St. Francis Xavier University
Department of Computer Science

CSCI 435: Algorithms and Complexity
Assignment 2
Due March 14, 2023 at 8:15am

Assignment Regulations.

- This assignment must be completed individually.
- Please include your full name and email address on your submission.
- You may either handwrite or typeset your submission. If your submission is handwritten, please ensure that the handwriting is neat and legible.

[10 marks] 1. Suppose you have a polynomial-time Las Vegas randomized algorithm solving a given problem. Show that you can always convert this to a polynomial-time Monte Carlo randomized algorithm with one-sided error solving the same problem.

Hint. To prove the probability that your Monte Carlo algorithm produces a correct answer, you may find *Markov's inequality* useful: given a nonnegative random variable X and a value $a > 0$, we have that $\mathbb{P}[X \geq a] \leq \mathbb{E}[X]/a$.

[12 marks] 2. (a) Suppose you ordered a physical random number generator online. You thought the deal you were getting was too good to be true, and it was: the generator produces bits independently with unequal probabilities. Specifically, it produces the bit `1` with probability p and the bit `0` with probability q, where $p \neq q$ and $p + q = 1$.

Design a randomized algorithm that uses this generator to produce a bit uniformly at random (i.e., producing `1` with probability $1/2$ and `0` with probability $1/2$), and prove the correctness and runtime of your algorithm. You can use any approach you like, but your algorithm should return the random bit in expected constant time.

(b) Returning to the online store, you spend a little more money to order a true random number generator that produces bits uniformly at random. By this point, you've grown tired of single bits, and now you want larger numbers.

Design a randomized algorithm that uses this generator to produce a random integer between 0 and N, where N may or may not be a power of two, and prove the correctness and runtime of your algorithm. You can use any approach you like.

[8 marks] 3. Suppose we want to create a binary counter that has two functions: INCREMENT, which increments the binary counter by one; and RESET, which sets all bits in the counter to be `0`.

Show how we can implement our binary counter such that any sequence of n calls to the INCREMENT or RESET functions takes $O(n)$ *amortized* time on a counter initialized to all-`0` bits. Describe (or give pseudocode for) both functions, and prove that each individual call has $O(1)$ *amortized* time complexity.