

MCS 177– Homework 4 make-up – Fall 2007

Due: 11/2

You can earn up to 20 more points on homework 4 by doing these problems. All of them involve the material in section 4.2 of the text. Numbers 2,5,6 and 7 present possible solutions for the last problem of homework 4.

1. (*2 pts*) Explain what `mod-expt` and `mod*` (p.90) do. Why do we use `mod*`? Would using `expt` or `power` in combination with `remainder` work?

2. (*2 pts*) Consider the following procedure, which uses `expt`. What happens when you run it with the number 2 and an extremely large prime number? You can find large prime numbers by using an applet available at <http://www-fs.informatik.uni-tuebingen.de/reinhard/krypto/English/2.3.2.e.html>. I used 1000000007 and got interesting results. Try this (or something somewhat larger) and see what happens. That is, evaluate `(index1 2 1000000007)` and tell me what happens. Why does this happen?

```
(define index1
  (lambda (n m)
    (define find-index
      (lambda (i)
        (if (= (remainder (expt n i) m) 1)
            i
            (find-index (+ i 1)))))
      (find-index 1)))
```

3. (*2 pts*) The time complexity of a procedure depends on the values of one or more of its parameters. Which of the parameters in `mod-expt` affect the time complexity of the procedure? Why?

4. (2 pts) When we analyze the time complexity of `mod-expt`, why do we count the number of multiplications (`mod*`'s) that it does?

5. (4 pts) Consider the following procedure, which uses `mod-expt` from page 90 of the text. Calculating its time complexity is hard, because it's not clear when `find-index` will terminate. However, if `m` is a prime number, then there will be at least one value of `n` which will have an index of `m-1`. Assuming that this is the `n` that we plug in, how many multiplications will `index2` do?

```
(define index2
  (lambda (n m)
    (define find-index
      (lambda (i)
        (if (= (mod-expt n i m) 1)
            i
            (find-index (+ i 1))))))
    (find-index 1)))
```

6. (2 pts) Suppose we replace the version of `mod-expt` that `index2` uses with the version on page 94, and call `(index2 2 m)`, where `m` is a large prime number. Then the index of 2 will be at worst `m-1`. Thus, the maximum number of `mod*`'s that such a call will do is roughly a constant times $(\log(m-1) + \log(m-2) + \dots + \log(2) + \log(1))$. Why?

7. (6 pts) Part of the problem with `index2` is that it computes n^1, n^2, n^3 and so on. Each time `find-index` is called, it computes n^i by multiplying n by itself i times and so it recomputes n^{i-1} along the way. You can save a considerable amount of time by adding another parameter, say `k`, to `find-index`. Then the value of `k` could be successive powers of `n`, stopping when you reach 1. Implement this approach and try it out on the same values as you did in problem 2.