Problem 1

Perfect hashing is nice, but does have the drawback that the perfect hash function has a lengthy description (since you have to describe the second-level hash function for each bucket). Consider the following alternative approach to producing a perfect hash function with a small description. Define *bi-bucket hashing*, or *bashing*, as follows. Given n items, allocate two arrays of size $2n^{3/2}$. When inserting an item, map it to one bucket in *each* array, and place it in the emptier of the two buckets.

1. Suppose a random function (i.e., all function values are uniformly random and mutually independent) is used to map each item to buckets. Give a good upper bound on the expected number of collisions (i.e., the number of pairs of items that are placed in the same bucket).

   Hint: what is the probablility that the k^{th} inserted item collides with some previously inserted item?

2. Argue that bashing can be implemented efficiently, with the same expected outcome, using the ideas from 2-universal hashing.

3. Conclude an algorithm with linear expected time (ignoring array initialization) for identifying a **perfect** bash function for a set of n items. You should prove that your scheme is perfect. How large is the description of the resulting function?

Problem 2

Consider the following examples of hash families. For each one, prove that it is 2-universal or give a counterexample.

1. Let p be a prime number and $n \leq p$ be an integer. Let

$$\mathcal{H} := \{h_a(x) = (ax \bmod p) \bmod n \mid a \in \{1, \dots, p-1\}\}$$

2. Let p be a prime number and $n \leq p$ be an integer. Let

$$\mathcal{H} := \{h_b(x) = (x + b \bmod p) \bmod n \mid b \in \{0, 1, \dots, p-1\}\}$$

3. Let m be an integer multiple of n. Let

$$\mathcal{H} := \{h_{a,b}(x) = (ax + b \bmod m) \bmod n \mid a \in \{1, \dots, m-1\}, b \in \{0, 1, \dots, m-1\}\}$$

4. Let p be a prime number and $n \leq p$ be an integer. Let

$$\mathcal{H} := \{h_{a,b}(x) = (ax + b \bmod p) \bmod n \mid a, b \in \{0, 1, \dots, p-1\}, a \neq 0\}$$

Problem 3

Consider the problem of deciding whether two integer *multisets* S_1 and S_2 are identical (that is, each integer occurs the same number of times in both sets). This problem can be solved by sorting the two sets in $O(n \log n)$ time, where $n = |S_1| = |S_2|$. In this question, you will devise 2 faster randomized algorithms for this problem.

You can assume that the multisets S_i only have integers of bit complexity w, and that integer operations of $O(w)$-bit integers can be executed in $O(1)$ time (RAM model), and that a prime with $O(w)$-bits can be found in $O(n)$ time.

1. Use polynomial identity testing to give a $O(n)$ time algorithm for the problem above.
2. Use hashing to give a $O(n)$ time algorithm for the problem above.

Your algorithm for both parts should succeed with probability $\geq 2/3$.

Problem 4

Another problem about Karger's randomized algorithm for minimum cut:

1. Suppose Karger's algorithm is applied to a tree. Show that it finds a minimum cut in the tree with probability 1.

2. Consider the following modification of Karger's algorithm: instead of choosing an edge uniformly at random and merging the endpoints, the algorithm chooses *any* two distinct vertices uniformly at random and marges them. Show that for any n there is a graph G_n with n vertices such that when the modified algorithm is run on G_n, the probability that it finds a minimum cut is *exponentially* small in n.

3. How many times would you have to repeat the modified algorithm of the previous part to have a reasonable chance of finding a minimum cut? What does this tell us about the practicality of the modified algorithm?

4. Show that for any $n \geq 3$ there is a graph G_n with n vertices that has $n(n-1)/2$ distinct minimum cuts.

Problem 5

To improve the probability of success of the randomized min-cut algorithm, it can be run multiple times.

1. Consider running the algorithm twice. Determine the number of edge contractions and bound the probability of finding a min-cut.

2. Consider the following variation. Starting with a graph with n vertices, first contract the graph down to k vertices using the randomized min-cut algorithm. Make ℓ copies of the graph with k vertices, and now run the randomized algorithm on these reduced graphs independently. Determine the number of edge contractions and bound the probability of finding a min-cut.

3. Find optimal (or at least near-optimal) values of k and ℓ for the variation in the previous part that maximizes the probability of finding a min-cut while using the same number of edge contractions as running the original algorithm twice.

Problem 6

Sublinear-time algorithms for connectedness in graphs with bounded degree.

Given a graph G of max degree d (*as adjacency list*), and a parameter $\epsilon > 0$, give an algorithm which has the following behavior: if G is connected, then the algorithm should pass with probability 1, and if G is ϵ-far from connected (at least $\epsilon \cdot n \cdot d$ edges must be added to connect G), then the algorithm should fail with probability at least 3/4. Your algorithm should look at a number of edges that is independent of n, and polynomial in d, ϵ.

For this problem, when proving the correctness of your algorithm, it is ok to show that if the input graph G is likely to be passed, then it is ϵ-close to a graph G_0 which is connected, without requiring that G_0 has degree at most d.