LAB 7

MyPowerMod

The goal of this lab is to write our own PowerMod function. We have already written functions for taking quotients and remainders, so feel free to use Mathematica's `Quotient` and `Mod` commands.

PROBLEM 7.1. Write a `PositivePowerMod[a_,b_,m_]` command that assumes that b is positive and computes REM (a^b, m). Remember to use recursion.

PROBLEM 7.2. Use the extended Euclidean algorithm to find the inverse of 53 mod 137. Write it out very carefully so you can tell Mathematica how to do it below. What numbers do you need to keep track of as you work through the algorithm?

Problem 7.3. Notice that to reconstruct the s and t values, you only need to know the quotients produced by the Euclidean algorithm.

Look at the example you worked out above, and write down the details of how s and t change.

Problem 7.4. Write a *recursive* `GCDQuotients[a_,b_]` command that computes the GCD of a and b and produces a list of quotients. The command `GCDQuotients[a_,b_]` should *not* return the GCD itself. Hint: use the Append command to add extra entries to a list.

Problem 7.5. Once we create the quotients, we want to reconstruct the s and t values. For this, we will give the list of quotients to a function `ListToST`, along with our starting values of s and t.

Write a `ListToST[QuotientList_, currents_, currentt_]` command that takes a list of quotients produced by `GCDQuotients` and produces the s and t for the Extended Euclidean Algorithm. It should also work inductively.

Problem 7.6. Write a `ModularInversion[a_, m_]` command that combines `GCDQuotients` and `ListToST` to find an inverse to a mod m.

Problem 7.7. Write a `MyPowerMod[a_,b_,m_]` function that computes rem (a^b, m) for any integer a by combining `ModularInversion` and `PositivePowerMod`.

Problem 7.8. How does `MyPowerMod` compare in speed and functionality to `PowerMod`?