Question 1 (special case of NP-complete problems).

Prove that the following special case of SAT remains NP-complete. An instance is given by a collection of clauses as in SAT, but each clause either has two literals, or has (at most) one complemented literal.

In contrast, verify that, if every clause has (at most) one complemented literal, the problem is in P.

That the problem remains in NP is easy to see: the certificate is the same (a truth assignment), the checking of the certificate the same, and in addition we verify that the clauses have the right form (two literals or at most one complemented literal).

To prove completeness, we reduce SAT (the general version) to our new problem. The basic idea is to provide a “substitute” variable for each variable in the SAT instance. If \( x \) is a variable in the SAT instance, we place it and the new variable \( X \) in the new instance and add two clauses, \( \{x, X\} \) and \( \{\bar{x}, \bar{X}\} \). In order to satisfy these two clauses, any satisfying truth assignment must assign to \( X \) the opposite of the truth value assigned to \( x \); in other words, we have \( X = \bar{x} \) and so we can replace complemented variables wherever we want by their substitute. Thus, if a SAT clause has one or two literals, we pass it unchanged; if it has at most one uncomplemented literal, ditto; but if it has at least three literals, of which at least two are complemented, then we replace all but one of the complemented literals by their substitutes. The result is a collection of clauses in the proper form and is logically equivalent to the collection of clauses from the SAT instance. The transformation clearly takes linear time, so our proof is complete.

To show that the case where all clauses have at most one complemented literal is in P, we give an algorithm. Assume first that all clauses have at least two literals (we will return to the 1-literal case later). Then every instance is satisfiable (so we can answer yes without any further work): merely set all variables to true. (In every clause, there is at least one non-complemented variable, which by itself satisfies the clause.) If there are 1-literal clauses, however, this statement no longer holds, but we can just add an initial phase as follows. While there remains a 1-literal clause, assign to the corresponding variable the appropriate truth value to make this clause true, then remove all clauses that contain the same literal (they are now satisfied) and remove the complemented version of that literal from all remaining clauses that contain it. If that last part causes a clause to become empty, stop and announce that the instance is not satisfiable. Otherwise iterate until every remaining clause has at least two literals, at which stage we can, as we just saw, answer yes.
**Question 2 (randomized algorithm).**

Consider the following algorithm to find the $k$th smallest element in a set $S$ of integers.

```
select(k, S):
    if $S = \{x\}$ then return $x$
    $x \leftarrow$ random element of $S$;
    $S_1 \leftarrow$ all elements of $S$ less than $x$
    $S_2 \leftarrow$ all elements of $S$ larger than $x$
    if $k \leq |S_1|$
        then call select($k, S_1$)
    else if $k > |S| - |S_2|$
        then call select($k - |S| + |S_2|, S_2$)
    else return $x$
```

Prove that the expected running time of this algorithm is linear, but that its worst-case running time is quadratic.

The worst-case running time occurs when, at each successive step, we eliminate only one element—because the $x$ picked at random happens to be the largest (or the smallest) of the remaining elements and the $k$ dictates that we go the other way. In such a case, we will need $n - 1$ successive iterations where iteration $i$ requires us to test all $n - i$ remaining elements against $x$, for a total cost of $\sum_{i=1}^{n-1} (n - i)$, which is quadratic in $n$. The expected running time is just the average over all possible cases at each step. At the first step, our set has $n$ elements and we may have to run a recursive call on $n - 1$, or $n - 2$, etc., down to possibly just one element; or we may be lucky and hit the right element immediately. Writing $\tilde{T}(n)$ for this expected running time, we thus have

$$\tilde{T}(n) = cn + \frac{1}{n} (1 + \sum_{i=1}^{n-1} \tilde{T}(i))$$

where the $cn$ term represents the cost of creating the subsets $S_1$ and $S_2$ and where we added 1 to the sum to account for the case of hitting the $k$th element immediately. To get rid of all these terms, we telescope the sum (subtract it from itself, shifted by one):

$$n \cdot \tilde{T}(n) - (n - 1) \cdot \tilde{T}(n - 1) = ncn - (n - 1)c(n - 1) + \tilde{T}(n - 1)$$

and, after simplifying:

$$\tilde{T}(n - 1) - \tilde{T}(n - 2) = c\frac{2n - 1}{n} < 2c$$

from which we can see that $\tilde{T}(n)$ is bounded by $2c(n - 1) + \tilde{T}(0)$, since the difference between consecutive values is less than $2c$. Thus the expected running time is linear in $n$.

**Question 3 (closure properties).**

Verify that ZPP (the class of decision problems solvable exactly in expected polynomial time or, equivalently, the class of decision problems solvable in polynomial time with zero error, but a bounded probability of not giving an answer) is closed under union and intersection.

The practice examination had almost exactly the same question: it posed the same question for RP instead of for ZPP. The same solution works with (very) minor modifications.
Suppose that $X$ and $Y$ are two sets in ZPP; let the corresponding zero-error randomized decision algorithms be $A_X$ and $A_Y$. Consider the set $X \cup Y$: to decide membership in it, we run both $A_X$ and $A_Y$; we return “yes” as soon as either one returns “yes” and return “no” if both return “no.” Intersection is symmetric: we return “no” as soon as either one returns “no” and return “yes” if both return “yes.” Clearly, answering as soon as one of the two routines answers gives us an expected running time no worse than that of either routine. It then remains to show that answering after both routines have answered still allows us to place a fixed bound on the probability of not finishing within polynomial time. Assume that $A_X$ exceeds some polynomial time bound $p(n)$ with probability at most $1 - \varepsilon_X$ and $A_Y$ exceeds the same bound with probability at most $1 - \varepsilon_Y$; then waiting for the slower of the two causes us to exceed the same bound with probability at most $1 - \varepsilon_X \varepsilon_Y$, which is just some larger value $1 - \varepsilon$. Thus we are done.

**Question 4 (no approximation).**
Show that, unless we have $P=NP$, the NP-hard problem defined below cannot have a polynomial-time approximation algorithm that returns a solution within a fixed ratio of the optimal.

An instance is given by a set $S$, a distance measure, $d: S \times S \rightarrow \mathbb{N}$, and a positive integer $k$; the problem is to find a partition of the set into $k$ subsets (clusters) such that the sum of the pairwise distances between elements in the same subset (taken over all subsets) is minimized.

(Hint: we know that $k$-coloring—deciding whether a graph is colorable using no more than $k$ colors—is NP-complete for any $k$ larger than 2. Show that a fixed-ratio approximation for the given problem would allow us to decide $k$-colorability.)

So suppose we have a fixed-ratio approximation algorithm for the clustering problem and let the ratio be $\alpha > 1$. Given a coloring problem (a graph $G = (V,E)$ and a bound on the number of colors, $k > 2$), we define a clustering problem as follows. The set $S$ will be $V$, the set of vertices of the graph. The number of clusters will be $k$, the number of colors. Finally, if $x$ and $y$ are vertices of the graph then we set the distance between $x$ and $y$ to $D = \lceil \alpha \rceil \cdot |V|^2$ if there is an edge $\{x,y\} \in E$, to $d = 1$ otherwise. This instance of clustering can obviously be produced in time polynomial in the size of the input graph.

Now, if the input graph has a $k$-coloring, then we can place all vertices of the same color in the same cluster, thus creating $k$ clusters. The pairwise distances within a cluster are all $d = 1$, because there cannot be an edge between two vertices of the same color in a legal coloring. Thus the cost of a clustering is simply the number of pairs of elements assigned to the same cluster, which cannot exceed $C = (|V| - k + 1)(|V| - k)/2$ (if all clusters have one element each, except for the last cluster, which has all remaining elements). On the other hand, if the graph is not $k$-colorable, then there is no way to assign elements to clusters so as to avoid assigning a large pairwise distance of $D$ to two elements, because at least one pair of vertices connected by an edge will be identically colored in any $k$-coloring. In that case, the cost of a clustering is at least $D$. Now, if we run our assumed fixed-ratio approximation algorithm and the graph had a $k$-coloring, the result will be a clustering of cost at most $\alpha C$; on the other hand, if the graph was not $k$-colorable, then any clustering costs at least $D$, which is larger, by construction, than $\alpha C$. Hence we could decide $k$-colorability in polynomial time by running our assumed fixed-ratio approximation algorithm; since $k$-colorability is NP-complete, this would imply $P=NP$. 