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.

Solution: False. Two entries from $\{1,\ldots,n\}^k$ that agree on all coordinates but one (e.g. $(1,1,1,\ldots 1)$ and $(2,1,1,\ldots,1)$) are mapped to the same element with probability $1-1/k > 1/n$.

(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%.

Solution: True. Suppose that the expected running time of the algorithm is at most cn for some constant $c \geq 0$. We create a new algorithm by running the original algorithm til time $100cn$. If the original algorithm stops by that time, we output the answer from the algorithm. Otherwise, we output some value. From Markov inequality, the original algorithm stops before time $100cn$ with probability at least 0.99, which means that the new algorithm outputs a correct answer with probability at least 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 $p_1 p_2$.

Solution: False. Unless the two steps are independent.

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

Solution: False. Consider elements 1 and 3: h_1 and h_2 both cause a collision between them, so in particular a uniformly random hash function chosen from H causes a collision between 1 and 3 with probability 1, greater than the $1/2$ allowed for universal hashing (since there are 2 hash buckets).

(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.

Solution: False. This scheme is not secure because the adversary can simply replace the file with any file and the hash of that file, and you cannot tell the difference.

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

Solution: True.

(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 $-\infty$.

Solution: False
It might converge to 0 depending on the value of $x^{(0)}$. If $x^{(0)} = 1$ for example, $x^{(t+1)} = x^{(t)} - \frac{1}{2}(x^{(t)})^2$ for all t. By induction it can be seen that $0 < x^{(t)} \leq 1 \forall t$ and hence the gradient descent algorithm doesn't converge 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 $h \neq h' \in \mathcal{H}$

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

Solution: False

We know that the set of all possible hash functions is universal. Hence consider this set as our universal family. Let $h(x) = 0 \forall x$ and $h'(x) = 0 \forall x \neq 0$ and $h'(0) = 1$. Clearly h and h' belong in our universal hash family but $\mathbb{P}_{k \in \{0,1\}^u}[h(k) = h'(k)] = 1 - \frac{1}{2^u} > \frac{1}{m}$ for an appropriately chosen m and u.

(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 $\beta \in (0,1)$

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

Solution: True
We denote $\sum_{i=1}^{n} X_i$ by X. Then,

$$X = \sum_{i=1}^{n} X_i = \sum_{i,j=1}^{n} c_{(i,j)}$$

So, X is a sum of n^2 independent random variables which take value in $[0,1]$. Also, $\mathbb{E}[X] = \sum_{i,j=1}^{n} \mathbb{E}[c_{(i,j)}] = \frac{n^2}{2}$. So, from Chernoff bound we get:

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

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

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

Solution: True
First, we will show that for S and S' such that $S \neq S'$, $m_S(v)$ and $m'_S(v)$ are pairwise independent. Without loss of generality we may assume that there exists an i such that $i \in S$, but $i \notin S'$.
Then,

$$\mathbb{P}[m_S(v) = \alpha \wedge m_{S'}(v) = \beta] = \mathbb{P}[x_i = \alpha + \sum_{j \in S \setminus i} x_j \pmod 2 \wedge m_{S'}(v) = \beta]$$

$$= \mathbb{P}[x_i = \alpha + \sum_{j \in S \setminus i} x_j \pmod 2]\mathbb{P}[m_{S'}(v) = \beta] = \mathbb{P}[m_S(v) = \alpha]\mathbb{P}[m_{S'}(v) = \beta]$$

Also, for every $S \subseteq \{1,\dots,k\}$

$$\text{Var}\left[m_S(v)\right] = \frac{1}{4}$$

because $m_S(v)$ is a random variable that follows a Bernoulli distribution with success probability $1/2$.
Therefore,

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

(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 $\Theta(n)$.

Solution: False
Expected runtime = $0.9999c_1 n + 0.0001c_2 n^2 = \Omega(n^2)$.

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. **Solution:**

What we need to do is to give, for each algorithm A, an adversary, B, which decides how to answer the leaf queries of A in order to force A to query every leaf. We say that an internal node is *determined* if, based only on the queries that A has made so far, there is only one possibility for the result of the vote at that node. Now, when A queries a leaf, l, B walks up from l until it finds the first ancestor a of l for which at most one of the siblings of a is currently determined. If a has no sibling in S, always answer true. Set l to be the opposite answer as that sibling has.

We claim that, with this algorithm, A can only determine the value of a node if it knows every value in that node's subtree. This is clearly true of the leaves of the tree. Now, assume that it is true for every node in level k, and consider a node a in level $k+1$ with children x, y, and z. If A determines the value of a, it must know the values of at least two of its children. Suppose WLOG that it knows x and y and that it determined x first. Then, at some point in the execution of A, it determined the value of x by querying leaf l; by our inductive hypothesis, A after that query must have known the value of every node in the subtree of a. Thus before the query, it knew the value of every node in that subtree other than ancestors of l, so that y was the first ancestor of l which had at most 1 sibling determined. Then B would have answered the query to l in order to make the value of y opposite the value of x. But then, since A has determined a and x and y disagree, A must also have determined z, so by induction A in fact has determined the value of every node in the subtree of a. This completes our proof.

(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.

Solution: The algorithm inspects the bits according to a random order. If the first two bits have the same value, that value is the majority, and the algorithm does not inspect the third bit. Otherwise, the algorithm inspects all bits. The reason that the third bit is not inspected with probability at least $\frac{1}{3}$ is that at least two out of the three bits have the same value, and there is probability at least $\frac{1}{3}$ that two identical bits come first.

(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).

Solution: Let $T(t)$ be the number of leaves queried for a node at height t. Then either

$$E[T(t)] = \frac{1}{3} \cdot 2E[T(t-1)] + \frac{2}{3} \cdot 3E[T(t-1)] = \frac{8}{3}T(t-1)$$

if exactly two children have the same value, or

$$E[T(t)] = 2E[T(t-1)]$$

if all three have the same value. In either case, we can bound this as

$$E[T(t)] \leq \frac{8}{3}E[T(t-1)]$$

and since $T(0) = 1$, this solves to $E[T(t)] = (\frac{8}{3})^t$. Thus $E[T(h)] \leq (\frac{8}{3})^{\log_3 n} = n^{\log_3 8/3} \leq n^{0.8928}$.

(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).

Solution: Take $c = 100$. By Markov inequality, we have

$$0.01 = \frac{1}{c} \geq \Pr[T(h) \geq cE[T(h)]] \geq \Pr[T(h) \geq cn^\alpha].$$

In other words, the probability that the algorithm from (c) inspects more than $cn^{0.8928}$ with probability at most 0.01, which yields the desired conclusion.

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.

Solution: These are all independent events. So linearity of expectation applies. Each given event has been remembered with probability $1-2^{-k}$. So in expectation $n(1-2^{-k})$ appointments are remembered.

(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., $1-1/n^{\alpha}$?

Solution: This problem is equivalent to how many times we must flip a coin to get a head with high probability. The probability of k tails in a row is $1/2^k$. Thus exactly $\alpha \lg n$ coin flips suffice.

(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., $1 - 1/n^{\alpha}$?

Solution: We must send at least $k = \Omega(\lg n)$ reminders, because we needed this many reminders to remember one event with high probability.

If we send $k = (\alpha + 1) \lg n$ reminders, then each event is remembered with probability $1-1/n^{\alpha+1}$. By a union bound, we know that all events are remembered with probability $1 - 1/n^{\alpha}$. So, the number of reminders needed is $k = O(\lg n)$.

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?

Solution:

This hash function is universal because a and b collide under only one of the two functions, so they collide with probability 1/2. It is not 2-universal because of the 4 possibilities for α, β, two occur with probability 1/2 and two occur with probability 0.

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

Solution:

The following hash function is one that is 2-universal but not 3-universal. You can verify that for each pair of elements, all 4 possibilities for α, β each occur once (so it is 2-universal). However, it is not 3-universal because you can't get all three elements to simultaneously hash to 1.

	a	b	c
h1	0	0	0
h2	1	0	1
h3	0	1	1
h4	1	1	0

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.

Solution: We will first prove the following claim:

Claim: At any step of the algorithm there is at most one non-full created bin.

proof: We prove the claim by induction on the item being placed on the bins:

- After we place the first item, there is only one created bin. So, the base case holds.
- Assume that we have at most one non-full created bin after we place item k.
- Then, for item $k+1$ there are two cases. The first case is that after the placement of item k, all the bins are full. Then, we create a new bin for item $k+1$ and this bin is the only non-full created bin. The second case is that the bin where item k is placed, b, is not full and all the other bins are full. We need to show that item $k+1$ fits in this bin. We know that the sizes of items are in decreasing order and that all the sizes are of the form 2^{-i}. So, all the item that are in bin b have sizes that are multiples of the size of item k. Assume that the size of item k is 2^{-s}. Then, the free space of bin b is of the form $1 - \lambda 2^{-s} > 0$ for some $\lambda \in \mathbb{N}$. Since λ is integer, it must be the case that $\lambda \in \{1, \ldots, 2^s - 1\}$. Thus, $1 - \lambda 2^{-s} \geq 2^{-s}$. Since the size of item $k+1$ is smaller than 2^{-s}, it fits in bin b.

□

From the claim above, we know that when the algorithm finishes, there will be at most one non-full created bin. If this algorithm was not optimal, there would be a solution with fewer bins. But, this cannot happen, since all the bins but one are already full.

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

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

Solution:

By the claim of part (a), we know that

$$Z = \lceil \sum_{i=1}^{n} x_i \rceil$$

where x_i is the size of item i.
Therefore,

$$\mathbb{E}[Z] = \lceil \sum_{i=1}^{n} \mathbb{E}[x_i] \rceil \leq n\mathbb{E}[x_1] + 1$$

since all the item are chosen independently from the same distribution.
Also,

$$\mathbb{E}[x_1] = \sum_{s=1}^{\infty} \frac{1}{2^s}\frac{1}{2^s} = \sum_{s=1}^{\infty} \frac{1}{4^s} = \frac{1}{3}$$

Therefore,

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

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

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

Solution: Let us define X as $\sum_{i=1}^{n} x_i + 1$. Then, X is the sum of n independent random variables with value in $[0, 1]$ and $\mathbb{E}[X] = \frac{n}{3} + 1$. So, by Chernoff bound:

$$\mathbb{P}\left(X \geq (1+\beta)\left(\frac{n}{3} + 1\right)\right) \leq e^{-\beta^2(\frac{n}{3}+1)/2}$$

For $\beta = \frac{\sqrt{n}-1}{1+\frac{n}{3}}$ and appropriately large n ($n > 64$), we get:

$$\mathbb{P}\left(X \geq \frac{n}{3} + \sqrt{n}\right) \leq e^{-\frac{(\sqrt{n}-1)^2}{2(\frac{n}{3}+1)}} \leq \frac{1}{e}$$

since for $n > 64$, $(\sqrt{n}-1)^2 \geq \frac{3n}{4}$ and $2\left(\frac{n}{3}+1\right) \leq \frac{3n}{4}$.
Now, we remark that since $Z \geq \alpha \Rightarrow X \geq \alpha$, we have:

$$\mathbb{P}(X \geq \alpha) \geq \mathbb{P}(Z \geq \alpha)$$

for every $\alpha \in \mathbb{R}$.
So, we get that

$$\mathbb{P}\left(Z \geq \frac{n}{3} + \sqrt{n}\right) \leq \frac{1}{e}.$$

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.

Solution:

Keep track of the number of elements in the stream, m, as well as the sum of the elements seen so far, $\sum_i A[i]$. Then, at the end, compute the missing element as

$$\frac{n(n+1)}{2} - \sum_i A[i]$$

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.

Solution:

Keep track of the number of elements in the stream, $\sum_i A[i]$ and $\sum_i A[i]^2$. Then, we get the missing numbers from the equations:

$$n = m + 2$$

$$x + y = \frac{n(n+1)}{2} - \sum_i A[i]$$

$$x^2 + y^2 = \frac{n(n+1)(2n+1)}{6} - \sum_i A[i]^2$$

(Need to argue that these equations give a unique solution that can be found efficiently.)