

# Ungraded Homework Solutions

## CSC 152 – Cryptography

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

1) 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. A multiplicative inverse needs to be in the same set and use the same operation (multiplication mod 5), so  $f^{-1}(y) = 2^{-1}y \bmod 5 = 3y \bmod 5$ .* We'll talk much more about multiplicative inverses later in the course.

2) (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  $a! = b!$ .

3) 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;
}
```

4) 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;
}
```

5) (a,b,c)  $1/b$  because each output is uniformly distributed for each different input and the size of the codomain is  $b$ . (d)  $1/a$  because without knowledge of the outputs associated with other inputs, the value of  $f(0)$  is equally likely to be any codomain element. (e)  $1/(a-1)$  because  $f(0) = 0$  which means  $f(1)$  cannot be 0 but is equally likely to be any of the other  $a-1$  codomain elements. (f) 0 because  $f(0) = 0$  which means  $f(1)$  cannot be 0.

6) (a) For eight bits, each can be a 0 or a 1, so there are  $2^8$  different patterns that can be created; or interpreting the eight bits as an integer, the smallest would be 0 and the largest would be 255, so there

must be 256 different bytes. (b) There are 64 bits and each can be 0 or 1, so there are  $2^{64}$  different patterns. These answers are both from the multiplication rule for counting: there are two choices for the first bit, and for each of those there are two choices for the next bit, so there are a total of four possible choices for the first two bits. For each of those there are two choices for the next bit, so there are a total of eight possible choices for the first three bits. Etc.