EECS 376 Final Exam

Instructions:
This exam is closed book, closed notebook. You may not provide your own scratch paper. No electronic devices are allowed. You may use **three** 8.5 × 11**-inch study sheets** (both sides) **that you prepared yourself**. Make sure you are taking the exam at the time slot and the classroom you were assigned by the staff. ***Please print your UNIQNAME at the top of each page.***

Any deviation from these rules may constitute an honor code violation. In addition, the staff reserves the right **not** to grade an exam taken in violation of this policy.

The exam consists of 10 multiple-choice questions, 3 short-answer questions, and 2 longer-answer questions. For the multiple-choice questions, please fill the selected circle completely and clearly. For the short- and long-answer sections, please write your answers clearly in the spaces provided. If you run out of room or need to start over, you may use the blank pages at the end, but you **MUST** make that clear in the space provided for the answer. The exam has 12 pages printed on both sides, including this page and the blank pages at the end.

You must leave all pages stapled together in their original order.

Write out and sign the full honor pledge below.

Honor pledge:
I have neither given nor received aid on this exam, nor have I concealed any violations of the Honor Code.
I will not discuss the exam with anyone before exam grades are released.
I attest that I am taking the exam at the time slot and the classroom I was assigned by the staff.

Signature: ______________________________

Name: ______________________________

Uniqname: ____________________

Standard Languages

You may rely on the following definitions without repeating them.

- $L_{\text{ACC}} = \{(\langle M\rangle, x) : M \text{ is a Turing machine that accepts } x\}$
- $L_{\text{HALT}} = \{(\langle M\rangle, x) : M \text{ is a Turing machine that halts on } x\}$
- $L_{\varepsilon\text{-HALT}} = \{\langle M\rangle : M \text{ is a Turing machine that halts on } \varepsilon\}$
- $L_{\emptyset} = \{\langle M\rangle : M \text{ is a Turing machine for which } L(M) = \emptyset\}$
- $L_{\text{EQ}} = \{(\langle M_1\rangle, \langle M_2\rangle) : M_1, M_2 \text{ are Turing machines for which } L(M_1) = L(M_2)\}$
- $\text{SAT} = \{\phi : \phi \text{ is a sastisfiable Boolean formula}\}$
- $\text{3SAT} = \{\phi : \phi \text{ is a satisfiable 3CNF formula}\}$
- $\textsc{Clique} = \{(G, k) : G \text{ is an undirected graph with a clique of size } k\}$
- $\textsc{VertexCover} = \{(G, k) : G \text{ is an undirected graph with a vertex cover of size } k\}$
- $\textsc{IndependentSet} = \{(G, k) : G \text{ is an undirected graph with an independent set of size } k\}$
- $\textsc{HamiltonianCycle} = \{G : G \text{ is a graph with a Hamiltonian cycle}\}$
- $\textsc{HamiltonianPath} = \{(G, s, t) : G \text{ is a graph with a Hamiltonian path from } s \text{ to } t\}$

UNIQNAME (print): ______________________________

Multiple Choice – 36 points (+4 extra credit)

1. Select the option that **correctly describes the following statement**:

   Some NP-Complete language is decidable in polynomial time if and only if *every* NP-Complete language is decidable in polynomial time.

   ● **Known to be true**
   ○ Known to be false
   ○ Unknown whether true or false

   Solution: In one direction, if some NP-Complete language is decidable in polynomial time (i.e., in P), then P = NP, so every NP-Complete language (which is a member of NP) is in P, i.e., decidable in polynomial time.

   The other direction holds trivially.

2. Select the option that **correctly describes the following statement**:

   The language

   $$\text{LogSAT} = \{\phi : \phi \text{ is a satisfiable Boolean formula having } 10\log_2|\phi| \text{ variables }\}$$

   is in P.

   (Recall that $|\phi|$ denotes the size of ϕ, when represented as a bit string.)

   ● **Known to be true**
   ○ Known to be false
   ○ Unknown whether true or false

   Solution: LogSAT is decidable in time polynomial in the length $|\phi|$ of the input formula ϕ by brute force: try all Boolean assignments to the $10\log_2|\phi|$ variables, of which there are $2^{10\log_2|\phi|} = |\phi|^{10}$, and accept if any of them satisfy ϕ. Because we can test whether an assignment satisfies ϕ in linear time, and there are only polynomially many assignments to try, the whole procedure is polynomial time.

3. Suppose that X is a non-negative random variable for which $\mathbb{E}[X] = 1/2$. Select the **tightest upper bound that *necessarily* holds** for $\Pr[X \geq 5/7]$.

   ○ $e^{-(3/14)^2}$
   ● $7/10$
   ○ $5/7$
   ○ 1

   Solution: Because X is a non-negative random variable, we can apply Markov's inequality, which says that $\Pr[X \geq 5/7] \leq \mathbb{E}[X]/(5/7) = 7/10$.

   This is the tightest bound that necessarily holds, because it could be that $X = 5/7$ with probability $7/10$ and $X = 0$ with probability $3/10$, making $\mathbb{E}[X] = 5/7 \cdot 7/10 + 0 \cdot 3/10 = 5/10 = 1/2$, as needed.

4. Let $n = 33$ be a public RSA modulus and $e = 7$ be the public exponent. **Select the corresponding decryption exponent** d and the **decrypted message** m for the ciphertext $c = 2$.

   ○ $d = 3$, $m = 29$

   [✓ $d = 3$, $m = 8$]

   ○ $d = 19$, $m = 17$

   ○ $d = 19$, $m = 29$

Solution: The factorization of n is $n = p \cdot q = 3 \cdot 11$, so $(p-1)(q-1) = 2 \cdot 10 = 20$. The public and private moduli are related by $e \cdot d = 7d = 1 \pmod{20}$, so by simple trial-and-error (or Euclid's algorithm) we can see that $d = 3$. The message for $c = 2$ is $c^d = 2^3 = 8 \pmod{33}$.

5. In an alternate (more chaotic) universe, n EECS 376 students take the final exam, and each graded exam is returned to a student *chosen uniformly at random*—but these choices *may or may not be independent*. In particular, it might be the case that a student could receive zero, one, or more than one exam(s).

   Select the option that correctly describes the **expected number of students who receive their own exams**.

   ○ It is $1/n$ if the choices are independent; otherwise, there is not enough information to determine the answer

   ○ It is $1/n$ regardless of whether the choices are independent

   ○ It is 1 if the choices are independent; otherwise, there is not enough information to determine the answer

   [✓ **It is 1 regardless of whether the choices are independent**]

   ○ Not enough information has been given to determine the answer, regardless of whether the choices are independent

Solution: For $i = 1, \dots, n$, define the indicator random variable X_i as:

$$X_i = \begin{cases} 1 & \text{if student } i \text{ receives their own exam} \\ 0 & \text{otherwise} \end{cases}$$

Then $X = \sum_{i=1}^{n} X_i$ is the number of students who receive their own exams back. Observe that

$$\mathbb{E}[X_i] = \Pr[X_i = 1] = \frac{1}{n},$$

because student i's exam goes to a student chosen uniformly at random. By linearity of expectation, which also holds regardless of whether the X_i are independent, we have

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \frac{n}{n} = 1.$$

Note that the (in)dependence of the X_i has no bearing on any of this, because linearity of expectation holds even for dependent random variables.

6. Select the **one language-input pair** for which the Cook-Levin theorem gives a **reduction from the language** to SAT, and **outputs a satisfiable Boolean formula** given the input.

○ Language: L_{HALT}
Input: $\langle M, x\rangle$ where M is a Turing machine and $M(x)$ rejects

○ Language: SAT
Input: Boolean formula $\phi = (x \text{ AND NOT}(x)) \text{ OR } (y \text{ AND NOT}(y))$

● **Language: Clique**
Input: $(G, 203)$**, where** G **is a graph that has a clique of size** 376

○ Language: $\text{GCD} = \{(x, y, g) : g = \gcd(x, y)\}$
Input: $(203, 376, 3)$

Solution: Recall that the Cook-Levin reduction proves that SAT is NP-hard, by showing that for every language $L \in \mathsf{NP}$, there exists an (efficiently computable) function f for which $x \in L \iff f(x) \in \text{SAT}$.

Of the options provided, only SAT, Clique, and GCD are in NP. Of these, only the provided input for Clique is an element of the corresponding language. This because G has a clique of size 376, so it also has a clique of size 203: any 203 of the vertices from the clique also form a clique. (By contrast, the formula ϕ is not satisfiable, and $\gcd(203, 376) \neq 3$, because neither number is divisible by 3.)

7. Select the **one statement that is known to be true, assuming that** $\mathsf{P} \neq \mathsf{NP}$.

○ No algorithm correctly solves all SAT instances having up to 1 billion variables in less than 1 second (on a standard laptop).

○ No polynomial-time algorithm correctly solves $\geq 99.9\%$ of all VertexCover instances of any given size.

○ There is no polynomial-time algorithm for any language in NP.

● **There is no polynomial-time algorithm for any NP-hard language.**

Solution: The first statement is not known to be true. Even though there is no polynomial-time algorithm for SAT (if $\mathsf{P} \neq \mathsf{NP}$), there could still be a fast, practical algorithm for inputs of a *fixed* (possibly even "large") size. "Polynomial time" refers to how the running time *scales* as the input size *grows*; it says nothing about fixed-sized inputs.

The second statement is not known to be true. Even though there is no polynomial-time algorithm that solves *all* instances of VertexCover (if $\mathsf{P} \neq \mathsf{NP}$), there could still be one that correctly solves a *large fraction* of all instances.

The third statement is not true. Every language in P is also in NP, and every language in P has a polynomial-time algorithm by definition.

The fourth statement is known to be true. A polynomial-time algorithm for some NP-hard language implies a polynomial-time algorithm for *every* language in NP, i.e., $\mathsf{P} = \mathsf{NP}$, which contradicts the $\mathsf{P} \neq \mathsf{NP}$ hypothesis from the prompt.

8. Define the language $L_{203} = \{\langle M\rangle : M \text{ does } \textit{not} \text{ accept the string "203"}\}$, and consider the following Turing machine:

```
1: function R(⟨M⟩)
2:     Run M("203") and output the opposite of whatever it outputs.
```

Select the **one true statement** about this machine.

- ● **It does not recognize L_{203} because it does not accept every $\langle M \rangle \in L_{203}$**
- ○ It does not recognize L_{203} because it does not reject or loop on every $\langle M \rangle \notin L_{203}$
- ○ It recognizes L_{203}, but does not decide L_{203}
- ○ It decides L_{203}

Solution: The first statement is true. Consider a Turing machine M that loops on input "203", so $\langle M \rangle \in L_{203}$. Hence, $R(\langle M \rangle)$ also loops, because its simulation of $M(\text{"203"})$ never halts. So $R(\langle M \rangle)$ does not accept, in violation of what's required of a recognizer for L_{203}.

The second statement is false. While R does not recognize L_{203} (as proved above), the stated reason is wrong. If $\langle M \rangle \notin L_{203}$, then M *does* accept "203", so $R(\langle M \rangle)$ rejects.

The third statement is false because R does not recognize L_{203}, as proved above.

The fourth statement is false: because R does not recognize L_{203}, it also does not decide L_{203} (recall that deciding is a stricter condition than recognizing).

9. **Assuming that** $\mathsf{P} = \mathsf{NP}$, select the option that **correctly describes the following statement**:

   There is a polynomial-time algorithm that, given only the public information from a run of the Diffie-Hellman protocol, outputs the resulting shared key.

   - ● **Known to be true**
   - ○ Known to be false
   - ○ Unknown whether true or false

Solution: The statement is true. If $\mathsf{P} = \mathsf{NP}$, then (the search version of) any NP problem is solvable in polynomial time. In particular, this applies to the discrete logarithm problem, because a candidate discrete log solution can be verified efficiently. It follows that Diffie-Hellman can be broken in polynomial time, because the eavesdropper can just solve for Alice's (or Bob's) secret exponent and then compute the shared key just as Alice (or Bob) does.

10. **(Extra credit.)** What are the names of Prof. Chris's kittens?

    - ● **Itchy and Scratchy**
    - ● **Biggie and Smalls**
    - ● **The Killers**
    - ● **Any/all of the above**

Solution: The kittens were called all of these at one time or another this semester! (Their names still haven't been permanently decided.) Any answer, or no answer at all, received the extra credit.

UNIQNAME (print): ____________________

Short Answer – 24 points

1. Recall the *decision* version of the subset-sum problem:

$$\textsc{SubsetSum} = \big\{(A,t) : \exists\, S \subseteq A \text{ whose elements sum to } t \in \mathbb{N}\big\}.$$

In this problem, **suppose that there is an efficient algorithm D that decides SubsetSum**.

The following *incomplete* algorithm uses D to efficiently solve the subset-sum *search* problem: given input (A,t), it should output a subset $S \subseteq A$ whose elements sum to t, if such a subset exists.

```
function SubsetSumSearch($A = \{a_1, \ldots, a_n\}, t$)
    if TODO#1 then
        return "no such subset exists"
    $S \leftarrow \emptyset$, $t' \leftarrow t$
    for $i = n$ down to 1 (inclusive) do
        if $D(\{a_1, \ldots, a_{i-1}\}, \text{TODO\#2})$ rejects then
            $S \leftarrow$ TODO#3
            $t' \leftarrow$ TODO#4
    return $S$
```

State what TODO#1 through TODO#4 should be to make the algorithm correct and efficient. You do *not* need to justify these answers.

Solution:

We give the missing pieces of the algorithm, along with a proof of correctness (which was not required in the exam).

TODO#1 is: $D(A,t)$ rejects. This means that there is no subset of A that sums to t, so we report this and halt. Otherwise, there is such a subset, and we continue.

The algorithm ensures that the following invariants hold at the start of each loop iteration:

(a) some subset of $\{a_1, \ldots, a_i\}$ sums to t';

(b) t' plus the sum of the elements in S equals t;

(c) $S \subseteq \{a_{i+1}, \ldots, a_n\}$.

By the above and the initialization of S and t', these invariants hold at the start (for $i = n$).

TODO#2 is: t'. If D rejects here, then *no* subset of $\{a_1, \ldots, a_{i-1}\}$ sums to t'. By the first invariant, there *is* a subset of $\{a_1, \ldots, a_i\}$ that sums to t', so a_i must be a member of any such subset.

TODO#3 is: $S \cup \{a_i\}$. Following the above reasoning, we add a_i to S. The third invariant will be preserved because i will be decremented at the end of the loop iteration.

TODO#4 is: $t' - a_i$. Because i will be decremented next and we have just added a_i to S, to maintain the first and second invariants we subtract a_i from t'.

After the final loop iteration we have $i = 0$, hence $\{a_1, \ldots, a_i\} = \emptyset$, so $t' = 0$ by the first invariant. Hence, by the second invariant, the elements in S sum to t. Finally, by the third invariant, $S \subseteq A$.

2. After studying all week for the EECS 376 final exam, you have prepared 3 perfect pages of notes, which have 240, 560, and 480 symbols, respectively. You need to print your masterpiece, but the University's printers are malfunctioning! Specifically, *each symbol* in your notes is *independently* printed with probability 3/4, and not printed with probability 1/4.

   Thanks to your diligent studying, you will be able to understand a page as long as *more than half* of its symbols are printed. (The symbols do not need to be contiguous.)

   In the following, **give the tightest bounds you can** for the probabilities in question.

   (a) Use the Chernoff-Hoeffding bound (with brief justification) to give an **upper bound** for the probability that you **will *not* be able to understand your first page**, i.e., at most half of its symbols will be printed.
   Also **do the same for the second and third pages**.
   Simplify the expressions as much as possible, but you *do not* need to evaluate any exponentials.

   Solution: For $i = 1, \dots, 240$, let X_i be the indicator random variable that is 1 if the ith symbol of the first page is printed, and 0 otherwise. Their sum $X = \sum_{i=1}^{240} X_i$ is then the total number of symbols printed on the first page, and $\mathbb{E}[X_i] = \Pr[X_i = 1] = 3/4$. Applying the (lower tail) Chernoff-Hoeffding bound, an upper bound on the probability that you will *not* be able to understand the first page is then

   $$\Pr[X/240 \leq 1/2] = \Pr[X/240 \leq \mathbb{E}[X_i] - 1/4] \leq e^{-2(1/4)^2 240} = e^{-30}.$$

   For the second page we proceed essentially identically with Y being the number of printed symbols, getting the bound

   $$\Pr[Y/560 \leq 1/2] \leq e^{-2(1/4)^2 560} = e^{-70},$$

   and for the third page, with Z being the number of printed symbols, we get

   $$\Pr[Z/480 \leq 1/2] \leq e^{-2(1/4)^2 480} = e^{-60}.$$

   All three of these quantities are very small: less than one in ten trillion! So, it is extremely likely that you will be able to understand each page.

   (b) Give, with brief justification, a **lower bound** for the probability that **you *will* be able to understand *all* of your pages**. (Simplify as in the previous part.)

   Solution: By the previous part and the union bound, the probability that you will fail to understand *some* page is at most $e^{-30} + e^{-70} + e^{-60}$. Therefore, the probability of the complement event—that you will be able to understand *all three* pages—is at least

   $$1 - (e^{-30} + e^{-70} + e^{-60}).$$

   (This is very close to a certainty.)
   Alternatively, observe that the readabilities of the pages are independent, so the probability that you will be able to read *all* of the pages is the product of the probabilities that you will be able to read each page, which at least

   $$(1 - e^{-30})(1 - e^{-70})(1 - e^{-60}).$$

   This is $\approx 1 - (e^{-30} + e^{-70} + e^{-60})$ (the bound we obtained above), but is even tighter by a very tiny amount. (Either bound received full credit.)

UNIQNAME (print): ______________________________

3. Consider a load balancer that needs to assign several tasks to worker machines, without overloading any machine. Let $T_i \in [0, 4]$ be the amount of work the ith task requires. **No machine should get more than 4 units of work** in total. The goal is to **minimize the number of machines** to which tasks are assigned. Consider the following assignment algorithm:

```
function AssignTasks($T_1, \ldots, T_k$)
    for $i = 1$ to $k$ do
        if task $i$ can be assigned to some already-provisioned machine (without overloading it) then
            assign task $i$ to such a machine
        else
            provision a new machine and assign task $i$ to it
    return the assignment
```

(a) Suppose that $\textsc{AssignTasks}(T_1, \ldots, T_k)$ provisions M machines. Show that $\sum_{i=1}^{k} T_i > 2(M-1)$.
Hint: argue that at most one provisioned machine can end up being assigned ≤ 2 units of work.

Solution: We claim that at most one provisioned machine can end up having ≤ 2 units of work. Suppose for contradiction that some two machines did. Then the first one of them to be provisioned would be able to take on every task the other one was assigned, so the algorithm never would have provisioned that other one—a contradiction. (This fact can also be proved by induction over the list of tasks, using the same main observation.)

Since at least $M - 1$ of the machines have > 2 units of work assigned to them, the total amount of work $\sum_i T_i$ is greater than $2(M-1)$.

(b) Let OPT be the minimum number of machines needed. Show that $M < 2 \cdot \mathsf{OPT} + 1$ (and hence $M \leq 2 \cdot \mathsf{OPT}$), where again M is the number of machines provisioned by $\textsc{AssignTasks}(T_1, \ldots, T_k)$. (You may use the statement from the previous part even if you did not manage to prove it.)
Hint: lower bound OPT in terms of $\sum_{i=1}^{k} T_i$.

Solution: Note that $\mathsf{OPT} \geq \sum_i T_i/4$, or equivalently, $4 \cdot \mathsf{OPT} \geq \sum_i T_i$. This is because no machine can be assigned more than 4 units of work, so we need at least $\sum_i T_i/4$ machines.

By the previous part, $\sum_i T_i > 2(M-1)$, so combining these two inequalities yields $4 \cdot \mathsf{OPT} > 2(M-1)$. Simplifying, we get $M < 2 \cdot \mathsf{OPT} + 1$, as needed. This further implies that $M \leq 2 \cdot \mathsf{OPT}$, because M and OPT are integers.

4. (Note: This question was not on the original exam.)

   An undirected graph $G = (V, E)$ is *3-colorable* if it is possible to assign each vertex one of three colors named R,W,B so that for every edge $(u, v) \in E$, the vertices u, v are have *different* colors.

   For example, a triangle graph is 3-colorable because we can assign each vertex a different color, but a complete graph on four vertices is not 3-colorable because any assignment of colors will result in (at least) two vertices having the same color, so the edge between those two vertices will be invalid. It is known that the decision problem of determining whether a given graph is 3-colorable is NP-complete.

   Consider the following zero-knowledge proof that a given graph $G = (V, E)$ is 3-colorable.

   - The prover (Merlin) starts with some fixed 3-coloring of G, and *recolors* it by randomly 'shuffling' the colors R,W,B. (For example, for the shuffling R,W,B $\to$ B,W,R, it would recolor every R vertex as B, every W vertex as W, and every B vertex as R.) It then conceals each vertex with a removable opaque cover, and shows the graph with its concealed vertices to the verifier (Arthur).
   - The verifier checks that the covered graph is indeed G (has the same vertices and edges), and then 'challenges' the prover by choosing a uniformly random edge $(u, v) \in E$.
   - The prover uncovers just the vertices u, v, revealing their assigned colors.
   - If u and v have *different* (valid) colors, the verifier accepts; otherwise, it rejects.

   This basic iteration can be repeated multiple times (reshuffling the colors each time) to improve the soundness, but below we consider only a single iteration.

   (a) Suppose that G is *not* 3-colorable, but the prover tries to 'fool' the verifier into accepting. Give, with brief justification, the tightest possible upper bound on the probability (over the verifier's random choice of challenge) that the prover can succeed.

   Solution: Since G is not 3-colorable, any hidden coloring that the prover chooses must have at least one edge e where both endpoints have the *same* color (or at least one of them has an invalid color). Because the verifier chooses a challenge edge uniformly at random, it has probability $1/|E|$ of choosing that edge e, and in this case the prover cannot make the verifier accept. Since the prover's chance of failure is at least $1/|E|$, the probability that it can fool the verifier is at most $1 - 1/|E|$.

   (b) Give a brief justification (1–2 sentences) why the protocol is zero knowledge.

   Solution: When G is 3-colorable, the verifier sees just the colors of the two endpoints of a single (randomly chosen) edge in the prover's recolored graph. Because the graph was recolored using a random shuffling of the original valid coloring, the two endpoints are simply two *random distinct* colors, and this is something that the verifier can generate itself.

UNIQNAME (print): ____

Longer Answer – 40 points

1. Define the language

$$\textsc{SetPacking} = \left\{ (A, S_1, \ldots, S_n, k) : \begin{array}{l} \text{each } S_i \subseteq A, \text{ and some } k \text{ of them} \\ \text{are pairwise disjoint} \end{array} \right\}.$$

(Recall that sets are pairwise disjoint if the intersection of any two of them is the empty set.)

Show that SetPacking is NP-Hard by proving that IndependentSet $\leq_p$ SetPacking.

Hint: give a reduction that outputs suitable subsets of *edges*.

Solution: We define a polynomial-time mapping reduction f that takes as input an instance of IndependentSet, i.e., an undirected graph $G = (V, E)$ and an integer k. It lets $A = E$, and for each $v \in V$, it lets S_v be the subset of edges that are incident to v, i.e.:

$$S_v = \{(u, v) : (u, v) \in E\}.$$

The reduction's output $f(G, k)$ is the SetPacking instance $A = E$, all the S_v for $v \in V$, and k. The reduction is efficient by inspection: essentially, it just iterates over each vertex and lists its incident edges.

We claim that $(G, k) \in$ IndependentSet $\iff$ $f(G, k) \in$ SetPacking, i.e., there exists an independent set I of size k in G if and only if k of the sets S_v are pairwise disjoint.

The key observation is that by construction, $S_u \cap S_v \subseteq E$ is the set of all edges that are incident to *both* u and v. So $S_u \cap S_v$ is the nonempty set $\{(u, v)\}$ if $(u, v) \in E$, and is the empty set $\emptyset$ otherwise.

$\Longrightarrow$ By assumption, there is an independent set I of size k in G. We claim that the k subsets S_v for $v \in I$ are pairwise disjoint. Consider any distinct $u, v \in I$. Because I is an independent set in G, we have $(u, v) \notin E$, so $S_u \cap S_v = \emptyset$ by the key observation, as desired.

$\Longleftarrow$ By assumption, k of the sets S_v are pairwise disjoint. Let I denote the k corresponding vertices in G; we claim that this is an independent set. Since $S_u \cap S_v = \emptyset$ for all distinct $u, v \in I$, by the key observation we have $(u, v) \notin E$, so I is an independent set.

Since IndependentSet is NP-hard, the above polynomial-time mapping reduction shows that SetPacking is as well.

2. In this problem, we consider a simple randomized approximation algorithm for the maximum independent set problem. The input is an undirected graph G with n vertices, each of which has *at most d neighbors* (and no self-loops). The algorithm works as follows:

   - Choose a uniformly random permutation (ordering) of the vertices, and assign each vertex its position in the permutation, from $1, \ldots, n$.
   - Output the set of *peak* vertices, where a vertex is a peak if it is assigned a *larger* number than all its neighbors.

   For example, if G is a cycle consisting of vertices u, v, w (so $n = 3$ and $d = 2$), and the permutation is (v, w, u), then u, v, w are assigned $3, 1, 2$, respectively, and u is the only peak vertex.

   (a) Define suitable indicator random variables and show that their sum is equal to the number of peak vertices.

   Solution: For each $v \in V$ (the vertex set), let X_v be the indicator random variable that is 1 if v is a peak vertex, and 0 otherwise. The sum $X = \sum_{v \in V} X_v$ is therefore the number of peak vertices.

   (b) Prove that the expected number of peak vertices is at least $n/(d+1)$.
   Hint: You may use (without proof) the fact that for any subset S of vertices, by symmetry, each vertex in S is equally likely to be assigned the largest number among all the vertices in S.

   Solution: By linearity of expectation, $\mathbb{E}[X] = \sum_{v \in V} \mathbb{E}[X_v]$, where X and X_v are as defined in the previous part. Using the hint with the set S consisting of a vertex v and all its $k \leq d$ neighbors, we have

   $$\mathbb{E}[X_v] = \frac{1}{k+1} \geq \frac{1}{d+1}.$$

   Therefore, $\mathbb{E}[X] \geq n/(d+1)$.

   (c) Prove that the set of peak vertices is an independent set of G, and that its expected size is at least $1/(d+1)$ times the size of any (maximum) independent set of G.

   Solution: Since the vertices are assigned distinct numbers, no two adjacent vertices can both be peaks. (The vertex assigned the smaller number is precluded from being a peak by its larger neighbor.) So, no two peak vertices have an edge between them, i.e., the set of peak vertices is an independent set in the graph.

   By the previous part, the expected number of peak vertices is at least $n/(d+1)$. Since every independent set has size at most n (the total number of vertices in the graph), the expected size of the output is at least $1/(d+1)$ times the size of any independent set of G, as claimed.