1. (10 points) **(Designing a Tournament)** This question will help you understand the applications of Chernoff bound and Union bound more, as you'll deal with tuning parameters to optimize the efficiency of the system. You'll also get an understanding of why NBA tournaments are designed the way they are, with more repeated matches being played as we get closer to the finals. For example, in the NBA, the early rounds are *best-of-five*, and the later rounds become *best-of-seven*. You will understand why, and also learn how to design tournaments with $n$ participants, for large $n$.

   Suppose there are $n$ teams, and they are totally ranked. That is, there is a well-defined best team, second ranked team and so on. It's just that we (the algorithm designer) don't know the ranking. Moreover, assume that for any given match between two players, the better ranked team will win the match with probability $p = \frac{1}{2} + \delta$, *independent of all other matches between these players and all other players also.* Here $\delta$ is a small positive constant.

   (a) (2 points) Let $n$ be a power of two, and fix an arbitrary tournament tree starting with $n/2$ matches, then $n/4$ matches and so on. What is the probability that the best team wins the tournament?

   **Solution:**

   *Proof.* $1 + 1 = 2$. □

   (b) (3 points) Use Chernoff bounds to bound the probability that the best team will not win the tournament, if each match-up occurs as a *best-of-k* series. How many games do you end up conducting in total to get a $1 - \epsilon$ probability of the best team winning?

   **Solution:**

   *Proof.* $1 + 1 = 2$. □

   (c) (5 points) Now design a tournament with a total of $O_\epsilon(n)$ games to get $1 - \epsilon$ probability of the best team winning eventually. $O_\epsilon(n)$ means $O(n)$ for all constant $\epsilon > 0$.

   **Solution:**

   *Proof.* $1 + 1 = 2$. □

2. (25 points) **(Counting Distinct Elements)** Universal hash families come in handy when designing *streaming algorithms* that make a *single pass* and use very little space in comparison to the size of the input stream. Here, we will construct a streaming algorithm to estimate the number of distinct elements in the stream. The technique here is from a seminal work of Alon, Matias and Szegedy (STOC 1996); the authors were awarded the Gödel prize in 2005 for this work.

(a) (5 points) Recall that for a prime $p$, and an integer $k$, where $1 \leq k \leq p$, the collection:

$$\mathcal{H}_k = \{h_{ab} : x \to (ax + b) \mod k \mid a, b \in \mathbb{F}_p, a \neq 0\}$$

is a 2-universal hash family. **Prove** the following lemma:
**Lemma 1.** *For every set $S \subseteq \mathbb{F}_p$,*

1. *if $|S| < k$, then*
$$\mathbf{Pr}_{h \leftarrow H_{4k}} [\exists x \in S : h(x) = 0] < 1/4;$$
2. *while, if $|S| \geq 2k$, then*
$$\mathbf{Pr}_{h \leftarrow H_{4k}} [\exists x \in S : h(x) = 0] \geq 3/8.$$

**Solution:**

*Proof.* $1 + 1 = 2$. □

(b) (10 points) Consider a stream of integers: $x_1, x_2, \ldots$; where each $x_i \in [n]$. For a parameter $k$, where $1 \leq k \leq n$, design a streaming algorithm with the following guarantees:

1. if the stream contains strictly less than $k$ distinct elements, the output is `No` with probability at least $1 - 1/n^2$;
2. while, if the stream contains at least $2k$ distinct elements, then the output is `Yes` with probability at least $1 - 1/n^2$.

The algorithm should proceed in three phases as in the template solution below. Fill in the details of the algorithm *in the given template*; you need not provide implementation details but should be precise in your description.

**Solution:**

- *Initialization*: (pre-processing done before making the pass)
    1. TODO: Fill in the steps.
- *Processing*: ($x \in [n]$ is the element to process)
- *Output*: (called after the input is processed)

Prove the guarantees of the algorithm designed above and calculate the space used by the algorithm (in $O(\cdot)$ notation).

**Solution:**

*Proof.* $1 + 1 = 2$. □

(c) (10 points) Using the above, design a streaming algorithm that outputs an estimate of the number of distinct elements, upto a factor 2 with probability at least $1 - 1/n$ ($n$ is a parameter passed to the algorithm). As before, the algorithm should be in three phases (and may use the above procedures). Prove the guarantees and calculate the total space used by the algorithm.

extra creditextra creditextra credit

(d) (5 extra credit points) Show how the above algorithm can be modified to obtain a factor $(1 + \epsilon)$-approximate with high probability.

extra creditextra creditextra credit

(e) (10 extra credit points) Can you reduce the space even further while maintaining the guarantees as in part c? The improvement should be asymptotic, and not just by a constant factor.

3. (15 points) **(Locality Sensitive Hashing)** Given a distance metric $d : X \times X \to \mathbb{R}^{\geq 0}$, a LSH family is a collection:

$$\mathcal{H} = \{\ h : X \to [M]\ \}$$

such that, for all $x, y \in X$:

1. if $d(x, y) \leq R$,
$$\mathbf{Pr}_{h \leftarrow \mathcal{H}}\left[h(x) = h(y)\right] \geq p_1;$$
2. while, if $d(x, y) > cR$,
$$\mathbf{Pr}_{h \leftarrow \mathcal{H}}\left[h(x) = h(y)\right] < p_2.$$
3. moreover, this (collision) probability is a non-increasing function of $d(x, y)$.

Here, $R, c, p_1, p_2$ are parameters that determine the quality of the hash family. As was outlined in class, Indyk and Motwani designed an algorithm to store $n$ data-points so that $c$-approximate near-neighbor queries may be answered in *sub-linear* time.

Recall that the algorithm consists of two components:

- **Preprocessing**: (given the data-points $x_1, \ldots, x_n$, and two integer parameters $k, l$ chosen appropriately.)
  1. Choose $l$ hash functions, $g_1, \ldots, g_l \leftarrow \mathcal{H}^k$ and initialize hash tables for each of them. In other words, each $g_i$ is the concatenation of $k$ (randomly chosen) hash functions from $\mathcal{H}$.
  2. Insert each data point in each of the hash functions.
- **Query**: (given data-point $y$.)
  1. Iterate over the points mapped to $g_i(y)$ for $1 \leq i \leq l$ in the hash table and output all points $x_j$ at distance at most $cR$.
  2. Abort the above search if more than $10l$ data-points have already been looked at among all the hash tables.

(a) (3 points) Calculate the value of $k$ so that, if $x, y \in X$ satisfy $d(x, y) > cR$, then:

$$\mathbf{Pr}_{g \leftarrow \mathcal{H}^k} [g(x) = g(y)] < 1/n.$$

Show hence that the probability of aborting the query procedure is at most $1/5$.

(b) (6 points) For the value of $k$ calculated above, calculate the collision probability, under a random $g$, of a data-point $x$ that is indeed within distance $R$ from $y$. Calculate $l$ such that the probability that none of the hash functions $g_i$ cause a collision between $x$ and $y$ is at most $1/5$.

(c) (6 points) State tha guarantees of the algorithm: output, space-complexity, time-complexity and confidence for the values of $k$, and $l$ calculated above.