15-451 Algorithms, Spring 2017
Recitation #3 Worksheet

Hashing: A *universal* hash family H from U to $[m] := \{0, 1, \ldots, m-1\}$ is a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ each mapping U to $[m]$, such that for any $a \neq b \in U$, when you pick a random function from H,

$$\mathbf{Pr}[h(a) = h(b)] \leq \frac{1}{m}.$$

Also, a *ℓ-universal* hash family H from U to $[m] := \{0, 1, \ldots, m-1\}$ is a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ each mapping U to $[m]$, such that for any *distinct* $a_1, \ldots, a_\ell \in U$, and for any $\alpha_1, \ldots, \alpha_\ell \in [m]$, when you pick a random function from H,

$$\mathbf{Pr}[h(a_1) = \alpha_1 \text{ and } \ldots \text{ and } h(a_\ell) = \alpha_\ell] = \frac{1}{m^\ell}.$$

1. Show that a 2-universal hash family is a universal hash family.

2. Is this hash family from $U = \{a, b\}$ to $\{0, 1\}$ (i.e., $m = 2$) universal? 2-universal?

	a	b
h_1	0	0
h_2	1	0

3. How about this one: is it universal? 2-universal? 3-universal?

	a	b	c
h_1	0	0	0
h_2	1	0	1
h_3	0	1	1
h_4	1	1	0

Las Vegas and Monte Carlo:

A *Las Vegas* algorithm is a randomized algorithm that always produces the correct answer, but its running time $T(n)$ is a random variable. That is, sometimes it runs faster and sometimes slower, based on its random choices; for instance, quicksort when choosing a random pivot, or treaps. A *Monte Carlo* algorithm is an algorithm with a deterministic running time, but that sometimes doesn't produce the correct answer. For example, you may be familiar with randomized primality testing algorithms, that given a number N will output whether it is prime or not and be correct with probability at least 99/100, say.

1. *Going from LV to MC:* Show that if you have a Las Vegas algorithm with expected running time $\mathbf{E}[T(n)] \leq f(n)$, then you can get a Monte Carlo algorithm with (a) worst-case running time at most $4f(n)$ and (b) probability of success at least 3/4.

2. *Going from MC to LV:* Suppose you have a MC algorithm `Algo` for a problem (e.g., think of factoring an n-bit number into a product of primes) with running time at most $f(n)$, that is correct with probablity p. Moreover, you have a "checking" algo `Check` that runs in time $g(n)$ and checks whether a given output is a correct solution for this problem. E.g., for factoring, you could just multiply the outputs together to make sure you get back the input, and also verify the outputs are indeed prime numbers by running a fast primality checker.

   Use these to get a LV algorithm that runs in expected time $\frac{1}{p}(f(n) + g(n))$.