Solutions 1

CSC 152 / 252 – Cryptography

Please notify me of any errors you find. If you need help, ask.

1) (a) Is not a function because $f(3)$ is not defined. (b) Is a function but is not invertible: $f(2) = f(3) = b$, so b has no unique inverse. (c) Is a one-to-one and onto function, and therefore invertible: $f^{-1} = \{(a,1),(b,2),(c,3)\}$.

2) Rewritten as a relation, $f = \{(0,0),(1,2),(2,4),(3,1),(4,3)\}$. This is a one-to-one and onto function, and therefore invertible: $f^{-1} = \{(0,0),(2,1),(4,2),(1,3),(3,4)\}$. *Note:* $(2x) \cdot 2^{-1} = x$ *which means that multiplying by 2's multiplicative inverse (if it exists) is the inverse operation for this function:* $f^{-1}(y) = 2^{-1}y \bmod 5 = 3y \bmod 5$.

3) (a) There are four domain elements and each has five possible mappings, so four times we have five candidates: 5^4 different functions. (b) There are a domain elements and each has b possible mappings, so a times we have b candidates: b^a different functions. (c) Zero. Since a permutation function must be one-to-one and onto, the domain and co-domain must be the same size. (d) When defining f there are four candidates for $f(0)$, three unused candidates for $f(1)$, two unused candidates for $f(2)$, and one unused candidate for $f(3)$, meaning there are 4! different ways to specify f. (e) It depends. If $a \neq b$ then just like (c) the answer is zero. If $a = b$ then the answer is like (d) and there are $a! = b!$.

4) This one simply requires the allocation of an array and loop to put a random value in each position.

```
unsigned* createRandomFunction(unsigned n) {
    unsigned *res = malloc(n * sizeof(unsigned));
    for (int i=0; i<n; i++) {
        res[i] = rand(n);
    }
    return res;
}
```

5) This one is harder because of the request for $O(n)$. To ensure that each number is in the array exactly once you must fill the array with each number and then randomize the order, and each step must be $O(n)$. Here's one way to do it: swap each element with a randomly chosen partner. Any reasonable $O(n)$ attempt will be considered correct.

```
unsigned* createRandomFunction(unsigned n) {
    unsigned *res = malloc(n * sizeof(unsigned));
    for (int i=0; i<n; i++) {
        res[i] = i;
    }
    for (int i=0; i<n; i++) {
        unsigned pos = rand(n);
        unsigned tmp = res[i]; res[i] = res[pos]; res[pos] = tmp;
    }
    return res;
}
```

P1) Simply declaring `hi` and `lo` as `unsigned` (a 32-bit type) or `unsigned short` (a 16-bit type) allowed the code to compile without error and work correctly. If you add `-Wconversion` to the compiler options, then using `unsigned short` causes lots of warnings requiring type-casting to fix.