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 $n^{3/2}$. 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. How large is the description of the resulting function?

Problem 2

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 3

One advantage of Karger's random contraction algorithm for the minimum cut problem is that it can be used to output all minimum cuts. In this question, we assume Karger's algorithm as a black box, which can be used to output a minimum cut with probability at least $2/n(n-1)$ in time $O(n^2)$, where n is the number of vertices in the input graph. Explain how Karger's algorithm can be used to output all minimum cuts and analyze its running time to output all minimum cuts with success probability at least 0.9999.

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

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.

Problem 6

In this problem we analyze a common algorithmic application of the Johnson-Lindenstrauss lemma.

Consider the k-means clustering problem. Given points $x_1, \ldots, x_n \in \mathbb{R}^d$, the goal is to partition the points into k disjoint sets (clusters) $C_1, \ldots, C_k$ to minimize the following cost:

$$Cost(x_1, \ldots, x_n, C_1, \ldots, C_k) = \sum_{i=1}^{k} \sum_{j \in C_i} \|x_j - \mu_i\|_2^2,$$

where the norm is the Euclidean norm and $\mu_i = \frac{1}{|C_i|} \cdot \sum_{j \in C_i} x_j$ is the mean of the points in cluster C_i.

1. Prove that:
$$Cost(x_1, \ldots, x_n, C_1, \ldots, C_k) = \sum_{i=1}^{k} \frac{1}{|C_i|} \cdot \sum_{\substack{j,\ell \in C_i \\ j<\ell}} \|x_j - x_\ell\|_2^2$$

2. Suppose we embed $x_1, \ldots, x_n$ into $O(\log n/\epsilon^2)$ dimensional vectors $y_1, \ldots, y_n$ using the Johnson-Lindenstrauss construction. Show that for all clusterings *simultaneously*:
$$(1-\epsilon) \cdot Cost(x_1, \ldots, x_n, C_1, \ldots, C_k) \leq Cost(y_1, \ldots, y_n, C_1, \ldots, C_k) \leq (1+\epsilon) \cdot Cost(x_1, \ldots, x_n, C_1, \ldots, C_k)$$
with high probability.

3. Suppose we find a set of clusters $\Gamma_1, \ldots, \Gamma_k$ such that:
$$Cost(y_1, \ldots, y_n, \Gamma_1, \ldots, \Gamma_k) \leq \gamma \cdot Cost(y_1, \ldots, y_n, C_1^*, \ldots, C_k^*)$$
where $C_1^*, \ldots, C_k^*$ is the optimal clustering for the points $y_1, \ldots, y_n$. That is, we found a γ-approximation to the optimal clustering for the low-dimensional points. Show that for $\epsilon \leq 1/2$, we have
$$Cost(x_1, \ldots, x_n, \Gamma_1, \ldots, \Gamma_k) \leq (1 + O(\epsilon)) \cdot \gamma \cdot Cost(x_1, \ldots, x_n, C_1^{opt}, \ldots, C_k^{opt}),$$
where $C_1^{opt}, \ldots, C_k^{opt}$ is the optimal cluster for $x_1, \ldots, x_n$.

In other words, we can compute an approximate clustering for our original points using the low dimensional data. This speeds up algorithms for k-means whenever $\log n/\epsilon^2 < d$.