Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Universal Hashing

Let $[m]$ denote the set $\{0, 1, ..., m-1\}$. Recall that a family of functions $\mathcal{H}$ is *universal* if for any $x \neq y, \Pr_{h\sim\mathcal{H}}[h(x) = h(y)] \leq 1/m$. That is, the chance that $h(x) = h(y)$ if we sample h uniformly at random from $\mathcal{H}$ is at most $1/m$.

For each of the following families of hash functions, determine whether or not it is universal. If it is universal, determine how many random bits are needed to choose a function from the family.

(a) $H = \{h_{a_1,a_2} : a_1, a_2 \in [m]\}$, where m is a fixed prime and

$$h_{a_1,a_2}(x_1, x_2) = a_1x_1 + a_2x_2 \mod m$$

Notice that each of these functions has signature $h_{a_1,a_2} : [m]^2 \to [m]$, that is, it maps a pair of integers in $[m]$ to a single integer in $[m]$.

(b) H is as before, except that now $m = 2^k$ for $k > 1$ is some fixed power of 2.

(c) H is the set of all functions $f : [m] \to [m-1]$.

2 Monte Carlo Games

Let's suppose we have a Monte Carlo algorithm (a randomized algorithm which has a deterministic bound on its runtime, but which only outputs the correct answer some of the time). Call this algorithm A; then $A(x, r)$ is the output of A on input x and random bits r. In this question, we will think of A as a distribution over many deterministic algorithms. Convince yourself that this makes sense: after all, if we fix a setting to the random bits r, we get $A_r(x)$, which is a deterministic algorithm (which may be wrong on some inputs). Let's fix a set of algorithms S (say, polynomial-time algorithms). Note that A has whatever property defines S if and only if it is a distribution over only algorithms in S (for example, we say a Monte Carlo algorithm is polynomial time if and only if it runs in polynomial time for all settings to the randomness, which is equivalent to all the deterministic algorithms in its distribution running in polynomial time).

We will define a function $c(a, x)$ which indicates whether the deterministic algorithm $a \in S$ is correct on input x; $c(a, x) = 1$ if a is correct on input x, and 0 if it is incorrect.

Let's use this function to define a zero-sum game; the row player will choose a and the column player will choose x; then a payoff of $c(a, x)$ will go to the row player.

(a) Describe the action and goal of the row and column players. Interpret these in the setting of 'correctness of the randomly chosen algorithm' that we constructed the game from. *Hint: Since $c(a, x)$ is an indicator, $\mathbb{E}[c(a, x)] = \Pr[c(a, x) = 1]$.*

(b) Using zero-sum game duality in conjunction with your interpretation above, what can we say about a problem if we know there exists a polynomial-time randomized algorithm which is correct with probability 2/3 on all inputs? What can we say if we know that there is a distribution of inputs on which no deterministic algorithm is correct with probability 2/3? *Hint: use the fact that a randomized algorithm induces a distribution over deterministic algorithms.*

3 Streaming Integers

In this problem, we assume we are given an infinite stream of integers x_1,x_2, ..., and have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to use in total. For all of the parts below, give a brief description of your algorithm and a brief justification of its correctness.

(a) Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.

(b) Show that we can compute whether the sum of all integers seen so far is divisible by some fixed integer N using $O(\log N)$ bits of memory.

(c) Assume N is prime. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible.

(d) Now let N be an arbitrary integer, and suppose we are given its prime factorization: $N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$. Give an algorithm to check whether N divides the product of all integers seen so far, using as few bits of memory as possible. Write down the number of bits your algorithm uses in terms of $k_1, \dots, k_r$.

4 Lower Bounds for Streaming

(a) Consider the following simple 'sketching' problem. Preprocess a sequence of bits $b_1, \dots, b_n$ so that, given an integer i, we can return b_i. How many bits of memory are required to solve this problem exactly?

(b) Given a stream of integers $x_1, x_2, \dots$, the *majority element* problem is to output the integer which appears most frequently of all of the integers seen so far. Prove that any algorithm which solves the majority element problem exactly must use $\Omega(n)$ bits of memory, where n is the number of elements seen so far.