St. Francis Xavier University
Department of Computer Science

CSCI 550: Advanced Algorithms
Assignment 2
Due November 16, 2026 at 10:30am

Assignment Regulations.

- This assignment may be completed individually or in a group of two people. If you are collaborating on an assignment as a group, your group must submit exactly one joint set of answers.
- Please include your full name and email address on your submission. For groups, every member must include their full name and email address on the joint 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 we want to divide the vertices of a simple undirected graph $G = (V, E)$ into three subsets A, B, and C, allowing for empty subsets. An edge is called a *bridge* if its two endpoints belong to different subsets. Our goal is to maximize the number of bridge edges.

Consider the following local search algorithm: start with any partition of the vertex set into three subsets, then repeatedly move one vertex to a different subset whenever doing so strictly increases the number of bridge edges. Stop when no such move is possible.

(a) Prove that this local search algorithm terminates after at most $|E|$ moves, and explain how it can be implemented in polynomial time.

(b) Prove that the final partition produces at least $\frac{2}{3} \cdot \mathrm{OPT}(G)$ bridge edges, where $\mathrm{OPT}(G)$ is the maximum possible number of bridge edges.

Hint. Upon termination, compare the number of neighbours a vertex has in its own subset with the number it has in each other subset.

[10 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.

[10 marks] 3. Suppose you have a Las Vegas randomized algorithm solving a given problem in expected polynomial time. Show that you can always convert this to a Monte Carlo randomized algorithm solving the same problem with one-sided error in polynomial time.

Hint 1. Remember that your Monte Carlo algorithm must produce a correct answer with probability $\geq 1/2$. This probability can be modelled by the expression $\mathbb{P}[T_{\text{LV}}(n) < T_{\text{MC}}(n)]$, where $T_{\text{LV}}(n)$ is the runtime of the Las Vegas algorithm and $T_{\text{MC}}(n)$ is the runtime of the Monte Carlo algorithm.

Hint 2. Building on the previous hint, you may find *Markov's inequality* useful: given a nonnegative random variable X and a value $a > 0$, we have that $\mathbb{P}[X \geq a] \leq \mathbb{E}[X]/a$.

[10 marks] 4. Consider the following sequence of page requests, starting with an empty cache:

$$1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.$$

Every request for a page not currently in the cache incurs one page fault, including requests made while the cache has unused space.

(a) Process this page request sequence using FIFO, first with a cache of size 3 and then with a cache of size 4. Show the cache contents after each request and count the number of page faults incurred. Does increasing the cache size always reduce the number of FIFO page faults?

(b) Prove that this behaviour cannot occur using LRU. That is, show that for every request sequence and for all $k \geq 1$, LRU incurs no more page faults with an initially empty cache of size $k + 1$ than with an initially empty cache of size k.

Hint. Describe the pages in an LRU cache in terms of their most recent requests.