Problem 1 (20 Points) - Competitive Analysis

You have reached a river (modelled as a straight line) and must find a bridge to cross it. The bridge is at some integer coordinate upstream or downstream.

1. Give a 9-competitive deterministic algorithm for optimizing the total distance travelled up and downstream before you find the bridge.
2. Give a randomized 7-competitive algorithm for the problem

Problem 2 (20 Points) - Paging Algorithms

A *conservative algorithm* is one that makes at most k page faults on any consecutive subsequence of the input that contains at most k pages. Here k is the size of the cache.

1. Prove that **LRU** and **FIFO** are conservative
2. Prove that any conservative algorithm is k-competitive.

Problem 3 (20 Points) - Multiplicative Weights Update

Here is a variation on the deterministic Weighted-Majority algorithm, designed to make it more adaptive.

1. Each expert begins with weight 1.
2. We predict the result of a weighted-majority vote of the experts.
3. If an expert makes a mistake, we penalize it by dividing its weight by 2, *but only if its weight was at least* 1/4 *of the* ***average weight*** *of experts.*

Prove that in any contiguous block of trials (e.g., the 51st day through the 77th day), the number of mistakes made by the algorithm is at most $O(m + \log n)$, where m is the number of mistakes made by the best expert *in that block*, and n is the total number of experts.

Problem 4 (20 Points) - Distinct Elements

Consider again the distinct elements problem that we saw in class. We are given a sequence of elements $a_1, \ldots, a_n$ from our universe $U = \{0, 1, \ldots, 2^b - 1\}$ as a stream, possibly with repetitions, and we would like to know how many distinct elements are there in the sequence. Since we are in the streaming setting, we will make only one pass through the sequence above, and we have little memory.

We will now analyze a different algorithm for the distinct elements problem:

- Let $N > n$ be an integer
- Pick at random a function $h : U \to [N]$ from a strongly 2-universal family.
- Let $m := \min\{h(a_1), \ldots, h(a_n)\}$

- Output N/m

1. Suppose that $a_1, \ldots, a_n$ contains k distinct elements. Show that

$$\Pr[\text{ algorithm outputs a number } > 4k] \leq \frac{1}{4}$$

2. Suppose that $a_1, \ldots, a_n$ contains k distinct elements. Show that

$$\Pr[\text{ algorithm outputs a number } < k/4] \leq \frac{1}{4}$$

3. Assuming that $U = [\mathsf{poly}(n)]$, what is the memory requirement of the algorithm above?

Problem 5 (20 Points) - Hardness of Approximation

Given a graph G, an *independent set* is a set of vertices such that no two are neighbors. The Maximum Independent Set (MIS) problem is a famous NP-complete problem.

Interestingly, the complement of an independent set is a vertex cover, so the complement of the MIS is the minimum vertex cover. We've seen (twice) how to get a two-approximation for vertex cover. Even though it is complementary, the situation for MIS is much worse.

Early in the study of approximation hardness, MIS was shown to be MAX-SNP-hard, meaning there is some constant to within which it *cannot* be approximated (unless P = NP).

Suppose one has an α-approximation algorithm for MIS. Consider the following "graph product" operation for a graph G. Create a distinct copy G_v of G for each vertex v of G. Then connect up the copies as follows: if (u, v) is an edge of G, then connect every vertex in G_u to every vertex in G_v.

(a) Prove that if there is an independent set of size k in G, then there is an independent set of size k^2 in the product graph.

(b) Prove that given an independent set of size s in the product graph, one can find an independent set of size $\sqrt{s}$ in G.

(c) Conclude from the MAX-SNP-hardness of MIS that MIS has *no* constant-factor approximation (unless P = NP).

Problem 6 (20 Points) - Matrix Multiplication and Determinant

Given a matrix $A \in \mathbb{Q}^{N \times N}$ where $N = 2^k$, prove that one can compute $\det(A)$ in time $O(N^\omega)$, where ω is the matrix multiplication exponent. You can assume that any matrix that you need to invert in the process is invertible.

Optional question: how would you remove the assumption given above?