COMPSCI 690RA: Problem Set 1

Due: 2/15 by 8pm in Gradescope.

Instructions:

- You are allowed to, and highly encouraged to, work on this problem set in a group of up to three members.
- Each group should **submit a single solution set**: one member should upload a pdf to Gradescope, marking the other members as part of their group in Gradescope.
- You may talk to members of other groups at a high level about the problems but **not work through the solutions in detail together**.
- You must show your work/derive any answers as part of the solutions to receive full credit.

Hint: The following two inequalities may be helpful in various places (and generally throughout the course): for any $x > 0$, $(1+x)^{1/x} \leq e$ and $(1-x)^{1/x} \leq 1/e$.

1. Probability Practice (6 points)

1. (2 points) Consider storing n items in a hash table with $m = n$ buckets, using a fully random hash function $\mathbf{h} : [n] \to [n]$ (i.e., each item is assigned independently to a uniform random bucket). What is the expected load factor of the hash table? I.e., what is the expected fraction of buckets that have at least one item in them? What is the limit of this value as $n \to \infty$? What about when $m = 2n$? **Hint:** Use linearity of expectation.

   Consider a single bucket in the hash table. The probability that none of the n items are placed in it is $(1-1/n)^n$. By linearity of expectation, the expected load factor is thus $1-(1-1/n)^n$. Additionally, $\lim_{n\to\infty} 1-(1-1/n)^n = 1-1/e \approx .632$.

   When $m = 2n$ buckets, the probability that none of the n items are placed in a bucket $(1-1/(2n))^n$. By linearity of expectation, the expected load factor is thus $1-(1-1/(2n))^n$. Additionally, $\lim_{n\to\infty} 1-(1-1/(2n))^n = 1-1/e^{.5} \approx .393$.

2. (2 points) Consider a random walk on a d-dimensional grid. At each step, the walk chooses one of the d-dimensions uniformly at random, and takes a step in that direction – up with probability 1/2 and down with probability 1/2. Assume that the walk starts at the origin and ends at position $x \in \mathbb{Z}^d$ and takes n steps. What is $\mathbb{E}[\|x\|_2^2]$, where $\|x\|_2^2$ denotes the squared Euclidean norm of x.

   We can write via linearity of expectation, $\mathbb{E}[\|x\|_2^2] = \sum_{i=1}^{n} \mathbb{E}[x_i^2]$, where x_i is the position in the i^{th} dimension. We can then write $x_i = \sum_{j=1}^{n} s_j$, where $s_j = 0$ with probability $\frac{d-1}{d}$, $s_j = 1$ with probability $\frac{1}{2d}$ and $s_j = -1$ with probability $\frac{1}{2d}$. We have $\mathbb{E}[s_j] = 0$

and $\text{Var}[s_j] = \mathbb{E}[s_j^2] = \frac{1}{d}$. Thus, since the s_j are independent, by linearity of variance, $\text{Var}[x_i] = \mathbb{E}[x_i^2] = n/d$. This then gives $\mathbb{E}[\|x\|_2^2] = \sum_{i=1}^{n} \mathbb{E}[x_i^2] = n$.

That is, no matter what d is, the expected squared Euclidean distance from the origin after n random steps is exactly n.

3. (2 points) A monkey types on a 26-letter keyboard that has lowercase letters only. Each letter is chosen independently and uniformly at random from the alphabet. If the monkey types 100,000,000 letters. what is the expected number of times the sequence 'random' appears? Give an upper bound on the probability that 'random' appears at least once.

   Use linearity of expectation. There are $10^8 - 5$ possible positions where the sequence 'random' can start. The probability that 'random' appears at each of these starting positions is $1/26^6$. So the expected number of times we see the sequence is $\frac{10^8-5}{26^6} = .324$. By Markov's inequality, we thus have $\Pr[\#\text{ appearances } \geq 1] \leq \frac{.324}{1} = .324$.

2. Translating Between Randomized Guarantees (4 points)

1. (2 points) Suppose I have a Las Vegas algorithm that solves a decision problem with expected runtime T. For any $\delta > 0$, describe and analyze a Monte-Carlo algorithm that correctly answers the decision problem with probability at least $1 - \delta$ and has worst case runtime $O(\log(1/\delta) \cdot T)$.

   Run the Monte-Carlo algorithm $\log_2(1/\delta)$ times independently, halting any run that takes $\geq 2T$ steps. By Markov's inequality, the probability than a run is halted is at most $\frac{T}{2T} = \frac{1}{2}$. So, the probability that all runs are halted is at most $\left(\frac{1}{2}\right)^{\log_2(1/\delta)} = \delta$. Thus, with probability $\geq 1 - \delta$ at least one run finishes, and if we return the output of that run, we correctly answer the decision problem. In total, the worse case runtime of this approach is $2T \cdot \log_2(1/\delta)$.

2. (2 points) Suppose I have a Monte-Carlo algorithm that solves a decision problem with worst case runtime T and at least $2/3$ probability of correctness. For any $\delta > 0$, describe and analyze a Monte-Carlo algorithm that correctly answers the decision problem with probability at least $1 - \delta$ and has worst case runtime $O(\log(1/\delta) \cdot T)$.

   Run the Monte-Carlo algorithm r times independently, and output the majority answer to the decision problem. Let $\mathbf{X}$ be the number of runs on which the Monte-Carlo algorithm is correct. Our output will be correct as long as $\mathbf{X} > r/2$. Thus, we need to bound $\Pr[\mathbf{X} \leq r/2]$. To do this, we use a Chernoff bound (one could also directly use the fact that $\mathbf{X}$ follows a binomial distribution).

   We have $\mathbb{E}[\mathbf{X}] = 2/3 \cdot r$ and thus

$$\Pr[\mathbf{X} \leq r/2] \leq \Pr[|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq r/6] = \Pr[|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq 1/4 \cdot \mathbb{E}[\mathbf{X}]]$$

   . Applying a Chernoff bound with $\mu = 2/3 \cdot r$ and $\delta = 1/4$, we thus have:

$$\Pr[\mathbf{X} \leq r/2] \leq 2\exp\left(-\frac{1/4^2 \cdot 2/3 \cdot r}{2 + 1/4}\right) \leq 2\exp\left(-r/54\right).$$

   If we set $r = 54(\ln(1/\delta)+1)$ then this probability is upper bounded by $2 \cdot \exp(-\ln(1/\delta)-1) \leq \delta$. Thus, our algorithm runs in time $r \cdot T = O(\log(1/\delta) \cdot T)$ and succeeds with probability at least $1 - \delta$.

3. Finding Random Primes (4 points)

A common application of randomized primality testing algorithms is in finding large primes for use in cryptographic schemes, such as RSA. Suppose you have a Monte-Carlo primality testing algorithm that, given an integer input x, outputs 'yes' or 'no'. If x is prime, the algorithm always outputs 'yes'. If x is composite, the algorithm outputs 'no' with probability at least $1/2$.

For any $\delta \geq 0$, describe an algorithm that makes $O(\log(n) \cdot \log(1/\delta))$ calls to this tester and outputs x such that, with probability at least $1 - \delta$, x is a uniformly random prime in $\{1, \ldots, n\}$.

Hint 1: Let $\pi(n)$ be the *prime counting function*: the number of prime numbers less than or equal to any integer n. You may use the fact that for any $n \geq 17$, $\frac{n}{\log n} \leq \pi(n)$. This fact is closely related to the prime number theorem (PNT).

Hint 2: It might be easier to first shoot for $O(\log(n) \cdot \log(1/\delta)^2)$ calls and then figure out how to tighten your analysis.

Our algorithms will be as follows:

1. Pick $r = \log(n) \cdot (\ln(1/\delta) + 1)$ integers $x_1, \ldots, x_r$ uniformly an independently at random from $\{1, \ldots, n\}$.
2. For $i = 1 \ldots r$:
   (a) Run the primality tester repeatedly on x_i, halting if the tester over outputs 'no' or if the total number of calls made (over the course of the full algorithm) exceeds $s = c \log(n) \cdot (\ln(1/\delta) + 1)$.
   (b) If the tester never outputs 0 on x_i, return x_i.
3. If the algorithm has not yes returned some x_i, output 'FAIL'.

Running Time: The algorithm only ever makes at most $s = c_2 \log(n) \cdot \log(1/\delta)$ calls to the primality tester, satisfying the desired running time.

Correctness: We first upper bound the probability that none of $x_1, \ldots, x_r$ are prime, in which case the algorithm will fail. By Hint 1, each x_i is prime with probability at least $\frac{1}{\log n}$. So the probability that none are prime is at most:

$$\left(1 - \frac{1}{\log n}\right)^r = \left[\left(1 - \frac{1}{\log n}\right)^{\log n}\right]^{r/\log n} \leq \frac{1}{e^{(\ln(1/\delta)+1)}} \leq \delta/e \leq \delta/2.$$

Now, given that at least one of the x_i is prime, the algorithm will only fail if there is some composite on which all calls made to the primality tester output 'yes'. We need to upper bound the probability of this happening by $\delta/2$. Then, that total probability if success is at least $(1 - \delta/2)^2 \geq 1 - \delta$.

Let x_i be the first prime in our list of numbers. Observe that once the algorithm starts testing x_i, it uses all its tests on that prime and outputs x_i. Thus, we need to upper bound the probability that the algorithm outputs some x_j for $j < i$. This will happen only if the algorithm runs at least s calls to the primality tester on the composites $x_1, \ldots, x_{i-1}$. For this to happen, the algorithm must make less than $(i - 1) \leq r = \log(n) \cdot (\ln(1/\delta) + 1)$ calls on these composites that output 'no'. Consider running s independent calls to the primality tester on composite numbers, and let $\mathbf{X}$ be a random variable equal to the number of 'no' outputs. To fail, we need $\mathbf{X} < (i - 1) \leq r = \log(n) \cdot (\ln(1/\delta) + 1) \leq 1/3 \cdot s$ if we set $c \geq 3$ so $s \geq 3 \log(n) \cdot (\ln(1/\delta) + 1)$. Each call outputs 'no' with probability at least $1/2$, and so $\mathbb{E}[\mathbf{X}] \geq s/2$. Thus, we can upper bound the probability of failure by:

$$\Pr(\mathbf{X} \leq 1/3 \cdot s) \leq \Pr(|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq 1/3 \cdot \mathbb{E}[\mathbf{X}]).$$

Applying a Chernoff bound with $\mu = s/2 = c/2 \cdot \log(n) \cdot (\ln(1/\delta) + 1)$ and $\delta = 1/3$ this probability is upper bounded by $2\exp\left(-\frac{1/3^2 \cdot c/2 \cdot \log(n) \cdot \ln(1/\delta)}{2+1/3}\right) \leq \delta/2$ if we set c large enough. This completes the argument.

4. Randomized Routing (4 points)

Consider the following simplified model of a routing problem under communication constraints: we have n nodes in a fully connected network. Each node has n messages that it would like to deliver to n (not necessarily unique) destinations. Also, for simplicity, we assume that each node is the intended recipient of exactly n messages. In each round, a node can send at most one message along each connection in the network. Assume that the message contains the id of its final intended recipient. Naively, sending these messages could take n rounds, if e.g., some node v needs to send n messages to some other node u.

Describe and analyze a randomized scheme that sends all messages in $O(\log n)$ rounds with high probability, i.e., with probability at least $1 - 1/n^c$ for a large constant c.

Hint: Have each node initially send each of its n messages to a random recipient, who will then forward the message to its final destination.

To think about: Can you do better than $O(\log n)$ rounds? What can you achieve deterministically?

Our protocol will be as follows:

- **Round 1:** Each node picks a random permutation of the n nodes in the graph (including itself). It then sends its at most n messages in order, to the nodes in this permutation. Thus, it sends at most one of the n messages to each other node. Note: It may 'send a message to itself' which is a no-op.
- **Rounds 2+:** Each node delivers sequentially all messages it has from Round 1 directly to their destinations.

Observe that if after Round 1, each node i has at most r messages destined for any other node j, then the whole protocol terminates in $r+1$ rounds. Thus, our goal is to show that for $r = O(\log n)$, all nodes have at most r messages to send to all other nodes after Round 1, with high probability.

Let $\mathbf{X}_{ij}$ be a random variable denoting the number of messages that node i has to send to node j after round 1. Observe that $\mathbf{X}_{ij} = \sum_{k=1}^{n} \mathbf{Y}_{ki}$, where $\mathbf{Y}_{ki} = 1$ if node k sent a message destined for j to node i in Round 1, and $\mathbf{Y}_{ki} = 0$ otherwise. Observe that the $\mathbf{Y}_{ki}$ are independent $\{0,1\}$ random variables. $\mathbb{E}[\mathbf{Y}_{ki}] = n_k/n$ where n_k is the number of messages that node k originally had to send to node j. Thus, $\mathbb{E}[\mathbf{X}_{ij}] = \sum_{k=1}^{n} \mathbb{E}[\mathbf{Y}_{ki}] = \frac{1}{n}\sum_{k=1}^{n} n_k \leq 1$, since at most n messages are destined for node j in total. Applying a Chernoff bound with $\delta = c\ln n$ for some constant c and $\mu = 1$,

$$\Pr[\mathbf{X}_{ij} \geq c\ln n + 1] \leq \frac{e^{c\ln n}}{(c\ln n + 1)^{c\ln n + 1}} \leq \left(\frac{e}{c}\right)^{c\ln n}. \tag{1}$$

If we set $c > e^2$, then this probability is at most $\frac{1}{e^{c\ln n}} = \frac{1}{n^c}$. Thus, $\Pr[\mathbf{X}_{ij} \geq c\ln n + 1] \leq \frac{1}{n^c}$ for all i, j. Taking a union bound over all n^2 variables $\mathbf{X}_{ij}$, we have $\Pr[\max_{i,j} \mathbf{X}_{ij} \geq c\ln n + 1] \leq \frac{1}{n^{c-2}}$. Thus, our algorithm runs in $c\ln n + 2 = O(\log n)$ rounds with probability $\frac{1}{n^{c-2}}$ for any constant c, completing the argument.

Note: One could be more careful in the analysis above, to get $O\left(\frac{\ln n}{\ln\ln n}\right)$ time, as we did with the balls-into-bins analysis in class. I was loose when I just bounded $c\ln n > c$ in (1).

5. Building on Freivald's Algorithm (4 points)

Consider two matrices $A, B \in \mathbb{R}^{n \times n}$ whose product AB has just k non-zero entries. Describe a randomized algorithm that computes AB in $O(n^2 \cdot k \cdot \log n)$ time, with probability at least 99/100.

Hint: Use repeated applications of Freivald's approach to determine which columns of AB are non-zero. Some students pointed out a much simpler and faster alternative to my solution that runs in $O(n^2(\log n + k))$ time. I think it is more natural, so don't be surprised if you get that runtime.

Bonus: (2 points) Give an algorithm that runs in $O(nk^2 + n^2 \log n)$ time, which for the interesting case of $k < n$ is faster than both my runtime and the students' runtime stated above. 3 points awarded if you achieve runtime $O(nk + n^2 \log n)$, which is even better. I believe this may be possible but am not sure.

Our algorithm will output $C \in \mathbb{R}^{n \times n}$ such that with high probability $C = AB$, as follows:

1. Let $t = c \log_2 n$ and let $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \{0,1\}^n$ be independent random vectors, each which has its entries set independently to 0 w.p. 1/2 and 1 w.p 1/2.
2. Compute $\mathbf{y}_i = AB\mathbf{x}_i$ for all $\mathbf{x}_i$.
3. For each $j \in [n]$ if $\mathbf{y}_i(j) = 0$ for all $i \in [t]$, set the j^{th} row of C to be all zeros. Else, set the j^{th} row of C to be $a_j^T B$, where $A_j \in \mathbb{R}^n$ is the j^{th} row of A.

Correctness: AB has at most k nonzero entries by assumption. Thus, AB has at most k nonzero rows. Consider any row j of AB that is all zeros. Then we can see that $\mathbf{y}_i(j) = 0$ always. Thus, in step 3, we set this row in C to be all zeros. So C is correct on all such rows.

Next, consider any row j of AB that is nonzero. By the Frievald's analysis shown in class, $\Pr[\mathbf{y}_i(j) = 0] \leq 1/2$. Thus, letting $\mathcal{E}_j$ be the event that row j is set to 0 in step 3, $\Pr[\mathcal{E}(j)] \leq 1/2^t \leq 1/n^c$. Let $\mathcal{E}$ be the event that *any nonzero row* j is set to 0 in step 3. Taking a union bound over the at most k non-zero rows, $\Pr[\mathcal{E}] \leq k/n^c \leq 1/100$ if we set c large enough. Thus, with probability, at least 99/100, in step 3, we correctly compute each nonzero row of AB as $a_j^T B$. So overall, with probability at least 99/100, $C = AB$.

Runtime: The first two steps take $O(n^2 \log n)$ time in total, since we must multiply AB by $O(\log n)$ random vectors. In step 3, we compute $a_j^T B$ for at most k rows (the non-zero rows of AB). Each of these matrix-vector products takes $O(n^2)$ time. Thus, the total runtime of this step is $O(n^2 k)$, giving overall runtime $O(n^2(\log n + k))$.

Bonus 1: To achieve runtime $O(nk^2 + n^2 \log n)$ we can additionally let $\mathbf{z}_1, \ldots, \mathbf{z}_t \in \{0,1\}^n$ be independent random vectors, and compute $\mathbf{w}_i = (AB)^T \mathbf{z}_i$. If the j^{th} column of AB is all zeros, we will have $\mathbf{w}_i(j) = 0$ for all i. However, if the j^{th} column has a nonzero in it, $\Pr[\mathbf{w}_i(j) = 0] \leq 1/2$. Thus, with high probability, we can identity the at most k non-zero columns of AB by checking if $\mathbf{w}_i(j) = 0$ for all i. Overall, using $O(n^2 \log n)$ operations to multiply AB and $(AB)^T$ by each $\mathbf{x}_i$ and $\mathbf{z}_i$, we can, with high probability, identify the at most k non-zero rows and columns of AB. The entries in the intersection of these rows and columns, of which there are $\leq k^2$, are the only entries that can be non-zero in AB. Thus, we just need to compute $(AB)_{ij} = a_j^T b_i$ for k^2 pairs i, j, where $a_j \in \mathbb{R}^n$ is the j^{th} row of A and $b_i \in \mathbb{R}^n$ is the i^{th} columns of B. This takes $O(nk^2)$ time, giving total runtime $O(nk^2 + n^2 \log n)$.

Bonus 2: Here is a rough idea of an approach that gives $\tilde{O}(nk + n^2)$ time – i.e., it achieves what I was asking for up to log factors. A similar runtime is achieved by `https://arxiv.org/pdf/1108.1320.pdf`. I'm not sure how to achieve exactly $O(nk + n^2 \log n)$, without losing any logs. Maybe I will learn from one of your solutions!

We want to identify just the k nonzero entries of AB. Once we have done this, we can compute $(AB)_{ij} = a_j^T b_i$ for just these entries in $O(nk)$ time. Start with the solution above – we identify a subset of $\leq k$ rows and $\leq k$ columns that, with high probability, contain all non-zero entries of AB. Let $A_S \in \mathbb{R}^{k \times n}$ and $B_S \in \mathbb{R}^{n \times k}$ be A, B restricted to these subsets of rows and columns respectively. So identifying the nonzero entries in $A_S B_S \in \mathbb{R}^{k \times k}$ suffices to solve the problem.

Let $\mathbf{S}_1, \ldots, \mathbf{S}_t \in \mathbb{R}^{k \times \log^c k}$ be independent ℓ_0 sampling matrices (Lecture 4), for $t = c \log k$. We can compute all sketches $A_S B_S \mathbf{S}_j$ in $O(t \cdot nk \log^c k) = O(nk \log^{c+1} k)$ time. From each sketch, we can recover a random nonzero entry in each row of $A_S B_S$. If a row has just 1 non-zero entry in it, we will recover the same entry with each sketch. If a row has ≥ 2 non-zero entries in it, we will with high probability recover ≥ 2 entries. We can then repeat the same process with columns.

After this, we can remove all rows/columns that have at most 1 non-zero entry in them – since we will have already recovered these entries with high probability. We will be left only with rows and columns (at most $k/2$ of each) with ≥ 2 non-zero entries in each. We can then repeat, but this time, set $t = 2c \log k$ times, and with high probability recover all entries in any row/column with between 2 and 4 non zero entries. Then remove such rows and repeat, but with $t = 4c \log k$ and recover all entries in any row/column with between 4 and 8 non zero entries, and so on. The maximum remaining number of rows/columns is cut in half each time. So our total runtime will be on order of

$$nk \cdot \log^{c+1} k + n \cdot \frac{k}{2} \cdot 2 \log^{c+1} k + n \cdot \frac{k}{4} \cdot 4 \log^{c+1} k + \ldots = O(nk \log^{c+2} k).$$

Note: This question was inspired by `https://d-nb.info/1114141399/34`, which along with `https://arxiv.org/pdf/1108.1320.pdf` could make an interesting starting point for a course project. Can you give a tight runtime bound? Can you simplify the existing algorithms? What about if AB is not exactly k sparse, but just has a small number of large entries?

6. Stacking Hash Tables (8 points)

1. (3 points) Consider storing n items in a hash table with m buckets, using a fully random hash function $\mathbf{h} : [n] \to [m]$ (i.e., each item is assigned independently to a uniform random bucket). Prove that if $m = cn^2$ for some sufficiently large constant c, then there are *no collisions* in the hash table with probability at least $9/10$. Thus, the table has worst case $O(1)$ lookup time. **Hint:** Use linearity of expectation, considering all pairs of items (i, j) that may collide.

   Let $\mathbf{X}_{ij} = 1$ if items i, j are mapped to the same bucket, and $\mathbf{X}_{ij} = 0$ otherwise. Let the total number of collisions be denoted $\mathbf{X} = \sum_{i,j} \mathbf{X}_{ij}$. We would like to upper bound $\Pr[\mathbf{X} \geq 1]$. Observe that $\mathbb{E}[\mathbf{X}_{ij}] = 1/m$ if we have uniformly random hash values in $[m]$. Thus, $\mathbb{E}[\mathbf{X}] = \sum_{ij} 1/m = \frac{\binom{n}{2}}{m}$. If we set $m = 5n^2$, then we have $\mathbb{E}[\mathbf{X}] \leq \frac{n^2/2}{5n^2} \leq 1/10$. Thus, by Markov's inequality, $\Pr[\mathbf{X} \geq 1] \leq \frac{\mathbb{E}[\mathbf{X}]}{1} \leq \frac{1}{10}$. This completes the proof – with probability $\geq 9/10$ there are no collisions at all, and so the hash table hash $O(1)$ lookup time.

   Note: You could instead solve this problem by talking about the events that any two items collide and using the union bound in place of Markov's inequality – the union bound is really just a special case of Markov's inequality.

2. (3 points) Consider storing n items in two hash tables with m buckets each, using two different fully random hash functions $\mathbf{h}_1 : [n] \to [m]$, $\mathbf{h}_2 : [n] \to [m]$. An item x is stored in bucket $\mathbf{h}_1(x)$ of the first table, unless this bucket already has an item in it. In that case, it is stored

in bucket $\mathbf{h}_2(x)$ of the second table. How large must m be such that, with probability at least 9/10, there are no collisions in this scheme? I.e., so that all buckets in both the first and second tables have at most 1 item in them. What is the worst case lookup time for this scheme?

Let $\mathbf{X}_{ij}$ be the indicator of the event that i, j 'collide' in the first table – i.e. that they are mapped to the same bucket and so that at least one of them must be mapped on to the second table. Let $\mathbf{Y}_{ij}$ be the indicator that i, j collide in the second table. Let $\mathbf{X} = \sum \mathbf{X}_{ij}$ and $\mathbf{Y} = \sum \mathbf{Y}_{ij}$ be the total number of collisions in the first and second table respectively. We need to show that $\Pr[\mathbf{Y} = 0] \geq 9/10$.

We do the analysis in two steps. Let $\mathcal{E}_x$ be the event that $\mathbf{X} \leq \frac{10n^2}{m}$ and let $\mathcal{E}_y$ be the event that $\mathbf{Y} = 0$. We first argue that $\Pr[\mathcal{E}_x] \geq 19/20$. This follows from Markov's inequality as in part one: $\mathbb{E}[\mathbf{X}_{ij}] = 1/m$ and thus $\mathbb{E}[\mathbf{X}] \leq \frac{n^2}{2m}$, and so by Markov's inequality, $\Pr[\mathbf{X} \geq \frac{10n^2}{m}] \leq 1/20$.

Next we argue that $\Pr[\mathcal{E}_y|\mathcal{E}_x] \geq 19/20$. In turn, this implies that

$$\Pr[\mathcal{E}_y] \geq \Pr[\mathcal{E}_y|\mathcal{E}_x] \cdot \Pr[\mathcal{E}_x] \geq (19/20)^2 \geq 9/10,$$

as desired.

Assume that $\mathcal{E}_x$ holds. Since each collision involves two items, this means that at most $\frac{20n^2}{m}$ items are hashed into the second table. There are thus at most $\binom{20n^2/m}{2} \leq \frac{200n^4}{m^2}$ pairs of items that are hashed into the second table. Again following part one, we have $\mathbb{E}[\mathbf{Y}|\mathcal{E}_x] \leq \frac{200n^4/m^2}{m} \leq \frac{200n^4}{m^3}$.

If we set $m = c \cdot n^{4/3}$ for $c = (200 \cdot 20)^{1/3}$, then this expectation is upper boundeed by 1/20, and thus by Markov's inequality, $\Pr[\mathbf{Y} \geq 1|\mathcal{E}_x] \leq 1/20$ and so $\Pr[\mathcal{E}_y|\mathcal{E}_x] \geq 19/20$ as required.

So overall, we have a scheme with $O(1)$ worse case lookup time and $O(m) = O(n^{4/3})$ space.

3. (2 points) Generalize the above approach to use any constant number of hash tables $t \geq 1$. What is the tradeoff between the amount of storage you must allocate to the hash tables and the worst-case lookup time?

The rough idea is that, following the analysis above, we expect $\approx \frac{n^2}{m}$ collisions in the 1^{st} table. In the second we expect $\approx \frac{\left(\frac{n^2}{m}\right)^2}{m} = \frac{n^4}{m^3}$. In the t^{th} we expect $\approx \frac{n^{2^t}}{m^{2^t-1}}$. We can use Markov's inequality to bound the actual number of collisions by $\approx \frac{n^{2^t}}{m^{2^t-1}}$ with good probability So, to have no collisions in the last table with good probability we need $m^{2^t-1} \approx n^{2^t}$ and so $m \approx n^{\frac{2^t}{2^t-1}} = n^{1+\frac{1}{2^t-1}}$. This quick becomes nearly linear as t grows – in particular, $m = O(n)$ when $t = O(\log \log n)$. The query time will be $O(t)$.

More formally, we define t random variables $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_t$, indicating the number of collisions in each table. We can let $\mathcal{E}_i$ be the event that $\mathbf{X}_i \leq \frac{(ct)^{2^i-1} \cdot n^{2^i}}{m^{2^i-1}}$ for some constant c. Observe that for $i = 1$, $\mathbb{E}[\mathbf{X}_1] \leq n^2/m$ and thus by Markov's inequality,

$$\Pr\left[\mathbf{X}_1 \leq \frac{(ct)^{2^1-1} \cdot n^{2^1}}{m^{2^1-1}}\right] = \Pr[\mathcal{E}_1] \geq 1 - \frac{1}{ct}.$$

For $i > 1$ we have:

$$\mathbb{E}[\mathbf{X}_i|\mathcal{E}_{i-1}] \leq \frac{(ct)^{2^i-2} n^{2^i}/m^{2^i-2}}{m} = \frac{(ct)^{2^i-2} n^{2^i}}{2m^{2^i-1}}.$$

Thus, by Markov's inequality, we have:

$$\Pr\left[\mathbf{X}_i \leq \frac{(ct)^{2^i-1} \cdot n^{2^i}}{m^{2^i-1}} \mid \mathcal{E}_{i-1}\right] = \Pr[\mathcal{E}_i | \mathcal{E}_i - 1] \geq 1 - \frac{1}{ct}$$

In turn, we have $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \ldots \cap \mathcal{E}_t] \geq \left(1 - \frac{1}{ct}\right)^t \geq 1 - 1/c$. So if we set $c = 20$, all events hold with probability at least $19/20$. When $\mathcal{E}_t$ holds, the number of collisions in the last table is bounded by $\mathbf{X}_t \leq \frac{(ct)^{2^t-1} \cdot n^{2^t}}{m^{2^t-1}}$. We need to set m such that $\frac{(ct)^{2^t-1} \cdot n^{2^t}}{m^{2^t-1}} < 1$, i.e., such that $m^{2^t-1} > (ct)^{2^t-1} \cdot n^{2^t}$. Thus, we set $\mathbf{m = ct \cdot n^{\frac{2^t}{2^t-1}} = ct \cdot n^{1+\frac{1}{2^t-1}}}$.

Observe that m grows linearly in t – however we never set $t \geq \log\log n$ since for $t = \log\log n$, $n^{1+\frac{1}{2^t-1}} = O(n)$, which is the best possible. So this linear in t term looses at most a $\log\log n$ factor.