, $g^2 = g \circ g$, $g^3 = g \circ g \circ g$, and so forth.
- $g^0 = \mathsf{Id}$ and $g^{-1}$ denotes the inverse of $g$.

**A Running Example: The Additive Group $\mathbb{Z}_N$.** Our running example will be the additive group $\mathbb{Z}_N = (S = \{0, 1, 2, \ldots, N-1\}, +)$ consisting of integers from $0$ to $N-1$, where the group operation is addition modulo $N$. The identity element is $0$, its inverse is $0$, and the inverse of $x \neq 0$ is $N - x$. We will call them the additive identity and the additive inverse, to distinguish them from their multiplicative friends who will show up soon, and who will turn out to be more useful.

**Order of a Group and the order of an element.** The order of a group is the number of elements in it, namely $|S|$. The order of an element $g \in S$ is the number of times one has to perform the group operation on $g$ to get to the identity element $\mathsf{Id}$. That is,

$$\mathsf{ord}(g) = \min_{i>0}\{g^i = \mathsf{Id}\} \ .$$

For example, the order of the group $\mathbb{Z}_N$ is $N$.

**Theorem 1.1** (Lagrange's Theorem). *The order of any element divides the order of the group.*

*Proof.* Let $g \in G$ be some group element. Note that multiplication by $g$ defines a bijection on $G$. That is, $f : G \to G$ defined by $f(x) = gx$ is a one-to-one and onto function.

We first claim that $g^{|G|} = \mathsf{Id}$. Let's multiply all the group elements in two ways. Notice that on the left and the right, we are computing the product over the same set of elements since multiplying by $g$ permutes the elements of the group. So,

$$\prod_{h \in G} h = \prod_{h \in G}(hg) = g^{|G|} \circ \Big(\prod_{h \in G} h\Big)$$

Dividing by $\prod_{h \in G} h$ from the left and the right gives us $g^{|G|} = \mathsf{Id}$.

Now let $x$ be the order of $g$. Let $x' = \mathsf{gcd}(x, |G|)$. By Extended Euclid, there are integers $a$ and $b$ such that

$$ax + b|G| = x'$$

Therefore,

$$g^{x'} = g^{ax+b|G|} = (g^x)^a (g^{|G|})^b = (\mathsf{Id})^a (\mathsf{Id})^b = \mathsf{Id}$$

If $x' < x$, this contradicts the assumption that $x'$ was the order of $G$, and therefore the smallest positive power of $G$ that gives $\mathsf{Id}$. So, it must be the case that $x' = x$ and thus $x = \mathsf{gcd}(x, |G|)$ and consequently, $x$ divides $G$. $\square$

**Generator of a Group.** A generator of a group $G$ is an element of order $|G|$. In other words,

$$G = \{g, g^2, \ldots, g^{|G|} = \mathsf{Id}\}$$

For example, $1$ is a generator of $\mathbb{Z}_N$ for any $N$.

**Cyclic group.** A group $G$ is called cyclic if it has a generator. By the above, $\mathbb{Z}_N$ is always cyclic. We also know:

**Theorem 1.2.** *Every group whose order is a prime number, is cyclic. Moreover, every element other than the identity is a generator.*

*Proof.* Exercise. $\square$

**Discrete Logarithms.** Let $G$ be a cyclic group. We know that $g$ has a generator, and that every $h \in G$ can be written as $h = g^x$ for *a unique* $x \in \{1, 2, \ldots, |G|\}$. We write

$$x = \mathsf{dlog}_g(h)$$

to denote the fact that $x$ is the discrete logarithm of $h$ to the base $g$.

We will look for groups where computing the group operations is easy (namely, polynomial time) but computing discrete logarithms is hard (namely, exponential or sub-exponential time). Our source for such groups will come from number theory, so let us move on to reviewing a few basic notions from (computational) number theory.

Discrete logarithms in $\mathbb{Z}_N$ are, for better or worse, easy. Indeed, the generators of $\mathbb{Z}_N$ are precisely those $g$ whose greatest common divisor with $N$ is 1. If you are told that

$$x \cdot g = h \pmod{N}$$

then $x$ can be computed as $h \cdot g^{-1} \pmod{N}$ where $g^{-1}$ now denotes the *modular multiplicative inverse* of $g$ which can be found easily using the extended Euclidean algorithm. (See next section). It is at this point that we abandon $\mathbb{Z}_N$ and move on to other groups, notably the multiplicative group $\mathbb{Z}_N^*$.

## 2 Baby Computational Number Theory

**Greatest Common Divisors, Euclid and Extended Euclid.** The greatest common divisor (gcd) of positive integers $a$ and $b$ is the largest positive integer $d$ that divides both $a$ and $b$. $a$ and $b$ are relatively prime if their gcd is 1.

One can find $d$ using Euclid's algorithm in time $O(n^2)$ where $n$ is the maximum of the bit-lengths of $a$ and $b$. One can do more: it turns out that there are always integers (not necessarily positive) $x$ and $y$ such that

$$ax + by = d$$

These integers can be found in essentially the same complexity using a modification of Euclid's algorithm, referred to as extended Euclid. We will assume that you have seen these algorithms and facts (say, in a previous course such as 6.042 or 6.006.)

**Modular Arithmetic.** We expect you to be familiar with modular arithmetic. For integers $a$ and $b$, $a \pmod{N}$ will denote the remainder upon dividing $a$ by $N$.

$b$ is the (multiplicative) inverse of $a$ if $ab = 1 \pmod{N}$. In this case, we say $b = a^{-1} \pmod{N}$. The multiplicative inverse of $a$ exists if and only if $\gcd(a, N) = 1$.

Exercise: Show a polynomial-time algorithm to compute the multiplicative inverse of a given number $a$ modulo a given $N$.

**Modular Exponentiation.** This is the operation of computing $a^b \pmod{N}$. Computing $a^b$ over the integers and then reducing mod $N$ is a horribly inefficient way to do this (think about it). Instead, one uses the repeated squaring algorithm that runs in time $O(n^3)$.

| Operation | Time Complexity | Remarks |
|---|---|---|
| $a+b$ | $O(n)$ | grade-school addition |
| $ab$ | $O(n^2)$ | grade-school multiplication |
| | $O(n \log n)$ | Harvey and van der Hoeven 2020 |
| $\mathsf{gcd}(a,b)$ | $O(n^2)$ | Euclidean algorithm or binary gcd algorithm |
| $a \pmod N$ | $O(n^2)$ | |
| $a+b \pmod N$ | $O(n^2)$ | |
| $ab \pmod N$ | $O(n^2)$ | |
| $a^{-1} \pmod N$ | $O(n^2)$ | extended Euclidean algorithm |
| $a^b \pmod N$ | $O(n^3)$ | repeated squaring algorithm |
| Factoring $N$ | $2^{O(n^{1/3}(\log n)^{2/3})}$ | Number field sieve |
| Discrete log in $\mathbb{Z}_N^*$ | $2^{O(n^{1/3}(\log n)^{2/3})}$ | Number field sieve |
| Discrete log in any group $G$ | $O(\sqrt{\lvert G \rvert})$ | baby step-giant step algorithm |

Figure 1: The complexity of basic operations with numbers. $n$ denotes the input length for each of these operations.

## 3 The Multiplicative Group $\mathbb{Z}_N^*$

The multiplicative group of numbers mod $N$, denoted $\mathbb{Z}_N^*$, consists of the set

$$S = \{1 \le a < N : \mathsf{gcd}(a,N) = 1\}$$

with multiplication mod $N$ being the group operation.

**Theorem 3.1.** *$\mathbb{Z}_N^*$ is a group. Operations in this group including multiplication and computing inverses can be done in time polynomial in the bit length of the numbers, namely* $\mathsf{poly}(\log N)$.

*Proof.* Exercise. ☐

**Examples.**

$$\begin{aligned}
\mathbb{Z}_2^* &= \{1\} \\
\mathbb{Z}_3^* &= \{1,2\} \\
\mathbb{Z}_4^* &= \{1,3\} \\
\mathbb{Z}_5^* &= \{1,2,3,4\} \\
\mathbb{Z}_6^* &= \{1,5\} \\
\mathbb{Z}_7^* &= \{1,2,3,4,5,6\}
\end{aligned}$$

Some further facts about $\mathbb{Z}_N^*$:

- The order of $\mathbb{Z}_N^*$, the number of positive integers smaller than $N$ that are relatively prime to it, is called the Euler totient function of $N$ denoted $\varphi(N)$.

- If $N = \prod_i p_i^{\alpha_i}$ is the prime factorization of $N$, then

$$\varphi(N) = \prod_i p_i^{\alpha_i - 1}(p_i - 1)$$

Prove this! For example, if $N$ is prime, then $\varphi(N) = N - 1$ and if $N = PQ$ is a product of two primes, then $\varphi(N) = (P-1)(Q-1)$.

- For every $a \in \mathbb{Z}_N^*$,

$$a^{\varphi(N)} = 1 \pmod{N}$$

This is called Euler's theorem, a direct consequence of Lagrange's theorem and the fact that the order of $\mathbb{Z}_N^*$ is $\varphi(N)$. In the special case where the modulus $P$ is prime, for every $a \in \mathbb{Z}_P^*$,

$$a^{P-1} = 1 \pmod{P}$$

a fact that is referred to as Fermat's little theorem.

## 4 The Multiplicative Group $\mathbb{Z}_P^*$ for a Prime $P$

Let's first focus on the case of $\mathbb{Z}_P^*$ when $P$ is prime. Then, $\mathbb{Z}_P^* = \{1, 2, 3, \ldots, P-1\}$ and its order is $\varphi(P) = P - 1$.

**$\mathbb{Z}_P^*$ is Cyclic.** The following is a very important property of $\mathbb{Z}_P^*$ when $P$ is prime.

**Theorem 4.1.** *If $P$ is prime, then $\mathbb{Z}_P^*$ is a cyclic group.*

For example, $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\} = \{5^6, 5^4, 5^5, 5^2, 5^1, 5^3\} = \{5^i \pmod{7} : i > 0\}$. So, 5 is a generator of $\mathbb{Z}_p^*$. We refer the reader to Angluin's notes for a proof.

It is very tempting to try to prove this theorem by appealing to Theorem 1.2 which says that every group with prime order is cyclic. Many before you have succumbed to this mistake. Be careful, and note that the order of $\mathbb{Z}_P^*$ is $P - 1$, which is *decidedly not prime*.

Even if $N$ is not prime, $\mathbb{Z}_N^*$ may end up being cyclic: in particular, it is known that this happens exactly when $N = 1, 2, 4, p^k$ or $2p^k$ where $p$ is an odd prime number. However, we will never encounter these beasts in our course.

One could ask several followup questions such as, how many generators are there for $\mathbb{Z}_p^*$, how to tell if a given element $g$ is a generator, and how to sample a random generator for $\mathbb{Z}_p^*$. We will answer them in the sequel, starting with the first question.

**$\mathbb{Z}_P^*$ has lots of generators.** The proof of the following theorem is contained in the proof of theorem 4.1 and can be found in Angluin's notes.

**Theorem 4.2.** *The number of generators in $\mathbb{Z}_P^*$ is $\varphi(P-1)$.*

Now, how large is $\phi(P-1)$ asymptotically? This is answered by the following classical theorem.

**Theorem 4.3.** *For every integer $N$, $\phi(N) = \Omega(N/\log\log N)$.*

In other words, if you pick a random element of $\mathbb{Z}_P^*$, you will see a generator with probability

$$\varphi(P-1)/(P-1) = \Omega(1/\log\log P)$$

which is polynomial in $1/\log P$. So, reasonably often!

**The multiplicative group $\mathbb{Z}_P^*$ and the additive group $\mathbb{Z}_{P-1}$.** Let us note the following structural fact about $\mathbb{Z}_P^*$ before proceeding further. These two groups are isomorphic with an isomorphism $\phi$ that maps $x \in \mathbb{Z}_{P-1}$ to $g^x \in \mathbb{Z}_P^*$. In particular, consider

$$\phi(x) = g^x \pmod{P}$$

we have $\phi(x+y) = \phi(x)\cdot\phi(y)$.

The isomorphism is efficently computable in the forward direction (exponentiation, using the repeated squaring algorithm) but not known to be efficiently computable in the reverse direction. The latter is the **discrete logarithm problem**.

**Primes, Primality Testing and Generating Random Primes.** The prime number theorem tells us that there are sufficiently many prime numbers. In particular, letting $\pi(N)$ denote the number of prime numbers less than $N$, we know that

$$\pi(N) = \Omega(N/\log N)$$

Thus, if you pick a random number smaller than $N$, with probability $1/\log N$ (which is 1/polynomial in the bit-length of the numbers in question) you have a prime number at hand.

The next question is how to recognize that a given number is prime. This has been the subject of extensive research in computational number theory with many polynomial-time algorithms, culminating with the deterministic polynomial-time primality testing algorithm of Agrawal, Kayal and Saxena (AKS) in 2002.

These two facts put together tell us how to generate a random $n$-bit prime number — just pick a random number less than $2^n$ and test if it is prime. In expected $n$ iterations of this procedure, you will find a $n$-bit prime number, even a random one at that.

**How to tell if a given $g$ is a generator of $\mathbb{Z}_P^*$?** We know that $g^{P-1} = 1 \pmod{P}$ and we want to check if there is some smaller power of $g$ that equals 1. We also know (by Lagrange) that any such power has to be a divisor of $P-1$. However, there are a large number of divisors of $P-1$, roughly $P^{1/\log\log P}$ which is not polynomial (in $\log P$.) It turns out, however, that you do not need to check all divisors, but rather only the *terminal* divisors.

More precisely, let $P-1 = \prod_i q_i^{\alpha_i}$ be the prime factorization of $P-1$. Then, the following algorithm works, on input $g$ and the prime factorization of $P-1$:

1. For each $i$, check if $g^{(P-1)/q_i} = q \pmod{P}$. If yes, say "not a generator" and otherwise say "generator".

That's nice. But can one tell if $g$ is a generator given only $g$ and $P$ (as opposed to the prime factorization of $P-1$ which is in general hard to compute)? We don't know, so we have to find a way around it. There are two solutions:

*Solution 1.* Pick $P = 2Q+1$ where $Q$ is prime. Such primes are called safe primes, and $Q$ is called a Sophie-Germain prime after the famous mathematician. While there are infinitely many primes, it has only been conjectured that there are infinitely many Sophie-Germain primes. This remains unproven.

*Solution 2.* Pick a random $P$ *together with its prime factorization.* This, it turns out, can be done due to a clever algorithm of Kalai. (reference on the webpage)

## 5 One-way Functions, PRGs and PRFs from Discrete Logarithms

A one-way function is a function $f$ that is easy to compute but hard to invert *on average*. A formal definition will come in later lectures, but for now, let us present an informal candidate.

$$f(P, g, x) = (P, g, g^x \pmod{P})$$

Computing this function can be done in time polynomial in the input length. However, inverting is the discrete logarithm problem (defined formally below) which is conjectured to be hard.

**Discrete Log Assumption (DLOG): For a random $n$-bit prime $P$ and random generator $g$ of $\mathbb{Z}_P^*$, and a random $x \in \mathbb{Z}_{P-1}$, there is no polynomial (in $n$) time algorithm that computes $x$ given $P, g, g^x \pmod{P}$.**

In fact, this is not just a one-way function, but can also be made into a family of one-way *permutations*. More on that later. As we will see in a couple of lectures, one-way permutations can be used to build pseudo-random generators; and as we saw already, pseudorandom generators can be used to build pseudorandom functions and stateless secret-key encryption and authentication. So, we can do all the crypto we saw so far based on the hardness of the discrete logarithm problem.

However, going via this route may not be the most efficient. So, we will look at related problems and try to build more efficient PRGs and PRFs.

### 5.1 The Diffie-Hellman Problems

Given $g^x$ and $g^y$ mod $P$, you can compute $g^{x+y} = g^x \cdot g^y \pmod{P}$, but can you compute $g^{xy} \pmod{P}$? If you can compute discrete logarithms, then you can compute $x$ from $g^x$, and raise $g^y$ to $x$ to get $(g^y)^x = g^{xy} \pmod{P}$. But discrete log is hard, so this isn't an efficient way to solve the problem.

Indeed, this problem, called the computational Diffie-Hellman (CDH) problem, appears to be computationally hard, in fact as hard as computing discrete logarithms!

**Computational Diffie-Hellman Assumption: For a random $n$-bit prime $P$ and random generator $g$ of $\mathbb{Z}_P^*$, and random $x, y \in \mathbb{Z}_{P-1}$, there is no polynomial (in $n$) time algorithm that computes $g^{xy} \pmod{P}$ given $P, g, g^x \pmod{P}, g^y \pmod{P}$.**

Moreover, it appears hard to even tell if you are given the right answer or not! That is, given $P, g, g^x, g^y$, it appears to be hard to distinguish between the right answer $g^{xy} \pmod{P}$ versus a random number $u$ mod $P$. The assumption that this decisional problem is hard is called the *decisional Diffie-Hellman* (DDH) assumption.

> **Decisional Diffie-Hellman Assumption: For a random $n$-bit prime $P$ and random generator $g$ of $\mathbb{Z}_P^*$, and random $x, y \in \mathbb{Z}_{P-1}$ and a random number $u \in \mathbb{Z}_P^*$, there is no polynomial (in $n$) time algorithm that distinguishes between $(P, g, g^x \pmod{P}, g^y \pmod{P}, g^{xy} \pmod{P})$ and $(P, g, g^x \pmod{P}, g^y \pmod{P}, u \pmod{P})$.**

We know that **DLOG $\rightarrow$ CDH $\rightarrow$ DDH** but no implications are known in the reverse directions.

**PRG from DDH.** Here is a candidate PRG whose pseudorandomness follows from the DDH assumption:

$$G(P, g, x, y) = (P, g, g^x, g^y, g^{xy})$$

Indeed, this expands two group elements into three. We can also speak about a family of pseudorandom generators which turns out to be more convenient. The family is indexed by $P$ and $g$ and works as follows:

$$G_{P,g}(x, y) = (g^x, g^y, g^{xy})$$

The security definition now says that given a random function chosen from the family $\{G_{P,g} : P \text{ prime and } g \text{ a generator}\}$, it is hard to distinguish between $g^x, g^y, g^{xy}$ and a sequence of three random group elements.

**PRF from DDH.** One could start from the PRG above and construct a PRF using the GGM construction. But there are more direct ways to do this by exploiting the number-theoretic structure in the function.

Here is a candidate PRF due to Naor and Reingold whose pseudorandomness follows from the DDH assumption. The key is a vector $\vec{x} = (x_0, x_1, \ldots, x_\ell)$ and the public description of the family if $P, g$. The PRF is defined as:

$$F_{P,g,\vec{x}}(a) = g^{x_0 \cdot \prod_{i=1}^{\ell} x_i} \pmod{P}$$

In other words, it computes a subset product in the exponent.