Collaboration Statement: Andrew Koulogeorge and Eric Richardson

Problem 2: Probabilistic Version of #DNF.

Solution.
Let x^* be a random vector where each entry x_i^* of x^* is sampled with probability p_i. Its important to note that our sample space is over all bit vectors of length n where each bit vector has a non uniform probability of being sampled. That is, each vector has a probability mass associated with it which equals the product of the probability of each entry occurring. Also note then that $p^* = \mathbb{P}[\varphi(x^*) = 1]$ where $\varphi(x^*)$ is 1 if x^* satisfies at least one of the clauses in the DNF formula and 0 otherwise. Now, let us observe the following key insights:

- The probability that x^* satisfies a clause C_i in the DNF formula can be computed in linear time. We can loop over the variables in C_i and keep a running product; for each literal x_i, if its negated we multiply our running probability by $(1 - p_i)$ and otherwise by p_i. Thus, we can compute the probability that x^* satisfies $C_i \ \forall \ i$ in $O(mn)$ time where n is the number of variables and m is the number of clauses in our DNF. Let this probability for a particular clause C_i be denoted q_i
- Let Q_i be the event that x^* satisfies clause C_i. It then follows x^* satisfies our DNF formula when it satisfies at least one of the clauses in the DNF: $p^* = \mathbb{P}[\varphi(x^*) = 1] = \mathbb{P}\left[\bigcup_{i=1}^{m} Q_i\right]$. By union bound, we can compute an upper bound for p^*: $p^* \leq \sum_{i=1}^{m} \mathbb{P}[Q_i] = \sum_{i=1}^{m} q_i = q$.

First, its very important to note what probability space these probabilities are over: they are over the space of all vectors x sampled with the probability masses $p_1 \dots p_n$. Later in the problem we are going to see another probability symbol which is over different space. For ease, lets define the probability a vector x is sampled in this space as $w(x)$.

Second, recall that the union bound can be a gross overestimate of the probability of a union of events occurring because it double counts the probability of events which occur simultaneously. Thus, the more clauses that a given x^* satisfies, the more of an overestimate q will be to p and our algorithm will have to take this into account. From here, we can construct an algorithm for an unbiased estimate $\hat{p}$ for p^* which has low variance. The below algorithm follows very closely the sampling algorithm seen in class for estimating #DNF.

procedure SAMPLER p^*

Compute upper bound on $p^* \rightarrow q = \sum_{i=1}^{m} q_i$

Sample clause C_i with probability $\frac{q_i}{q}$

Sample y s.t is satisfies C_i and sample the remaining literals y_i from p_i.

Compute the number of clauses that y satisfies $\rightarrow$ $N(y)$

Compute estimator $\rightarrow$ $\hat{p} = \frac{q}{N(y)}$

return $\hat{p}$

First, I claim that $\hat{p}$ is an unbiased estimate for p^*. Let U be the set of bit vectors x of length n such that $\varphi(x) = 1$, let $Sampler(y)$ be the probability that our algorithm samples bit vector y and $\hat{p}(y)$ be the value of our estimator (which is a random variable and thus a map from our sample space containing bit vectors to $\mathbb{R}$) for a given sample y.

$$\mathbf{E}\left[\hat{p}\right] = \sum_{y \in U} \mathbb{P}\left[Sampler(y)\right] * \hat{p}(y)$$

Note that this expectation is over a new sample space; that is, the probability our algorithm samples a vector y is not the same as the probability that x^* is sampled. Indeed, our sampler never samples a vector which does not satisfy the DNF formula. Therefore, our algorithm induces a **new probability distribution** over the which vectors can be sampled. Now, lets take a closer look at for a fixed vector $y \in U$ what the probability our algorithm samples y. Let us apply the law of total probability and condition on the event of which clause C_i we sample. Conditioned on which clause we sample, we can then compute the probability that y is sampled. Note that we need only consider the clauses C_j which y is satisfies since we impose that constraint before we sample the remaining values for y. We denote the number of these clauses that y satisfies at $N(y)$

$$\mathbb{P}\left[Sampler(y)\right] = \sum_{C_j} \mathbb{P}\left[Sampler(C_j)\right] \mathbb{P}\left[Sampler(y)|C_j\right]$$

$$\mathbb{P}\left[Sampler(y)\right] = \sum_{C_j} \mathbb{P}\left[Sampler(y)|C_j\right] \frac{q_i}{q}$$

Now to roll up our sleeves: what is the probability we sample our fixed vector y given we have sampled a clause which it satisfies. Note that the randomness under consideration is only over the entries of y which are not present in C_j since our algorithm fixes those entries deterministically. Lets define the indices in y which were not set by our algorithm and whose variables are positive in y as I_{pos} and the indices which are negated in y as I_{pos}. Then, the probability that we sample y conditioned on C_j is equal to the product of these probabilities; we are computing the chance that each of these entries was set:

$$\prod_{i \in I_{pos}} p_i \prod_{j \in I_{neg}} 1 - p_j$$

Here comes the trick; observe what occurs to this above expression if we multiple both the numerator and denominator by the probability that y satisfies C_j:

$$\frac{\prod_{i \in I_{pos}} p_i \prod_{j \in I_{neg}} 1 - p_j * q_i}{q_i}$$

The numerator is exactly the probability of sampling y in the original space discussed above where a vector is drawn from the weights of $p_1 \dots p_n$:

$$\mathbb{P}\left[Sampler(y)\right] = \sum_{C_j} \frac{w(y)}{q_i} \frac{q_i}{q} = \frac{N(y)w(y)}{q}$$

$$\mathbf{E}\left[\hat{p}\right] = \sum_{y \in U} \frac{N(y)w(y)}{q} * \frac{q}{N(y)} = \sum_{y \in U} w(y) = p^*$$

Second, I claim that $\hat{p}$ has low variance:

$$\mathbf{Var}\left[\hat{p}\right] \leq \mathbf{E}\left[\hat{p}^2\right] \leq M\mathbf{E}\left[\hat{p}\right]$$

where M is the max value which the random variable $\hat{p}$ can take on. Note that $q_i \leq p*$ since p^* is the probability that any of the clauses are satisfied which is at least as large as the probability that exactly 1 is satisfied. Since $N(y) \geq 1$, we have that:

$$\hat{p} \leq q = \sum_{i=1}^{m} q_i \leq mp^* \implies M \leq mp^*$$

$$\mathbf{Var}\left[\hat{p}\right] \leq mp^{*2}$$

From here, since we have obtained an unbiased estimator with an estimator quality that is good ($\frac{\mathbf{Var}\left[\hat{p}\right]}{\mathbf{E}\left[\hat{p}\right]} \leq m$) we can apply the boosting theorem and obtain an (ε, δ) esimtator by sampling $\hat{p}$ $k = O(\frac{mlog(2/\delta)}{\varepsilon^2})$ times and taking the median. □

Problem 4: Implementing Karger's RANDMC Algorithm.

Solution.

(a) We assume access to the following two data structures:

  (a) $deg(v)$: a dictionary which counts the number of edges incident on v

  (b) A_{uv}: an $n \times n$ matrix containing the number of edges between u and v

Consider the fact that G is a multigraph, i.e. there may be multiple edges between the same pair of vertices. Because of this, we cannot simply select two vertices at random and return an edge between them (if one exists). However, we do have access to the number of edges between any pair of vertices via the adjacency matrix A. As such, our sampling procedure should, for any edge $e = \{u, v\}$, return e with probability $\frac{A_{uv}}{|E|}$. For instance, if vertices x and y contain s edges between them, then the probability we sample edge $\{x, y\}$ should be $\frac{A_{xy}}{|E|}$, where $A_{xy} = s$. With this in mind, consider the following sampling procedure.

1: **procedure** SAMPLE(deg, A)
2: Compute $S \leftarrow \sum_{v \in V} deg(v)$
3: Sample vertex u with probability $\frac{deg(u)}{S}$
4: Sample a neighbor v of u with probability $\frac{A_{uv}}{deg(u)}$
5: **return** $\{u, v\}$

We wish to show that SAMPLE returns an arbitrary edge $\{u, v\}$ with probability $\frac{A_{uv}}{|E|}$. First, consider the quantity S computed in Line 2. Because $deg(v)$ gives us the degree of any vertex v, we can compute the sum of all degrees in linear time. It follows from the Handshake Lemma that

$$S = \sum_{v \in V} deg(v) = 2|E|$$

Hence, the probability we sample any vertex u in Line 3 is $\frac{deg(u)}{2|E|}$. Intuitively, this means we are more likely to sample vertices with higher degrees. Using the adjacency matrix A, we can determine all vertices adjacent to u (as well as the number of edges) in linear time. Using this information, we sample a neighbor v of u with probability $\frac{A_{uv}}{deg(u)}$. Now, consider what needs to happen in order for our procedure to sample some edge $\{u, v\}$. This occurs if we initially sample the vertex u, then sample v, or if we initially sample v, then sample u (since G is undirected). Let X_t be the event that vertex t is sampled in line 3, and Y_t the event that vertex t is sampled

in line 4. Then, we have

$$\begin{aligned}
\mathbf{Pr}[\text{SAMPLE returns } \{u,v\}] &= \mathbf{Pr}[X_u] \cdot \mathbf{Pr}[Y_v|X_u] + \mathbf{Pr}[X_v] \cdot \mathbf{Pr}[Y_u|X_v] \\
&= \frac{deg(u)}{S} \cdot \frac{A_{uv}}{deg(u)} + \frac{deg(v)}{S} \cdot \frac{A_{vu}}{deg(v)} && \text{(by lines 3 and 4 of SAMPLE)} \\
&= \frac{A_{uv}}{S} + \frac{A_{vu}}{S} \\
&= \frac{2A_{uv}}{S} && \text{(because } A_{uv} = A_{vu}\text{)} \\
&= \frac{2A_{uv}}{2|E|} && \text{(because } S = 2|E|\text{)} \\
&= \frac{A_{uv}}{|E|}
\end{aligned}$$

Therefore, SAMPLE returns a random edge $e = \{u, v\} \in E$ with probability $\frac{A_{uv}}{|E|}$. As discussed, lines 2-4 can each individually be computed in linear time. Thus, our algorithm samples a random edge in E in $O(n)$ time.

(b) Suppose we sampled some edge $e = \{x, y\}$ via the procedure above and contracted it in the RANDMC algorithm. Then, we must modify both the deg dictionary and the adjacency matrix A. Specifically, the contraction process removes x and y from G and combines them into a supervertex xy. Consider the following algorithm which performs these operations in place:

```
1: procedure UPDATE(deg, A, x, y)
2:     deg(x) ← deg(x) + deg(y) − 2A_xy
3:     deg(y) ← 0
4:     A_xy ← A_yx ← 0
5:     for u ∈ V do
6:         A_xu ← A_xu + A_yu
7:         A_ux ← A_xu
8:         A_yu ← A_uy ← 0
```

In this procedure, we modify the vertex x such that it has the properties of the supervertex xy. Specifically, the new degree of x is given by $deg(x) + deg(y) - 2A_{xy}$. This ensures that the degree of x accounts for all edges previously connected to x as well as all edges previously connected to y, except for those edges connecting x and y ($2A_{xy}$ accounts for the fact that the edges between x and y contribute to both $deg(x)$ and $deg(y)$). Then, by setting $deg(y) = 0$, we effectively remove it from further consideration in the sampling process since vertices are sampled with probability proportional to their degree. Next, we zero out A_{xy} and A_{yx}, which are the entries in the adjacency matrix associated with edges between x and y. Then, for each vertex $u \in V$, we update the number of edges between x and u to include the edges between y and u. Additionally, we update the corresponding entry A_{ux} to the new value A_{xu}, and zero out A_{uy} and A_{yu}. This process covers all necessary modifications to the data structures resulting from the contraction process, and requires iteration

through n vertices, resulting in a total time complexity of $O(n)$. Note that the size of the degree dictionary and adjacency matrix remain n and n^2, respectively, regardless of how many times edge contraction is performed. The "removal" of vertices is mimicked by zeroing out one of the vertices associated with the newly contracted edge in both data structures, which, when considered in conjunction with our sampling algorithm, means that only the updated vertex (with its new degree and edge counts) will be sampled in future iterations of RANDMC. Since both SAMPLE and UPDATE require $O(n)$ operations individually, and RANDMC samples and contracts until only two vertices remain, the total time complexity of RANDMC using these subroutines is $O(n^2)$.

Problem 5: Taking Care of False Positives.

Solution. 1: **procedure** SAMPLER(K, ε, δ)

2: Sample from given importance sampling algorithm $N = \frac{S \log 2n/\delta}{k\varepsilon^2}$ times and keep an hash map of frequencies for how many times we have seen each element.

3: Include an element in H if we sampled it at least $(1-\varepsilon)\frac{NK}{S}$ times

4: return H

Let us define two random variables X and Y where X equals the number of times we sample a fixed element $x \in U \; s.t \; score(x) \geq k$ in N samples and Y equals the number of times we sample a fixed element $y \in U \; s.t \; score(y) < (1-\varepsilon)k$. For shorthand, let us define $T = \frac{NK}{S}$. This gives us a threshold value of $T(1 - \frac{\varepsilon}{2})$. We wish to bound the probability that X is smaller than our threshold and the probability that Y is larger than the threshold. If X is smaller than the threshold, that means that there existed a heavy element which we didn't include in H (bad recall) and if Y is larger than the threshold that means there existed a light element which we did include (bad precision). If we can bound the probability for a single heavy element x not occurring enough and bound the probability for a single light element occurring too often, then we can apply union bound and bound the overall probability of failure.

Lets begin by computing the expected value for X and Y. First, observe that X and Y can be expressed as sum of independent Bernoulli Random variables where $X_i = 1$ if the ith sample drawn from the importance sampler is x and 0 otherwise. A symmetric defining is applied to Y. Now observe that since we are using a given importance sampler, the probability that x or y is sampled using A is $\frac{score(x)}{S}$ and $\frac{score(y)}{S}$ respectively. By applying the law of total expectation we can see that:

$$\mathbf{E}[X] = \sum_{i=1}^{\frac{TS}{K}} \frac{score(x)}{S} = \frac{TS}{K}\frac{score(x)}{S} = \frac{T * score(x)}{K} \geq T$$

$$\implies \mathbf{E}[X] \geq T$$

$$\mathbf{E}[Y] = \sum_{i=1}^{\frac{TS}{K}} \frac{score(y)}{S} = \frac{TS}{K}\frac{score(y)}{S} = \frac{T * score(y)}{K} < T(1-\varepsilon)$$

$$\implies \mathbf{E}[Y] < T(1-\varepsilon)$$

First, lets bound the probability of X being less than the threshold. Let $q = \frac{\varepsilon}{2} \in (0,1)$

$$\mathbb{P}[X \leq (1-q)\mathbf{E}[X]] \leq \exp O(-q^2\mathbf{E}[X])$$

Since $\mathbf{E}[X] \geq T$:

$$\mathbb{P}[X \leq (1-q)T] \leq \mathbb{P}[X \leq (1-q)\mathbf{E}[X]] \leq \exp O(-q^2\mathbf{E}[X])$$

Plugging in for $q = \frac{\varepsilon}{2}$:

$$\mathbb{P}\left[X \leq (1-\frac{\varepsilon}{2})T\right] \leq \exp O(-\varepsilon^2\mathbf{E}[X]) \leq \exp O(-\varepsilon^2 T)$$

Thus, we have bounded the probability that a given element $x \in U$ where $score(x) \geq k$ is sampled less than our threshold. We can now take the union bound over **any** element x failing in this way to obtain a bound on the probability of this mistake occurring for any element in U:

$$\mathbb{P}[\textit{Probability that any element falls below the threshold}] \leq n \exp O(-\varepsilon^2 T)$$

Note that this upper bound is pretty course since we are even considering elements that are not heavy, but it does the trick.

Now lets apply the same procedure for the false positive case Y. Let $w = \frac{\varepsilon}{2}$. Applying Chernoff on Y:

$$\mathbb{P}[Y \geq (1+w)\mathbf{E}[Y]] \leq \exp O(-w^2\mathbf{E}[X = Y])$$

Since $\mathbf{E}[Y] < T(1-\varepsilon)$:

$$\mathbb{P}[Y \geq (1-\varepsilon)(1+w)T] \leq \mathbb{P}[Y \geq (1+w)\mathbf{E}[Y]] \leq \exp O(-w^2\mathbf{E}[Y])$$

Now lets analyze how our chosen value for $w = \frac{\varepsilon}{2}$ compares to the threshold value of $(1-\frac{\varepsilon}{2})T$. When we plug in w we get that:

$$(1-\varepsilon)(1+w)T = (1-\varepsilon)(1+\frac{\varepsilon}{2})T = (1-\frac{\varepsilon}{2}-\frac{\varepsilon^2}{4})T < (1-\frac{\varepsilon}{2})T$$

Since $(1-\varepsilon)(1+w)T < (1-\frac{\varepsilon}{2})T$:

$$\mathbb{P}\left[Y \geq (1-\frac{\varepsilon}{2})T\right] \leq \mathbb{P}[Y \geq (1-\varepsilon)(1+w)T] \leq \exp O(-\varepsilon^2\mathbf{E}[Y]) \leq \exp O(-\varepsilon^2 T)$$

Lastly, note that we got rid of the expectation of Y in the exponent but we didn't use the same argument as the analysis with X. Recall from the fine print of DeepC's notes that the bounds from Chernoff still apply when $\mathbf{E}[Y]$ is upper bounded. Therefore, we substituted in $\mathbf{E}[Y] < T(1-\varepsilon)$. Now, we apply union bound like we did for the false negative case to bound the bad event of an example being a false negative across all examples in U

$$\mathbb{P}[\textit{Probability that any element falls above the threshold}] \leq n \exp O(-\varepsilon^2 T)$$

With this analysis we have shown that the probability that our algorithm fails, that is, the probability that there exists a heavy element $x \in U$ which is not in H **OR** that there exists a light element $y \in U$ which is in H is bounded above by the following expression. By including either of these cases as a failure, we are able to ensure both high precision and recall with high probability. We wish to solve for the number of samples N our algorithm must take from U in order to ensure that the probability of failure is at most δ so lets rewrite T in its form of $\frac{NK}{S}$:

$$\mathbb{P}[\textit{fail}] \leq 2n \exp O(-\varepsilon^2 \frac{NK}{S}) = \delta$$

$$\implies N > \frac{S}{K}\frac{1}{\varepsilon^2}\log(2n/\delta) = O(\frac{S}{K}\frac{1}{\varepsilon^2}\log(n/\delta)) \quad \square$$

Problem 7: Importance Sampler for Matrix Multiplication of Three Matrices.

Solution.
We follow a similar procedure to what was explored in the second approach to the solution for the previous problem. First, observe that

$$(ABC)_{ij} = \sum_{s=1}^{n}\sum_{r=1}^{n} A_{ir}B_{rs}C_{sj}$$

We define the score of row s in matrix C as

$$\text{score}_s^C = \sum_{j=1}^{n} C_{sj}$$

where score_s^C yields the sum of row s in C. Now, let score_r^B represent the sum of row r in B weighted with the corresponding score_s^C, i.e.

$$\text{score}_r^B = \sum_{s=1}^{n} B_{rs} \cdot \text{score}_s^C$$

Similarly, we let score_i^A be the sum of entries in row i weighted with the corresponding score_r^B (which itself is weighted with a corresponding score_s^C), i.e.

$$\begin{aligned}
\text{score}_i^A &= \sum_{r=1}^{n} A_{ir} \cdot \text{score}_r^B \\
&= \sum_{r=1}^{n} A_{ir} \sum_{s=1}^{n} B_{rs} \cdot \text{score}_s^C \\
&= \sum_{r=1}^{n} A_{ir} \sum_{s=1}^{n} B_{rs} \sum_{j=1}^{n} C_{sj} \\
&= \sum_{r=1}^{n} A_{ir} \sum_{j=1}^{n} \sum_{s=1}^{n} B_{rs} C_{sj} \\
&= \sum_{j=1}^{n} \sum_{s=1}^{n} \sum_{r=1}^{n} A_{ir} B_{rs} C_{sj}
\end{aligned}$$

We see that score_i^A is the sum of entries in row i of the matrix product ABC. Now, we define $\Gamma(A, B, C) = \sum_{i=1}^{n} \text{score}_i^A$, which is the total sum of all entries in ABC. With this setup, we can sample an (i, j) as follows:

1. Sample row i of A with probability $p(i) = \dfrac{\text{score}_i^A}{\Gamma(A, B, C)}$
2. Sample column r of A with probability $q_A(r|i) = \dfrac{A_{ir} \cdot \text{score}_r^B}{\text{score}_i^A}$
3. Sample column s of B with probability $q_B(s|r) = \dfrac{B_{rs} \cdot \text{score}_s^C}{\text{score}_r^B}$
4. Sample column j of C with probability $q_C(j|s) = \dfrac{C_{sj}}{\text{score}_s^C}$
5. Return (i, j)

Thus, the probability of sampling (i, j) is

$$\begin{aligned}
\textbf{Pr}[(i,j) \text{ is sampled}] &= p(i) \cdot \sum_{s=1}^{n} \sum_{r=1}^{n} q_A(r|i) \cdot q_B(s|r) \cdot q_C(j|s) \\
&= \frac{\text{score}_i^A}{\Gamma(A, B, C)} \cdot \sum_{s=1}^{n} \sum_{r=1}^{n} \frac{A_{ir} \text{score}_r^B}{\text{score}_i^A} \cdot \frac{B_{rs} \text{score}_s^C}{\text{score}_r^B} \cdot \frac{C_{sj}}{\text{score}_s^C} \\
&= \frac{1}{\Gamma(A, B, C)} \cdot \sum_{s=1}^{n} \sum_{r=1}^{n} A_{ir} B_{rs} C_{sj} \\
&= \frac{(ABC)_{ij}}{\Gamma(A, B, C)}
\end{aligned}$$

which is exactly the probability proportional to $(ABC)_{ij}$. Since each of the score vectors are computed in $nnz(A)$, $nnz(B)$, and $nnz(C)$ time respectively, we have that the total time complexity of sampling an (i, j) is $O(n + nnz(A) + nnz(B) + nnz(C))$.

Problem 8: Estimating the number of connected components using graph queries.

Solution. (a) Let $conn(v)$ be the number of nodes in the connected component of G which contains nodes v. Consider a given connected component C_k with n_k nodes.

$$\forall\, x \in C_k, conn(x) = \frac{1}{n_k} \implies \sum_{x\,\in\, C_k} \frac{1}{n_k} = 1$$

where the last equality follows from the fact that there is n_k nodes in connected component C_k. Since each vertex in G only belongs to a single connected component, we can break up the sum of $conn(x)$ across all nodes in G by which component they belong to. The following summation is really summing up the value of 1, C^* number of times: $\sum_{x\in G} \frac{1}{conn(x)} = C^*$

(b) Let $\hat{est} = \frac{1}{conn(x)}$ be our estimate for the statistic $N = \frac{C^*}{n}$ where we sample x from G uar. Note that n is the number of nodes in G and C^* is the number of connected components in G. Note that the randomness over our random variable $\hat{est}$ is only over the node we sample.

$$\mathbf{E}\left[\hat{est}\right] = \sum_{v\in G} \mathbb{P}\left[v\right] \hat{est}(v) = \frac{1}{n}\sum_{v\in G} \frac{1}{conn(v)} = \frac{C^*}{n}$$

$$\mathbf{E}\left[\hat{est}^2\right] = \frac{1}{n}\sum_{v\in G} \frac{1}{conn(v)^2} \leq \frac{1}{n}\sum_{v\in G} \frac{1}{conn(v)} = \frac{C^*}{n}$$

$$\mathbf{Var}\left[\hat{est}\right] = \mathbf{E}\left[\hat{est}^2\right] - \mathbf{E}\left[\hat{est}\right]^2 \leq \frac{C^*}{n} - \frac{C^{*2}}{n^2} = \frac{C^*}{n}(1 - \frac{C^*}{n})$$

where the inequality in the computation for $\mathbf{E}\left[\hat{est}\right]^2$ follows from the fact that $\frac{1}{conn(x)} \leq 1$. We can see from the following analysis that $\hat{est}$ is unbiased and has variance which is maximized when there are twice as many nodes as components in G. Since our estimate is unbiased we can apply the boosting theorem for additive estimates to obtain the following samples in order to get an (ε, δ) estimate:

$$K = O(\frac{C^*}{n}(1 - \frac{C^*}{n})\frac{1}{\varepsilon^2}\log(2/\delta)$$

(c) Observe that the assumption that we have access to $conn(v)$ for all $v \in G$ is not realistic because in the worst case where we have a graph with a single component, we would have to apply a traversal algorithm to compute $conn(v)$ which would visit

all nodes and edges in the graph taking $O(n+m)$ time where m is the total number of edges in the graph. Instead of computing the exact value $conn(v)$, let us define an approximation $conn^*(v) = min(conn(v), \frac{2}{\varepsilon})$. First, note that our approximate connectivity metric is always an underestimate ($\leq$) of the true connectivity of node v and thus $\frac{1}{conn^*(v)}$ is always an overestimate of $\frac{1}{conn(v)}$. We wish to show that we can compute this underestimate rather quickly. We do this by executing a partial traversal of the graph G with BFS:

```
1:  procedure BFS VARIANT(G, v, ε)
2:      init Set "seen"
3:      init Queue "queue"
4:      init count = 0
5:      push v onto queue; add v to seen
6:      while queue not empty and count < 2/ε do:
7:          popleft x from queue
8:          for y in neighbors(x) if y not in seen do:
9:              add y to seen
10:             count++
11:             if count ≥ 2/ε: break
12:             push y to queue
13:     return count
```

I claim that the above traversal algorithm returns $conn^*(v)$ in $O(\frac{1}{\varepsilon^2})$ time. Consider the following two cases. First, if the connectivity of v is less than $\frac{2}{\varepsilon}$ then we will visit each node before count gets larger than $\frac{2}{\varepsilon}$, our queue will become empty, and we will return $conn(v)$. Second, consider if $conn(v) > \frac{2}{\varepsilon}$. Note that for any edge that is seen during this traversal, both of its endpoints are already seen by the traversal. Thus, when the count variable reaches $\frac{2}{\varepsilon}$, we know that we have seen $\frac{2}{\varepsilon}$ nodes which implies we have traversed at most $O(\frac{1}{\varepsilon^2})$ edges since the number of edges in this sub graph is at most on the order of the number of nodes in the sub graph squared.

(d) We wish to prove the following identity:

$$\left|\sum_{v\in V}\frac{1}{conn^*(v)} - \sum_{v\in V}\frac{1}{conn(v)}\right| \leq \frac{n\varepsilon}{2}$$

Recall that $conn^*(v) = min(conn(v), \frac{2}{\varepsilon})$. Note that we are reasoning over the entire vertex set of G. For each vertex $v \in G$ there are 2 cases. If $conn(v) \leq \frac{2}{\varepsilon} \implies$

$conn^*(v) = conn(v)$. For each of the vertex's in G with this property, the corresponding term will cancel out. Thus, we need only consider the case where $conn(v) > \frac{2}{\varepsilon} \implies conn^*(v) = \frac{2}{\varepsilon} \implies \frac{1}{conn^*(v)} = \frac{\varepsilon}{2}$. WLOG, assume there are k elements $x \in G$ with this property where k is a natural number between 0 and n. Let K be the set of nodes with this property where $|K| = k$ and note that the left hand summation is always at least as large as the summation over the vanilla connectivity since $conn^*(v)$ is always an underestimate to $conn(v)$ and thus the inverse is always an overestimate. Thus, we can write the left hand side of the expression as:

$$|\sum_{v\in V}\frac{1}{conn^*(v)}-\sum_{v\in V}\frac{1}{conn(v)}| = \sum_{v\in V}(\frac{1}{conn^*(v)}-\frac{1}{conn(v)}) = \sum_{x\in K}\frac{\varepsilon}{2}-\frac{1}{conn(x)} \leq \sum_{x\in K}\frac{\varepsilon}{2} = \frac{k\varepsilon}{2} \leq \frac{n\varepsilon}{2}$$

where the last inequality follows from the fact that k could be as large as n. □

(e) Lastly, we wish to use our new metric $conn^*(v)$ to compute an approximate estimate to $\frac{C^*}{n}$. Let us define $\overline{est}$ to be an estimator for $\frac{C^*}{n}$ where we randomly sample $v \in G$ and return $\frac{1}{conn^*(v)}$ where $conn^*(v) = min(conn(v), \frac{2}{\varepsilon})$. Note that this new estimator is no longer unbiased; that is, its expectation is not equal to the statistic $\frac{C^*}{n}$ we are after:

$$\mathbf{E}\left[\overline{est}\right] = \frac{1}{n}\sum_{v\in G}\frac{1}{conn^*(v)}$$

Note from part (d) however that we showed this expectation is very close to the true parameter:

$$\sum_{v\in V}\frac{1}{conn^*(v)} - \sum_{v\in V}\frac{1}{conn(v)} \leq \frac{n\varepsilon}{2} \implies (\mathbf{E}\left[\overline{est}\right] - \frac{C^*}{n}) \leq \frac{\varepsilon}{2}$$

Where the first equality is from the result proved in part (d) and the logical implication divides both sides by n so that the summations can are placed in the form of the expectation of $\overline{est}$ and $\hat{est}$ respectively where $\hat{est}$ is unbiased yet $\overline{est}$ is not. In words, even though $\overline{est}$ is a biased estimator, its still very close and becomes a more accurate estimate as ε, which is defined from the estimate $\overline{est}$, gets smaller (and the threshold we stop the BFS search gets larger). From here, the idea is very clean. We will apply the boosting theorem on $\overline{est}$ in order to obtain an (ε, δ) estimate for $\mathbf{E}\left[\overline{est}\right]$ which is biased but is "close" to the estimate with high probability. Its easy to show that the variance of $\overline{est}$ is upper bounded by 1 since $\overline{est}$ takes on values between 1 and $\frac{1}{n} \implies$ the additive boosting theorem tells us we need $O(\frac{1}{t^2}\log(2/\delta))$ samples in order to obtain an (t, δ) estimator. Note that for each vertex we sample, we need to compute $conn^*(v)$ which we showed can be done in $O(\frac{1}{\varepsilon^2})$ time. Let $t = \frac{\varepsilon}{2}$. Then, we will need $O(\frac{1}{\varepsilon^4}\log(2/\delta))$ queries, where a part of the this expression

comes from the total number of nodes we need to query at random (which comes from the boosting of our biased estimator: $O(\frac{1}{\varepsilon^2}\log(2/\delta))$) and the other comes from the algorithm to compute $conn^*(v)$ which needs to be run for each node v and which we showed can be done in $O(\frac{1}{\varepsilon^2})$. Thus, with probability at least $1-\delta$ we will have:

$$\overline{est} - \mathbf{E}\left[\overline{est}\right] \leq \frac{\varepsilon}{2}$$

$$\mathbf{E}\left[\overline{est}\right] - \frac{C^*}{n} \leq \frac{\varepsilon}{2}$$

$$\implies \overline{est} - \frac{C^*}{n} \leq \varepsilon$$

Indeed, even without an unbiased estimator, we were able to get an epsilon close additive approximation for the true statistic with probability at least $1-\delta$. □

Problem 10: Sum of Squared Degrees.

Solution.

(a) Let x be an arbitrary vertex in V. We wish to determine the probability that our sampling procedure returns x. First, we sample an edge $(u,v) \in E$ uniformly at random. In particular, the probability we sample an edge (u,v) is simply $\frac{1}{m}$, where m is the number of edges. Since there are $deg(x)$ edges incident on x, the probability we sample any of these $deg(x)$ edges is $\frac{deg(x)}{m}$. Now, assuming that some edge containing x has been sampled, we have that x itself will be sampled from that edge with probability $\frac{1}{2}$. Thus, **Pr[**x is sampled] = $\frac{deg(x)}{2m}$. With this, we can calculate the expectation of Z:

$$\begin{aligned}
\mathbf{Exp}[Z] &= \sum_{v\in V} 2m \cdot deg(v) \cdot \mathbf{Pr}[v \text{ is sampled}] \\
&= 2m \cdot \sum_{v\in V} deg(v) \cdot \mathbf{Pr}[v \text{ is sampled}] \\
&= 2m \cdot \sum_{v\in V} deg(v) \cdot \frac{deg(v)}{2m} \\
&= 2m \cdot \frac{1}{2m} \sum_{v\in V} deg(v) \cdot deg(v) \\
&= \sum_{v\in V} deg^2(v) \\
&= ||deg||_2^2
\end{aligned}$$

(b) From the previous problem, we have that $\mathbf{Exp}[Z] = \sum_{v \in V} deg^2(v)$. Thus, $\mathbf{Exp}[Z]^2 = \left(\sum_{v \in V} deg^2(v) \right)^2$. We find $\mathbf{Exp}[Z^2]$ as follows:

$$\begin{aligned}
\mathbf{Exp}[Z^2] &= \sum_{v \in V} (2m \cdot deg(v))^2 \cdot \frac{deg(v)}{2m} \\
&= \sum_{v \in V} 4m^2 \cdot deg^2(v) \cdot \frac{deg(v)}{2m} \\
&= 2m \sum_{v \in V} deg^3(v)
\end{aligned}$$

We wish to show that $\frac{\mathbf{Exp}[Z^2]}{\mathbf{Exp}[Z]^2} = O(\sqrt{n})$. Hence, we have

$$\begin{aligned}
\frac{\mathbf{Exp}[Z^2]}{\mathbf{Exp}[Z]^2} &= \frac{2m \sum_{v \in V} deg^3(v)}{\left(\sum_{v \in V} deg^2(v)\right)^2} \\
&\leq \frac{2m \left(\sum_{v \in V} deg^{\frac{3}{(\frac{3}{2})}}(v)\right)^{\frac{3}{2}}}{\left(\sum_{v \in V} deg^2(v)\right)^2} \\
&= 2m \cdot \left(\sum_{v \in V} deg^2(v)\right)^{\frac{3}{2}} \cdot \left(\sum_{v \in V} deg^2(v)\right)^{-2} \\
&= 2m \cdot \left(\sum_{v \in V} deg^2(v)\right)^{-\frac{1}{2}} \\
&= \frac{2m}{\sqrt{\sum_{v \in V} deg^2(v)}} \\
&= \frac{\sum_{v \in V} deg(v)}{\sqrt{\sum_{v \in V} deg^2(v)}} \\
&\leq \frac{\sqrt{\sum_{v \in V} deg^2(v)} \sqrt{\sum_{v \in V} 1^2}}{\sqrt{\sum_{v \in V} deg^2(v)}} \\
&= \sqrt{\sum_{v \in V} 1} \\
&= \sqrt{n}
\end{aligned}$$

Therefore, $\frac{\mathbf{Exp}[Z^2]}{\mathbf{Exp}[Z]^2} = O(\sqrt{n})$.