

Class 1. Handouts. Note. The course web page is at www.cs.umb.edu/cs320 (that includes Math cs320) Text: Discrete Mathematics and its Applications 6th Edition, Rosen, McGraw Hill.

Discrete Math typically reasons about finite sets of objects (graphs) or infinite sets that are fully distinct (integers), rather than "continuous" sets (real numbers, continuous and differentiable functions).

We start with logic and "propositional combinations", the basis of mathematical reasoning and proofs.

Def. A *proposition* is a declarative statement that is either true or false. We say that the *truth value* of a proposition is either true (T) or false (F).

Examples:

- o "What time is it?" IS NOT a proposition, since it is neither true nor false.
- o "Washington, D.C. is the capital of the U.S." IS a proposition and it is true (T).
- o " $1 + 1 = 2$." IS a proposition and it is true (T).
- o " $x + 1 = 2$." IS NOT a proposition, neither true nor false (depends on value of x).
- o " $2 + 2 = 3$." IS a proposition and it is false (F).
- o " $x < y$ if and only if $y > x$." IS a (compound) proposition, and it is true (T).

But this statement's truth is based on our understanding of the context: x and y are numbers and we are willing to assign tentative truth values T or F to $x < y$, and note that whichever truth value we assign will be the same as the one assigned to $y > x$. As we indicated with example 3 above, $x + 1 = 2$ is not a statement, and neither is $x < y$. But either one can be a statement *variable*, to which T or F can be assigned under varying conditions.

Def. A *propositional variable* (or *statement variable*) is typically represented by letters p, q, r , or s , just as x and y variables represent numbers. Each propositional variable such as p is assumed to have either T or F value, and multiple variables, when used together, are assumed to take on T and F values in all possible combinations: (p,q) will have values (T,T), (T,F), (F,T) and (F,F).

We now define ways to create *compound propositions*, formed from existing propositions using logical operations and parentheses (). Here are some logical operations.

Unary Operator: Negation (NOT), Symbol: \neg

Binary Operators:

Conjunction (AND), Symbol: \wedge

Disjunction (OR), Symbol: \vee (also called Inclusive Or)

Exclusive Or (XOR), Symbol: \oplus

Implication (if – then), Symbol: \rightarrow

Biconditional (if and only if), Symbol: \leftrightarrow

Def. 1. If p is a proposition, the negation of p , $\neg p$, can be stated as: "It is not the case that p ."

The proposition p can be read as "not p ". The truth value of $\neg p$ is the opposite of the truth value of p , T when p has value F and F when p has value T. We can represent truth values of compound propositions in a Truth Table such as the following.

Truth Table 1

Negation

p	$\neg p$
T	F
F	T

Truth Table 2

Conjunction

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

We define $p \wedge q$ to be T exactly when p is T and q is T, but F otherwise. For $p \wedge q$ read p AND q.

Truth Table 3

Disjunction

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

We define $p \vee q$ to be T exactly when p is T or q is T or both, but F otherwise. For $p \vee q$ read p or q, or for emphasis, p inclusive or q. (See "exclusive or" below.)

Truth Table 4

Exclusive Or: Was T for $p = T, q = T$, should be F

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

We define $p \oplus q$ to be T exactly when p is T or q is T but NOT BOTH, or F otherwise. For $p \oplus q$ read p xor q, or for emphasis, p exclusive or q.

Truth Table 5

Implication

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

We define $p \rightarrow q$ to be T exactly when q is T or p is F, but F otherwise. For $p \rightarrow q$ read p implies q. This is also called a *conditional statement*. This is an odd definition, so we need to justify it.

If we take $p \rightarrow q$ to mean, "if p is true then q is true", then that certainly takes care of the first two rows of Truth Table 5, where p is T in both cases and $p \rightarrow q$ is T when q is T and F when q is F. But why do we want to include "if p is F then p implies any truth value, T or F"?

Assume we use the following False statement S1 to reason about arithmetic: $1 = 2$

By standard arithmetical operations we can derive equality of any two numbers from this false statement. E.g., $27.1 = 27.2$. Divide both sides of S1 by 10 and get $.1 = .2$, and add the true statement $27 = 27$.

This is just one example, but you can derive many more: $2 = 3$ (add $1 = 1$) or $1 = 3$ (double $1 = 2$ to get $2 = 4$ and subtract $1 = 1$), and so on. From a false statement in many logical systems, we can derive all T and F statements (i.e., one F statement implies all other F and T statements).

Truth Table 6

If and only if

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

We define $p \leftrightarrow q$ to be T exactly when p and q are both T or p and q are both F. For $p \leftrightarrow q$ read p if and only if q. This is also called a *biconditional statement*.

We can show that $p \leftrightarrow q$ is implied by the pair of propositions $p \rightarrow q$ and $q \rightarrow p$; not only that but $p \rightarrow q$ and $q \rightarrow p$ together imply that $p \leftrightarrow q$.

Rewriting what we have just said as a compound logical statement we have:

$$(p \leftrightarrow q) \leftrightarrow (p \rightarrow q \wedge q \rightarrow p)$$

and we claim that this statement is always true. A statement that is always true is called a *tautology*! We can prove this is always true by using a Truth Table.

Truth Table: Proof of tautology

p	q	$p \leftrightarrow q$	$p \rightarrow q$	$q \rightarrow p$	$(p \rightarrow q) \wedge (q \rightarrow p)$	$(p \leftrightarrow q) \leftrightarrow ((p \rightarrow q) \wedge (q \rightarrow p))$
T	T	T	T	T	T	T
T	F	F	F	T	F	T
F	T	F	T	F	F	T
F	F	T	T	T	T	T

Why do we use parentheses in these expressions? When an expression is enclosed in parentheses the expression is evaluated before any operation outside the parentheses. For example, what is the value of this arithmetic expression? $7 \cdot 5 + 3$

Of course it is 38, because multiplication happens before addition when there are no parentheses. This is called precedence of operations! Times (\cdot) has a higher precedence than Plus ($+$). If we wanted addition to happen first, we would have to write: $7 \cdot (5 + 3)$ to get 56!

Here is the precedence of Logical operations.

Table 8

Precedence of

Logical Operators (Note \oplus has been placed in precedence position, not in Text.)

Operator	Precedence
\neg	1
\wedge	2
\oplus	3
\vee	4
\rightarrow	5
\leftrightarrow	6

Are all parentheses in the truth table above needed? What if leave them out in $(p \rightarrow q) \wedge (q \rightarrow p)$? Then it would be as if we wrote $p \rightarrow (q \wedge q) \rightarrow p$, which doesn't look like a proper expression. If p is F and q is T, then we have $F \rightarrow T \rightarrow F$. This is not associative: $(F \rightarrow T) \rightarrow F = T \rightarrow F = F$. But $F \rightarrow (T \rightarrow F) = F \rightarrow F = T$.

If $s \leftrightarrow t$ is a tautology, we write $s \Leftrightarrow t$ or $s \equiv t$. (Rosen uses \equiv and we will too from now on.)

But $s \leftrightarrow t$ is just a proposition, it can be T or F. $s \equiv t$ is more. It stands for $s \leftrightarrow t$ is a Tautology

More Example of Tautologies:

- $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$
- $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$

These two tautologies are known as De Morgan's laws. Find in text.

For an exercise, prove with Truth Tables.

A contradiction is a statement that is always false.

Examples:

- $r \wedge (\neg r)$
- $\neg(\neg(p \wedge q) \leftrightarrow (\neg p) \vee (\neg q))$

A negation of any tautology is a contradiction, and negation of any contradiction is a tautology.

Given an implication $p \rightarrow q$, we can read this as "p implies q" or "q is implied by p" or "p is a sufficient condition for q" or "q is a necessary condition for p" or "p only if q". We have three other implication defined in terms of $p \rightarrow q$.

Converse: $q \rightarrow p$; Inverse: $\neg p \rightarrow \neg q$; contrapositive: $\neg q \rightarrow \neg p$.

$p \rightarrow q$ and the contrapositive $\neg q \rightarrow \neg p$ are equivalent in meaning (\equiv)(in Text symb \Leftrightarrow)

The converse and inverse are also equivalent (the inverse is the contrapositive of the converse).

You should prove the following with truth tables; they can be used as rules of logic in proofs.

First: $p \rightarrow q \equiv \neg p \vee q$; It follows from this that: $\neg(p \rightarrow q) \equiv p \wedge \neg q$

Another: $(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$

Also: $((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$ is an important Tautology

Class 2.

Translating English to Logical Forms. Pg 11 of Text.

Example, "You can access the Internet from campus only if you are a computer science major or you are not a freshman." (We assume inclusive or.) "p only if q" means the same as $p \rightarrow q$.

Here's how to remember this. Note that "p if q" means the same as "if q then p", i.e.: $p \rightarrow q$, and "p if and only if q" means $p \leftrightarrow q$; recall that $((p \leftrightarrow q) \equiv ((q \rightarrow p) \wedge (p \rightarrow q)))$ (Table 8, Pg 25). Read this as "p if and only if q" means the same as "p if q" ($q \rightarrow p$) and "p only if q" ($p \rightarrow q$).

Thus if a stands for the phrase "You can access the Internet from campus", f for "You are a freshman" and c is "You are a computer science major", then the logical form of the sentence is:

$$a \rightarrow (c \vee \neg f)$$

Try this example as an exercise. "You cannot ride the roller coaster if you are under four feet tall unless you are at least sixteen years old." One way to do this is interpret it as: "if you are under four feet tall and under sixteen years old (negation of "you are at least sixteen years old") then you cannot ride the roller coaster.

Let s stand for the phrase "You are at least sixteen years old," r stand for "You can ride the roller coaster," and f stand for "you are at least four feet tall." Then the logical form of the sentence is:

$$\neg f \wedge \neg s \rightarrow \neg r$$

Note that the equivalent contrapositive is: $r \rightarrow f \vee s$. This makes the original statement more clear: to ride the roller coaster, this implies you must be either four feet tall OR sixteen years of age; thus you cannot ride the roller coaster if you are under four feet tall UNLESS you're sixteen.

System Specifications.

A (partial) system specification written by a design engineer should unambiguously describe at least one state) of a system during operation. Here is a set of specification requirements.

"The router can send packets to the edge system only if it supports the new address space."

"For the router to support the new address space, the latest software release must be installed."

"The router can send packets to the edge system if the new software release is installed."

"The router does not support the new address space."

We wish to check if this set of specifications requirements is *Consistent*, meaning that some possible configuration will make all these requirements true!

Let s be "The router can send packets to the edge system." Let a be "The router supports the new address space." Let r be "The latest software release is installed." Then the specification requirements are: $s \rightarrow a$, $a \rightarrow r$, $r \rightarrow s$. and $\neg a$.

For the specification requirements to be Consistent, all of these requirements must be shown to be True for some choice of T and F for the three propositions. One way to do this is to create a Truth table with these three proposition variables and four requirements and see if all four are True on some line of the Truth Table.

But we can usually show it is consistent without truth tables if we reason about it. For $\neg a$ to be T, a must be F. But then for $s \rightarrow a$ to be T, s must be F as well. And now for $r \rightarrow s$ to be true r must be F. With all three F, the requirements all hold.

This set of requirements is quite weird, since the fact that the router does not support the new address space means NOTHING ELSE WILL WORK. The latest software release cannot be installed, the router will not send packets to the edge system. What happens if we assume the router DOES support the new address space instead (this is the only change). Then we have:

$s \rightarrow a, a \rightarrow r, r \rightarrow s.$ and $a.$

We assume all these are true. Since a is true and $a \rightarrow r$ is true, r must be true; but since $r \rightarrow s$ is true, s must be true as well, and then $s \rightarrow a$ is also true. This shows that the set of requirements is consistent across a change of state: from $\neg a$, where the system has no working capabilities, to a , where the system has all capabilities. This is more impressive.

1.3 Predicates and Quantifiers.

The forms of logic expressions we have seen so far cannot adequately express the meaning of certain statements in mathematics and natural language.

For example, suppose we know names for a set A representing all computers on the university network. Given a proposition $P(x)$, "a specific computer x in the set A is functioning properly", how would we say: "Every computer on the university network is functioning properly"?

We define a *Propositional Function* $P(x)$ or a *Condition*, where x ranges over A , with the property that $P(x)$ is T or F for each $x \in A$, and x is functioning properly if and only if $P(x)$ is T.

The set A is called the *domain* of $P(x)$, and the set T_P of all elements x of A for which $P(x)$ is true is called the truth set of $P(x)$: $T_P = \{x \mid x \in A \text{ and } P(x) \text{ is T}\}$ or, if we assume the set A is understood and a condition of the right of " \mid " must be T for the element to exist, $T_P = \{x \mid P(x)\}$.

Frequently, when A is some set of numbers, the condition $P(x)$ has the form of an equation or inequality involving the variable x .

Quantifiers.

We define the *Universal Quantifier* to provide a notation $\forall x P(x)$, which means, "For all x (in the assumed domain A), $P(x)$ is true." In the Example given above this would mean, "All computers on the university network are functioning properly." Note that $\forall x P(x)$ itself can be either T or F.

We read $\forall x P(x)$ as "for all x , $P(x)$ " or "for every x , $P(x)$ ". Notice that $P(x)$ itself is a condition that is meaningful on multiple elements of a domain, and cannot have a T or F value: it might be T or F on different elements x . However $\forall x P(x)$ itself is saying something about all elements in the domain and DOES have a Truth value. We can show that $\forall x P(x)$ is false by providing a counterexample, an element a in the domain A such that $P(a)$ is false.

If there were more than one possible domain for elements to be quantified, say A_1 and A_2 , we could write the universal quantifier as: $(\forall x \in A_2) P(x)$.

Example. Consider the domain A of positive integers less than 10, i.e., $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. If $P(x)$ is the condition $x+3 \leq 12$, then $\forall x P(x)$ is True. But if $Q(x)$ is the condition $x+9 \leq 12$, then $\forall x Q(x)$ is False (certainly for $x = 9$, $x + 9$ is greater than 12).

Observe that $\forall x P(x)$ is false (for Domain A) if and only if there is at least one x in A such that $P(x)$ is false. This is what is called a *counterexample*!

Note that if elements of a domain A can be listed as x_1, x_2, \dots, x_n , then universal quantification in that range, $\forall x P(x)$, is the same as the conjunction: $P(x_1) \wedge P(x_2) \wedge \dots \wedge P(x_n)$.

Example. Let the range A be positive integers not exceeding four and $P(x)$ be $x^2 \leq 10$. Then $\forall x P(x)$ is identical in meaning to $P(1) \wedge P(2) \wedge P(3) \wedge P(4)$, but since $P(4)$ stands for $4^2 < 10$ and $4^2 = 16$, $P(4)$ is false and so is $\forall x P(x)$.

We define the *Existential Quantifier* to provide a notation $\exists x P(x)$, which means, "There exists (at least one) x (in the given domain A), for which $P(x)$ is true." Note once again that $P(x)$ is a conditional on multiple elements of a domain, and cannot have a T or F value. But $\exists x P(x)$ is potentially saying something about all elements in the domain and DOES have a Truth value.

Why is $\exists x P(x)$ potentially saying something about all elements in the domain? Because if there is only one x in a domain for which $P(x)$ is T, it might be the last one examined, and if $\exists x P(x)$ is F, we would have to be sure that $\neg P(x)$ is true for every x in the domain.

Example: if $G(x)$ means "x is a genius" and $P(x)$ means "x is a UMB professor" (which defines our domain A), then what does $\exists x (P(x) \wedge G(x))$ mean? It means: "There is an x such that x is a UMB professor and x is a genius" or "At least one UMB professor is a genius."

Example: if $G(x)$ means "x is a genius" and $S(x)$ means "x is a UMB student", what does $\forall x (S(x) \rightarrow G(x))$ mean? Well, it depends on the Domain A we're working with. If A is the set of all UMB students, then it means all UMB students are geniuses (and it may be T or F, of course -- but probably F).

If the Domain A is all people in the world not UMB students, then any x for which $S(x)$ is F will mean $S(x) \rightarrow G(x)$ is certainly true, so it implies nothing about people in the Domain outside of UMB students and has the same meaning as when the Domain A is the set of all UMB students.

But if the Domain A is the set of all UMB students who are about to graduate with a perfect grade point average, then $\forall x (S(x) \rightarrow G(x))$ is much more likely to be true than when it contains all UMB students. Knowing the Domain A is crucial in determining meaning!

What does $\forall x \exists y (x + y = 320)$ mean? For every x , there exists a y such that $x + y = 320$. A Domain A for which this is true is the set of all integers (positive, negative, and zero). A Domain for which it is NOT true is the set of non-negative integers: if $x = 321$, no y gives $x + y = 320$.

On the inside-front cover of Rosen, the set of non-negative (natural) integers is represented by \mathbf{N} , and the set of all integers (positive, negative or zero) is represented by \mathbf{Z} (Definition, pg. 112).

Recall that we have a counterexample to $\forall x P(x)$ for Domain A, when there is an element a in A such that $P(a)$ is false. Such a counterexample means that $\forall x P(x)$ is FALSE! This is one of *de Morgan's laws for Quantifiers* (Table 2, Pg 41), written logically as:

$\neg(\forall x P(x)) \equiv \exists x (\neg P(x))$. The two are logically equivalent, so each implies the other.

Similarly we can say that it is false that $\exists x P(x)$ if and only if, for all values x , $P(x)$ is FALSE. This is the second (and last) of *de Morgan's laws for Quantifiers*, written logically as:

$$\neg(\exists x P(x)) \equiv \forall x (\neg P(x)).$$

See Table 2 in Section 1.3 in Rosen (pg. 41) for de Morgan's Laws for Quantifiers.

I'm jumping ahead a little bit here to **Section 1.4**.

Now recall above $\forall x \exists y (x + y = 320)$ means for every x , there exists a y such that $x + y = 320$. The proper way to read this is $\forall x (\exists y (x + y = 320))$, i.e., for all x the following is true: there exists a y such that $x + y = 320$. Now how would I negate this?

$\neg(\forall x (\exists y (x + y = 320)))$. Applying \neg to the outer $\forall x$ (statement) gives us: $\exists x \neg(\text{statement})$ or in other words: $\exists x \neg(\exists y (x + y = 320))$; now applying \neg to the inner $\exists y (x + y = 320)$ statement gives us $\forall y \neg(x + y = 320)$. Of course $\neg(x + y = 320)$ can be simplified to $x + y \neq 320$ so we end up with $\exists x \forall y (x + y \neq 320)$.

Just work from the outside, in. Recall for $x, y \in \mathbf{N}$ (non-negative integers), it is NOT TRUE that $\forall x (\exists y (x + y = 320))$, in other words $\neg(\forall x (\exists y (x + y = 320))) \equiv \exists x \neg(\exists y (x + y = 320))$, i.e., there is an $x \in \mathbf{N}$ ($x = 321$) such that there does not exist a $y \in \mathbf{N}$ such that $x + y = 320$ (since $y = -1$, which would make $x + y = 320$, is not in \mathbf{N}).

Continuing, $\exists x \neg(\exists y (x + y = 320)) \equiv \exists x \forall y \neg(x + y = 320)$, where the statement on the right says: There exists an $x \in \mathbf{N}$ ($x = 321$) such that for all $y \in \mathbf{N}$, $x + y \neq 320$. Right?

Example. For Rational numbers \mathbf{Q} (positive and negative integers and fractions and zero) is it true that $x^2 > x$? I.e., is the following True? $\forall (x \in \mathbf{Q}) (x^2 > x)$ No, because for $0 < x < 1$, $x^2 < x$, e.g. $(0.5)^2 = 0.25 < 0.5$. But of course for negative numbers, it IS true that $x^2 > x$.

To put it a different way, $x^2 > x$ if and only if $(x^2 - x) > 0$ (adding $-x$ to both sides) or, factoring $(x - 1) \cdot (x) > 0$. This will be True if and only if both factors are positive, i.e.: $x > 1$ (which means $(x - 1)$ and x are both positive, or else $x < 0$ (which means $(x - 1) < 0$ as well).

All right, so it's NOT True that $\forall (x \in \mathbf{Q}) (x^2 > x)$; that means there exists a counterexample, $\exists (x \in \mathbf{Q}) \neg(x^2 > x) \equiv \exists (x \in \mathbf{Q}) (x^2 < x)$, and we've just seen $x^2 < x$ when $x = 0.5$.

We have just covered most of Section 1.4, though there is more English to Logic translation (and the reverse) covered there -- you need to know this, but read it and do hw (solved problems as well as Exercises for you to solve, to get practice. I will take questions anytime.

Class 3.

1.5 Rules of Inference

Mathematical reasoning allows us to construct mathematical arguments (proofs), which have a sequence of steps that are known to be true because they follow the **Rules of Inference**. A type of formal proof is covered, but I won't give Exam questions about long chains of reasoning, nor will I ask you to identify errors in proofs (**fallacies**). Just learn how individual rules work.

Understand everything through Example 7, then skip to pg. 70, for quantified statements and understand Examples 12 and 13; learn to use the rules in Table 1 on pg. 66. There are only a few homework exercises for this Section.

Here is a simple valid argument known to ancient Greeks that was quoted through the Middle Ages in Western countries with the Latin name **modus ponens**, meaning **mode that affirms**.

"If you have a current password, then you can log in to the network."

"You have a current password."

Therefore: "You can log in to the network."

In logical form, this is a **Rule of Inference**, written

$$\begin{array}{l} p \rightarrow q \\ \underline{p} \\ \therefore q \end{array}$$

The two statements $p \rightarrow q$ and p are known as **Hypothesis statements**, and they are both assumed to be True. The symbol \therefore should be read: **therefore**. If we translated this argument into pure logic, we would have the statement:

$$((p \rightarrow q) \wedge p) \rightarrow q$$

And this is a tautology, easily seen by examining a Truth table. See Table 1, Pg. 66, top, second column. So why is this Rule of Inference such a big deal? *Because it actually causes us to come to a conclusion in the real world!*

There was a joke (I think by Lewis Carroll, who wrote **Alice in Wonderland**) where a character agreed, when told, that these two statements were True: p and $(p \rightarrow q)$, but didn't see how from these two statements he could conclude that q was true as he was asked to do. Because, he was told, it was also true that $(p \wedge (p \rightarrow q)) \rightarrow q$, and in fact this was a tautology!

He agreed that the three statements were True: p and $(p \rightarrow q)$ and $(p \wedge (p \rightarrow q)) \rightarrow q$, but he still didn't see how to conclude that q was True. So he was told that it was also a tautology that:

$(p \wedge (p \rightarrow q) \wedge ((p \wedge (p \rightarrow q)) \rightarrow q)) \rightarrow q$, so the four statements together now imply that q is True. He still didn't see why and was provided with a yet more complicated tautology.

Here is an Example argument. Given propositions: s : "it snows today" and h : "we stay home", argue that, given: $s \rightarrow h$ and $\neg h$ then $\neg s$ must be true.

1. $s \rightarrow h$ hypothesis
2. $\neg s \vee h$ equivalent to 1.
3. $\neg h$ hypothesis
4. $\neg s$ 2, 3 and disjunctive syllogism (Pg. 66, Table 1, fourth line)

(or use one step with modus ponens. Remember, Quizzes are open book! Alternatively try this:

1. $s \rightarrow h$ hypothesis
2. $\neg h \rightarrow \neg s$ contrapositive of 1, Tautology: Table 7, pg. 25
3. $\neg h$ hypothesis
4. $\neg s$ 2, 3 and modus ponens

To show you a ridiculous puzzle of this kind, here is one by Lewis Carroll, written around 1895 (don't worry about trying to solve it -- I merely think of it as "cute" or "whimsical").

The only animals in this house are cats.
Every animal is suitable for a pet that loves to gaze at the moon.
When I detest an animal, I avoid it.
No animals are carnivorous, unless they prowl at night.
No cat fails to kill mice.
No animals ever take to me, except what are in this house.
Kangaroos are not suitable for pets.
None but carnivores kill mice,
I detest animals that do not take to me.
Animals, that prowl at night, always love to gaze at the moon.

See Table 2, pg. 70, and understand the steps in Examples 12 and 13. I would not give you an Exam problem that required anywhere near the number of steps of Example 13, but you should understand why the steps are true. I might ask you to fill in the Reasons for some steps!!!

1.6 Introduction to Proofs

Study through Example 14, pg. 83. I won't ask you to find mistakes in proofs (last few pages).

Terminology. A **theorem** is a statement (proposition) that can be shown to be true. The argument by which a theorem is demonstrated is called a **proof** and has steps using rules of inference based on **axioms** and previously proven theorems **lemmas**, or **corollaries**.

A **lemma** is a less important statement that is proven in preparation for proving a more important theorem. As for **axioms**, there are exactly eleven on which all of Plane Geometry is based, for example: two triangles are congruent if two angles and an included side are equal -- ASA -- or another, through a point outside a line, there is one and only one line parallel to the given line.

A **corollary** is a (usually) less important theorem easily proved once a major theorem is proved.

A **conjecture** is a statement someone believes is true that has not yet been proved (and which might turn out to be false). For example, **Goldbach's Conjecture** states that every even number greater than 2 is the sum of two primes (counting 1 as a prime, not standard). It was conjectured in 1742 and is still open (believed True but may be False). See the Wikipedia article.

The seeming aim of Sections 1.6 and 1.7 is to list methods of proof, give some examples, and help you learn how to perform proofs. But performing proofs take a lot more practice than this.

However, let's cover some of this, and maybe it will help. To start, consider how conjectures are stated in mathematics. "If $x > y$, where x and y are positive real numbers, then $x^2 > y^2$."

This really seems to mean, "For all positive real numbers x and y , if $x > y$ then $x^2 > y^2$." But the standard convention in Math is to omit such a universal quantifier, simply assuming it when two variables x and y are named in a proposition. I didn't see this conjecture proved, so here's a proof.

Since $x > y$ and they are both positive, I can multiply both sides by the same positive real number and the result will be true. Multiplying by x , we get $x^2 > xy$; multiplying by y , we get $xy > y^2$. So by transitivity, $x^2 > xy > y^2$, and $x^2 > y^2$.

A **direct proof** of $p \rightarrow q$ starts by assuming p is true, then contains a sequence of valid steps of inference to end up by concluding that q is true. Here is a definition on which to base proofs.

Definition 1. An integer n is even if (and only if) there exists an integer k such that $n = 2k$, and n is odd if (and only if) there exists an integer k such that $n = 2k+1$.

First of all, the text says only "if" rather than "if and only if" which is what is meant. It then states that an integer n can be either even or odd and never both, which would imply the only if part. We can easily prove this.

Proposition (minor Theorem). An integer n must be either odd or even and cannot be both.

Proof. To show it must be one, start with the integer 1. This is equal to $2k+1$ where $k = 0$. If we then add 1 to get 2, 2 is equal to $2k$, where $k = 1$. Clearly (this means simple argument is left out) we can add 1 to n as long as we like and will get $n = 2k$ and $n = 2k + 1$ alternately. We could similarly subtract 1 from n as often as we like and get the same result. This is true for all integers n , positive and negative. Now we show that an integer cannot be both even and odd at once. If $n = 2k$ for some integer k and we also have that $n = 2m+1$ for some integer m , then $2k = 2m+1$, but then (dividing both sides by 2): $k = m+(1/2)$, which is impossible for two integers k and m .

Example 1. Give a direct proof of the theorem: "If n is an odd integer, then n^2 is odd." **Proof.** If n is odd, then $n = 2k+1$, and $n^2 = 4k^2+4k+1 = 2(2k^2+2k)+1$, so $n^2 = 2m+1$, where $m = 2k^2+2k$, and thus is also odd. QED

Example 2. Give a direct proof that if the integers m and n are both perfect squares, then mn is a perfect square. **Proof.** If m and n are both perfect squares, then $m = s^2$ and $n = t^2$ for two integers s and t . But then $mn = s^2t^2 = (st)^2$, which is also a perfect square of the integer st . QED

A **proof by contraposition** proves $p \rightarrow q$ by starting with the assumption $\neg q$ and proving $\neg p$. This is MORE COMMONLY KNOWN AS an indirect proof.

Example 3. Prove that if n is an integer and $3n+2$ is odd, then n is odd. **Proof.** If we assumed instead that n were even ($\neg q$) then we would have $n = 2k$. Then $3n+2$ would be $6k+2 = 2(3k+1)$ and thus $3n+2$ would be even ($\neg p$). QED

Example 8. Prove that if n is an integer and n^2 is odd (p) then n is odd (q). **Proof.** This is difficult, if not impossible, to prove directly. But if n is not odd, i.e., if n is even ($\neg q$), then n^2 is even ($\neg p$). Note that this requires that not-even means odd and the reverse.

Proof by Contradiction proves a statement p by starting with the assumption $\neg p$ and showing that a contradiction can be derived as a result.

Definition 2. A **rational** number r has the property that there exist integers p and q , with $q \neq 0$, such that $r = p/q$. A real number that is not rational is called **irrational**, e.g., $2^{1/2}$ or Π .

Example 10. Prove that $2^{1/2}$ (i.e., SQRT(2)) is irrational. **Proof.** Proof by Contradiction. Assume that SQRT(2) is rational, and thus SQRT(2) = a/b for a, b integers and $b \neq 0$. Assume a and b have no common divisors -- since we could factor them out and get smaller a and b . But then by squaring both sides we get $2 = a^2/b^2$, or by transposition, $2b^2 = a^2$. But then a^2 must be even, so a is even and $a = 2k$ and $a^2 = 4k^2$. Then $2b^2 = a^2 = 4k^2$ and $b = 2k$, so b is also even, contradicting the assumption that a and b had no common divisors. QED

1.7 Proof Methods and Strategy.

Study through Example 7, skip to looking for counterexamples on pg. 96, Example 17, then skip rest of section.

To begin, Exhaustive Proofs are obvious -- just show for finite set of a given domain.

Proof by Cases. Likely to be an infinite domain but proof naturally breaks up into a few cases.

Example 4. Prove that if n is an integer, $n^2 \geq n$. **Proof.** Case (i). If $n = 0$, then $0 \geq 0$ is True. Case (ii) If n is positive, then $n \geq 1$; multiplying both sides by n , $n^2 \geq n$. Case (iii) If n is negative, n^2 is positive, so again $n^2 \geq n$. QED

CHECKERBOARD EXAMPLE (TALK IT)

CHAPTER 2. Sets, Functions, Sequences and Sums.

2.1 Sets. Study entire Section except skip Example 20.

Definition: A set is an unordered collection of *distinct* objects. The objects are called the *elements* (or *members*) of the set and the set is said to *contain* its elements.

Examples. The set of vowels in English, $V = \{a, e, i, o, u\}$. The set of odd numbers less than 10, $O = \{1, 3, 5, 7, 9\}$. The set of positive integers less than 100 can be denoted by $\{1, 2, 3, \dots, 99\}$.

Set Builder Notation: Examples

$O = \{x \mid x \text{ is an odd positive number less than } 10\}$, or $O = \{x \in \mathbf{Z}^+ \mid x \text{ is odd and } x < 10\}$

Define \mathbf{Z} = set of integers, \mathbf{N} = set of non-negative integers, \mathbf{Z}^+ = set of positive integers,

\mathbf{Q} = set of rational numbers $\{p/q \mid p, q \in \mathbf{Z} \text{ and } q \neq 0\}$, \mathbf{R} = set of real numbers.

Definition 3. Two sets A and B are equal ($A = B$) iff they have the same elements.

E.g. $\forall x (x \in A \leftrightarrow x \in B)$. (This begs the question a bit -- what is the domain for x in $\forall x$? It should be $x \in A \cup B$.)

Definitions. A is a *subset* of B , $A \subseteq B$ iff $\forall x (x \in A \rightarrow x \in B)$. If A is a subset of B and B contains elements that A does not, we say A is a *proper* subset of B ($A \subset B$).

The *empty set* or *null set* (\emptyset) contains no elements and is a subset of all other sets.

Class 4.

The Power Set.

Definition. Given a set S , its power set $P(S)$ is the set of all subsets of S . In set builder notation:

$$P(S) = \{x: x \subseteq S\}$$

If S is a finite set containing n elements, then $P(S)$ will always contain 2^n elements (including the empty set). E.g., $P(\{1,2,3\}) = \{\{1,2,3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1\}, \{2\}, \{3\}, \emptyset\}$

The power set of the empty set is the set containing the empty set alone $P(\emptyset) = \{\emptyset\}$.

Definition. A Cartesian product of two sets A and B is $A \times B$, defined as follows:

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

We can have a Cartesian product of multiple sets:

$$A_1 \times A_2 \times \dots \times A_n = \{(a_1, a_2, \dots, a_n) \mid a_i \in A_i \text{ for all } i \text{ from } 1 \text{ to } n\}.$$

Section 2.2 Set Operations.

Define intersect ($A \cap B$), union ($A \cup B$), difference ($A - B$), complement A^c , finite set, infinite set. Note, the complement of A in the text is A -with-a-line-over-it, but I can't print that. Getting Graphics soon. The idea of a complement is that there is a Universal set U , and $A^c = U - A$.

Illustrate with Venn Diagrams various operations. $(A^c)^c = A$. $(A \cap B)^c = A^c \cup B^c$. See Table 1 Set Identities on pg.124.

Note one can create a Membership Table (like a Truth Table) to demonstrate set identities. Naming k sets (e.g., 2 sets), assume 1's and 0's on 2^k (4) levels to represent all possible combinations of elements to differentiate all the sets by their element content.

A	B	$A \cap B$	$(A \cap B)^c$	A^c	B^c	$A^c \cup B^c$	$(A \cap B)^c \leftrightarrow A^c \cup B^c$
1	1	1	0	0	0	0	1
1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1
0	0	0	1	1	1	1	1

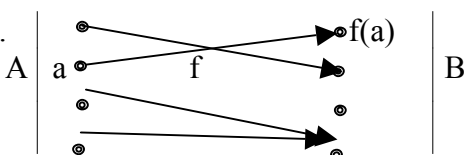
Union (intersection, Cartesian product) of a collection of sets -- Notation.

$$A_1 \cup A_2 \cup \dots \cup A_n \text{ can be written } \bigcup_{\{i=1 \text{ to } n\}} A_i.$$

Section 2.3 Functions.

Definition. Let A and B be non-empty sets. A *function* f from A to B will assign exactly one element b of B to every element a of A , and this is written $f(a) = b$. If f is a function from A to B we write $f: A \rightarrow B$ (read: f sends A to B ; the arrow is NOT an implication here.)

Here is a function representation.



Here f is a *function* from A to B ; A is the *Domain* of f and B is the *Codomain*; for $a \in A$, $f(a) = b$ (f of a is b), and we say f maps A to B ; the *Range of f* is the set of images of A by f .

Example 4. Let $f: \mathbf{Z} \rightarrow \mathbf{Z}$, with Domain and Codomain the set of integers, defined as $f(x) = x^2$. The Range is the set of positive integers that are perfect squares: 1, 4, 9, ...

Definition 4. The floor function sends a real number x (such as $x = 1.512$) to the largest integer not exceeding x (x may be positive or negative): $\text{floor}(1.512) = 1$; $\text{floor}(-1.513) = -2$. The function $\text{floor}(x)$ is sometimes represented by $[x]$. In the C language, the function is declared:

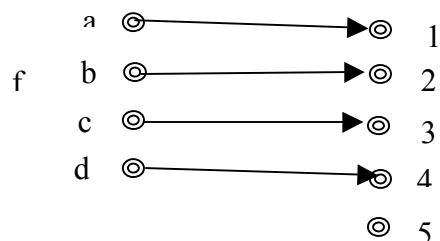
```
int floor(double );
```

Two functions with the same domain and value ranges of the same type (int, real) can be added:

$$(f_1 + f_2)(x) = f_1(x) + f_2(x), \text{ or multiplied: } (f_1 f_2)(x) = f_1(x)f_2(x).$$

Definition 5. A function f is said to be **one-to-one**, or **injective**, if and only if different values x and y in the domain are always mapped to different elements $f(x)$ and $f(y)$ in the range.

The one-to-one definition used in the Text is that f is one-to-one if and only if $f(a) = f(b)$ implies that $a = b$; this is the contrapositive of my definition, but I think mine is easier to grasp: if $a \neq b$ then $f(a) \neq f(b)$. The function diagram I used at the bottom of the prior page is NOT one-to-one. Here is a diagram that is.



Note that the function of Example 4, $f: \mathbf{Z} \rightarrow \mathbf{Z}$, $f(x) = x^2$ is not injective (one-to-one), because $f(1) = f(-1)$, but if we restrict the domain to \mathbf{Z}^+ , then $f(x) = x^2$ is injective! The functions $g: \mathbf{Z} \rightarrow \mathbf{Z}$, defined by $g(x) = -x$, and $h: \mathbf{Z} \rightarrow \mathbf{Z}$, defined by $h(x) = x+1$ however, are injective. In both cases, if $a \neq b$ then $g(a) \neq g(b)$ since $-a \neq -b$, and $h(a) \neq h(b)$ since $a+1 \neq b+1$.

Definition 7. The two functions $g(x)$ and $h(x)$ are both *onto*, or *surjective*, meaning that for every b in the codomain (\mathbf{Z}) there is an a in the domain (also \mathbf{Z}) such that $g(a) = b$ (or $h(a) = b$). For g , given x in the codomain choose $-x$ and $g(-x) = -(-x) = x$; for h , given x in the codomain, choose $x-1$, and $h(x-1) = (x-1)+1 = x$.

Definition 8. A function f is a *one-to-one correspondence* or a *bijection*, if and only if it is both one-to-one and onto.

Definition 9. Let f be a one-to-one onto function from Domain A onto Codomain B . Then there is an inverse function of f denoted f^{-1} that maps every element of B (f was onto) back to a unique element of A (f was one-to-one) in such a way that $f^{-1}(f(x)) = x$ in A and $f(f^{-1}(y)) = y$ in B .

Definition 10. Let g be a function from the set A to the set B and f be a function from the set B to the set C . Then the *composition* of the functions f and g , written $f \circ g$, is defined by:

$$(f \circ g)(a) = f(g(a)).$$

Definition. If we define I as an identity function on some domain (say A), then $f^{-1} \circ f = I$, and $f \circ f^{-1} = I$ (on the domain B).

Section 2.4 Sequences and Summations

Definition 1. A *sequence* is a function from a subset of the integers (usually either the set $\{0, 1, 2, \dots\}$ or the set $\{1, 2, 3, \dots\}$) to a set S . The notation a_n denotes a *term* of the sequence, the image of the integer n .

Definition 2. A *geometric progression* is a sequence of the form: $a, ar, ar^2, \dots, ar^n, \dots$, where the *initial term* a and the *common ratio* r are real numbers.

Jumping ahead to Summations, we ask how to add the first n (or $n+1$) terms of a geometric progression: can we represent such a sum with a simple formula? The answer is yes.

For $a = 1$ and $r = 0.5$, the sequence is $1, 1/2, 1/4, 1/8, 1/16, \dots$. The sum of these terms converges to 2. ($1 + 1/2 = 2 - 1/2$; $1 + 1/2 + 1/4 = 2 - 1/4$; $1 + 1/2 + 1/4 + 1/8 = 2 - 1/8$; $1 + 1/2 + 1/4 + 1/8 + 1/16 = 2 - 1/16$, and so on, coming closer and closer to 2, but never quite reaching it.

We would write this summation as $\sum_{i=0}^n (1/2)^i$ (I am waiting for a formula package.)

In general, the sum $a + ar + ar^2 + \dots + ar^n$ has a closed form value $(a - ar^{n+1})/(1 - r)$ if $r \neq 1$ (or $(n+1)a$ if $r = 1$). You can test this by multiplying $(1 - r)$ by $(a + ar + ar^2 + \dots + ar^n)$ to get $(a - ar^{n+1})$. If $0 < r < 1$. (See Theorem 1, pg 155). This sum converges to $a/(1 - r)$ as n increases without bound.

Definition 3. An *arithmetic progression* is a sequence of the form: $a, a+d, a+2d, \dots, a+nd, \dots$, where the *initial term* a and the *common difference* d are real numbers.

If a is 1 and d is 2, this is the sequence of odd numbers, $1, 3, 5, \dots, 2n+1, \dots$. It is easy to show by mathematical induction that the sum of the first n odd numbers is n^2 . $1 = 1^2$, $1 + 3 = 2^2$, $1 + 3 + 5 = 3^2$. The n^{th} odd number is $2n-1$, and assume that the sum of the first n odd numbers is n^2 . Then add the next odd number, $2n + 1$, and clearly $n^2 + 2n + 1$ is $(n+1)^2$. QED

It is also easy to show that the sum of the first n positive integers, $1 + 2 + 3 + \dots + n$, is equal to $n(n+1)/2$. True for 1 ($1 = 1(1+1)/2$), true for 2 ($1 + 2 = 3 = 2(2+1)/2$). Assume for n and prove for $n+1$. If the sum through n is $n(n+1)/2$, then the sum through $n+1$ will be $(n+1)(n+2)/2 = n(n+1)/2 + 2(n+1)/2 = n(n+1)/2 + (n+1)$. QED Therefore: $d + 2d + 3d + \dots + nd = nd(n+1)/2$.

In general, the sum $a + (a+d) + (a+2d) + \dots + (a+nd) = (n+1)a + n(n+1)d/2 = (n+1)(2a+nd)/2$.

There is a story that when Euler (probably the greatest mathematician of the 1700s) was eight or nine years old, his instructor at school gave the class a problem to keep them busy, to add up all the multiples of 7 less than 1000. This is an arithmetic series, and Euler figured out the formula for the sum and answered within a few seconds.

There is no way to give a formula for the sum: $1 + 1/2 + 1/3 + \dots + 1/n + \dots$ (although it is approximated by $\log(n)$).

Skip Special Integer Sequences, pgs. 451-453 and most of Table 2, pg. 157.

Cardinalities of Infinity. Two sets (even infinite sets) are said to have the same Cardinality iff the elements of the two sets can be put in one-to-one correspondence.

The set of positive integers \mathbf{Z}^+ has the same cardinality as the set of even positive integers, $\{2, 4, \dots, 2n, \dots\}$ (call it $2\mathbf{Z}^+$). **Proof.** Simply use $f: \mathbf{Z}^+ \rightarrow 2\mathbf{Z}^+$ by $f(i) = 2i$; then $f^{-1}(i) = i/2$, an integer since all integers i in $2\mathbf{Z}^+$ must be even. QED. Sets in 1-1 correspondence with the integers are called *Countable* sets.

In Cardinality Theory there are sets that are LARGER than the infinite set of integer, such as the set of real numbers. Constructive mathematicians disagree with this statement, since the set of all possible English definitions is countable (i.e., have cardinality of \mathbf{Z}^+), so how can mathematics postulate a LARGER set of real numbers?

Proof. Any possible English-language statement is convertible to integer form (each statement would be represented by bit sequences in a computer). This includes mathematical definitions of real numbers, using the standard approach to define an infinite sequence and applying the Least Upper Bound axiom. No real number can be constructed without such a sequence definition.

But all such definitions can be defined by an arbitrarily long but always finite binary sequence in a computer, and there is a 1-1 correspondence with the integers. QED

Typical Example of Non-Countable set. The set of all subsets of \mathbf{Z}^+ , $P(\mathbf{Z}^+)$ was shown by Georg Cantor to be larger than the set \mathbf{Z}^+ in the late 1800s. I heard from Paul Erdos that in one of Cantor's letters he said he didn't trust the proof (at first) since it seemed like a contradiction. All of this theory has been VERY controversial, although it is now taught in all college texts.

Note that $P(\mathbf{Z}^+)$ includes all infinite subsets of \mathbf{Z}^+ as well as all finite subsets, e.g., all even positive numbers, all odd positive numbers, all perfect primes (2, 3, 5, 7, 13, ...), and so on.

Proof. We will prove that there is no 1-1 onto function $f()$ from \mathbf{Z}^+ to $P(\mathbf{Z}^+)$. For if there were we would be able to list all subsets of \mathbf{Z}^+ , named S_1, S_2, S_3, \dots , where the subscripts list the integers that correspond to these subsets, i.e., $f(1) = S_1, f(2) = S_2, f(3) = S_3, \dots$, and so on.

To show this correspondence $f()$, we will create an infinite array, where the rows form the sequence of subsets and the columns represent integers 1, 2, 3, ... that sit in the rows; for each set S_1, S_2, S_3, \dots , we show whether successive integers 1, 2, 3, 4, 5, ... sit in the set by listing a bit string such as 01011... This bit string says that 1 is not in the set, 2 is, 3 is not, 4 and 5 are, etc.

Here is the array for a specific correspondence $f()$:

Int:	123456789...	
S_1	<u>1</u> 0000000...	S_1 might be the finite set containing only the integer 1 (or more > 9)
S_2	1 <u>0</u> 1010101...	S_2 might be the set of odd numbers (we're not showing them all)
S_3	01 <u>0</u> 101010...	S_3 might be the set of even numbers
S_4	011 <u>0</u> 10100...	S_4 might be the set of perfect primes: 2, 3, 5, 7, ...
S_5	1101 <u>1</u> 0100...	
S_6	1001 <u>0</u> 0101...	
S_7	10100 <u>1</u> 100...	
...		

What we do next is show a set of numbers that are not the image of $f: \mathbf{Z}^+ \rightarrow P(\mathbf{Z}^+)$, that is we show a bitmap sequence that does not correspond to any of the subsets S_i for any i in \mathbf{Z}^+ .

Note the underlining of successive bits in bit strings corresponding to successive subsets S_1, S_2, S_3, \dots . The underlined bits are: 1000101... We now flip all these bits for all sets listed from $P(\mathbf{Z}^+)$. It should be clear that for bit string 0111010... is not any of the strings pictured, and if we keep flipping bits the resulting string will not be ANY of the strings listed.

This means that by selecting integers where this flipped string is 1 and no integers where it is 0, we will find a subset S of \mathbf{Z}^+ that is NOT in the range of $f: \mathbf{Z}^+ \rightarrow P(\mathbf{Z}^+)$. But $f()$ was supposed to be a 1-1 onto function, and we have just found an element of that is not in the range. Therefore our assumption that there exists such a function is contradicted and $P(\mathbf{Z}^+)$ is not Countable.

Class 5.

In the Proof last time that $P(\mathbf{Z}^+)$ is not Countable, we assumed the ability to construct an infinite set (the diagonal set of integers in $f: \mathbf{Z}^+ \rightarrow P(\mathbf{Z}^+)$). But how would you construct an infinite set? What algorithm would you use? We used an "Infinite Axiom of Choice"; it is not constructive in the same sense it is for a finite set, thus it is controversial among serious mathematicians.

Note that I used Mathematical Induction to validate the formulas for the sum of a geometric progression and the sum of an arithmetic progression, although it is not defined until Section 4.1. Let us therefore jump forward to Section 4.1 and define it now.

4.1 Mathematical Induction. (Pg. 263)

The idea of mathematical induction is that we are confronted with a ladder whose top we can't see -- can we climb to an arbitrarily high rung on the ladder?

Say we step up and stand on the first rung so we know we can do that. Then let us say that we step from the first to the second rung, and we're assured by someone we trust that all the rungs are the same distance apart, so since we can go from rung 1 to rung 2 we can go from k to $k+1$ for any k . (This ignores hunger and being able to rest, but assume we can solve those problems.)

Then we can climb to any level n on the ladder. (There is nothing magic about using k for the *induction step*, going from k to $k+1$ -- any variable can be used.)

OK, so let's prove once again that $1 + 2 + \dots + n = n(n + 1)/2$. Call this assertion $P(n)$. We want to prove $\forall (n \in \mathbf{Z}^+) P(n)$. Step 1. $P(1)$ is True because $1 = 1(1 + 1)/2$.

Step 2. Assume we have reached the k^{th} rung: $1 + 2 + \dots + k = k(k + 1)/2$. Prove we can reach the $(k+1)^{\text{st}}$ rung: $1 + 2 + \dots + k + (k+1) = k(k + 1)/2 + (k + 1)$ (known formula for sum of first k terms plus $(k+1)^{\text{st}}$ term), and $k(k + 1)/2 + (k + 1) = k(k + 1)/2 + 2(k + 1)/2 = (k(k + 1) + 2(k + 1))/2$.

Now adding the terms of the numerator, we get: $(k + 2)(k+1)/2 = (k+1)((k+1) + 1)/2$, and if we let $k+1 = m$, we have: $1 + 2 + \dots + (m - 1) + m = m(m + 1)/2$, which is the formula we are trying to prove, and since we showed from $P(k)$ that $P(m)$ is true, $m = k + 1$, we have proven the formula.

We can skip Section 4.1 Examples 7, 8, 10 and the rest of the Section.

Example 9. A finite set S with n elements has a Power Set $P(S)$ with 2^n elements. (Notation, given a set S , we write $|S|$ to represent the cardinality of S , i.e. the number of elements in S . See Inside front cover under Sets, about the middle for $|S|$. Thus we are trying to show that if $|S| = n$ then $|P(S)| = 2^n$.)

We argued that this was true by listing all elements of S and choosing which elements were in each particular subset by corresponding to an arbitrary bit sequence to select the elements.

Say the elements of S can be named by the integers $1, 2, 3, 4, \dots, n$, and we write below these integers the 0-1 bit selector sequences $0, 1, 1, 0, \dots, 0$, e.g. this would NOT select 1 and 4 and n , and WOULD select 2 and 3.

There are clearly 2^n bit-selector sequences of length n , and we put these in 1-1 correspondence with the subsets of S , so the two sets have equal cardinality.

But let us do it with Mathematical Induction. Call the assertion $P(n)$ that a set S with $|S| = n$ has $|P(S)| = 2^n$. We want to prove $\forall (n \in \mathbf{Z}^+) P(n)$. **Proof.** Step 1. $P(1)$ is True because a set S with a single element, $\{1\}$, has a power set with two subsets: $\{\{1\}, \emptyset\}$.

Step 2. Assume we have shown for any set S with $|S| = k$, that this is true, e.g., if S has elements $\{1, 2, \dots, k\}$, $|P(S)| = 2^k$. We prove it is True for a set S with $|S| = k+1$ that $|P(S)| = 2^{k+1}$ as follows.

Proof: Assume one element of S is x , and let $S' = S - \{x\}$, i.e. $|S'| = k$, so $|P(S')| = 2^k$. But now consider the set T of elements of $P(S')$ with each set in $P(S')$ having the additional element x .

Clearly $T \subset P(S)$, since each of its subsets has only elements of S , and $|T| = |P(S')|$ since all element s of T are created by adding an element x to subset elements in $P(S')$. In fact $T \cap P(S') = \emptyset$ since all elements of T are subsets containing x and x does not occur in elements of $P(S')$. Also $T \cup P(S') = P(S)$: thus $|T| = |P(S')| = 2^k$, and since these are disjoint, $P(S) = 2^{k+1}$. QED.

Class 6.

Chapter 3. Algorithms, the Integers, and Matrices.

Section 3.1 Algorithms.

Study everything except **The Insertion Sort** (pg 174) and **The Halting Problem** (pg 176).

Def. 1 An *algorithm* is a finite set of precise instructions for performing a computation or solving a problem.

ALGORITHM 1 Finding the maximum element of a finite sequence.

procedure $max(a_1, a_2, \dots, a_n)$: integers)

$max := a_1$;

for $i := 2$ to n

if $max < a_i$ **then** $max := a_i$

{max returned is the largest element}

Obviously this algorithm takes n steps to complete -- compare max so far to each a_i in list.

Searching Algorithms

The general searching algorithm is this: Locate an element x in a list of distinct elements a_1, a_2, \dots, a_n , or determine that x is not in the list. The solution is the location in the list that equals x (this is i if $x = a_i$, and 0 if x is not in the list. NOTE: Here we are assuming that the list has no duplicate elements -- if there were duplicates (not assumed here), then we might search for *Multiple Locations* that contain a value x .

ALGORITHM 2 The Linear Search Algorithm.

procedure $linear_search(x)$: integer; a_1, a_2, \dots, a_n : distinct integers)

$i := 1$;

while ($i \leq n$ and $x \neq a_i$)

$i := i + 1$

if $i \leq n$ **then** $location = i$

else $location := 0$

{location is the subscript of the term equal to x or is 0 if x not found}

This is another n step algorithm

Now: binary search algorithm. Old Twenty Questions quiz program: Identify anything in the world by being told in advance if it is animal, vegetable or mineral, then asking twenty yes or no questions. (Twenty questions identifies one from a million objects by well-posed questions.)

I'll do it with 1 to 1000 in ten questions, Someone think of an integer. My first question: is it greater than 512?

ALGORITHM 3 The Binary Search Algorithm.

procedure *binary_search*(*x*: integer; a_1, a_2, \dots, a_n : distinct increasing integers)

i := 1; (*i* is left endpoint of search interval)

j := *n*; (*j* is right endpoint of search interval)

while *i* < *j*

begin

$m := (i + j)/2$ -- using integer arithmetic, so assume $(i + j)/2 = \lfloor (i + j)/2 \rfloor$

if $x > a_m$ **then** *i* := *m* + 1

else *j* := *m*

end

if $x = a_i$ **then** *location* = *i*

else *location* := 0

{*location* is the subscript of the term equal to *x* or is 0 if *x* not found}

This is a very fast algorithm to find a particular *x* among *n* elements. If there are 1024 (2^{10}) elements, 10 **steps**, if 1048576 (2^{20}) elements, 20 steps. Basically $\log_2 n$ steps for *n* elements.

But look! There are for each probe in Algorithm 3 there are (like) 4.5 evaluations in a **step** (between **begin** and **end**: counting (1) the loop step evaluation $i < j$, (2) setting *m*, (3) the **i...then** (2 evaluations) and sometimes the **else** (0.5 evaluations), while for Algorithms 1 and 2 there are about 3 evaluations per loop.

Does this matter? Should we worry about what the constant multiple *c* of the number of steps is needed for evaluations in these algorithms? Answer: No. That would be asking too much.

If we had a million elements a_1, a_2, \dots, a_n in Algorithms 1 or 2, the algorithms would take $1000000c$ evaluations to execute, while if we had the same million elements (in order) in Algorithm 3, it would take $20k$ evaluations to execute, where there are *k* evaluations per step. The difference between 1000000 and 20 is much more important than the difference between *c* and *k*.

Jumping ahead to Section 3.2, Big-Oh Notation: If *f* and *g* are functions from the set of integers or real numbers to the set of real numbers, we say *f*(*x*) is $O(g(x))$ if there are constants *c* and *k* such that $|f(x)| \leq c|g(x)|$ whenever $x > k$.

E.g., Theorem 1, pg 184: If $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$, where $a_n, a_{n-1}, \dots, a_1, a_0$ are real numbers, then *f*(*x*) is $O(x^n)$. Note that it is the x^n term of the polynomial on the right that grows the fastest, and we're saying that big-oh notation cares only about that term, and ignores constant multiple a_n .

For example: $17x^2 + 29x + 1000$ is $O(x^2)$. Try $x = 1,000,000$: then $17x^2 + 29x + 1000 = 17,000,000,000,000 + 29,000,000 + 1000$. We see why the x^2 term is most important for large *x*,

Note that a lot of examples given in Section 1.2 are confusingly weak results, (Example 2: $7x^2$ is $O(x^3)$); in fact, $7x^2$ is $O(x^2)$, which is a stronger result.

In big-oh terms, Algorithm 1 (finding a maximum by full search) and Algorithm 2 (Linear Search) are $O(n)$, and Algorithm 3, Binary Search, is $O(\log n)$ -- it needn't be $\log_2 n$, since $\ln n = \log_e n$ and $\log_{10} n$ are both constant multiples of $\log_2 n$.

Sorting

ALGORITHM 4 The Bubble Sort.

procedure *bubblesort*(a_1, a_2, \dots, a_n : real numbers with $n \geq 2$)

for $i := 1$ **to** $n - 1$

for $j := 1$ **to** $n - i$

if $a_j \geq a_{j+1}$ **then** *interchange a_j and a_{j+1}*

{(a_1, a_2, \dots, a_n is in increasing order)}

Do an example on the board. Note that for each loop through $j := 1$ **to** $n - i$ we are simply finding the maximum remaining element in the top $n - i$ remaining elements and moving that maximum down to the end.

This is a double loop and the $i = 1$ step loops through n elements, the $i = 2$ step loops through $n - 1$ elements, and the $i = m$ step loops through $n - m$ elements, until $m = n - 1$. The total number of inner loop steps is $n + (n - 1) + (n - 2) + \dots + 2 + 1$. This is a sum we've studied. What is it?

It sums to $n(n+1)/2$. In terms of big-oh notation, then, the Bubble sort is $O(n^2)$.

Greedy Algorithms

Assume you have a large number of every kind of coin in the list: pennies, nickels, dimes, quarters and half-dollars. Say that you were paying \$2.87 for some item, and wanted to use up some change, but the seller was your friend and you didn't want to load her down with more coins than necessary. How can you make exact payment with the minimum number of coins?

Answer. Use a greedy method of paying with the largest denomination coins first until using such a coin would pay too much, then go to the next lower denomination coin and continue in this way. E.g.: 5 half-dollars, leaves 37 cents; 1 quarter leaves 12 cents; one dime and then two pennies.

Algorithm 6 is specified on pg 175, using any set of denominations of coins. On pg 176 we have a proof by contradiction that the greedy algorithm works as advertised for U.S. coins.

But NOTE: If we didn't have nickel coins, the algorithm wouldn't work optimally for 30 cents: three dimes would give a smaller number of coins than a quarter and five pennies!!

Section 3.2 The Growth of Functions. Reading: I have already covered big-oh notation. This should be quite easy for people who have had calculus. Read up to pg 189 and **DO NOT skip** big-omega and big-theta notation.

Class 7.

Section 3.2. The Growth of Functions.

Recall that from Definition 1, pg 180, we say that $f(x)$ is $O(g(x))$ iff there are positive constants C and k such that $|f(x)| \leq C|g(x)|$ whenever $x > k$. In other words, $f(x)$ is $O(g(x))$ means sooner or later in x , $|f(x)|$ is bounded above by some constant times $|g(x)|$. Thus $|f(x)|$ is bounded **above** by $g(x)$, but it might not be CLOSE to $g(x)$ -- perhaps $f(x) = x$ and $g(x) = x^2$. (See Fig 3, pg 187.)

Definition 2, pg 189, says $f(x)$ is $\Omega(g(x))$ (read " $f(x)$ is 'big-omega' of $g(x)$ ") iff there are also positive constants C' and k' such that $|f(x)| \geq C'|g(x)|$ whenever $x > k'$. In other words, $f(x)$ is $\Omega(g(x))$ means sooner or later in x , $|f(x)|$ is bounded **below** by some constant times $|g(x)|$. Thus we might have $f(x) = x$ and $g(x) = \log x$. (See Fig 3, pg 187 again.)

But if we have **BOTH** $f(x)$ is $O(g(x))$ **AND** $f(x)$ is $\Omega(g(x))$, this means that for sufficiently large $x > k$ (max of k & k' above) $|f(x)|$ lies between two constant multiples of $|g(x)|$. When this is true we say that $f(x) = \Theta(g(x))$ (read $f(x)$ is "big theta" of $g(x)$), but this naming is less suggestive than saying $f(x)$ **is of order** $g(x)$.

See Example 11 of Section 3.2: the sum $f(n) = 1 + 2 + 3 + \dots + n$ is $= n(n+1)/2$ is of order n^2 . And Example 12, $3x^2 + 8x \log x$ is of order x^2 .

Section 3.3. Complexity of Algorithms. We only worry about worst-case complexity.

The **Time Complexity** of an algorithm is the number of operations used by an algorithm when the input has a certain size.

E.g., for the **Linear Search Algorithm** on n elements, finding a value x in a sequence of distinct elements a_1, a_2, \dots, a_n , one comparison is used on each successive element to test if it is equal to x and one to test if we have reached the end of the loop: thus $2n+c$ comparisons in the worst case is $\Theta(n)$, or order of n . The average case would find the result about half-way through, but still $\Theta(n)$.

Note there is a cute trick to knock out the test for end of loop: when searching for x , create an element a_{n+1} with value x , so we will certainly find x at that point and can then check if it is the x we created; if so, we return location 0. We never need to test for end of loop this way.

The **Binary Search Algorithm** to find a value x in an increasing sequence on n numbers a_1, a_2, \dots, a_n , we have $\log_2 n$ program steps to home in on the element that might match x . Each of the steps needs to calculate $m = \lfloor (i + j)/2 \rfloor$ to determine a new "probe point", but this merely affects the constant multiple: the algorithm takes time $\Theta(\log n)$ or order of $\log n$.

The Bubble Sort algorithm has n passes through an outer loop and during pass i , we interchange elements in the first $n - i + 1$ elements going from having the $i-1$ largest elements at the end of the sequence to having the i largest at the end. The number of elements accessed on successive passes is $(n - 1) + (n - 2) + \dots + 2 + 1 = (n - 1)n/2$, thus the time is $\Theta(n^2)$ or order of n^2 .

If we look at Table 2, pg 198, we see a list of orders of time used going from the ones we've seen to 2^n to $n!$. I don't know any algorithms that take $\Theta(n!)$ but I can give one that takes time $\Theta(2^n)$. Note that for $n = 10$, $2^{10} = 1024$ (roughly 1000 or 10^3). Note from Table 2 that if operations take 10^{-9} seconds, 2^{10} operations takes 10^{-6} seconds, but 2^{100} takes 4×10^{13} years!!

While 2^{10} is about 10^3 , 2^{100} is about $10^{30} = 1,000,000,000,000,000,000,000,000,000$ or 1 nonillion. Never heard of that before? Imagine waiting for the algorithm to complete! Here is an algorithm that takes 2^n steps.

It is called the "Tower of Hanoi" problem. We are given three tall pegs mounted on a board on which n disks of different sizes with holes in the center (to fit on pegs) sit. See pg. 452 of text.

Initially, the n disks are placed on one of the pegs, each disk sitting on a larger one so the sizes are decreasing from bottom to top. Our job is to slide disks off pegs on which they sit and place them on different pegs until we have duplicated this arrangement of n disks on a different peg. There is but one rule. **At no time can a larger disk rest on a smaller one!**

Let's try to do this inductively. Say we have $n = 1$, i.e. only one disk: then we can move this disk to a different peg in a single step.

How many steps does it take if we have two disks? Say we start on peg 1. Move the smaller top disk to another peg, say peg 2. Then move the second disk (the larger one) to peg 3. Then move the one on disk 2 to peg 3 and we have two disks on peg 3, smaller disks sitting on larger ones and we are done. This took 3 steps. Call this sequence of moves *a 2-disk movement*.

What if we have 3 disks? Well we can start by moving the top 2 disks to other pegs, but at some point we have to have another peg that is unoccupied so we can move the (largest) 3rd disk to another peg. We can only do this by duplicating the 2-disk movement above, since we have to end up with the smaller disks on larger disks. We can call this a *three-disk movement*.

THEN we can move the largest of the 3 disks to another peg (say peg 2) and by duplicating the 2-disk movement again, end up with all 3 disks on peg 2 in proper order. Note we moved the largest disk once and performed two 2-disk movements, each taking 3 steps. So solving this for 3 disks took $1 + 3 + 3 = 7$ steps. We can call this a *three-disk movement*.

Note that we can always treat pegs with larger disks than what we are working on with a k -disk movement as if they were empty, since we will never place a smaller disk on a larger one.

OK, now let us say it takes H_n steps to move n disks from one peg to another while following this rule. First note that with 0 disk, the number of steps is 0 (this is true, and it might help us find a pattern). Here is what we've discovered for our sequence of H_n for n from 0 to 3.

0, 1, 3, 7, ...

Does this sequence give you any ideas? Look at the sequence 2^n . 1, 2, 4, 8, ... Could it be that $H_n = 2^n - 1$? Well, let's see if we can prove it by induction. Assume we have shown this to be true up to k , that $H_k = 2^k - 1$. What can we say for $k+1$?

Well to start with, we need to get the top k disks off the largest disk (say the $k+1^{\text{st}}$). Not only that, but we need to leave another peg empty in order to move the largest disk to another peg. But that means we must have solved the H_k problem for k disks.

We move the largest disk to the empty peg and we now need to move the k disks to sit on top of that largest disk, so that's solving the H_k problem again. Thus to solve H_{k+1} we needed to solve H_k twice and move the largest disk in the middle. Under our induction hypothesis:

$H_{k+1} = 2H_k + 1 = 2(2^k - 1) + 1 = 2^{k+1} - 1$. And we have proved $H_n = 2^n - 1$. Don't try to solve this problem on your home computer! Note that http://en.wikipedia.org/wiki/Tower_of_hanoi has an animated solution for this problem with four disks. Problems that are worse than polynomial are said to be *intractable*.

The text also mentions NP-complete problems, a class that is THOUGHT to be intractable, but nobody has proved it. Here is a seemingly simple one. Say that one has $2n$ integers, X_1, X_n, \dots, X_n . Assume all the integers are VERY large with at least n digits each. Also assume that any $k+1$

elements of this set always give a higher sum than any k elements of the set. Give an algorithm to find the two n -element subsets with sum closest to $\frac{1}{2}$ the sum of the $2n$ elements.

This is believed to be intractable (worse than polynomial).

Section 3.4. Integers and Division. Skip Applications of Congruences & Cryptology at the end.

Definition 1. If a and b are integers with $a \neq 0$, we say that a divides b iff there is an integer c such that $b = ac$. Then we say a is a factor of b and b is a multiple of a . The notation $a \mid b$ means that a divides b and $a \nmid b$ means a does not divide b .

Example 2. If n and d are positive integers, how many positive integers not exceeding n are divisible by d ? They are all values for integers k with $kd \leq n$; thus the number is $k = \lfloor n/d \rfloor$.

Theorem 1. Let a , b , and c be integers. Then

- (i) if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$
- (ii) if $a \mid b$, then $a \mid bc$ for all integers c .
- (iii) if $a \mid b$ and $b \mid c$ then $a \mid c$.

Proof of (i). If $a \mid b$ and $a \mid c$ then by Definition there are integer s and t such that $as = b$ and $at = c$. Thus $as + at = a(s + t) = (b + c)$ and $a \mid (b + c)$.

Proof of (ii). If $a \mid b$ then as above $as = b$ and so $a(sc) = bc$ and $a \mid bc$.

Proof of (iii). If $a \mid b$ then $as = b$ and if $b \mid c$ then $bu = c$ (u an integer), so $a(su) = c$ and $a \mid c$.

But you need to know all these instinctively!

Corollary to Theorem 1. If a , b , and c are integers and $a \mid b$ and $a \mid c$ then $a \mid (mb + nc)$ for any integers m and n . Follows from part (ii) of Thm 1 that $a \mid mb$ and $a \mid nc$ and then apply part (i).

This will turn out to have important ramifications because in fact if b and c have no common factors except a , there will be specific integers m' and n' (one of them negative) such that $a = m'b + n'c$. Try it with $4 \mid 12$ and $4 \mid 20$. Since $12 = 3 \times 4$ and $20 = 5 \times 4$, and $7 \times 3 - 4 \times 5 = 1$, we see that $7 \times 12 + (-4) \times 20 = 84 - 80 = 4$.

The Division Algorithm

Theorem 2. Division Algorithm. Let a be an integer and d a positive integer. Then there are unique integers q and r such that $a = dq + r$, with $0 \leq r < d$. Here d is called the *divisor*, a is the *dividend* (the quantity d is divided into), q is the *quotient*, and r is called the *remainder*.

Example 3. Divide 101 by 11. We get $101 = 11 \times 9 + 2$. Here 9 is the quotient and 2 is the remainder. This always works the way you've been taught if the dividend is positive.

Example 4. Divide -11 by 3 and find the quotient and remainder. NOTE that the remainder cannot be negative, so acting as if -11 is +11, and getting $11 = 3 \times 3 + 2$, then making it negative: $-11 = 3 \times (-3) + (-2)$ doesn't work. We have to add 3 to this negative remainder to get +1, and thus get $-11 = 3 \times (-4) + 1$. This is the way computers do division of negative binary numbers!

Modular Arithmetic

Definition 3. If a and b are integers and m is a positive integer, then a is congruent to b mod m if m divides $a - b$. We write $a \equiv b \pmod{m}$ to indicate this, or $a \not\equiv b \pmod{m}$ if it's not true.

Typically we write $a \equiv b \pmod{m}$ if there is some quotient q such that $a = mq + b$ (where m is the divisor of a , q is the quotient, and b is the remainder). But in fact we can add any positive or

negative multiple c of m to b to get B , $B = b + cm$, and it will still be true that $a \equiv B \pmod{m}$. The set of all such integers B is called the **congruence class** of $a \pmod{m}$.

Theorem 4. Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that $a = b + km$. (Just what we said above after Def. 3.)

Theorem 5. Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

So we can add and multiply congruence classes. Note that for any integer c , $cm \equiv 0 \pmod{m}$ and if we have $a \pmod{m}$ that $a + 0 \pmod{m} = a \pmod{m}$. Furthermore, for every $a \pmod{m}$ there is $-a \pmod{m}$: given a , we take the integer $(a - m)$ and $a + (a - m) \pmod{m} = 0 \pmod{m}$.

In terms of multiplication, we have $1 \pmod{m}$ and for any $a \pmod{m}$, $1 \times a \pmod{m} = a \pmod{m}$. However the inverse multiplicative element a^{-1} such that $aa^{-1} \pmod{m} = 1 \pmod{m}$ does NOT always exist. (It will if m is a prime, but we leave that for later.)

Applications of Congruences. Skip from here to the end of Chapter.

3.5 Primes and Greatest Common Divisors.

Definition 1. A positive integer is called **prime** if it is greater than 1 and the only positive factors of p are 1 and p . A positive integer greater than 1 that is not prime is called **composite**.

Theorem 1. The Fundamental Theorem of Arithmetic Every positive integer greater than 1 can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of non-decreasing size.

E.g. $100 = 2 \times 2 \times 5 \times 5 = 2^2 5^2$, 641 (prime), $999 = 3 \times 3 \times 3 \times 37 = 3^3 37$, $1024 = 2^{10}$.

Theorem 2. If n is a composite integer, then n has a prime divisor less than or equal to $\text{SQRT}(n)$.

Proof. If n is a composite integer it is a product of at least two primes. Say p and q are primes whose product divides n (where p might be equal to q). If we assume that there is no prime factor less than or equal to $\text{SQRT}(n)$ then p and q are both greater than $\text{SQRT}(n)$ and their product is greater than n : this is impossible, since the product of p and q was assumed to divide n . Thus at least one of the two primes must be less than or equal to $\text{SQRT}(n)$.

Example 3. Show that 101 is prime.

First note that all primes $\leq \text{SQRT}(101)$ are 2, 3, 5, 7 (the next one is 11). TRICKS. Any number divisible by 2 ends with an even digit (not 101). Any number divisible by 3 has digits that sum to a number divisible by 3 (such as 111, not 101). Any number divisible by 5 has final digit 0 or 5 (not 101). There is no trick I know for 7, but 7 divided into $101 = 14$ with a remainder of 3, so not a factor. Thus 101 is a prime.

Theorem 3. There are infinitely many primes.

Proof. Assume there were a finite number of primes. List them as p_1, p_2, \dots, p_n , and now take their product plus 1: $p_1 p_2 \dots p_n + 1$. This is equal to 1 mod any of the primes, therefore it is not divisible by any of them. Therefore it is either a new prime itself or it is composite as a product of primes that have not been listed. Therefore the finite list was incorrect and there must be an infinite number of primes.

Class 8.

Continuing 3.5 Primes and Greatest Common Divisors.

If $P(x)$ is the number of primes less than or equal to the integer x , then $P(x)$ approaches $x/\ln x$ as x gets larger. ($\ln x$ is natural log or log base e of x .) Another way of saying this is that a random integer less than x has probability $1/\ln x$ of being a prime. This is said to be THEOREM 4 in our text, but it is much too complicated for us to prove, and you are only responsible for the idea.

You are also not responsible for the next subsection, Conjectures and Open Problems About Primes, but they're fun to look at if you have time. I've already mentioned Goldbach's conjecture (Example 7), and the Twin Prime Conjecture is also amusing (Example 9).

Greatest Common Divisors and Least Common Multiples.

Definition 2. Let a and b be integers, not both zero. The largest integer d such that $d \mid a$ and $d \mid b$ is called the *greatest common divisor* of a and b , denoted by $\gcd(a, b)$.

Examples. (1) What is $\gcd(0, 17)$? Answer: only 17 divides the second integer. Does it divide 0? Recall from our definition we say that a divides b iff there is an integer c such that $b = ac$. Thus 17 divides 0 because there is an integer 0 such that $0 = 17 \times 0$. Thus 17 is $\gcd(17, 0)$. Indeed, $\gcd(a, 0) = a$ for any a (note from the definition, a cannot also be zero).

(2) What is $\gcd(100, 420)$? Write the prime factorization of each: $100 = 2^2 5^2$; $420 = 2^2$ (brings down to 105) $3 \times 5 \times 7$, or all together: $2^2 3 5 7$; the factors they have in common are 2^2 and 5, so $\gcd(100, 420) = 20$.

We can find $\gcd(a, b)$ by taking prime factorizations of a and b , then choosing the largest power of each prime that divides both a and b , which is the minimum power in each factorization. I.e. if $a = 2^3 3 5^2 = 600$ and $b = 2^1 5^3 7 = 1750$, we take $2^{\min(3,1)} 3^{\min(1,0)} 5^{\min(2,3)} 7^{\min(0,1)} = 2^1 5^2 = 50$.

Definition 3. Two integers a and b are said to be relatively prime if $\gcd(a, b) = 1$.

The integers a and b are relatively prime if either a or b is a prime and the other is not equal to the first and is non-zero. E.g.: $\gcd(1, 11) = 1$, $\gcd(2, 11) = 1$, $\gcd(3, 11) = 1$, $\gcd(4, 11) = 1$, $\gcd(5, 11) = 1$, $\gcd(6, 11) = 1$, $\gcd(7, 11) = 1$, $\gcd(8, 11) = 1$, $\gcd(9, 11) = 1$, $\gcd(10, 11) = 1$.

Of course two integers don't have to have one of them prime in order for them to be relatively prime; they just need to have no matching prime factors. If $a = 2^1 5^1 = 10$ and $b = 3^1 7^1 = 21$, the two are relatively prime.

Definition 4. The sequence of integers a_1, a_2, \dots, a_n are pairwise relatively prime if $\gcd(a_i, a_j) = 1$ for any $i \neq j$ between 1 and n . (This definition might appear in homework, but is not important.)

Definition 5. The least common multiple of the positive integers a and b is the smallest positive integer that is divisible by both a and b . It is denoted $\text{lcm}(a, b)$.

Recall finding $\gcd(a, b)$ by choosing minimum powers of prime factorizations: if $a = 2^3 3 5^2 = 600$ and $b = 2^1 5^3 7 = 1750$, $\gcd(a, b) = 2^{\min(3,1)} 3^{\min(1,0)} 5^{\min(2,3)} 7^{\min(0,1)} = 2^1 5^2 = 50$. Similarly, $\text{lcm}(a, b) = 2^{\max(3,1)} 3^{\max(1,0)} 5^{\max(2,3)} 7^{\max(0,1)} = 2^3 3 5^3 7 = 21000$. Note that $ab = 2^{3+1} 3^{1+0} 5^{2+3} 7^{0+1} = 1050000$ and that's the same as $\gcd(a, b) \times \text{lcm}(a, b) = 50 \times 21000$.

This is clear since $\gcd(a, b)$ is the product of primes to power $\min(p_1, p_2)$, where p_1 is the prime power for a and p_2 for b , while $\text{lcm}(a, b)$ is the product of primes to power $\max(p_1, p_2)$, and when $\gcd(a, b)$ and $\text{lcm}(a, b)$ are multiplied, the prime power will be $\min(p_1, p_2) + \max(p_1, p_2) = p_1 + p_2$. This is also the prime power for the product ab . Thus $\gcd(a, b) \times \text{lcm}(a, b) = ab$.

3.6 Integers and Algorithms

We commonly use a base 10 representation of integers. For example, 965 represents $9 \times 10^2 + 6 \times 10^1 + 5 \times 10^0$, where 10^0 is of course 1, so this can be written $9 \times 10^2 + 6 \times 10^1 + 5$. But there are many other possible bases than 10.

Theorem 1. Let b be a positive integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k \times b^k + a_{k-1} \times b^{k-1} + \dots + a_1 \times b^1 + a_0,$$

where k is a non-negative integer, a_0, a_1, \dots, a_k are nonnegative integers less than b , and $a_k \neq 0$.

Consider 165 in base 10, and let us represent it in base 2 (called a binary expansion). What is the least significant bit of the result, one or zero? I.e., will we represent it as $xx \dots xx1$ or $xx \dots xx0$? (Wait.) It's an odd number, so what's the least significant bit? A one, yes.

Using the division algorithm in dividing by 2, we get $165 = 2 \times 82 + \underline{1}$ (a remainder of 1) and it should be clear that the binary expansion of this gave the $xx \dots xx1$ now has all the earlier digits $xx \dots xx$ representing 82. Now does a binary expansion of 82 end in zero or one? 82 is even, so zero, and the full binary representation of the original 165 is now $xx \dots xx01$.

We see this because $82 = 2 \times 41 + \underline{0}$ (a remainder of 0). Let's keep doing this, dividing by 2 and finding the remainder. $41 = 2 \times 20 + \underline{1}$, and we have $165 = xx \dots x101$. And so on: $20 = 2 \times 10 + \underline{0}$, $10 = 2 \times 5 + \underline{0}$, $5 = 2 \times 2 + \underline{1}$, $2 = 2 \times 1 + \underline{0}$, $1 = 2 \times 0 + \underline{1}$, and reading the remainders **backwards**, we end up with 10100101.

Note that as we keep dividing by 2 we always end up with 1 (3 or 2 both give 1 and every division by 2 must go through 3 or 2 -- prove it by induction), so that's the 1 that starts the integer represented (see Theorem 1: $a_k \neq 0$).

OK, let's check this worked properly. $10100101 = 2^7 + 2^5 + 2^2 + 2^0 = 128 + 32 + 4 + 1 = 165$.

How would we give an octal (base 8) representation? Write the bit representation as 10'100'101' (blocks of 3 from the right) and we get: $2 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 2 \times 64 + 4 \times 8 + 5 = 128 + 32 + 5 = 165$ (same as above). But we can do this a different way!

Starting with 165, we successively divide by 8 (as before we did with 2), what will be the remainder? First, $165 = 8 \times 20 + \underline{5}$, then $20 = 8 \times 2 + \underline{4}$ and finally $2 = 8 \times 0 + \underline{2}$, so we get 245₈ (which stands for 245-base-8), or $2 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 2 \times 64 + 4 \times 8 + 5 = 165$, as above.

We can do this with any base. For example, base 7: $165 = 7 \times 23 + \underline{4}$, then $23 = 7 \times 3 + \underline{2}$, and finally $3 = 7 \times 0 + \underline{3}$, so we get $324_7 = 3 \times 49 + 2 \times 7 + 2 = 147 + 14 + 4 = 147 + 18 = 165$.

I have just illustrated Algorithm 1, pg 221 in the text. Note too Table 1, pg. 222 illustrating decimal, binary, octal, and hexadecimal. Hexadecimal has 16 digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A, B, C, D, E, F, up through value 15, and 16 is represented as 10_{16} . Note that patterns of 0^5 and 1^5 can be put in place of hex digits to create a binary representation of the same number.

See Table 1, pg. 222. The binary equivalent of $A5C9_{16} = 1010'0101'1100'1001'$.

I expect you to understand how to add in binary (it's easy: $1 + 1 = 0$ and carry 1, $1 + 1$ with a carried 1 = 1 and carry 1). I don't expect you to be quick with addition, multiplication, div and mod in an arbitrary base -- you know how to do it in decimal. (But LOOK at Algorithms 2 & 3.)

Also skip Modular Exponentiation, Algorithm 4. But the Euclidean Algorithm is fundamental.

The Euclidean Algorithm.

Consider $\gcd(91, 287)$. Divide 287 (the larger integer) by 91 to get $287 = 91 \times 3 + 14$. Any number d that divides 287 and 91 will also divide $287 - 91 \times 3 = 14$. So since we are finding $\gcd(91, 287)$, this will be the same as $\gcd(91, 14)$ (two smaller numbers, but 14 is a linear combination of 287 and 91).

Now divide 91 (the larger integer) by 14 to get $91 = 14 \times 6 + 7$. But then any common divisor of 14 and 91 will also divide $91 - 14 \times 6 = 7$, so we can write $\gcd(91, 14)$ as $\gcd(14, 7) = 7$. Now we have $\gcd(91, 287) = \gcd(91, 14) = \gcd(14, 7) = 7$.

This represents a way to find a gcd of any pair of integers by successively dividing the larger by the smaller and replacing the larger by the remainder.

Note that in Algorithm 6. Pg 229, if we start $r := x \bmod y$, with y larger than x , we will end up with $r = x$; then in the next pass x and y will interchange so the larger will be x , and r will be less than either x or y . You ought to be able to perform this without having to memorize Algorithm 6, but you might want to start by looking at the Algorithm to do homework.

3.7 Applications of Number Theory

In my opinion, the purpose of CS320 is to make you aware of mathematics you might be expected to know in later courses in Math and CS here at UMB. In my opinion, this test is not met by Lemmas 1 and 2 on pg 233 and Theorem 2 on pg 234, nor by the Chinese Remainder Theorem nor Pseudoprimes (although Theorem 5 is important).

We also skip Public Key Cryptography and RSA. (I will explain the concept of Public Key encryption, but give no details nor hold you responsible.) It is not that none of these topics are sure to have no relevance in later courses, just that I would expect the courses to explain these concepts as they introduce their material.

Theorem 1. If a and b are positive integers, then there exist integers s and t such that $\gcd(a, b) = sa + tb$. Typically either s or t is negative and the other positive.

Proof. Recall in the Euclidean Algorithm above, we have $\gcd(287, 91) = 7$, and $287 - 91 \times 3 = 14$, then $91 - 14 \times 6 = 7$, but substituting for 14, we have $91 - (287 - 91 \times 3) \times 6 = 7$, and gathering terms for 91 and 287 we have $91(1 + 3 \times 6) - 6 \times 287 = 19 \times 91 - 6 \times 287 = 1729 - 1722 = 7$.

Looking at the Euclidean Algorithm (Algorithm 6, pg 229) we find that a $\gcd(a, b)$ is always a linear combination of a and b in this way.

There is a cute problem that if we have two stamps whose denominations a and b are relatively prime then for n any integer greater than some integer N we can represent n precisely with these stamps. That result can be shown with this theorem. We just have to make N large enough to make the negative factor zero.

OK, now a couple of Lemmas that will lead to showing that a factorization of an integer into primes is unique.

Lemma 1. If a , b , and c are positive integers such that $\gcd(a, b) = 1$ and $a \mid bc$, then $a \mid c$.

Proof. Because $\gcd(a, b) = 1$, by theorem 1 there exist integers s and t such that $sa + tb = 1$. Multiplying both sides by c we get $sac + tbc = c$. Given the assumption that $a \mid bc$ and Thm. 1 on pg 202, $a \mid tbc$. Clearly $a \mid sac$, so $a \mid sac + tbc$ and therefore $a \mid c$.

Lemma 2. If p is a prime and $p \mid a_1 a_2 \dots a_n$ and each a_i is an integer, then $p \mid a_i$ for some i .

This is a generalization of Lemma 1. If for $n-1$ out of n values a_i (say all except a_k) $p \nmid a_i$, then we also have $\gcd(p, a_i) = 1$ for those values, but by Lemma 1, since $p \mid a_1 a_2 \dots a_n$, it must be the case that $p \mid a_k$. Of course it might also be that $p \mid a_k$ for more than one subscript k .

Proof of uniqueness of prime factorization. Say $n = p_1 p_2 \dots p_s$ and also $n = q_1 q_2 \dots q_t$, with both sets of primes in increasing order. First remove all common primes from these two sequences, and if there are any left renumber them and we will have fewer primes on each side (r and v) $p_1 p_2 \dots p_r$ and also $n = q_1 q_2 \dots q_v$, where no identical prime appears on both sides of this equation. By Lemma 2 it follows that p_1 divides q_i for some i . But since there are no common primes and no prime divides another prime, this is impossible.

I am going to skip Theorem 2.

Linear Congruences

A congruence of the form $ax = b \pmod{m}$, where m is a positive integer, a and b are integers, and x is a variable, is called a *linear congruence*. We wish to find all integers x that satisfy this congruence.

The typical way to solve for x is to find an *inverse*, that is a **multiplicative inverse**, of the number a , written a^{-1} (it was written a with a bar over it in the text) such that $a^{-1}a = aa^{-1} = 1 \pmod{m}$

Theorem 3. If a and m are relatively prime integers and $m > 0$, then the inverse of a modulo m exists and is unique. That is, there is a unique positive integer a^{-1} less than m that is the inverse of $a \pmod{m}$, and every other inverse of a modulo m is congruent to $a^{-1} \pmod{m}$.

Proof. By Theorem 1 of this chapter, because $\gcd(a, m) = 1$, there are integers s and t such that $sa + tm = 1$.

This implies that: $sa + tm = 1 \pmod{m}$, but because $tm = 0 \pmod{m}$ this means $sa = 1 \pmod{m}$. Thus $s = a^{-1}$. That this inverse is unique modulo m is the topic of Exercise 9 of this Section.

Example 3. Find an inverse of 3 modulo 7. (Note that 7 is a prime so $\gcd(k, 7) = 1$ if k is not equal to 0 modulo m . Use the Euclidean algorithm to find $\gcd(7, 3)$, dividing 7 by 3, and we get $7 = 2 \times 3 + 1$, or $-2 \times 3 + 1 \times 7 = 1$, so -2 (or 5 modulo 7) is the inverse of 3, e.g. $3 \times 5 = 1 \pmod{7}$.

When we speak of modulo m and m is a prime, then every modulo class has an inverse. These integer modulo classes $(0, 1, \dots, 6 \pmod{7})$ thus form a finite field. You can add, there is an additive identity 0, there is an additive inverse, you can multiply there is a multiplicative identity 1, and there is always a multiplicative inverse.

Example 4. What are the solutions of the linear congruence $3x = 4 \pmod{7}$? In example 3, we saw that -2 (or 5) is the inverse of 3, so $x = -2 \times 4 = -8 \pmod{7}$ so $x = -8 = -1 = 6 \pmod{7}$. Clearly -8 and -1 and 6 and 13 and... are all in the same modulo class relative to 7, all solutions to x .

Theorem 5. Fermat's Little Theorem. If p is a prime and a is an integer not divisible by p then

$$a^{p-1} = 1 \pmod{p} \quad \text{This is the same as } a^p = (a \pmod{p})$$

Difficult proof: Exercise 17, pg. 244.

Example: $3^6 = 1 \pmod{7}$ $3^2 = 9 = (2 \pmod{7})$; $3^6 = 2^3 = 8 = 1 \pmod{7}$

Class 12.

I am going to skip **Section 3.8: Matrices** since CS/Math320 has a prerequisite of Math 260, Linear Algebra. I will not hold you responsible for matrices on Quizzes and Exams. Recall that we already covered **Section 4.1: Mathematical Induction**, although we skipped Examples 7, 8, 10 and the rest of the Section.

4.2 Strong Induction and Well-Ordering

Strong Induction is something you might have thought was included in Induction when you first learned it, but in fact the definition was rather limited.

Mathematical Induction: To prove the propositional function $P(n)$ is true for all positive integers n we complete two steps: (1) Prove $P(1)$ is true, (2) Assuming $P(k)$, show this implies $P(k+1)$ for all positive integers n .

Strong Induction modifies step (2): given the truth of $P(1) \wedge P(2) \wedge \dots \wedge P(k)$, we show this implies $P(k+1)$ is true. The difference is that we can use not just that $P(k)$ is true but that $P(i)$ is true for $i = 1, 2, \dots, k$.

Example 3. Consider a game where two players start with two rows of the same number of matches, and each in turn can remove any (positive) number of matches from one row. The winner is the player removing the last match. Then player 2 can win by matching the player 1's move in a different row, so both rows end with the same number after the second play.

Let $P(n)$ be true if the second player can win starting with n matches in each row.

Proof by induction. If we start with 1 match in each row the second player wins, since player 1 must remove the first match and player 2 can remove the second and last. Thus $P(1)$ is true.

Assume $P(k)$: player 2 can win starting with any number of matches in each row up to k . Then prove for $P(k+1)$. Assume player 1 removes some number of matches reducing $(k+1)$ in a row to some smaller number m . If player 2 then removes the same number of matches we either reduce the problem to $P(m)$, $m < k+1$, or if $m = 0$, the second player will have already won.

Generalization. Is there a consistent winner starting with 4 rows with an equal number of matches? What about other even numbers? What about three rows? (There's a simple answer.)

Example 4. Prove that every amount of postage of 12 cents or more can be formed using just 4-cent and 5-cent stamps.

Proof. Let $P(n)$ be the statement that either $n < 12$ or $n \geq 12$ and the postage for n can be constructed using 4-cent and 5-cent stamps. (I don't see how the text's statement worked.)

Basis step. Let us show $P(12)$, $P(13)$, $P(14)$ and $P(15)$ are true with constructions: $(4,4,4)$, $(4,4,5)$, $(4,5,5)$, $(5,5,5)$.

Induction Step. Now for $n \geq 16$, we can prove $P(n)$ by considering the construction for $P(n - 4)$ (with $n \geq 12$) and adding a 4-cent stamp.

The Well-Ordering Property introduces a seemingly trivial axiom. Every non-empty set of nonnegative integers has a least (smallest) element. This is obvious for a finite set, but perhaps harder to see for an infinite set of nonnegative integers (depending on how the set is described).

Example 5. Recall the division algorithm states that if a is an integer and d is a positive integer, then there are unique integers q and r with $0 \leq r < d$ and $a = dq + r$. We can show the uniqueness of r by using the well-ordering property,

Proof. Let S be the set of nonnegative integers of the form $a - dq$ (value of $r \bmod d$), where q is an integer. If a is positive then so is $a - dq$ with $q = 0$, and we can make q larger as long as $a - dq$ remains positive (until we get $0 \leq r = a - dq < d$); if a is negative, then $a - dq$ with q negative as well will be increasing as $|q|$ increases, so sooner or later $a - dq$ will represent positive integers. Taking the smallest nonnegative integer of the form $r = a - dq$ will give an r with $0 \leq r < d$, because if $r \geq d$ we can reduce $a - dq$ by d by increasing or decreasing q by 1.

Example 6 is rather hard to grasp, but you might want to look at it if you enjoy this sort of proof. Skip Strong Induction in Computational Geometry and the rest of the Section.

4.3 Recursive Definitions and Structural Induction

In a typical recursive definition of a function $f(n)$ on positive integers n , the values of $f()$ at the first k positive integers is defined, and a rule is given for determining the value of the function at larger integers from its values at some or all of the preceding k integers.

Example 2. $F(n) = n!$ can be defined as follows: $F(0) = 1$ and $F(n) = n \times F(n-1)$. This allows us to evaluate a recursive function to return $n!$: $F(5) = 5 \times F(4) = 5 \times 4 \times F(3) = \dots = 5 \times 4 \times 3 \times 2 \times 1 \times F(0) = 5 \times 4 \times 3 \times 2 \times 1 \times 1 = 120$.

Definition 1. The Fibonacci Numbers, f_0, f_1, f_2, \dots are defined as follows: $f_0 = 0, f_1 = 1$, and

$$f_n = f_{n-1} + f_{n-2}, \quad \text{for } n = 2, 3, 4, \dots$$

E.g.

$$f_2 = f_1 + f_0 = 0 + 1 = 1,$$

$$f_3 = f_2 + f_1 = 1 + 1 = 2,$$

$$f_4 = f_3 + f_2 = 2 + 1 = 3,$$

$$f_5 = f_4 + f_3 = 3 + 2 = 5,$$

$$f_6 = f_5 + f_4 = 5 + 3 = 8. \text{ Then } 13, 21, 34, 55, \dots$$

Example 6. Show that for $n \geq 3, f_n > \alpha^{n-2}$, where $\alpha = (1 + \text{SQRT}(5))/2$.

We use strong induction, starting with $n = 3$. First note: $\alpha < 2 = f_3, \alpha^2 = (3 + \text{SQRT}(5))/2 < 3 = f_4$, So $P(3)$ and $P(4)$ are true.

Inductive step. Assume $P(j)$ is true for all integers $j, 3 \leq j \leq k$, and show $P(k+1)$ is true. First note $\alpha = (1 + \text{SQRT}(5))/2$ is a solution of $x^2 - x - 1$ (Solutions are: $((-b \pm \text{SQRT}(b^2 - 4 \times a)) / (2 \times a))$ and in the $+$ case we have $(1 + \text{SQRT}(1 + 4))/2$.) Thus we have: $\alpha^2 = \alpha + 1$. Thus, multiplying both sides by α^{k-3} , we get $\alpha^{k-1} = \alpha^{k-2} + \alpha^{k-3}$.

But recall we have that $f_j > \alpha^{j-2}$ up to $j = k$ and wish to show it for $k+1$. This can be rewritten:

$$f_{k+1} = f_k + f_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1}.$$

Note that the quantity α is called the "Golden Mean" because if we make a rectangle of sides 1 and α , then cut a square of side 1 out of one end of the rectangle, the resulting rectangle still has a ratio of edge lengths 1: α . Doing this continually, one can create a spiral, and this ratio was felt to be particularly pleasing by the Greeks.

I do not go into Lamé's Theorem, but it gives an upper bound to the number of successive divisions that might be needed in finding $\text{gcd}(a, b)$ with the Euclidean Algorithm. The number of divisions is less than or equal to five times the number of decimal digits in the larger a or b .

Recursively Defined Sets and Structures

Consider the set S of integers defined by: BASIS STEP: $3 \in S$; RECURSIVE STEP: If $x \in S$ and $y \in S$ then $x + y \in S$.

It would be a good quiz or Exam question to demonstrate that S is the set of all integers divisible by 3. Note you are showing two things: (1) if $x \in S$ then x is divisible by 3, (2) if y is an integer divisible by 3 then $y \in S$.

Definition 2. The set Σ^* of strings over the alphabet Σ is defined as follows. BASIS STEP: $\lambda \in \Sigma^*$ where λ is the empty string containing no symbols. RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Example 8. If $\Sigma = \{0, 1\}$, then Σ^* consists of the empty string λ and all possible strings of 0's and 1's.

I skip Example 9 and thereafter, including Structural Induction and Generalized Induction.

4.4 Recursive Algorithms

Definition 1. An Algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.

Example 1. Give a recursive algorithm for computing $n!$, where n is a non-negative integer. I provide the function factorial coded in the C language.

```
int factorial(int n)
{ /* There's high risk of int overflow if calculate n! See http://en.wikipedia.org/wiki/Factorial */
  if (n == 0) return 1;
  else return n*factorial(n - 1); /* All C functions can call themselves with new arguments */
} /* 12! is last value that fits in 32 bit int, 20! is last int that fits in 64 bit int. */
```

Of course a more efficient function (because recursion is very CPU-intensive) is this:

```
int factorial(int n)
{ /* Same risk here of int overflow, of course. */
  if (n == 0) return 1;
  for(i = 1, m = 1; i <= n; i++)
    m = i*m;
  return m;
}
```

Example 3. Devise a recursive algorithm for computing $b^n \bmod m$, where b , n , and m are integers with $m \geq 2$, $n \geq 0$ and $1 \leq b \leq m$.

A rather inefficient solution uses the fact that $b^n \bmod m = (b \times (b^{n-1} \bmod m)) \bmod m$. (Note that even for a large integer a , $a \bmod m$ is always an integer i with $1 \leq i < m$, the remainder when a is divided by m . Thus for example if $b = 24$ and $m = 17$, $b^1 \bmod m = 24 \bmod 17 = 7$, and $(b \times (b^1 \bmod m)) \bmod m = (24 \times (24 \bmod 17)) \bmod 17 = (24 \times 7) \bmod 17 = (168 \bmod 17) = 15 \bmod 17$ (because $168 = 9 \times 17 + 15 = (153 + 15)$).

In fact, if we're taking everything mod 17, we don't ever have to use a factor larger than 17. Since $24 = 7 \bmod 17$, $24^2 = 7^2 \bmod 17 = 49 \bmod 17 = 15 \bmod 17$ because $49 = 2 \times 17 + 15$.

A much more efficient recursive algorithm can be based on the observation that:

$$b^n \bmod m = (b^{n/2} \bmod m)^2 \bmod m, \text{ when } n \text{ is even and}$$

$$b^n \bmod m = (((b^{\text{FLOOR}(n/2)} \bmod m)^2 \bmod m) \times b) \bmod m, \text{ when } n \text{ is odd}$$

Algorithm 3 Recursive Modular Exponentiation (in C language)

```
int mpower(int b, int m, int n) /* integers m ≥ 2, n ≥ 0, b > 0 */
{
    if (n == 0) return 1;
    if (n%2 == 0) return(((mpower(b, n/2, m)*mpower(b, n/2, m))%m); /* %m means mod m */
    else return((((mpower(b, (n-1)/2, m)*mpower(b, (n-1)/2, m))%m)*(b%m))%m);
}
```

Algorithm 4 Recursive Algorithm to compute gcd(a, b) (in C language)

```
int gcd(int a, int b) /* integers a and b with 0 ≤ a < b */
{
    if (a == 0) return b; /* gcd(0, b) = b */
    return(gcd(a, b%a));
}
```

I skip a few Examples and the approach to proving recursive algorithms correct. Typically, such proofs lead to algorithms that don't work properly or don't even compile.

The Merge Sort Algorithm

I will skip a formal description of the algorithm and will merely describe it. It depends upon the idea that if one has two sorted lists L1 and L2, it is easy to merge the two into a sorted list L.

One starts with a cursor at the beginning of each list, finds the smaller of the two elements under the two cursors, then moves that element to be the first (next) element in L, and advances the cursor over that element to the next element in the list. Repeating this, one ends with one of the two cursors having run out of elements; then move all remaining elements in the other list to L.

We start by merging successive pairs of lists of length 1 to get lists of length 2, then merging pairs of length 2 to get lists of length 4, then 8, then 16. Lists of length 2 requires n passes, so if we start with M elements, we can sort it in log₂ M passes, that is in time O(M log M).

Example Of 2-Way Merge Sort

Initially	14	12	5	9	2	16	1	10	7	3	11	8	13	4	6	15
After Pass 1	12	14	5	9	2	16	1	10	3	7	8	11	4	13	6	15
After Pass 2	5	9	12	14	1	2	10	16	3	7	8	11	4	6	13	15
After Pass 3	1	2	5	9	10	12	14	16	3	4	6	7	8	11	13	15
After Pass 4	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

When one sorts on disk, access is VERY SLOW compared to memory, so want to reduce the number of passes as much as possible, to 2 if possible. Start by sorting in memory and write long (100 MByte) sorted lists. Then, instead of two way merge, use K-way merge. Maintain K cursors and place values under each cursor into a data structure called a *heap*.

See: [http://en.wikipedia.org/wiki/Heap_\(data_structure\)#Variants](http://en.wikipedia.org/wiki/Heap_(data_structure)#Variants) (Note we can have min at top instead of max to perform Merge above.)