0 Review of Concepts & Notation

The material in this section is meant as a review. Despite that, **many students report that they find this review useful** for the rest of the book.

0.1 Logs & Exponents

You probably learned (and then forgot) these identities in middle school or high school:

$$(x^a)(x^b) = x^{a+b}$$
$$(x^a)^b = x^{ab}$$
$$\log_x(ab) = \log_x a + \log_x b$$
$$a \log_x b = \log_x(b^a)$$

Well, it's time to get reacquainted with them again.

In particular, **never ever** write $(x^a)(x^b) = x^{ab}$. If you write this, your cryptography instructor will realize that life is too short, immediately resign from teaching, and join a traveling circus. But not before changing your grade in the course to a zero.

0.2 Modular Arithmetic

We write the set of integers as:

$$\mathbb{Z} \overset{\text{def}}{=} \{\ldots, -2, -1, 0, 1, 2, \ldots\},$$

and the set of natural numbers (nonnegative integers) as:

$$\mathbb{N} \overset{\text{def}}{=} \{0, 1, 2, \ldots\}.$$

Note that 0 is considered a natural number.

Definition 0.1 *For $x, n \in \mathbb{Z}$, we say that n* ***divides*** *x (or x* ***is a multiple of*** *n), and write $n \mid x$, if there exists an integer k such that $x = kn$.*

Remember that the definitions apply to both positive and negative numbers (and to zero). We generally only care about this definition in the case where n is positive, but it is common to consider both positive and negative values of x.

Example *7 divides 84 because we can write $84 = 12 \cdot 7$.*
7 divides 0 because we can write $0 = 0 \cdot 7$.
7 divides -77 because we can write $-77 = (-11) \cdot 7$.
-7 divides 42 because we can write $42 = (-6) \cdot (-7)$.
1 divides every integer (so does -1). The only integer that 0 divides is itself.

Definition 0.2 (% operator) *Let n be a positive integer, and let a be any integer. The expression $a \,\%\, n$ (usually read as "a **mod** n") represents the remainder after dividing a by n. More formally, $a \,\%\, n$ is the **unique** $r \in \{0, \ldots, n-1\}$ such that $n \mid (a - r)$.*[1]

Pay special attention to the fact that $a \,\%\, n$ is always a *nonnegative* number, even if a is negative. A good way to remember how this works is:

$$\boxed{a \text{ is } (a \,\%\, n) \text{ more than a multiple of } n.}$$

Example $21 \,\%\, 7 = 0$ *because* $21 = 3 \cdot 7 + \underline{0}$.
$20 \,\%\, 7 = 6$ *because* $20 = 2 \cdot 7 + \underline{6}$.
$-20 \,\%\, 7 = 1$ *because* $-20 = (-3) \cdot 7 + \underline{1}$. *($-20$ is one more than a multiple of 7.)*
$-1 \,\%\, 7 = 6$ *because* $-1 = (-1) \cdot 7 + \underline{6}$.

Unfortunately, some programming languages define % for negative numbers as $(-a) \,\%\, n = -(a \,\%\, n)$, so they would define $-20 \,\%\, 7$ to be $-(20 \,\%\, 7) = -6$. This is madness, and it's about time we stood up to these programming language designers and smashed them over the head with some mathematical truth! For now, if you are using some programming environment to play around with the concepts in the class, be sure to check whether it defines % in the correct way.

Definition 0.3 ($\mathbb{Z}_n$) *For positive n, we write $\mathbb{Z}_n \stackrel{\text{def}}{=} \{0, \ldots, n-1\}$ to denote the set of **integers modulo n**. These are the possible remainders one obtains by dividing by n.*[2]

Definition 0.4 ($\equiv_n$) *For positive n, we say that integers a and b are **congruent modulo** n, and write $a \equiv_n b$, if $n \mid (a - b)$. An alternative definition is that $a \equiv_n b$ if and only if $a \,\%\, n = b \,\%\, n$.*

You'll be in a better position to succeed in this class if you can understand the (subtle) distinction between $a \equiv_n b$ and $a = b \,\%\, n$:

$a \equiv_n b$: In this expression, a and b can be integers of any size, and any sign. The left and right side have a certain relationship modulo n.

$a = b \,\%\, n$: This expression says that two integers are equal. The "=" rather than "≡" is your clue that the expression refers to equality over the integers. "$b \,\%\, n$" on the right-hand side is an operation performed on two integers that returns an integer result. The result of $b \,\%\, n$ is an integer in the range $\{0, \ldots, n-1\}$.

Example *"$99 \equiv_{10} 19$" is true. Applying the definition, you can see that 10 divides $99 - 19$.*
On the other hand, "$99 = 19 \,\%\, 10$" is false. The right-hand side evaluates to the integer 9, but 99 and 9 are different integers.

[1] The fact that only one value of r has this property is a standard fact proven in most introductory courses on discrete math.

[2] Mathematicians may recoil at this definition in two ways: (1) the fact that we call it $\mathbb{Z}_n$ and not $\mathbb{Z}/(n\mathbb{Z})$; and (2) the fact that we say that this set contains integers rather than congruence classes of integers. If you appreciate the distinction about congruence classes, then you will easily be able to mentally translate from the style in this book; and if you don't appreciate the distinction, there should not be any case where it makes a difference.

In short, expressions like $a \equiv_n b$ make sense for any a, b (including negative!), but expressions like $a = b \,\%\, n$ make sense only if $a \in \mathbb{Z}_n$.

Most other textbooks will use notation "$a \equiv b \pmod{n}$" instead of "$a \equiv_n b$." I dislike this notation because "(mod n)" is easily mistaken for an operation or action that only affects the right-hand side, when in reality it is like an adverb that modifies the *entire* expression $a \equiv b$. Even though $\equiv_n$ is a bit weird, I think the weirdness is worth it.

If $d \mid x$ and $d \mid y$, then d is a **common divisor** of x and y. The largest possible such d is called the **greatest common divisor (GCD),** denoted $\gcd(x, y)$. If $\gcd(x, y) = 1$, then we say that x and y are **relatively prime**. The oldest "algorithm" is the recursive process that Euclid described for computing GCDs (ca. 300 BCE):

```
GCD(x, y): // Euclid's algorithm
  if y = 0 then return x
  else return GCD(y, x % y)
```

Tips & Tricks

You may often be faced with some complicated expression and asked to find the value of that expression mod n. This usually means: find the unique value in $\mathbb{Z}_n$ that is congruent to the result. The straightforward way to do this is to first compute the result *over the integers*, and then reduce the answer mod n (*i.e.*, with the $\%\, n$ operator).

While this approach gives the correct answer (and is a good anchor for your understanding), it's usually advisable to **simplify intermediate values mod n.** Doing so will result in the same answer, but will usually be easier or faster to compute:

Example *We can evaluate the expression* $6 \cdot 7 \cdot 8 \cdot 9 \cdot 10 \,\%\, 11$ *without ever calculating that product over the integers, by using the following reasoning:*

$$\begin{aligned}
6 \cdot 7 \cdot 8 \cdot 9 \cdot 10 &= (42) \cdot 8 \cdot 9 \cdot 10 \\
&\equiv_{11} 9 \cdot 8 \cdot 9 \cdot 10 \\
&= (72) \cdot 9 \cdot 10 \\
&\equiv_{11} 6 \cdot 9 \cdot 10 \\
&= (54) \cdot 10 \\
&\equiv_{11} 10 \cdot 10 \\
&= 100 \\
&\equiv_{11} 1
\end{aligned}$$

In the steps that only work mod 11, we write "$\equiv_{11}$". We can write "$=$" when the step holds over the integers, although it wouldn't be wrong to write "$\equiv_{11}$" in those cases too. If two expressions represent the same integer*, then they surely represent values that are congruent mod 11.*

My advice is to simplify intermediate values mod n, but "simplify" doesn't always mean "reduce mod n with the $\%\, n$ operation." Sometimes an expression can by "simplified" by substituting a value with something congruent, but *not* in the range $\{0, \ldots, n-1\}$:

Example *I can compute $7^{500} \% 8$ in my head, by noticing that $7 \equiv_8 -1$ and simplifying thusly:*

$$7^{500} \equiv_8 (-1)^{500} = 1.$$

Similarly, I can compute $89^2 \% 99$ in my head, although I have not memorized the integer 89^2. *All I need to do is notice that $89 \equiv_{99} -10$ and compute this way:*

$$89^2 \equiv_{99} (-10)^2 = 100 \equiv_{99} 1$$

You can compute either of these examples the "hard way" to verify that these shortcuts lead to the correct answer.

Since addition, subtraction, and multiplication are defined over the integers (*i.e.*, adding/subtracting/multiplying integers always results in an integer), these kinds of tricks can be helpful.

On the other hand, dividing integers doesn't always result in an integer. Does it make sense to use division when working mod n, where the result always has to lie in $\mathbb{Z}_n$? We will answer this question later in the book.

0.3 Strings

We write $\{0, 1\}^n$ to denote the set of n-bit binary strings, and $\{0, 1\}^*$ to denote the set of all (finite-length) binary strings. When x is a string of bits, we write $|x|$ to denote the length (in bits) of that string, and we write $\overline{x}$ to denote the result of flipping every bit in x. When it's clear from context that we're talking about strings instead of numbers, we write 0^n and 1^n to denote strings of n zeroes and n ones, respectively (rather than the result of raising the *integers* 0 or 1 to the n power). As you might have noticed, I also try to use a different font and color for characters (including bits, anything that could be used to build a string through concatenation) vs. integers.

Definition 0.5 ($\oplus$, XOR) *When x and y are strings of the same length, we write $x \oplus y$ to denote the bitwise exclusive-or (XOR) of the two strings. The expression $x \oplus y$ is generally not defined when the strings are different lengths, but in rare occasions it is useful to consider the shorter string being padded with 0s. When that's the case, we must have an explicit convention about whether the shorter string is padded with leading 0s or trailing 0s.*

For example, $0011 \oplus 0101 = 0110$. The following facts about the XOR operation are frequently useful:

$x \oplus x = 000\cdots$	XOR'ing a string with itself results in zeroes.
$x \oplus 000\cdots = x$	XOR'ing with zeroes has no effect.
$x \oplus 111\cdots = \overline{x}$	XOR'ing with ones flips every bit.
$x \oplus y = y \oplus x$	XOR is symmetric.
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$	XOR is associative.

See if you can use these properties to derive the very useful fact below:

$$a = b \oplus c \iff b = a \oplus c \iff c = a \oplus b.$$

There are a few ways to think about XOR that may help you in this class:

- **Bit-flipping:** Note that XOR'ing a bit with `0` has no effect, while XOR'ing with `1` flips that bit. You can think of $x \oplus y$ as: "starting with x, flip the bits at all the positions where y has a `1`." So if y is all `1`'s, then $x \oplus y$ gives the bitwise-complement of x. If $y = \texttt{1010}\cdots$ then $x \oplus y$ means "(the result of) flipping every other bit in x."

  Many concepts in this course can be understood in terms of bit-flipping. For example, we might ask "what happens when I flip the first bit of x and send it into the algorithm?" This kind of question could also be phrased as "what happens when I send $x \oplus \texttt{1000}\cdots$ into the algorithm?" Usually there is nothing special about flipping just the first bit of a string, so it will often be quite reasonable to generalize the question as "what happens when I send $x \oplus y$ into the algorithm, for an arbitrary (not-all-zeroes) string y?"

- **Addition mod-2:** XOR is just addition mod 2 in every bit. This way of thinking about XOR helps to explain why "algebraic" things like $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ are true. They are true for addition, so they are true for XOR.

  This also might help you remember why $x \oplus x$ is all zeroes. If instead of XOR we used addition, we would surely write $x + x = 2x$. Since $2 \equiv_2 0$, we get that $2x$ is congruent to $0x = 0$.

Definition 0.6 (||, concatenation) *We write $x \| y$ to denote the result of **concatenating** x and y.*

0.4 Functions

Let X and Y be finite sets. A function $f : X \to Y$ is:

injective (1-to-1) if it maps distinct inputs to distinct outputs. Formally: $x \neq x' \Rightarrow f(x) \neq f(x')$. If there is an injective function from X to Y, then we must have $|Y| \geqslant |X|$.

surjective (onto) if every element in Y is a possible output of f. Formally: for all $y \in Y$ there exists an $x \in X$ with $f(x) = y$. If there is a surjective function from X to Y, then we must have $|Y| \leqslant |X|$.

bijective (1-to-1 correspondence) if f is both injective and surjective. If there is a bijective function from X to Y, then we must have $|X| = |Y|$.

0.5 Probability

Definition 0.7 (Distribution) *A **(discrete) probability distribution** over a set X of **outcomes** is usually written as a function "$\Pr$" that associates each outcome $x \in X$ with a probability $\Pr[x]$. We often say that the distribution **assigns** probability $\Pr[x]$ to outcome x.*

For each outcome $x \in X$, the probability distribution must satisfy the condition $0 \leqslant \Pr[x] \leqslant 1$. Additionally, the sum of all probabilities $\sum_{x \in X} \Pr[x]$ must equal 1.

Definition 0.8 (Uniform) *A special distribution is the **uniform distribution** over a finite set X, in which every $x \in X$ is assigned probability $\Pr[x] = 1/|X|$.*

We also extend the notation Pr to **events**, which are collections of outcomes. If you want to be formal, an event A is any subset of the possible outcomes, and its probability is defined to be $\Pr[A] = \sum_{x \in A} \Pr[x]$. We always simplify the notation slightly, so instead of writing $\Pr[\{x \mid x \text{ satisfies some condition}\}]$, we write Pr[condition].

Example *A 6-sided die has faces numbered* $\{1, 2, \dots, 6\}$. *Tossing the die (at least for a mathematician) induces a uniform distribution over the choice of face. Then* $\Pr[\textit{3 is rolled}] = 1/6$, *and* $\Pr[\textit{an odd number is rolled}] = 1/2$ *and* $\Pr[\textit{a prime is rolled}] = 1/2$.

Tips & Tricks

Knowing one of the probabilities $\Pr[A]$ and $\Pr[\neg A]$ (which is "the probability that A *doesn't* happen") tells you exactly what the other probability is, via the relationship

$$\Pr[A] = 1 - \Pr[\neg A].$$

This is one of the most basic facts about probability, but it can be surprisingly useful since one of $\Pr[A]$ and $\Pr[\neg A]$ is often much easier to calculate than the other. If you get stuck trying to come up with an expression for $\Pr[A]$, try working out an expression for $\Pr[\neg A]$ instead.

Example *I roll a six-sided die, six times. What is the probability that there is some repeated value? Let's think about all the ways of getting a repeated value. Well, two of the rolls could be 1, or three of rolls could be 1, or all of them could be 1, two of them could be 1 and the rest could be 2, etc. Oh no, am I double-counting repeated 2s and repeated 1s? Uhh...*

An easier way to attack the problem is to realize that the probability we care about is actually $1 - \Pr[\textit{all 6 rolls are distinct}]$. *This complementary event (all 6 rolls distinct) happens exactly when the sequence of dice rolls spell out a permutation of* $\{1, \dots, 6\}$. *There are* $6! = 720$ *such permutations, out of* $6^6 = 46656$ *total possible outcomes. Hence, the answer to the question is*

$$1 - \frac{6!}{6^6} = 1 - \frac{720}{46656} = \frac{45936}{46656} \approx 0.9846$$

Another trick is one I like to call **setting breakpoints** on the universe. Imagine stopping the universe at a point where some random choices have happened, and others have not yet happened. This is best illustrated by example:

Example *A classic question asks: when rolling two 6-sided dice what is the probability that the dice match? Here is a standard (and totally correct way) to answer the question:*

> *When rolling two 6-sided dice, there are* $6^2 = 36$ *total outcomes (a pair of numbers), so each has probability* $1/36$ *under a uniform distribution. There are 6 outcomes that make the dice match: both dice 1, both dice 2, both dice 3, and so on. Therefore, the probability of rolling matching dice is* $6/36 = 1/6$.

A different way to arrive at the answer goes like this:

> *Imagine I roll the dice one after another, and I pause the universe (set a breakpoint) after rolling the first die but before rolling the second one. The universe has already decided the result of the first die, so let's call that value d. The dice will match only if the second roll comes up d. Rolling d on the second die (indeed, rolling any particular value) happens with probability* $1/6$.

This technique of setting breakpoints is simple but powerful and frequently useful. Some other closely related tricks are: (1) postponing a random choice until the last possible moment, just before its result is used for the first time, and (2) switching the relative order of independent random choices.

Precise Terminology

It is tempting in this course to say things like "x is a random string." But a statement like this is sloppy on several accounts.

First, is 42 a random number? Is "heads" a random coin? What is even being asked by these questions? Being "random" is not a property of an *outcome* (like a number or a side of a coin) but a property of the *process* that generates an outcome.[3] Instead of saying "x is a random string," it's much more precise to say "x was chosen randomly."

Second, usually when we use the word "random," we don't mean any old probability distribution. We usually mean to refer to the *uniform distribution.* Instead of saying "x was chosen randomly," it's much more precise to say "x was chosen uniformly" (assuming that really *is* what you mean).

Every cryptographer I know (yes, even your dear author) says things like "x is a random string" all the time to mean "x was chosen uniformly [from some set of strings]." Usually the meaning is clear from context, at least to the other cryptographers in the room. But all of us could benefit by having better habits about this sloppy language. Students especially will benefit by internalizing the fact that **randomness is a property of the *process*, not of the individual outcome.**

0.6 Notation in Pseudocode

We'll often describe algorithms/processes using pseudocode. In doing so, we will use several different operators whose meanings might be easily confused:

$\leftarrow$ When $\mathcal{D}$ is a probability distribution, we write "$x \leftarrow \mathcal{D}$" to mean "sample x according to the distribution $\mathcal{D}$."

If $\mathcal{A}$ is an algorithm that takes input and also makes some internal random choices, then it is natural to think of its output $\mathcal{A}(y)$ as a distribution — possibly a different distribution for each input y. Then we write "$x \leftarrow \mathcal{A}(y)$" to mean the natural thing: "run $\mathcal{A}$ on input y and assign the output to x."

[3] There is something called Kolmogorov complexity that can actually give coherent meaning to statements like "x is a random string." But Kolmogorov complexity has no relevance to this book. The statement "x is a random string" remains meaningless with respect to the usual probability-distribution sense of the word "random."

We overload the "$\leftarrow$" notation slightly, writing "$x \leftarrow X$" when X is a *finite set* to mean that x is sampled from the *uniform distribution* over X.

$:=$ We write $x := y$ for assignments to variables: "take the value of expression y and assign it to variable x."

$\overset{?}{=}$ We write comparisons as $\overset{?}{=}$ (analogous to "==" in your favorite programming language). So $x \overset{?}{=} y$ doesn't modify x (or y), but rather it is an expression which returns `true` if x and y are equal.

You will often see this notation in the conditional part of an if-statement, but also in return statements as well. The following two snippets are equivalent:

return $x \overset{?}{=} y$	$\Leftrightarrow$	if $x \overset{?}{=} y$:
		  return `true`
		else:
		  return `false`

In a similar way, we write $x \overset{?}{\in} S$ as an expression that evaluates to true if x is in the set S.

Subroutine conventions

We'll use mathematical notation to define the *types* of subroutine arguments:

FOO $(x \in \{0,1\}^*)$: $\ldots$	means	"`void foo(string x) { ... }`"

0.7 Asymptotics (Big-O)

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We characterize the asymptotic growth of f in the following ways:

$$\begin{aligned}
f(n) \text{ is } O(g(n)) &\overset{\text{def}}{\Leftrightarrow} \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \\
&\Leftrightarrow \exists c > 0 : \text{for all but finitely many } n : f(n) < c \cdot g(n) \\
f(n) \text{ is } \Omega(g(n)) &\overset{\text{def}}{\Leftrightarrow} \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \\
&\Leftrightarrow \exists c > 0 : \text{for all but finitely many } n : f(n) > c \cdot g(n) \\
f(n) \text{ is } \Theta(g(n)) &\overset{\text{def}}{\Leftrightarrow} f(n) \text{ is } O(g(n)) \text{ and } f(n) \text{ is } \Omega(g(n)) \\
&\Leftrightarrow 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \\
&\Leftrightarrow \exists c_1, c_2 > 0 : \text{for all but finitely many } n : \\
&\qquad\qquad c_1 \cdot g(n) < f(n) < c_2 \cdot g(n)
\end{aligned}$$

Exercises

0.1. Rewrite each of these expressions as something of the form 2^x.

(a) $(2^n)^n =$??
(b) $2^n + 2^n =$??
(c) $(2^n)(2^n) =$??
(d) $(2^n)/2 =$??
(e) $\sqrt{2^n} =$??
(f) $(2^n)^2 =$??

0.2. (a) What is $0 + 1 + 2 + \cdots + (n-2) + (n-1) \% n$, when n is an odd integer? Prove your answer!

(b) What is $0 + 1 + 2 + \cdots + (n-2) + (n-1) \% n$, when n is even? Prove your answer!

0.3. What is $(-99) \% 10$?

0.4. Without using a calculator, what are the last two digits of 357998^6?

0.5. Without using a calculator, what is $1000! \% 427$? (That's not me being excited about the number one thousand, it's one thousand *factorial!*)

0.6. Which values $x \in \mathbb{Z}_{11}$ satisfy $x^2 \equiv_{11} 5$? Which satisfy $x^2 \equiv_{11} 6$?

0.7. What is the result of XOR'ing every n bit string? For example, the expression below is the XOR of every 5-bit string:

$$\texttt{00000} \oplus \texttt{00001} \oplus \texttt{00010} \oplus \texttt{00011} \oplus \cdots \oplus \texttt{11110} \oplus \texttt{11111}$$

Give a convincing justification of your answer.

0.8. Consider rolling several d-sided dice, where the sides are labeled $\{0, \ldots, d-1\}$.

(a) When rolling two of these dice, what is the probability of rolling *snake-eyes* (a pair of 1s)?

(b) When rolling two of these dice, what is the probability that they *don't* match?

(c) When rolling **three** of these dice, what is the probability that they all match?

(d) When rolling three of these dice, what is the probability that they **don't** all match (including the case where two match)?

(e) When rolling three of these dice, what is the probability that at least two of them match (including the case where all three match)?

(f) When rolling three of these dice, what is the probability of seeing at least one 0?

0.9. When rolling two 6-sided dice, there is some probability of rolling snake-eyes (two 1s). You determined this probability in the previous problem. In some game, I roll both dice each time it is my turn. What is the smallest value t such that:

$$\Pr[\text{I have rolled snake-eyes in at least one of my first } t \text{ turns}] \geqslant 0.5?$$

In other words, how many turns until my probability of getting snake-eyes exceeds 50%?