Instructions

This homework assignment must be completed individually (and not in groups). Completing this assignment will set you up to use the tools you’ll need for the rest of the course, and will give you and us an idea of your mastery of the prerequisite knowledge before you start the course.

For this assignment, you will receive full credit for completing all required steps. For the pre-test, this means an honest attempt at answering all the questions; correctness of the answers will not be used to calculate your score for this first homework.

Required Reading Rosen Chapters 1 and 2, Sections 5.1-5.2 (covers prerequisite concepts).
1. (10 points) Prove that for every \( a > 1 \) and \( b > 1 \), \( a^{\log_b b} = b^{\log_b a} \).

\[
a^{\log_b b} = (2^{\log_b a})^{\log_b b} = 2^{\log_b a \log_b b} = (2^{\log_b b})^{\log_b a} = b^{\log_b a}
\]

The first equation uses the definition of logs, the second uses the algebraic rule of exponentiation \((x^y)^z = x^{yz}\), the third uses the same rule in reverse, and the fourth uses the definition of log. We need \( a > 0 \) and \( b > 0 \) since otherwise log is not well defined (at least as a real number).

2. (a) Give a formula for the number of digits in the decimal expansion for a positive integer \( n \).

(b) Give a formula for the number of digits in the binary expansion for a positive integer \( n \).

(c) Describe (in words) a rule to decide, if \((i, j)\) and \((i_2, j_2)\) have both been printed for some \( n \) then which ordered pair was printed first?

We print from smallest \( i \) to largest \( i \). So if \( i_1 < i_2 \), the first pair is printed first, and if \( i_1 > i_2 \) the second pair. If \( i_1 = i_2 \), since within the same \( i \), we print in increasing order of \( j \), the first pair is first if \( j_1 < j_2 \) and second otherwise. Summing up, the first pair is printed first if and only if \( i_1 < i_2 \) or \( i_1 = i_2 \) and \( j_1 < j_2 \).

3. Consider the following algorithm

\begin{verbatim}
procedure Loops(n: a positive integer)
  1. for i := 1 to n
  2.   for j := 1 to n
  3.     print (i, j)
\end{verbatim}

(a) Write what the algorithm prints when \( n = 4 \).

\((1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), (4,4)\)

(b) Describe what the algorithm prints in general terms.

It prints all pairs of positive integers where both integers are at most \( n \), ordered first by the first integer and then by the second

(c) How many times does \textbf{print} routine get called?

Since there are \( n \) possible \( i \)'s and for each \( n \) \( j \)'s, print is called \( n^2 \) times.

(d) Describe (in words) a rule to decide, if \((i_1, j_1)\) and \((i_2, j_2)\) have both been printed for some \( n \) then which ordered pair was printed first?

We print from smallest \( i \) to largest \( i \). So if \( i_1 < i_2 \), the first pair is printed first, and if \( i_1 > i_2 \) the second pair. If \( i_1 = i_2 \), since within the same \( i \), we print in increasing order of \( j \), the first pair is first if \( j_1 < j_2 \) and second otherwise. Summing up, the first pair is printed first if and only if \( i_1 < i_2 \) or \( i_1 = i_2 \) and \( j_1 < j_2 \).

4. Consider a room of people and over the course of the evening, some people shook hands. Prove that whatever pairs of people shake hands, there are an even number of people who have shaken hands an odd number of times.

There are two arguments I know. The first uses induction on \( t \), the number of hand shakes. Let \( Odd_t \) represent the number of people who’ve shaken hands an odd number of times after the first \( t \) hand shakes. We’ll prove by induction on \( t \) that \( Odd_t \) is always even.

The base case is \( t = 0 \). When zero hand shakes have taken place, everyone has shaken hands 0 times, so \( Odd_0 = 0 \). Since 0 is even, \( 0 = 2 \times 0 \), the claim holds for \( t = 0 \).

Assume \( Odd_t \) is even. (We need to show \( Odd_{t+1} \) is even). Look at the \( t + 1 \)’st hand shake. There are three cases: Both people involved have previously shaken hands an odd number of times, exactly
one has shaken hands an odd number of times and the other has shaken hands an even number of
times, or both have previously shaken hands an even number of times. In the first case, the two were
both counted in \( \text{Odd}_t \), but after this handshake, they have both now shaken hands an even number
of times (since an odd number +1 is even). Thus, \( \text{Odd}_{t+1} = \text{Odd}_t - 2 \), and an even number -2 is
even. In the second case, the two switch whether they’ve shaken hands an even number of times, and
\( \text{Odd}_{t+1} = \text{Odd}_t \) is even. In the last case, they both now have shaken an odd number of times, and
\( \text{Odd}_{t+1} = \text{Odd}_t + 2 \) is still even. So in all cases, \( \text{Odd}_{t+1} \) is even.

Since we showed our claim for \( t = 0 \), and shown that if it is true for one value of \( t \) it is true for the
next, by induction, the claim holds for all \( t \), in particular for the last handshake.

An alternate proof: Since each handshake involves two people, after \( t \) hand shakes, the total sum of
the times people have shaken hands is \( 2t \), an even number. We can break this sum into the sum over
all people who’ve shaken hands an even number of times and over all people who’ve shaken hands an
odd number of times. The first sum is always an even number because the sum of even numbers is
even. So the second must also be an even number. Since the sum of an odd number of odd numbers is
odd, there must be an even number of terms in the second sum, i.e., an even number of people who’ve
shaken hands an odd number of times.