(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't do last week's homework, choose a problem from it that looks challenging to you, and in a few sentences, explain the key ideas behind its solution in your own words.

Solution: My solution for 5. d. is as follows:

> **Solution:** For each f, construct an instance with n elements $e_1, \ldots, e_n$. Partition the subsets into n disjoint groups of f each, where each group covers only one distinct element. Each element appears in exactly f subsets. The algorithm picks all f subsets for every newly uncovered element, while an optimal cover picks exactly one subset per element. The algorithm picks a total of fn subsets, while the optimal cover picks a total of n. Thus, the algorithm's cover can be f times as large as an optimal one.

My solution is far more complex than what was necessary to complete this problem. I use n elements on fn subsets, where the correct solution clevely uses 1 element on f subsets. If my solution found this, it would be much more efficient and correct.

(6 pts) 2. **Uniform sampling with constant memory.**

You are given a sequence of items $(a_1, \ldots, a_n)$ arriving one by one. Suppose you can only store *one item* in memory at a time. Consider the following algorithm:

```
1: function UNIFORM(a_1, ..., a_n)
2:     for i = 1, ..., n do
3:         Overwrite the current memory with a_i with probability 1/i
4:         Otherwise, the memory stays unchanged (with probability 1 − 1/i)
5:     Output the item in the memory
```

Prove that UNIFORM selects a uniformly random sample from the sequence $(a_1, \ldots, a_n)$. In other words, show that each element a_i has an equal probability $1/n$ being in the memory after processing all items.

Hint: If A and B are independent events, then $\Pr[A \text{ and } B] = \Pr[A] \cdot \Pr[B]$.

Solution: Consider any element a_i. It is selected with probability $\frac{1}{i}$ when processed. For each j such that $i < j \leq n$, the probability that a_j does not overwrite the current memory is $1 - \frac{1}{j} = \frac{j-1}{j}$. Thus,

$$\Pr[a_i \text{ is output}] \;=\; \frac{1}{i} \prod_{j=i+1}^{n} \frac{j-1}{j} \;=\; \frac{1}{i} \cdot \frac{i}{n} \;=\; \frac{1}{n}.$$

3. **Recording record-breaking moments.**

The EECS 376 staff are competing in a Rubik's cube Speedrun to see who can solve the cube the fastest. Each of the n staff members attempts a run, and their times are recorded in sequence $(t_1, \ldots, t_n)$, where t_i is the time used by the i-th staff member.

The *Hall of Fame* has only *one seat*, reserved for the current fastest time. Every time a record is broken, i.e., $t_i = \min\{t_1, \ldots, t_i\}$, the seat changes hands to the new record-holder. Lambda, who is responsible for updating the Hall of Fame, starts to wonder how often they will have to make these updates.

For simplicity, you may assume all t_i's are *distinct* in this problem.

(5 pts) (a) Define an indicator random variable that is suitable for this problem. When defining an indicator random variable, be sure to:

1. clearly describe the event for which the variable equals 1 and 0 (you may write "0 otherwise" to be concise), and
2. derive the probability that the indicator random variable equals 1.

Solution: Define the indicator random variable I_i for $1 \leq i \leq n$ as follows:

$$I_i = \begin{cases} 1, & \text{if } t_i = \min\{t_1, t_2, \ldots, t_i\}, \\ 0, & \text{otherwise.} \end{cases}$$

Since all t_i's are distinct, each of the first i times is equally likely to be the minimum, so $\Pr[I_i = 1] = \frac{1}{i}$.

(3 pts) (b) Let X be the number of record-breaking moments. Show that

$$\mathbb{E}[X] = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$$

(The above number is known as the harmonic sum, which is approximately $\ln n$.)

Solution: Let $X = \sum_{i=1}^{n} I_i$ be the total number of record-breaking moments. By linearity of expectation,

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[I_i] = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

4. **Equi-satifiability.**

In this problem, we define a clause to be in *4-disjunctive-unique-form* (4DUF) if it has *exactly 4 literals* involving *4 distinct variables*, connected by disjunctions ($\vee$).

Given a truth assignment, we say that 4DUF clause is *equi-satisfied* if it contains *exactly two* true literals and *exactly two* false literals.

For example, for the clause $c = (\overline{w} \vee x \vee \overline{y} \vee z)$:

- The assignment $w = x = y = z =$ true equi-satisfies C because it contains exactly two literals (x, z) and exactly two false literals $(\overline{w}, \overline{z})$.
- The assignment $w = x = y =$ true, $z =$ false does not equi-satisfy c because it contains one true literal and three false literals.

Now, consider a a collection of m 4DUF clauses $C = \{c_1, \ldots, c_m\}$. While a variable may appear in multiple clauses, no clause contains repeated variables. Let Rand-Assign be the same randomized algorithm as in lecture: assign random truth values (true or false) to the formula's variables, uniformly and independently. That is, for each variable, assign it to be true or false each with 1/2 probability, independently of all the others.

(8 pts) (a) Show that Rand-Assign equi-satisfies a constant fraction $\rho \in [0, 1]$ of the clauses of C in expectation. Analyze and determine the exact value of ρ.

Solution: For a 4DUF clause, each literal is true with probability 1/2 independently. Thus, the probability that exactly two of the four literals are true is

$$\Pr[\text{equi-satisfied}] \;=\; \binom{4}{2}\left(\frac{1}{2}\right)^4 \;=\; \frac{6}{16} \;=\; \frac{3}{8}.$$

Thus, in expectation, $\rho m = \frac{3}{8}m$ clauses are equi-satisfied, so $\rho = \frac{3}{8}$.

(6 pts) (b) Using Markov's inequality, give a **lower bound** on the probability that Rand-Assign equi-satisfies at least $\rho/2$-fraction of the clauses.

Solution: Let X be the number of equi-satisfied clauses. Then $\mathbb{E}[X] = \frac{3}{8}m$. The probability of equi-satisfying at least $\frac{\rho}{2} = \frac{3}{16}$ fraction of clauses is $\Pr\left[X \geq \frac{3}{16}m\right]$. Equivalently,

$$X \geq \frac{3}{16}m \quad\Longleftrightarrow\quad m - X \leq m - \frac{3}{16}m = \frac{13}{16}m\,.$$

Since $\mathbb{E}[m - X] = m - \mathbb{E}[X] = \frac{5}{8}m$, by Markov's inequality,

$$\Pr\left[m - X \geq \frac{13}{16}m\right] \;\leq\; \frac{\mathbb{E}[m - X]}{\frac{13}{16}m} \;=\; \frac{(5/8)m}{(13/16)m} \;=\; \frac{10}{13}\,.$$

Therefore,

$$\Pr\left[X \geq \frac{3}{16}m\right] \;\geq\; 1 - \frac{10}{13} \;=\; \frac{3}{13}\,.$$

5. **Near median.**

In class we showed that the randomized Quicksort algorithm has *expected* running time $O(n \log n)$ on an n-element array. However, this does not tell us much about the *distribution* of the running time, i.e., how likely it is to have various specific running times.

Ideally, we would like to use a pivot that is "nearly" the median value of the array. Such an element partitions the array into two roughly equal-sized parts for recursive sorting, and this would allow us to guarantee that the overall running time is $O(n \log n)$. In this problem, we will develop a linear-time randomized algorithm for finding a near-median with high probability.

Let S be a set of n orderable elements, which are distinct without loss of generality. We say that $a \in S$ has *rank* $\text{rank}(a) = k$ if a is the kth smallest element in S. Given input S, the goal is to find a *near median* of S, which is an element having rank between $2n/5 + 1$ and $3n/5$ (inclusive). For simplicity, *we assume that n is divisible by 5* (because we can "pad" the set with up to four extra elements).

(6 pts) (a) Consider the simplest randomized algorithm: output a uniformly random element $a \in S$. Prove that this algorithm outputs a near median of S with probability at least $1/5$.

Solution: Let $n = 5k$. There are exactly k elements with rank between

$$\frac{2n}{5} + 1 = 2k + 1 \quad \text{and} \quad \frac{3n}{5} = 3k.$$

A uniformly random element is near median with probability

$$\frac{k}{5k} = \frac{1}{5}.$$

(6 pts) (b) Briefly describe an $O(n)$-time algorithm that, given S and an element $a \in S$, determines whether a is a near median.

Solution: Given S and an element $a \in S$, compute

$$r = |\{x \in S : x < a\}| + 1.$$

Then a is a near median if and only if

$$\frac{2n}{5} + 1 \leq r \leq \frac{3n}{5}.$$

This process takes $O(n)$ time.

(8 pts) (c) Give an $O(n)$-time randomized algorithm that, on input S, outputs a near median with probability at least 99%.

Solution: Repeat the following process 21 times:

1. Uniformly select an element $a \in S$.
2. Compute $r = |\{x \in S : x < a\}| + 1$.
3. If
$$\frac{2n}{5} + 1 \leq r \leq \frac{3n}{5},$$
output a and halt.

If no near median is found after 21 iterations, compute the median of S using a deterministic linear-time selection algorithm and output it.

The probability of not finding a near median in one iteration is at most $4/5$. Thus, the probability of not finding a near median in 21 iterations is at most

$$\left(\frac{4}{5}\right)^{21} \leq \frac{1}{100}.$$

The expected running time of the algorithm is $O(n)$ because each iteration takes $O(n)$ time, and the expected number of iterations is constant.

(15 pts) 6. **Random binary search trees.**

Recall that a binary search tree on distinct values is a binary tree with the following property. For any node, where x denotes its value, all the nodes in its left subtree have values less than x, and all nodes in its right subtree have values greater than x.

Consider the following random process that generates a binary search tree on the integers $\ell, \ldots, r$ for given $\ell \leq r$, as follows.

- Choose $x \in \{\ell, \ldots, r\}$ uniformly at random, and take x as the root. If $\ell = r$, output the tree consisting of just x.
- If $\ell < x$, for x's left subtree, recursively build a random binary search tree on $\ell, \ldots, x-1$.
- If $x < r$, for x's right subtree, recursively build a random binary search tree on $x+1, \ldots, r$.

For a random binary search tree on $1, \ldots, n$, prove that the expected depth of the node with any particular value is $O(\log n)$. In other words, for any particular value in a random binary search tree, searching for it takes $O(\log n)$ time.

Hint: Use ideas from the analysis of the Quicksort algorithm from class.

Solution: Fix a key $k \in \{1, \ldots, n\}$. Define the indicator variable

$$X_{k,j} = \begin{cases} 1 & \text{if key } j \text{ is an ancestor of } k, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the depth of k is

$$D(k) = \sum_{j \neq k} X_{k,j}.$$

For $j \neq k$, key j is an ancestor of k if and only if it is chosen as the root before any key in the set $\{\min\{j,k\}, \min\{j,k\}+1, \ldots, \max\{j,k\}\}$ is chosen. Hence,

$$\Pr[X_{k,j} = 1] = \frac{2}{|j-k|+1}.$$

Expectation gives

$$E[D(k)] = \sum_{j \neq k} \Pr[X_{k,j} = 1] \leq 2 \sum_{i=1}^{n} \frac{1}{i+1} = 2H_n = O(\log n).$$

7. **Randomized vertex cover.**

In this problem, we will consider randomized algorithms for approximating vertex-cover problems. Recall that the "double cover" algorithm *deterministically* gives a 2-approximation to the minimum vertex cover problem, but in this problem you will show that a randomized approach is more versatile.

(6 pts) (a) Consider a probability experiment with set S with k distinct items. In each round:

- With probability **at least** p (where $p \in (0, 1]$), one *new* item (not previously chosen) is selected.
- Otherwise, no item is selected on that round.
- The experiment ends when all k items are selected.

Let R be a random variable denoting the number of rounds until the experiment ends. Prove that

$$\mathbb{E}[R] \leq \frac{k}{p}$$

Hint: For $i = 1, \ldots, k$, let R_i be the random variable representing the number of rounds needed to select the i-th item *after* $i - 1$ items have already be chosen. You may also want to review geometric distribution.

Solution: For $i = 1, \ldots, k$, let R_i be the number of rounds needed to select a new item once $i - 1$ items have been chosen. Since a new item is chosen in each round with probability at least p, $\mathbb{E}[R_i] \leq \frac{1}{p}$. By linearity of expectation,

$$\mathbb{E}[R] = \sum_{i=1}^{k} \mathbb{E}[R_i] \leq \frac{k}{p}.$$

(12 pts) (b) Consider the following algorithm

```
1: function RandomSingleCover(G = (V, E))
2:     C ← ∅
3:     while there is some edge e = (u, v) that is not covered by C do
4:         Add exactly one of u or v to C, each with probability 1/2
5:     return C
```

Prove that RandomSingleCover outputs a 2-approximation, in expectation, for the minimum vertex cover problem. In other words, show that

$$\mathbb{E}[|C|] \leq 2 \cdot |C^*|$$

where $C^* = \{v_1, \ldots, v_k\}$ is some minimum vertex cover in the input graph.

Hint: In each iteration, if the algorithm processes edge $e = (u, v)$, at least one of u or v must be in C^* (briefly explain why in your solution). First, relate the algorithm's iterations to rounds in the probability experiment in Part a. Then, show that the algorithm halts *no later* than when the experiment ends.

Solution: In any iteration of RandomSingleCover processing an uncovered edge $e = (u, v)$, at least one endpoint belongs to C^*; thus, the probability of selecting a vertex in C^* is at least $1/2$. Modeling the iterations as rounds in a probability experiment with success probability at least $1/2$, Part (a) (with $p = \frac{1}{2}$) implies that the expected number of rounds is at most $2|C^*|$.

$$\mathbb{E}[|C|] \leq 2|C^*|.$$

(9 pts) (c) Now, consider the *minimum-weight* vertex cover problem: given a graph G equipped with *positive vertex weights*, to find a vertex cover of minimum total weight.

For each of the following algorithms, show that it can yield an *arbitrarily large* approximation ratio for this problem, by describing an explicit graph and showing how the (expected) weight of the algorithm's output cover can be very large relative to the optimal weight.

(i) The double-cover algorithm from lecture.

(ii) The RandomSingleCover algorithm from Part b.

(iii) The following greedy algorithm:

```
1: function Greedy(G = (V, E, w))
2:     S ← ∅
3:     while G has an edge do
4:         Remove all isolated vertices of G
5:         Add a lowest-weight vertex in G to S and remove all its incident edges
6:     return S
```

Solution:

(i) **Double-Cover Algorithm:**
Construct $G_1 = (\{u, v\}, \{(u, v)\})$ with $w(u) = 1$ and $w(v) = M$, where $M > 0$ is arbitrarily large. An optimal vertex cover is $\{u\}$ with weight 1, whereas the double-cover algorithm outputs $\{u, v\}$ with weight $1 + M$.

(ii) **RandomSingleCover:**
Using the same graph G_1 as in the Double-Cover Algorithm answer, the algorithm in each iteration processes the sole edge (u, v) and selects u with probability $1/2$ and v with probability $1/2$. The expected weight is $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot M = \frac{1+M}{2}$, which can be arbitrarily large as M increases.

(iii) **Greedy Algorithm:**
Construct G_2 as a star with center u and leaves $v_1, \ldots, v_n$, where $w(u) = 1 + \delta$ (for some $\delta > 0$) and $w(v_i) = 1$ for all i. An optimal vertex cover is $\{u\}$ with weight $1 + \delta$, while the greedy algorithm always selects a leaf (the lowest-weight vertex) in each iteration, eventually outputting $\{v_1, \ldots, v_n\}$ with total weight n. For arbitrarily large n, the approximation ratio is very large compared to the optimal weight $1 + \delta$.