Practice Quiz 2

- The following practice quiz is a compilation of relevant problems from previous semesters.
- This practice quiz may not be taken as a strict gauge for the difficulty level of the actual quiz.
- The quiz contains multiple problems, several with multiple parts. You have 120 minutes to complete this quiz.
- Write your solutions in the space provided. If you run out of space, continue on a scratch page and make a notation.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to *give an algorithm* in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem.
- **Good Luck!**

Problem-1: T/F Questions

Answer *true* or *false* to each of the following. **No justification is required for your answer**. Space is provided for scratch work.

(a) T / F Let n and k be integers greater than 10. Consider the following family $\mathcal{H}$ of hash functions that map elements from the set $\{1, \ldots, n\}^k$ to the set $\{1, \ldots, n\}$. $\mathcal{H}$ contains for every $1 \leq i \leq k$ a function h_i that maps $(x_1, \ldots, x_k) \in \{1, \ldots, n\}^k$ to x_i. $\mathcal{H}$ is a universal hash family.

(b) T / F A randomized algorithm that is always correct and has expected linear running time can be converted to a randomized algorithm that always runs in linear time, but is correct only with probability 99%.

(c) T / F Suppose algorithm $\mathcal{A}$ has two steps, and $\mathcal{A}$ succeeds if both the steps succeed. If the two steps succeed with probability p_1 and p_2 respectively, then $\mathcal{A}$ succeeds with probability $p_1 p_2$.

(d) The hash family $H = \{h_1, h_2\}$ is universal, where $h_1, h_2 : \{1, 2, 3\} \to \{0, 1\}$ are defined by the following table:

	1	2	3
h_1	0	1	0
h_2	1	0	1

(For example, $h_1(3) = 0$.)

(e) Suppose a file server stores a hash of every file in addition to the file contents. When you download a file from the server, you also download the hash and confirm that it matches the file. This system securely verifies that the downloaded file has not been modified by an adversary, provided the hash function has collision resistance.

(f) A Monte Carlo algorithm runs in worst-case polynomial time.

(g) Consider $f(x) = \frac{x^3}{3}$ and $x^{(0)} \neq 0$. The gradient descent algorithm with the following iteration

$$x^{(t+1)} \leftarrow x^{(t)} - \frac{1}{2} f'(x^{(t)})$$

always converges to $-\infty$.

(h) The following definition is equivalent to that of a universal hash family:
Let $\mathcal{H}$ be a family of hash functions $h : \{0,1\}^u \to \{0,1\}^m$, then for any $h \neq h' \in \mathcal{H}$

$$\mathbb{P}_{k \in \{0,1\}^u}[h(k) = h'(k)] \leq \frac{1}{m}$$

(i) We define n random variables $X_i = \sum_{j=1}^{n} c_{(i,j)}$, where for every (i,j), $c_{(i,j)}$ is determined by an independent unbiased coin flip, so $c_{(i,j)}$ is one with probability 1/2 and zero otherwise. Then, for every $\beta \in (0,1)$

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i > (1+\beta)\frac{n^2}{2}\right] < e^{-\beta^2 n^2/6}$$

(j) Consider a k-bit vector $v = x_1 \ldots x_k$ chosen uniformly at random from $\{0,1\}^k$. For any $S \subseteq \{1, \ldots, k\}$ such that $S \neq \emptyset$, define $m_S(v) = \sum_{i \in S} x_i \pmod 2$. Then,

$$\operatorname{Var}\left[\sum_{S \subseteq \{1,\ldots,k\}, S \neq \emptyset} m_S(v)\right] = \frac{2^k - 1}{4}$$

(k) If an algorithm runs in time $\Theta(n)$ with probability 0.9999 and in time $\Theta(n^2)$ with the remaining probability, then its expected run-time is $\Theta(n)$.

Problem-2: Democratic Feudal System *[25 points]*

A kingdom has a single monarch. The monarch has three loyal dukes who reign over three parts of his kingdom. Each of these dukes in turn has three loyal advisors, who each have three loyal subjects, and so on, down to the peasants. This feudal society can be modeled by a tree with branching factor 3 and height h. Voting day has now arrived, and each peasant votes either for or against some public policy. This is modeled by assigning a value of 0 or 1 to each of the leaves of the tree. Any intermediate officer votes according to the majority of his immediate constituents. That is, the value assigned to any non-leaf node of the tree is the majority of the values of its immediate children, whose own value is the majority of the values of their own immediate children, and so on so forth down to the leaves of the tree. If the value thus assigned to the root of the tree (the king's node) is 1, the policy passes; otherwise, the policy fails. Since collecting votes is expensive, the king would like to determine an efficient algorithm to evaluate the final vote.

(a) *[5 points]* Show that in the worst case, a deterministic algorithm will have to query the vote of all of the $n = 3^h$ leaves in order to determine the value assigned to the root. That is, argue that it is impossible for a deterministic algorithm to correctly determine the value at the root of the tree without knowing every single leaf's value.

(b) *[5 points]* Design a randomized algorithm for computing the majority of three bits b_1, b_2, b_3, such that, no matter what the values of b_1, b_2, b_3 are, with probability at least $\frac{1}{3}$, the algorithm does not inspect one of the values.

(c) *[10 points]* Design a randomized algorithm that evaluates the final vote correctly with probability 1, but only inspects the value of n^α leaves in expectation, for some constant $\alpha < 1$ (independent of n).

(d) *[5 points]* Show that there exists some constant $c \geq 1$ (independent of n) such that, with probability at least .99, your algorithm from (c) inspects at most cn^α leaves, where α is the constant you derived in (c).

Problem-3: Forgetful Forrest *[25 points]*

Prof. Forrest Gump is very forgetful, so he uses automatic calendar reminders for his appointments. For each reminder he receives for an event, he has a 50% chance of actually remembering the event (decided by an independent coin flip).

(a) Suppose we send Forrest k reminders for each of n events. What is the expected number of appointments Forrest will remember? Give your answer in terms of k and n.

(b) Suppose we send Forrest k reminders for a *single* event. How should we set k with respect to n so that Forrest will remember the event with high probability, i.e., $1 - 1/n^\alpha$?

(c) Suppose we send Forrest k reminders for each of n events. How should we set k with respect to n so that Forrest will remember *all* the events with high probability, i.e., $1 - 1/n^\alpha$?

Problem-4: Hashing **[8 points]**

Recall that a universal hash family H from U to $M = \{0, 1, \ldots, m-1\}$ is a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ each mapping U to M such that for any $a \neq b \in U$, when you pick a random function h from H,

$$\Pr[h(a) = h(b)] \leq \frac{1}{m}$$

An l-universal hash family H from U to $M = \{0, 1, \ldots, m-1\}$ is a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ each mapping from U to M such that for any distinct $a_1, a_2, \ldots, a_l \in U$, and for any $\alpha_1, \alpha_2, \ldots, \alpha_l \in M$, when you pick a random function from H,

$$\Pr[h(a_1) = \alpha_1, h(a_2) = \alpha_2, \ldots, h(a_l) = \alpha_l] = \frac{1}{m^l}$$

Consider this hash function from $U = \{a, b\}$ to $\{0, 1\}$ (where $m = 2$):

	a	b
h_1	0	0
h_2	0	1

(a) Is it a universal hash family? 2-universal?

(b) Give an example of a hash function that is 2-universal but not 3-universal.

Problem-5: **Bin Packing** Let X_i, $1 \leq i \leq n$ be independent and identically distributed random variables following the distribution

$$\mathbb{P}\left(X_i = \frac{1}{2^k}\right) = \frac{1}{2^k} \quad \textit{for} \quad 1 \leq k \leq \infty$$

Each X_i represents the size of the item i. Our task is to pack the n items in the least possible number of bins of size 1. Let Z be the random variable that represents the total number of bins that we have to use if we pack the n items optimally. Consider the following algorithm for performing the packing:

```
BIN-PACKING($x_1, ..., x_n$):
    Sort ($x_1, ..., x_n$) by size in decreasing order. Let ($z_1, ..., z_n$) be the sorted array of items
    Create a bin $b_1$
    For each $i \in \{1, ..., n\}$
        If $z_i$ fits in some of the bins created so far
            Choose one of them arbitrarily and add $z_i$ to it
        Else create a new bin

    **return** the number of created bins.
```

(a) **[8 points]** Prove that the above packing algorithm is optimal for this problem.

(b) **[7 points]** Prove

$$\mathbb{E}[Z] \le \frac{n}{3} + 1$$

(c) **[5 points]** Prove that

$$\mathbb{P}\left(Z \ge \frac{n}{3} + \sqrt{n}\right) \le \frac{1}{e}$$

Problem-6: [15 points] Who is missing? : Parts (a) – (b)

In this problem, your input is a stream of m integers, all of which are from the set $\{1, \ldots, n\}$.

(a) **[7 points]** Suppose $m = n - 1$. This means that there is at least one element from $\{1, \ldots, n\}$ that does not appear in the stream. Suppose you are also guaranteed that there is *exactly* one element that does not appear (so all other elements appear exactly once). Give a deterministic streaming algorithm to find this element with one pass using only $O(\log n)$ space.

where n is computed as $m + 1$. As the elements in the stream are restricted to the range $[1, n]$, storing the running sum can be done in $O(\log n)$ space.

(b) **[8 points]** Now suppose $m = n - 2$. This means that there are at least two elements that do not appear in the stream. Again you are guaranteed that exactly two elements do not appear (so all other elements appear exactly once). Give a deterministic streaming algorithm to find both the missing elements with one pass using $O(\log n)$ space.