CS3460: Data Structures

Lab 02: Hash Functions

Total Points: 20

Problems

Notes on Grading: Unless otherwise stated, all programs will receive input via `System.in` and will output solutions via `System.out`.

To simplify the grading process, all grading will be automated. You will be provided with sample input/output files for testing. You can ensure that your program will receive full marks by testing it with these provided files.

```
$ java YourProgram < input.txt > output.txt
$ diff output.txt correct.txt
```

The first line executes the Java program, redirecting input from a file `input.txt` and writing the output to a file `output.txt`. The second line compares your program's output (now stored in `output.txt`) with the correct answer (stored in `correct.txt`). If these files match exactly, the `diff` program will print nothing. Otherwise, it will list the differences.

Submission: Please submit the file `Adversary.java`. Please do not include any other files (including `HashFunctions.java`).

1. **Fun with Hash Functions (20 points):** Welcome to the team! You have been hired for your skill and expertise in adversarial input, assigned to defeat your professor. With full malice of intent, you have acquired privileged knowledge of four (4) hash functions that you will use to cause collisions in a hash table, turning operations that should run in $O(1)$ in expectation to an absolutely embarrassing $O(n)$!

   Each key k will be mapped to a table of size n. If C is the largest possible key k ($0 \leq k \leq C$), then C/n is the fraction of keys that can be hashed to the same index. If $C/n \geq n$, then, according to the pigeonhole principle, we can find a set of n keys that map to the same table index.

   We will assume $C = n^2$. Let k be the key being hashed. Then we have the following hash functions:

   (a) `hash1(k)` $= k \mod n$

   (b) `hash2(k)` $= \lfloor nk/C \rfloor$

   (c) `hash3(k)` $= ((2971k + 101923) \mod 128189) \mod n$

   (d) `hash4(k)` is the first integer generated using `nextInt(n)` from `java.util.Random` when given seed k

   You do not need to and should not modify the file `HashFunctions.java` (and do not submit it with your work either). Some startup code is provided in `Adversary.java`. This program will take two arguments from the command line, n and p, where n is the input size and p is which hash function we are testing.

   Please complete the functions `func1` through `func4` so that they output (print) n keys that all hash to the same table index using the corresponding hash functions.

```
$ java Adversary 100 2
```

   When executing the above command, your code should print 100 different keys (one per line) that all produce the same hash value using `hash2(k)`. For full credit, your program should work for any $n \leq 10000$, and be appropriately fast with respect to the input size. Please make sure your output is formatted correctly with no extraneous information.