0. [From the instructions on the first page of the answer booklet:]
   • here come dat boi!!!!

   Solution: o shit waddup!

   **Rubric:** Jeff wins $5 from Patrick if anyone actually writes this.

   Solution:
1. **Prove** that it is NP-hard to determine whether a given undirected graph has a triple-Hamiltonian circuit.

**Solution:** here is a polynomial-time reduction from the standard Hamiltonian cycle problem.

Given an arbitrary undirected graph $G$, we transform it into another undirected graph $H$ as follows. For each vertex $v$ in $G$, we add four vertices $v_m$, $v_p$, $v_n$, and $v_o$ and six edges $vv_m$, $vv_p$, $vv_n$, $vv_o$, $v_mv_p$, and $v_nv_o$, as illustrated below. This construction clearly requires only polynomial time.

![Graph Transformation](image)

First, suppose $G$ has a Hamiltonian cycle $u \rightarrow v \rightarrow w \rightarrow \cdots \rightarrow z \rightarrow u$. If we replace each vertex $v$ in this cycle with the walk

$$v \rightarrow v_m \rightarrow v_p \rightarrow v_m \rightarrow v_p \rightarrow v_m \rightarrow v_p \rightarrow z,$$

then the resulting closed walk in $H$ is a triple-Hamiltonian circuit.

On the other hand, suppose $H$ has a triple-Hamiltonian circuit. For each vertex $v$ in $G$, the three visits to $v$ subdivide the triple-Hamiltonian circuit into three subwalks:

- one containing all visits to $v_m$ and $v_p$ and nothing else
- one containing all visits to $v_n$ and $v_o$ and nothing else, and
- one containing all visits to other vertices in $H$.

If we contract the first and second subwalks, for every vertex $v$, the resulting closed walk is a Hamiltonian cycle in $G$.

We conclude that $G$ has a Hamiltonian cycle if and only if $H$ has a triple-Hamiltonian circuit. ■

**Rubric:** 10 points: Standard NP-hardness rubric.
2. Lafayette needs to choose a subset \( S \) of his ragtag volunteer army of \( m \) soldiers to complete as many of a set of \( n \) important tasks as possible. Exactly \( k \) soldiers are qualified for each task. Each task will be completed if and only if exactly one soldier in \( S \) is qualified for that task.

(a) Suppose Lafayette chooses each soldier independently with probability \( p \). What is the exact expected number of tasks that will be completed, in terms of \( n \), \( p \), and \( k \)?

**Solution:** The probability that Lafayette chooses exactly one of the \( k \) soldiers qualified for any particular task is \( kp(1 - p)^{k-1} \). Thus, linearity of expectation implies that the expected number of completed tasks is \( nkp(1 - p)^{k-1} \).

**Rubric:** 3 points.

(b) What value of \( p \) maximizes this expected value?

**Solution:** We set the derivative of the success probability \( kp(1 - p)^{k-1} \) to zero and solve for \( p \).

\[
\frac{\partial}{\partial p} kp(1 - p)^{k-1} = 0 \\
\iff k(1 - p)^{k-1} - kp(k - 1)(1 - p)^{k-2} = 0 \\
\iff (1 - p) - p(k - 1) = 0 \\
\iff 1 - pk = 0
\]

So the unique optimum occurs at \( p = \frac{1}{k} \). Since the success probability is zero when \( p = 0 \) or \( p = 1 \), this optimum is in fact a maximum.

**Rubric:** 3 points.

(c) Describe a randomized polynomial-time \( O(1) \)-approximation algorithm for Lafayette’s problem. What is the expected approximation ratio for your algorithm?

**Solution:** Lafayette chooses each soldier with probability \( \frac{1}{k} \). By the analysis in part (a), the expected number of tasks completed is

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

The optimal number of tasks completed is trivially at most \( n \). Thus, the expected approximation ratio for this algorithm is

\[
\left( \frac{k}{k - 1} \right)^{k-1} = \left( 1 + \frac{1}{k - 1} \right)^{k-1} \approx e.
\]

**Rubric:** 4 points.
3. Describe and analyze an algorithm to nest a given set of \( n \) boxes, with distinct dimensions strictly between 10cm and 20cm, so that the number of visible boxes is as small as possible.

**Solution:** We need to **assign** as many boxes as possible to a unique larger box. To that end, we define a bipartite graph \( G = (L \cup R, E) \), where \( L \) contains a vertex for each box, \( R \) contains a vertex for each box, and \( uv \in E \) if and only if box \( u \) can nest inside box \( v \). This graph has \( 2n \) vertices and \( O(n^2) \) edges, and we can construct it in \( O(n^2) \) time by brute force.

Now we compute a maximum matching in \( G \), using the algorithm described in the notes (which is based on Ford-Fulkerson), in \( O(VE) = O(n^3) \) time. Then for each edge \( uv \) in the matching, put box \( u \) inside box \( v \). A box is visible if and only if the corresponding node in \( L \) is not adjacent to an edge in the matching. Thus, minimizing the number of visible boxes is equivalent to maximizing the number of edges in the matching.

Overall, the algorithm runs in \( O(n^3) \) time. ■

**Rubric:** 10 points: standard graph-reduction rubric. No correctness proof required.
+2 extra credit for faster algorithms: \( O(n^{2.5}) \) using Hopcroft and Karp algorithm, \( O(n^{2.373}) \) using Mucha and Sankowski’s algorithm based on fast matrix multiplication, or \( O(E^{10/7} \text{polylog} n) \) using Madry’s algorithm based on electrical flows.
4. Describe an efficient algorithm that computes the smallest number of towns where the revolutionary army should set up ambush points, given Mulligan’s map as input.

Solution: This is the vertex-capacitated version of minimum cut. Given Mulligan’s graph $G = (V, E)$, we construct a new directed graph $H = (V', E')$ by splitting each vertex $v$ into two vertices connected by an edge $v_{in} \rightarrow v_{out}$ with capacity 1, and then replacing each undirected edge $uv$ with two directed edges $u_{out} \rightarrow v_{in}$ and $v_{out} \rightarrow u_{in}$, each with infinite capacity. The resulting graph has $2V$ vertices and $V' + 2E$ edges.

Let $s$ be the node in $G$ corresponding to Charleston, and let $t$ be the node in $G$ corresponding to Yorktown. We compute the minimum $(s_{out}, t_{in})$ cut in $H$ in $O(V'E') = O(VE)$ time, using Orlin’s algorithm, and then return the capacity of this cut. (Specifically, for each edge $v_{in} \rightarrow v_{out}$ that crosses the minimum cut, the Americans should set up an ambush point in town $v$.)

But in fact, we can improve the running time using Ford-Fulkerson instead of Orlin’s algorithm. The cost of the minimum $(s_{out}, t_{in})$ cut in $H$ is at most the number of edges with unit capacity, which is $V - 2$. Thus, Ford-Fulkerson finds this minimum cut in $O(V^2)$ time.

Rubric: 10 points: standard graph reduction rubric. No correctness proof required.
-1 for $O(VE)$ time instead of $O(V^2)$ time.
5. Assign a random priority to each vertex of a graph $G$, and let $S$ be the set of all vertices $v$ such that the priority of $v$ is larger than the priority of at least one neighbor of $v$.

(a) What is the probability that the set $S$ is a vertex cover of $G$? **Prove** your answer is correct.

**Solution:** Consider an arbitrary edge $uv$. If $u$ has higher priority than $v$, then $u \in S$. If $v$ has higher priority than $u$, then $v \in S$. In both cases, at least one endpoint of $uv$ lies in $S$. We conclude that $S$ is always a vertex cover of $G$. ■

**Rubric:** 2 points = 1 for “always” + 1 for proof.

(b) Suppose the input graph $G$ is a cycle of length $n$. What is the exact expected size of $S$?

**Solution:** Let $v$ be an arbitrary vertex in $G$, and let $u$ and $w$ be the neighbors of $v$. We have $v \in S$ if and only if $v$ does not have the lowest priority in the set of three vertices $\{u, v, w\}$. Because priorities are independent and uniform, it follows that $\Pr[v \in S] = 2/3$. Linearity of expectation now implies that $\mathbb{E}[|S|] = 2n/3$. ■

**Rubric:** 3 points. 2 points for “$O(n)$” or “$\Theta(n)$” with no leading constant. 1 point for $\Theta(n)$ but wrong leading constant (for example, “$n/2$”). No proof required.

(c) Suppose the input graph $G$ is a **star**. What is the exact probability that $S$ is the smallest vertex cover of $G$?

**Solution:** Let $z$ denote the unique vertex with degree $n-1$ in $G$. The smallest vertex cover of $G$ is the singleton set $\{z\}$. An arbitrary leaf lies in $S$ if and only if its priority is larger than the priority of $z$. Thus, $S = \{z\}$ if and only if $z$ has the largest priority of all $n$ vertices. We conclude that $\Pr[S = \{z\}] = 1/n$. ■

**Rubric:** 3 points. No proof required.

(d) Again, suppose $G$ is a star. Suppose we run the randomized algorithm $N$ times, generating a sequence of subsets $S_1, S_2, \ldots, S_N$. How large must $N$ be to guarantee with high probability that some $S_i$ is the minimum vertex cover of $G$?

**Solution:** For all $i$, part (c) implies $\Pr[S_i \neq \{z\}] = 1 - 1/n$. Thus, the probability that none of the sets $S_1, S_2, \ldots, S_N$ is equal to $\{z\}$ is exactly

$$\left(1 - \frac{1}{n}\right)^N$$

If we set $N = n \ln n$, then this probability becomes

$$\left(1 - \frac{1}{n}\right)^{n \ln n} < e^{-n} < 1/n.$$

**Rubric:** 2 points. “$O(n \log n)$” or “$\Theta(n \log n)$” are both fine. No proof required.
6. Burr computes two arrays \( \text{profit}[1..n] \) and \( \text{skip}[1..n] \) that describe a sequence of \( n \) cases, where for each index \( i \),

- \( \text{profit}[i] \) is the amount of money Burr would make by taking the \( i \)th case, and
- \( \text{skip}[i] \) is the number of consecutive cases Burr must skip if he accepts the \( i \)th case.

Design and analyze an algorithm that determines the maximum total profit Burr can secure.

Solution (dynamic programming): Let \( \text{MaxProfit}(i) \) be the maximum profit that Burr can secure from cases \( i \) through \( n \), assuming Burr can actually accept case \( i \). The top-level value we want is \( \text{MaxProfit}(1) \). This function obeys the following recurrence:

\[
\text{MaxProfit}(i) = \begin{cases} 
0 & \text{if } i > n \\
\max \left\{ \begin{array}{l} 
\text{MaxProfit}(i + 1) \\
\text{profit}[i] + \text{MaxProfit}(i + 1 + \text{skip}[i])
\end{array} \right. & \text{otherwise}
\end{cases}
\]

We can memoize this function into a one-dimensional array \( \text{MaxProfit}[1..n] \), which we can fill in decreasing order (right to left) in \( O(n) \) time. ■

\textbf{Rubric:} 10 points: Standard dynamic programming rubric. This is not the only correct solution.