1 Hashing Bashing

1.1 Solution:

1. $O(\log |U|)$ for the first level and for each of the $O(n)$ second level functions, giving a total of $O(n \log |U|)$

2. Suppose we are using two arrays of size $cn^{1.5}$. Consider the $(k+1)^{st}$ item inserted. Since only $k$ buckets (at worst) are occupied, the probability that both candidate locations are occupied is only $(k/(cn^{1.5}))^2$. Thus, the expected number of times an item is actually inserted into an already-occupied bucket is at most

$$\sum_{k=0}^{n-1} \frac{(n-1)n(2n-1)}{6c^2n^3} = O(1)$$

Another approach. Now let’s consider pairwise collisions. Item $k$ collides with item $j < k$ only if (i) one of the candidate locations of item $k$ is the location of item $j$ (this has probability at most $2/(cn^{1.5})$) and (ii) the other candidate location for item $k$ contains at least one element (probability at most $k/(cn^{1.5})$). Thus, the probability $k$ collides with $j$ is at most $2k/(c^2n^3)$. Summing over the $k$ possible values of $j < k$, we find the expected number of collisions for item $k$ is at most $2k^2/(c^2n^3)$. Summing over all $k$, we get the same result as above: $O(1)$ expected collisions.

3. Start with a 2-universal family of hash functions mapping $n$ items to $2n^{1.5}$ locations. Consider any particular set of $n$ items. Consider choosing a random function from the hash family. The probability that item $k$ collides with item $j$ is $1/2n^{1.5}$ by pairwise independence, implying by the union bound that the probability $k$ collides with any item is at most $1/2\sqrt{n}$.

Now suppose that we allocate two arrays of size $2n^{1.5}$ and choose a random 2-universal hash function from the family independently for each array. If an item does not have a collision in both arrays, then it will be placed in an empty bucket by the bash function.
We need merely analyze the probability that this happens for every item (this would make the bash function perfect).

The probability that item \( k \) has a collision in both arrays is at most \((1/2\sqrt{n})^2 = 1/4n\). It follows that the expected number of items colliding with some other item is at most 1/4. This implies in turn that with probability 3/4, every item is placed in an empty bucket by the (perfect) bash function. This in turn implies that some pair of 2-universal hash functions defines a perfect bash for our set of \( n \) items.

Since every set of items gets a perfect bash from this scheme, it follows that the family of pairs of 2-universal functions above is a perfect bash family. Since the 2-universal family has size polynomial in the universe, so does the family of pairs of 2-universal functions. Accordingly, the hash function can be described using \( O(\log |U|) \) bits.

4. When we sample a hash function from the above 2-universal family, we get a 3/4 probability of having no collisions. It follows that if we make 2 or more attempts, we can expect to find a collision-free hash function.

5. If we map our \( n \) items to \( k \) candidate locations in an array of size \( n^{1+1/k} \), our collision odds work out as above and we get a constant number of collisions. Similarly, \( k \) random 2-universal hash families, each mapping to a set of size \( n^{1+1/k} \), has a constant probability of being perfect for any particular set of items, so the set of all such functions provides a perfect family (of polynomial size for any constant \( k \)). This gives a tradeoff of \( k \) probes for perfect hashing in space \( O(n^{1+1/k}) \).

Note that while we can achieve perfect hashing to \( O(n) \) space, the resulting family does not have polynomial size (since a different, subsidiary hash function must be chosen for each sub-hash-table).
2 Sampling with Few Bits

2.1 Solution

1. Let \(0.b_1b_2\ldots\) be the binary fraction representation of \(p_1\). We can treat the random bit stream \(r_1, r_2, \ldots\) also as a binary fraction and return the first item if \(r = 0.r_1r_2\ldots < p_1\) and the second item otherwise. Since the random bit stream is unbiased, we will return the first item with probability \(p_1\). We will make the comparison of the two binary fractions by comparing \(b_i\) and \(r_i\) starting from \(i = 1\). If the two bits differ (with probability \(0.5\)), we can immediately conclude if \(r < p_1\) and return the appropriate item from \(S\). If they are the same (with probability \(0.5\)), we need to compare additional bits to determine the direction of the inequality. Thus, for any arbitrary \(p_1\), the probability that it will take exactly \(n\) comparisons to choose one of the two items is \(\left(\frac{1}{2}\right)^n\). Thus, the expected number of bits to examine is \(B = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n + \sum_{n=2}^{\infty} \left(\frac{1}{2}\right)^n + \cdots = 1 + \frac{1}{2} + \cdots + \frac{1}{2^n} + \cdots = 2 \in O(1)\).

2. When there are more than 2 items in \(S\), we associate \(S\) with the root node of a binary tree that at each node divides the associated set into 2 roughly even partitions. For each node \(i\), we can compute the probability \(p_i\) of drawing an element from the left subtree by summing up the probabilities associated with the elements associated with the subtrees and normalize such that probability of descending into the two subtrees sum up to 1. To draw a sample from the set, we recursively compare the current random bit stream against \(p_i\) at the node starting at the root, and descend down the appropriate subtree based on the result of the comparison. The recursion ends when we reach a subtree with only a single element and return that element. Since the height of the tree is \(O(\log n)\) and each comparison uses \(O(1)\) random bits in expectation, the overall algorithm uses \(O(\log n)\) random bits in expectation per sample. \(^1\)

3. Assume that \(m\) unbiased random bits suffice for the uniform distribution on \(\{1, 2, 3\}\). This means that at the highest resolution, we are able to divide the \([0,1]\) interval into \(2^m\) equally sized subintervals. But we need to apportion intervals so that equal numbers of intervals are assigned to elements 1, 2, and 3. This is impossible since 3 does not divide \(2^m\). More generally, we have \(2^m\) equi-probable states, which must be evenly divided among three elements. Therefore, finite bits will not suffice in the worst case.

\(^1\) By computing cumulative probabilities \(c_i = p_1 + \cdots + p_i\) and reusing the random bit stream when performing binary comparison over \(c_i\), we can further reduce the constant factor of the \(O(\log n)\) algorithm.
3 Lower bound for Balls in Bins

3.1 Solution

1. We computed the following probability in the class,

\[ P[k \text{ balls in bin } 1] = \binom{n}{k} \left( \frac{1}{n} \right)^k \left( 1 - \frac{1}{n} \right)^{n-k}. \]

Then by using the mentioned inequality,

\[ P[k \text{ balls in bin } 1] \geq \binom{n}{k} \left( \frac{1}{n} \right)^k \left( 1 - \frac{1}{n} \right)^{n-k} \]

\[ = \left( \frac{1}{k} \right)^k \left( 1 - \frac{1}{n} \right) \]

\[ \geq \left( \frac{1}{k} \right)^k \left( 1 - \frac{1}{e} \right)^{\frac{1}{n}} \text{, for } n \geq 2 \]

\[ \geq \frac{1}{2e} \left( \frac{1}{k} \right)^k. \]  

(1)

Now, we just set \( k = \frac{c \lg n}{\lg \lg n} \), giving us

\[ P\left[ c \frac{\lg n}{\lg \lg n} \text{ balls in bin } 1 \right] \geq \frac{1}{2e} \left( \frac{1}{c \frac{\lg n}{\lg \lg n}} \right)^{c \frac{\lg n}{\lg \lg n}} \]

\[ = \frac{1}{2e} \left( \frac{\lg \lg n}{c \lg n} \right)^{c \frac{\lg n}{\lg \lg n}} \]

\[ \geq \left( \frac{1}{c \lg n} \right)^{c \frac{\lg n}{\lg \lg n}} \text{, for large enough values of } n \]

\[ = \left( \frac{1}{c 2^{\lg \lg n}} \right)^{c \frac{\lg n}{\lg \lg n}} \]

\[ = \frac{1}{c 2^{\lg \lg n - (c \frac{\lg n}{\lg \lg n})}} \]

\[ = \frac{1}{c 2^{\lg n}} \]

\[ = \frac{1}{cn^c} \]

\[ = \Omega(n^{-c}). \]  

(2)

Setting \( c = 1/2 \), we get \( P[\lg n/2 \lg \lg n \text{ balls in bin } 1] \geq \Omega(1/\sqrt{n}). \)
2. Let $B_i$ be the event that bin $i$ has at least $k$ balls. We first show that conditioning on $\neg B_1, \cdots, \neg B_{i-1}$ only increases the probability that the next bin $i$ does have $k$ balls. We use induction on the number of bins we are conditioning on. The base case is as follows, for $i > 1$:

$$P[B_i] = P[B_i|B_1] \cdot P[B_1] + P[B_i|\neg B_1] \cdot P[\neg B_1]$$

$$\leq P[B_i|\neg B_1] \cdot P[B_1] + P[B_i|\neg B_1] \cdot P[\neg B_1]$$

$$= P[B_i|\neg B_1](P[B_1] + P[\neg B_1])$$

$$= P[B_i|\neg B_1]. \quad (4)$$

In line 3, we notice that $B_i$ is more likely if the first bin does not have $k$ balls comparing to the case in which the first bin has at least $k$ balls, because then there are more balls that can be in bin $i$.

Next we assume that $P[B_i] \leq P[B_i|\neg B_1 \land \cdots \land \neg B_j]$ (for $i > j + 1$). Then by the same argument we made for line 3, we have:

$$P[B_i] \leq P[B_i|\neg B_1 \land \cdots \land \neg B_j]$$

$$= P[B_i|\neg B_1 \land \cdots \land \neg B_j \land B_{j+1}] \cdot P[B_{j+1}]$$

$$+ P[B_i|\neg B_1 \land \cdots \land \neg B_j \land \neg B_{j+1}] \cdot P[\neg B_{j+1}]$$

$$\leq P[B_i|\neg B_1 \land \cdots \land \neg B_j \land \neg B_{j+1}]$$

$$\leq P[B_i|\neg B_1 \land \cdots \land \neg B_j \land \neg B_{j+1}] \quad (5)$$

Thus, we have concluded that conditioning on bins not having $k$ balls increases the chances that the next bin does. Specifically, the induction ends at proving

$$P[B_i] \leq P[B_i|\neg B_1 \land \cdots \land \neg B_{i-1}] .$$

Conversely, we have

$$P[\neg B_i] \geq P[\neg B_i|\neg B_1 \land \cdots \land \neg B_{i-1}] ,$$

because this is exactly $1 - P[B_i]$.

So now let’s solve the real problem, with $k = \lg n/2 \lg \lg n$. From part (a), we have

$$P[B_i] = P[\text{Bin } i \text{ has at least } \lg n/2 \lg \lg n \text{ balls}] \geq \frac{1}{2\sqrt{n}} .$$

Thus, we have

$$P[\neg B_i] = P[\text{Bin } i \text{ has at most } \lg n/2 \lg \lg n \text{ balls}] \leq 1 - \frac{1}{2\sqrt{n}} .$$
So now we just solve for all bins having at most this many balls:

\[
\mathbb{P}[\text{all bins have } \leq \frac{\lg n}{2} \lg \lg n \text{ balls}] = \mathbb{P}[\neg B_1] \cdot \mathbb{P}[\neg B_2 | \neg B_1] \cdots \mathbb{P}[\neg B_n | \neg B_1 \land \ldots \land \neg B_{n-1}]
\leq \mathbb{P}[\neg B_1] \cdot \mathbb{P}[\neg B_2] \cdots \mathbb{P}[\neg B_n]
\leq \left(1 - \frac{1}{2\sqrt{n}}\right)^n
\leq e^{-\left(\frac{1}{2\sqrt{n}}\right)^n}
= e^{-\frac{\sqrt{n}}{2}}.
\]

(6)

So the probability is exponentially small that all bins have have fewer than \(\frac{\lg n}{2} \lg \lg n\) balls. Therefore, we conclude that with high probability, some bin has \(\Omega(\lg n/ \lg \lg n)\) balls.
4 Balls and Bins with replacement

4.1 Solution

Here we assume that the number of processors is $10n$.

1. We consider ordering the jobs arbitrarily and processing the jobs in that order. Only the first job that assigns to a processor get executed in that round.

Let $X_i$ be indicator random variables as follows:

$$X_i = \begin{cases} 
1 & \text{if job } i \text{ assigns to a previously occupied processor} \\
0 & \text{otherwise} 
\end{cases}$$

The $X = \sum X_i$ is the number of unprocessed job in a round.

In the worst case, when we’re considering job $i$, all previous jobs have been placed in different processors. Thus, there are at most $i - 1$ occupied processors. Therefore, we have $\mathbb{P}[X_i] \leq \frac{i-1}{10n}$.

At this point we may be tempted to apply a Chernoff bound, but $X_i$s are not independent, so we define a new set of random variables $Y_i$ such that $Y_i = 1$ with probability $\frac{i-1}{10n}$. Thus, $X$ is stochastically dominated by $Y = \sum Y_i$.

Accordingly, to upper bound $\mathbb{P}[X \geq \alpha^2 n]$, it suffices to upper bound $\mathbb{P}[Y \geq \alpha^2 n]$.

Now:

$$\mathbb{E}[Y] = \mathbb{E}[\sum Y_i] = \sum \mathbb{E}[Y_i] = \sum_{i=1}^{an} \mathbb{P}[Y_i] \leq \sum_{i=1}^{an} \frac{i-1}{10n} = \frac{(an)(an-1)}{20n} = \frac{a(an-1)}{20} \leq \frac{\alpha^2 n}{20}. \quad (7)$$

So, $\mathbb{P}[Y \geq (\frac{\alpha}{2})^2 n] \leq \mathbb{P}[Y \geq 5\mathbb{E}[Y]]$.

Applying Chernoff, we conclude that, $\mathbb{P}[Y \geq 5\mathbb{E}[Y]] \leq e^{-O(\alpha^2 n)}$ which is $\leq 1/2n^2$ as long as $\alpha^2 n \geq c \log n$ for a sufficiently large constant $c$.

We conclude that $\mathbb{P}[X \geq (\frac{\alpha}{2})^2 n] \leq 1/2n^2$.
2. We call an unlucky round one which has too many unprocessed balls at the end (i.e., more than $\alpha^2 n$). Since the probability of getting unlucky rounds is $1/n^2$, we can just use union bound to argue that if we run for $n$ rounds (we can’t possibly run for more), it’s high probability that none of those rounds will be unlucky. Thus, with probability $(1 - 1/2n)$, all rounds are lucky. So how many rounds do we need with $T(\alpha n) = T((\frac{\alpha}{2})^2 n) + 1$? Well, that’s just $O(\log \log n)$. So what happens when we get down to something like $\log n$ jobs? Well, then we can just look at the probability that two particular jobs colliding is $1/n$. Thus, the probability of any pair of jobs colliding is at most $\log^2 n/n$. Thus, running for a constant $c$ number of rounds, we can get a probability that is at most $(\log^2 n/n)^c$, which is $O(n^{-(c-1)})$. So once we have few enough jobs left, we’re likely to finish in a constant number of rounds, and we’re done after a union bound with the probability that all rounds are lucky.
5 Consistent Hashing with Inconsistent Views

5.1 Solution

Let there be $m$ machines and $n$ clients interested in a specific data item. Using the consistent hashing scheme, each machine and data item is mapped to the cyclical interval $[0, 1]$. For convenience, label the machines $1, \ldots, m$ in order of appearance after the position of the data item. Thus, machine $1$ owns the data item. However, if a particular client believes that machines $1, \ldots, k$ are down, it will query machine $k + 1$ for the data.

Let $z^j_i = 1$ if client $j$ believes machine $i$ is up and $0$ otherwise. Thus, for $k < \frac{m}{2}$, the probability that a client $j$ queries machine $k + 1$ is bounded by:

$$\Pr[\text{client } j \text{ queries machine } \geq k + 1] = \Pr[z^j_1 = z^j_2 = \cdots = z^j_k = 0] < 2^{-k}$$

Choosing $k = c \log n$ and applying the union bound, the probability that any of the $n$ clients query machine $\geq k + 1$ is bounded by $n2^{-k} = n^{-(c-1)}$. Thus, with high probability, no client will query machine $\geq O(\log n)$. Alternatively, at most $O(\log n)$ machines will be queried with high probability.