Remember: you may work in groups of up to three people, but must write up your solution entirely on your own. Collaboration is limited to discussing the problems – you may not look at, compare, reuse, etc. any text from anyone else in the class. Please include your list of collaborators on the first page of your submission. You may use the internet to look up formulas, definitions, etc., but may not simply look up the answers online.

Please include proofs with all of your answers, unless stated otherwise.

1 Hashing (50 points)

We saw in class that universal hashing lets us give guarantees that hold for arbitrary (i.e. worst case) sets $S$, in expectation over our random choice of hash function. Let’s work out some more of those guarantees.

(a) (25 points) Let $H$ be a universal hash family from $U$ to a table of size $m$. Let $S \subseteq U$ be a set of $m$ elements which we want to hash (so we’re hashing the same number of elements as the size of the table). Prove that if we choose $h$ from $H$ uniformly at random, the expected number of pairs $x, y \in S$ that collide is at most $m - \frac{1}{2}$.

(b) (25 points) Prove that with probability at least $\frac{3}{4}$, no bin in the table gets more than $2\sqrt{m} + 1$ elements.

Hint: use part (a), and consider using Markov’s Inequality. To remind you: if $X$ is a non-negative random variable with expectation $\mathbb{E}[X]$, then $\Pr[X > k\mathbb{E}[X]] < 1/k$ for any $k > 0$. For example, the probability that $X$ is more than 100 times its expectation is less than $1/100$.

2 Union-Find (50 points)

In class we proved that if we use trees to represent disjoint sets and use both union-by-rank and path compression, then the amortized cost of any operation (Make-Set, Find, or Union) is only $O(\log^* n)$. In this question we’ll analyze what happens if we change our data structure in two ways: we do not use path compression, and we do union-by-cardinality rather than union-by-rank.

Slightly more formally, we change our data structure as follows. As before, every node has three values: an element, a parent pointer, and a value. But the value isn’t a rank, but is rather the number of nodes in the subtree rooted at it, i.e., the cardinality of the subtree. So the root of any tree stores the cardinality of the set represented by the tree. The basic operations work as follows:

- Make-Set$(x)$ simply returns a single node with the element $x$, where the parent pointer points to itself and the initial cardinality is 1.

- Find$(x)$ follows the parent pointer from $x$ recursively until it ends up at the root (which it knows since only the root will have its parent pointer point to itself), and then it returns the element at this root.
• Union$(x, y)$ first does Find$(x)$ to find the root $r_x$ of the tree containing $x$ and Find$(y)$ to find the root $r_y$ of the tree containing $y$. If $(r_x \to \text{cardinality}) \geq (r_y \to \text{cardinality})$, then we set the parent of $r_y$ to be $r_x$ and set $(r_x \to \text{cardinality}) = (r_x \to \text{cardinality}) + (r_y \to \text{cardinality})$. Similarly, if $(r_y \to \text{cardinality}) > (r_x \to \text{cardinality})$, then we set the parent of $r_x$ to be $r_y$ and set $(r_y \to \text{cardinality}) = (r_x \to \text{cardinality}) + (r_y \to \text{cardinality})$.

(a) (25 points) Prove that the worst-case running time of any operation is $O(\log n)$, where $n$ is the number of Make-Set operations (i.e., the number of elements). Hint: can you bound the depth/height of a tree by its cardinality?

(b) (25 points) We now want to provide a matching lower bound not just on the worst-case running time, but even on the amortized running time. So suppose that we first do $n$ Make-Set operations (so there are $n$ elements). Give a sequence of $O(n)$ Union operations which cumulatively take $\Omega(n \log n)$ time (implying that the amortized cost of an operation is $\Omega(\log n)$). Each union operation should be a union of two distinct sets, i.e., you should never call Union$(x, y)$ on elements $x$ and $y$ that are in the same set.