EECS 376: Foundations of Computer Science
University of Michigan, Winter 2024

Solutions to Homework 1

This homework has 9 questions, for a total of 100 points and 10 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

(a) Carefully read Handout 1 before starting this assignment, and apply it to the solutions you submit.

(b) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

1. **Properties of logarithms.**

The objective of this problem is to recall some important properties of logarithms that we'll use throughout the course. First recall the following properties of exponentiation for any real numbers b, x, y, assuming that $b \neq 0$:

P1: $(b^x)^y = b^{xy}$

P2: $b^x = b^y$ if and only if $x = y$, additionally assuming that $b \neq 1$.

Now, for positive real numbers $b \neq 1$ and x, the definition of the base-b logarithm is:

D1: $y = \log_b x$ is the unique real number for which $b^y = x$. That is, it satisfies the identity $b^{\log_b x} = x$. (Uniqueness of this value follows directly from property P2.)

Combining D1 with P2, we see that $x = y$ if and only if $\log_b x = \log_b y$ (assuming that b, x, y are positive and $b \neq 1$). So, equality is preserved by taking logarithms or exponentials (with the same base) on both sides of an equation, assuming the hypotheses are met.

Prove each of the following properties of logarithms, for any positive real numbers (*not necessarily integers*) a, b, c, n where $b, c \neq 1$. Cite definition D1 and properties P1 and P2 where you use them; you may also use (with citation) the results of previous parts of the problem. Do not use any other assumptions about, or properties of, logarithms and exponentials.

For reference, we have provided a solution to the first part, which you can use as a model for later solutions.

(0 pts) (a) $\log_b(a^n) = n \log_b a$. (Be careful: n is not necessarily an integer!)

Solution: Notice that n is not necessarily an integer; the problem stipulates only that it is a positive real number.

By P2, the statement we want to prove is equivalent to $b^{\log_b(a^n)} = b^{n \log_b a}$ (since $b \neq 1$). By D1, the left-hand side is equal to a^n, so it suffices to prove that $a^n = b^{n \log_b a}$. Indeed,

$$b^{n \log_b a} = (b^{\log_b a})^n = a^n,$$

where the first equality is by P1, and the second equality is by D1.

(As a remark, this proof does not rely on the assumption that n is positive, so the statement holds even more generally for any real number n. However, later parts require n to be positive.)

(5 pts) (b) $\log_b a = \frac{\log_c a}{\log_c b}$. (This is sometimes called the "change-of-base theorem," because it lets us change the base of the logarithm by dividing by an appropriate factor. Note that $\log_c b \neq 0$ because $b \neq 1$, so the division is well defined.)

Solution: By D1, we have that $a = b^{\log_b a}$. Taking the base-c logarithm of both sides (which, by D1 and P2, preserves equality) and applying part (a) above, we get that

$$\log_c a = \log_c(b^{\log_b a}) = (\log_b a) \cdot (\log_c b).$$

Dividing both sides by $\log_c b$ yields the desired equality.

(5 pts) (c) The ratio $\frac{\log_b n}{\log_c n}$ (for $n \neq 1$) depends only on b and c, not n.
This implies that we can often omit the (constant) base when writing logarithms in asymptotic notation, e.g., we can write $O(\log n)$ instead of $O(\log_2 n)$, because $O(\cdot)$ hides constant factors that do not depend on n. However, we cannot omit the base when the logarithm appears in the exponent, as in $O(3^{\log_2 n})$.

Solution: By applying part (b) twice, we get that

$$\frac{\log_b n}{\log_c n} = \frac{\log_2 n / \log_2 b}{\log_2 n / \log_2 c} = \frac{\log_2 c}{\log_2 b},$$

which depends only on b and c, as desired.

2. **Practice with asymptotics ("big-Oh, big-Omega, big-Theta").**

Let $f(n)$ and $g(n)$ be positive functions. Recall that $f(n) = O(g(n))$ if there exist constants $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

In words, we often phrase this as: "$f(n)$ is *at most* (or: is *upper-bounded by*) a *constant multiple* of $g(n)$, for *all large enough* n." ("All large enough n" refers to the clause "all $n \geq n_0$", i.e., all n beyond some fixed threshold.)

A *sufficient* condition for $f(n) = O(g(n))$ is that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists and is finite. However, this condition is not *necessary*; it may be that $f(n) = O(g(n))$ even if the limit does not exist.

Similarly, $f(n) = \Omega(g(n))$ if there exist constants $c, n_0 > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

- In words: "$f(n)$ is *at least* (or: is *lower-bounded by*) a (positive) constant multiple of $g(n)$, for all large enough n."
- A sufficient condition is that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists and is nonzero (this includes the case where the limit is infinite).
- It can be seen that $f(n) = \Omega(g(n))$ is equivalent to $g(n) = O(f(n))$, since $f(n) \geq c \cdot g(n)$ if and only if $g(n) \leq (1/c) \cdot f(n)$, and c is a positive constant if and only if $1/c$ is one.

Finally, $f(n) = \Theta(g(n))$ if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

For the following pairs of functions, state with justification whether or not each of the following hold: $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$.

Hint: You might find L'Hôpital's Rule useful: it says that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$ when the latter limit exists, where f' and g' are respectively the derivatives of f and g.

(5 pts) (a) $f(n) = 3n^2 + 10n + 6$, $g(n) = n^2$.
(Optional challenge: generalize to similar-looking f and g, beyond specific constant coefficients or exponents.)

Solution: By noticing that the dominant terms of f and g, ignoring constant factors, are both n^2, we can conjecture that $f(n) = \Theta(g(n))$. This is indeed the case, which we formally prove by showing both required conditions.

To prove that $f(n) = \Omega(g(n))$, we can simply take constants $c = n_0 = 1$; it is immediate that $f(n) \geq 1 \cdot g(n)$ for all $n \geq 1$.

To prove that $f(n) = O(g(n))$, we can take constants $c = 4$ (chosen to be slightly larger than f's coefficient of n^2) and $n_0 = 11$ (chosen to be slightly larger than all of f's other coefficients). Then we wish to show that for all $n \geq 11$, we have $f(n) \leq 4g(n)$, i.e., $3n^2 + 10n + 6 \leq 4n^2$. By rearranging, this is equivalent to showing that $n^2 \geq 10n + 6$ for all $n \geq 11$. Indeed, for all $n \geq 11$ we have that

$$n^2 > n^2 - 1 = (n-1)(n+1) \geq 10(n+1) > 10n + 6.$$

Alternatively, we could have used limits to very concisely prove both required conditions. Applying L'Hôpital's Rule twice, we get that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f''(n)}{g''(n)} = \lim_{n\to\infty} \frac{6}{2} = 3,$$

which is finite (implying that $f(n) = O(g(n))$) and nonzero (implying that $f(n) = \Omega(g(n))$), as needed.

More generally, if $f(n), g(n)$ are polynomials in n of the same (finite) degree, then $f(n) = \Theta(g(n))$. Try to prove this!

(5 pts) (b) $f(n) = \log_2(n^2)$, $g(n) = \log_{10}(n^{10})$.
(Optional challenge: generalize to similar-looking f and g, beyond specific constant bases and exponents.)

Solution: All three conditions hold, i.e., $f(n) = \Theta(g(n))$. By the previous problem on logarithms, $f(n) = 2\log_2 n$ and $g(n) = 10\log_{10} n = \frac{10}{\log_2 10} \cdot \log_2 n$. So $f(n)/g(n) = (\log_2 10)/5$ is a nonzero constant, and therefore $f(n) = \Theta(g(n))$.

More generally, if $f(n), g(n)$ are logarithms, with *any* (not necessarily the same) constant bases greater than one, of *any* polynomials in n (not necessarily of the same degree), then $f(n) = \Theta(g(n))$. Try to prove this!

(5 pts) (c) $f(n) = n^5 \cdot 2^{3n}$, $g(n) = 3^{2n}$.

(Optional challenge: generalize to similar-looking f and g, beyond specific constants in the bases and exponents.)

Solution: Notice that $f(n) = n^5 \cdot 8^n$ and $g(n) = 9^n$; we will use these forms in what follows.

We first show that $f(n) = O(g(n))$. We give a proof directly according to the definition, by giving suitable constants c, n_0 and showing that $n^5 \cdot 8^n \leq c9^n$ for all $n \geq n_0$. Letting $c = 1$ for simplicity, what we want to show is equivalent to $n^5 \leq (9/8)^n$ for all $n \geq n_0$. Using a calculator, we see that this inequality holds at $n = 232$, so we take $n_0 = 232$. Then we observe that beyond this threshold, the right-hand side grows much more quickly than the left-hand side: as n increases by one, the right-hand side grows multiplicatively by a factor of $9/8 = 1.125$, while the left-hand size grows by a factor of *at most* $(233/232)^5 \leq 1.022$. So, the inequality does indeed hold for all $n \geq n_0$.

Now we show that $f(n) \neq \Omega(g(n))$. We give a proof according to the definition. We need to prove the logical negation of the statement "$n^5 8^n = \Omega(9^n)$," which is: *for all* positive constants c, n_0, we have $n^5 8^n < c \cdot 9^n$ for *some* $n \geq n_0$. So, letting $c, n_0 > 0$ be some arbitrary constants, we need to derive some $n \geq n_0$ *which may depend on* c, n_0, for which $n^5 < c(9/8)^n$. The key intuition is that the exponential on the right eventually grows arbitrarily larger than the polynomial on the left, so some large enough n does indeed do the job.

More rigorously, with the help of a computer algebra system like Mathematica, we can determine that the inequality $n^5 < c(9/8)^n$ holds whenever n is greater than some complicated-looking expression we call K_c, which involves various logarithms and esoteric-sounding "product log functions." But the key point is that K_c *depends only on* c, and hence is a constant because c is a constant. Therefore, by taking any $n > \max(n_0, K_c)$, we do indeed satisfy the inequality for some $n \geq n_0$, as needed.

Alternatively, we can approach this without much difficulty using limits. Note that in $f(n)/g(n)$, we can move factors between the numerator and denominator as is convenient, to get nicer derivatives when applying L'Hôpital's Rule. In particular,

$$\frac{f(n)}{g(n)} = \frac{n^5 \cdot 8^n}{9^n} = \frac{n^5}{(9/8)^n}$$

so

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^5}{(9/8)^n} = \lim_{n\to\infty} \frac{5n^4}{(9/8)^n \ln(9/8)} = \lim_{n\to\infty} \frac{20n^3}{(9/8)^n \ln^2(9/8)}$$
$$= \cdots = \lim_{n\to\infty} \frac{c}{c'(9/8)^n} = 0 \,.$$

for constants $c = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ and $c' = \ln^5(9/8)$. Since the limit exists and is zero, we have that $f(n) \neq \Omega(g(n))$. (To be precise, the problem preamble did not state that when the limit exists, a nonzero limit is *necessary* for $f(n) = \Omega(g(n))$, only that it is *sufficient*. But this is indeed true, and it was stated in discussions and Piazza that it could be used.)

3. **Comparing asymptotic running times.**

Suppose that Algorithm X has running time $T_X(n) = O(n^2)$, Algorithm Y has running time $T_Y(n) = \Theta(n \log n)$, and Algorithm Z has running time $T_Z(n) = \Theta(n)$.

As usual, all running times are stated in terms of the *worst case* for inputs of size n. That is, $T_X(n)$ is the *maximum* number of steps for which X runs, taken over all inputs of size n (and similarly for T_Y, T_Z).

Solution: Before considering the specific claims, we highlight the main conceptual points for this problem:

- An algorithm can take varying amounts of time on different inputs, even ones of the same size. (For example, consider insertion sort, which is fast when its input array is already sorted, but much slower when its input is far from sorted.) In this class, we care mainly about the *worst-case* running time over all inputs of a particular size, so that we have a runtime guarantee no matter what the input is. But on certain inputs, an algorithm might perform better than its worst-case behavior.
- An $O(\cdot)$ bound is merely an (asymptotic) *upper bound*. A function that grows much more slowly than $g(n)$ is still considered to be $O(g(n))$. Similarly, an $\Omega(\cdot)$ bound is merely an (asymptotic) *lower bound*; a function that grows much more quickly than $g(n)$ is still considered to be $\Omega(g(n))$. By contrast, a $\Theta(\cdot)$ bound is both an (asymptotic) upper *and* lower bound. Any function that is $\Theta(g(n))$ must be 'sandwiched' between two constant multiples of $g(n)$ (above a certain threshold for n), i.e., it must scale just as $g(n)$ does, as n grows large.
- Asymptotic bounds like $O(\cdot)$ and $\Omega(\cdot)$ say how a function (often, an algorithm's runtime) scales *as the input size n grows large*, beyond some threshold (denoted n_0). They do not say anything about the function's behavior for "small" values of n (below the threshold). Without any additional hypotheses, the function's behavior below the threshold can be arbitrary.

(10 pts) (a) Consider the claim: "on every input, Y runs faster than X." Clearly explain why this claim is (i) not necessarily true, and (ii) not necessarily false. To do this, provide concrete running times that are consistent with the stated asymptotic bounds, and which make the claim true or false, as appropriate.

Solution: The claim is not necessarily true; it can be false since X might be faster than Y for some *small* inputs (i.e., small input size n). For example, it could be the case that $T_X(n) = n^2$ and $T_Y(n) = 100n \log_2 n$. Then on *any* input of size $n = 2$, X runs in time *at most* $T_X(2) = 4$. By contrast, on a *worst-case* input of size $n = 2$, Y runs in time *exactly* $T_Y(2) = 100 \cdot 2 \cdot 1 = 100$. So, X is faster than Y on such an input.

Important subtlety: For the previous argument, we need to consider a *worst-case* input to Y. This is because $T_Y(n)$ refers to Y's *maximum* running time over all inputs of size n; it says nothing about how Y performs on non-worst-case inputs. So, even if we know that $T_X(n) < T_Y(n)$ for some small value of n, we still cannot conclude that X is faster than Y on *every* size-n input. Y might still run faster than X on some of these inputs, but not all of them.

The claim is not necessarily false; it can be true because on *all* inputs of size n, X might run in time exactly n^2 (which qualifies as $O(n^2)$), and Y might run in time at most $n \log_2 n$ (which qualifies as $O(n \log n)$). Then because $n > \log_2 n$ for all $n \geq 1$, Y runs faster than X on every input.

Important subtlety: For the previous argument, we need to assume something about the running time of X for *all* inputs, not just about T_X. This is because $T_X(n)$ merely refers to the *worst-case* running time of X over all inputs of size n; it says nothing about how X performs on other (non-worst-case) inputs. So, even if we know that $T_X(n) > T_Y(n)$ for all n, we still cannot conclude that Y is faster than X on *all* inputs, only that this is the case for X's worst-case inputs. It could still be that X is faster than Y on other inputs.

(10 pts) (b) Consider the claim: "for all large enough n, there is an input of size n for which Y runs faster than X." Clearly explain why this claim is (i) not necessarily true, and (ii) not necessarily false.

Solution: The claim is not necessarily true; it can be false because the $O(n^2)$ upper bound on $T_X(n)$ could be very *loose*, and X might actually run in much faster than quadratic time (for all inputs). For example, it could be the case that $T_X(n) = 10 = O(n^2)$, whereas Y takes time $100n \log_2(2n) = \Theta(n \log n)$ for *every* input of size n. Then because $100n \log_2(2n) \geq 100n > 10$ for every $n \geq 1$, X is faster than Y on *every* input.

Important subtlety: For the previous argument, we need to assume something about the running time of Y for *all* inputs, not just about T_Y. This is for the same reason given in the previous part: we need a lower bound on Y's runtime for *every* input, but T_Y refers only to Y's worst-case runtime.

The claim is not necessarily false; it can be true because the $O(n^2)$ upper bound on

$T_X(n)$ might actually be tight. For example, it could be that $T_X(n) = n^2 = O(n^2)$ and $T_Y(n) = n \log_2 n = \Theta(n \log n)$. Then for each value of n, there is some *worst-case* input x_n for X, i.e., X run on x_n takes time exactly n^2. For this input, Y takes time *at most* $n \log_2 n$. Because $n \log_2 n < n^2$ for all $n \geq 1$, Y is faster than X on all the inputs x_n.

Important subtlety: For the previous argument, we need to consider *worst-case* size-n inputs for X. (An alternative way to do this is by adopting an example where X takes time $T_X(n)$ for *every* size-n input; this makes every input a worst-case one.) This is for the same reason discussed previously: we need a lower bound on X's runtime for certain inputs, and $T_X(n)$ is merely the *worst-case* runtime of X on size-n inputs; X might run much faster on non-worst-case inputs.

(10 pts) (c) Consider the claim: "for all large enough n, there is an input of size n for which Z runs faster than Y." Is this claim necessarily true, necessarily false, or not necessarily either? Clearly justify your answer.

Solution: This is necessarily true. For each value of n, there is some *worst-case* size-n input y_n for Y, i.e., Y run on y_n takes time exactly $T_Y(n)$. Then because $T_Y(n) = \Omega(n \log n)$, there is a positive constant $c > 0$ such that for all large enough n, Y run on y_n takes time *at least* $cn \log_2 n$. Because $T_Z(n) = O(n)$, there is a positive constant $c' > 0$ such that for all large enough n, Z run on y_n takes time *at most* $c'n$. Finally, because $c'n < cn \log_2 n$ for all large enough n (namely, when $n > 2^{c'/c}$), we conclude that Z runs faster than Y on y_n for all large enough n.

Important subtlety: For the previous argument, we need consider *worst-case* size-n inputs to Y, not arbitrary ones. This is because we need a lower bound on Y's runtime for certain inputs. As discussed in the previous parts, $T_Y(n)$ represents Y's runtime on worst-case size-n inputs, and Y could be much faster on non-worst-case inputs.

As a remark, it is important to carefully understand what the claim does *not* say:

- It does not say that Z is faster than Y on *all* inputs, or even those of large enough size. Indeed, it could be the case that Z is *slower* than Y on almost all inputs—say, more than 99.9% of them!

  Again, $T_Y(n)$ and $T_Z(n)$ only refer to the *worst-case* running times of Y and Z on size-n inputs, not the "best case" or "typical case."

- It does not say that for *every* n, there is a size-n input on which Z is faster than Y. Indeed, it could be the case that Z is *slower* than Y for *all* inputs of size up to (say) 2^{1000}, which is vastly more than the number of atoms in the universe!

  The asymptotic expressions only say what happens for "large enough" values of n, i.e., as n grows toward infinity; but they do not say what the specific threshold of "large enough" is.

(10 pts) 4. **Homework solver!**

Consider the following homework solver:

```
function SOLVE-HW(X):              // X is a non-negative integer
   for i = 1 to floor(X^(1/3)):    // floor of cube root of X
      print('hey grader, give me ' + X + ' points')
   return X
```

Give an asymptotic (big-O) bound on the worst-case number of characters printed by the program as a function of the *size* of the (non-negative integer) input X, written in binary. Make this bound as tight as possible. Is the program "efficient," as defined by this course?

(Note: our providing this program does not constitute a guarantee of any particular lower bound on the number of points you should expect to receive if you choose to run it for your own homeworks.)

Solution: Recall that the size of an integer input is the *number of digits* in its binary representation (or decimal, or any other fixed base greater than one). Let n denote the number of digits in the binary representation of X.

Then each iteration prints out $n + 27 = \Theta(n)$ characters. (The exact count $n + 27$ assumes that X is printed in binary, but the ultimate $\Theta(n)$ bound is valid no matter what constant base greater than one is used, by the earlier problem on logarithms.) The same also goes for any constant number of extra characters printed around X.) Since there are $\lfloor \sqrt[3]{X} \rfloor = \Theta(\sqrt[3]{X})$ iterations, the function prints $\Theta(n\sqrt[3]{X})$ characters in total.

Because X is an n-bit integer, it may be as large as $2^n - 1$. The runtime is dominated by the number of characters it prints, so written purely in terms of the input size, the runtime is $\Theta(n\sqrt[3]{2^n}) = \Theta(n \cdot 2^{n/3})$. So, this program runs in *exponential* time in its input size n, and therefore it is *not* efficient.

As a remark, we instead could have measured the size of the input X in *decimal* (or any other base greater than one). In this case, X could be as large as $10^n - 1$, which results in a worst-case runtime $\Theta(n \cdot 10^{n/3})$, which is still exponential and hence *not* efficient. In general, regardless of which constant bases (greater than one) we use for measuring the input size or for printing X, this algorithm will be exponential time, and hence not efficient as defined in this class.

(10 pts) 5. **Practice with induction.**

Use induction to prove that for any integer $n \geq 0$, the number of binary strings having length *at most* n (including length zero) is exactly $2^{n+1} - 1$.

(You may use the fact that there are exactly 2^k binary strings of length exactly k, which was proven in discussion.)

Solution: Base case: for $n = 0$, there is exactly one string of length zero (the empty string), and $2^{n+1} - 1 = 2 - 1 = 1$, as needed.

Now as the inductive hypothesis, assume that the statement is true for some $n \geq 0$. We will prove that it is true for $n + 1$, i.e., that there are $2^{n+2} - 1$ strings of length at most $n + 1$.

Each string of length at most $n+1$ either has length *exactly* $n+1$, or has length *at most* n (and there are no overlaps between these two cases). There are 2^{n+1} of the former strings, and by the inductive hypothesis, there are $2^{n+1} - 1$ of the latter strings. So in total there are $2^{n+1} + 2^{n+1} - 1 = 2 \cdot 2^{n+1} - 1 = 2^{n+2} - 1$ strings of length at most $n + 1$, as desired.

(10 pts) 6. **Representing data.**

Suppose you want to unambiguously represent the elements of $\{0, 1, \ldots, n-1\}$ (for some positive integer n) using ternary strings of a certain length k, i.e., strings with exactly k characters from the set $\{a, b, c\}$. What is the *smallest* possible choice of k for which this is possible?

For example, to represent the elements of $\{0, 1, 2, 3, 4, 5\}$, we can use the ternary strings aa, ab, ac, ba, bb, and bc of length $k = 2$. This is the smallest length possible to uniquely represent these elements, because there are only 3 ternary strings of length $k = 1$.

Solution: We want to determine the smallest possible k that allows us to represent n elements using distinct strings of length k. First observe that for any integer $k \geq 0$, the number of distinct strings of length k is exactly 3^k. To see this, note that such a string has three possibilities for each of the k positions, so the total number of possibilities is $\underbrace{3 \times \cdots \times 3}_{k \text{ times}} = 3^k$. (For length $k = 0$, we have just the single "empty" string ε.)

So, to be able to represent each of the n elements by a distinct string of length k, it is both necessary and sufficient to choose k so that

$$3^k \geq n.$$

By taking the base-3 log of both sides, this is equivalent to

$$k \geq \log_3 n.$$

Since k must be an integer, the smallest k we can use is therefore $k = \lceil \log_3 n \rceil$.

7. **Proofs: direct, and by contrapositive.**

Recall that a real number is *rational* if it can be written as p/q for some integers p, q, where $q \neq 0$.

(5 pts) (a) Give a direct proof that if $r \neq 0$ is rational, then $1/r$ is rational. What is the contrapositive of this statement? (There is more than one valid way to write it.)

Solution: If $r \neq 0$ is rational then $r = p/q$ for some integers p, q, with $p \neq 0$ (because $r \neq 0$). Then $1/r = q/p$ is also rational, because it is a ratio of two integers. (Here we are using the fact that $p \neq 0$.)

There are a couple valid ways to express the contrapositive, depending on how one formally quantifies the original statement.

One form of the contrapositive is "If $1/r$ is irrational, then r is irrational or $r = 0$." However, notice that in order for the $1/r$ in the hypothesis to make sense, we must (implicitly) assume that $r \neq 0$, so the "or $r = 0$" clause in the conclusion never applies.

Alternatively, we can express the original statement as "Let $r \neq 0$. If r is rational then $1/r$ is rational." In this case, the contrapositive is "Let $r \neq 0$. If $1/r$ is irrational then r is irrational."

In either case, the contrapositive statement says that the reciprocal of a nonzero irrational number is irrational.

(5 pts) (b) Give a proof by contrapositive (*not* by contradiction!) that if $r \neq 0$ is rational and x is irrational, then $x \cdot r$ is irrational. First state the contrapositive (there is more than one valid way to write it), then prove that directly. Be sure to explicitly say where you use the hypothesis that $r \neq 0$ (because the statement is false without it!).

Solution: As in the previous part, there is more than one way to phrase the contrapositive:

- We can say "Let $r \neq 0$ be rational. If $x \cdot r$ is rational, then x is also rational."
- Or, we can say "Let $r \neq 0$. If $x \cdot r$ is rational, then r is irrational or x is rational (or both)."
- Or, we can say "If $x \cdot r$ is rational, then $r = 0$ or r is irrational or x is rational."

We prove the first of these. Let $r \neq 0$ be rational, and suppose that $x \cdot r$ is rational, so $x \cdot r = p/q$ for some integers p, q with $q \neq 0$. Then $x = (p/q) \cdot (1/r)$, and $1/r$ is rational by the previous part. (This is where we have used the hypothesis that $r \neq 0$, which the previous part requires.) This product (and more generally, any product) of rational numbers is also rational, which we prove as follows: by definition, $1/r = a/b$ for some integers a, b with $b \neq 0$. So, $x = (p/q) \cdot (a/b) = (pa)/(qb)$, where pa and qb are both integers (since p, q, a, b are integers), and $qb \neq 0$ because $q \neq 0$ and $b \neq 0$. Therefore, by definition x is rational, which is what we needed to show.

A proof for the third statement above would go like this. Suppose that $x \cdot r$ is rational. We proceed by cases. In the case where $r = 0$, then the statement's conclusion holds trivially (it says "$r = 0$ or ..."), so we are done. So now consider the case where r is irrational. In the (sub)case where r is irrational, the conclusion again holds trivially. So now consider the case where $r \neq 0$ and r is rational; we need to prove that x is rational. But these are exactly the hypotheses and conclusion of the first version of the statement, which we proved in the previous paragraph.

(5 pts) (c) Prove that any nonzero rational number may be written as the product of two irrational numbers. You may use the fact that $\sqrt{2}$ is irrational without proof. Again, explicitly say where you use the hypothesis that the number in question is nonzero (because the statement it false without it!).

Solution: Let r be a nonzero rational number, so $r = p/q$ for some nonzero integers p, q. We can then write $r = \frac{p}{\sqrt{2}} \cdot \frac{\sqrt{2}}{q}$. By the previous parts and the fact that $p \neq 0$, neither $\frac{p}{\sqrt{2}}$ nor $\frac{\sqrt{2}}{q}$ is rational: the former is a nonzero rational number p times the (irrational) reciprocal of the irrational number $\sqrt{2}$, and similarly for the latter. (Here we have used the fact that $r \neq 0$ and hence $p \neq 0$.)

(10 EC pts) 8. **Extra credit:** *You are not required to do this question to receive full credit on this assignment.* To receive the bonus points, you must typeset this **entire** assignment in LaTeX, and draw a table with two columns that includes the *name* (e.g., "fraction") and an *example* of each of the following:

- fraction (using `\frac`),
- less than or equal to,
- union of two sets,
- summation using Sigma ($\sum$) notation,
- the set of real numbers ($\mathbb{R}$); write a mathematically correct statement that applies to *all* real numbers $x \in \mathbb{R}$.

Solution: The following LaTeX code, which uses macros from the provided `header.tex`, was used to create the table below.

```
\renewcommand{\arraystretch}{2}
\begin{tabular}[t]{|c|c|}
  \hline
  fraction
  & $\frac{4}{8} = \frac{1}{2}$ \\
  \hline
  less than or equal to
  & $\abs{\inner{\vec{x}, \vec{y}}}^2 \leq
    \length{\vec{x}}^2 \cdot \length{\vec{y}}^2$ \\
  \hline
  union of two sets
  & $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$ \\
  \hline
  sum using Sigma notation
  & $\displaystyle
    \sum_{n=1}^\infty \frac{1}{n^2} = \frac{\pi^2}{6}$ \\
```

```
  \hline
  true statement $\R$, $\C$, and $\Z$
  & $\Z\subset \R\subset \C$ \\
  \hline
\end{tabular}
```

fraction	$\frac{4}{8} = \frac{1}{2}$
less than or equal to	$\lvert\langle \vec{x}, \vec{y}\rangle\rvert^2 \leq \lVert\vec{x}\rVert^2 \cdot \lVert\vec{y}\rVert^2$
union of two sets	$\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$
sum using Sigma notation	$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$
true statement $\mathbb{R}$, $\mathbb{C}$, and $\mathbb{Z}$	$\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$