

- 3(b) **Claim:** Any algorithm that solves the two-consecutive-zeros problem must perform, for some worst case input string, at least $n - 1$ operations if $n \equiv 1 \pmod{3}$ and at least n operations otherwise.

Proof. If $n = 0, 1$, the claim is trivially true. If $n = 2$, the adversary answers '0' to the algorithm's first query and thus forces any correct algorithm to query the other bit.

Assume that the claim is true for strings of length at most $n - 1$. Given a string of length $n > 2$, let i be the position of the algorithm's first query. If both $i - 1 \not\equiv 1 \pmod{3}$ and $n - i \not\equiv 1 \pmod{3}$ then the adversary answers '1' for position i , and considers positions $1 \dots i - 1$ and $i + 1 \dots n$ as two separate subproblems. By induction, any correct algorithm must query every position in both subproblems in the worst case, since $i - 1 \not\equiv 1 \pmod{3}$ and $n - i \not\equiv 1 \pmod{3}$. (Here, we assume that the adversary never reveals two consecutive zeros in either subproblem.) Thus the claim holds in this case.

If either $i - 1 \equiv 1 \pmod{3}$ or $n - i \equiv 1 \pmod{3}$ then the adversary answers '0' for position i and sets positions $i - 1$ and $i + 1$ to '1'. This means that if, in the future, the algorithm queries position $i - 1$ or $i + 1$ the adversary answers '1'. Notice that the algorithm must query both position $i - 1$ and $i + 1$ (if they are in the range $1 \dots n$) at some point or it cannot be correct. The adversary treats positions $1 \dots i - 2$ and $i + 2 \dots n$ as two separate subproblems. If $i = 1$ or $i = n$ then there is only one subproblem, its size is equivalent to $0 \pmod{3}$, and by induction the claim holds. Otherwise, either $i - 2$ or $n - i - 1$ is equivalent to $0 \pmod{3}$ and thus the other is equivalent to $(i - 2) + (n - i - 1) = n - 3 \equiv n \pmod{3}$. If $n \not\equiv 1 \pmod{3}$ then the sizes of both subproblems are not equivalent to $1 \pmod{3}$. If $n \equiv 1 \pmod{3}$ then only one subproblem has size equivalent to $1 \pmod{3}$. In either case, the claim holds. \square

- 4(a) We start by comparing x with the last element in the first row of M . If $x < M[1, n]$ then x cannot be in $M[1 \dots n, n]$, i.e., x cannot be in the last column of M . If $x > M[1, n]$ then x cannot be in $M[1, 1 \dots n]$, i.e., x cannot be in the first row of M . No matter how the comparison turns out, we eliminate a row or column from M . We repeat this process (always comparing x to the last element in the first row of what remains of M) until we find x or all the elements of M have been eliminated. There are a total of $2n$ rows and columns in M , however the last comparison eliminates both a row and a column, so the total number of comparisons is at most $2n - 1$.

- 4(b) The adversary answers any algorithm's queries using the following matrix M :

$$M[i, j] = \begin{cases} -\infty & \text{if } i + j \leq n \\ +\infty & \text{otherwise.} \end{cases}$$

If the algorithm fails to compare x to any entry $M[i, j]$ with $i + j = n$ or $i + j = n + 1$ (a total of $n - 1 + n = 2n - 1$ entries), then that entry could contain x without violating the ordering property of M . Thus the algorithm must make at least $2n - 1$ comparisons.

- 5(a) Find the minimum of the first $n - k + 1$ elements in the array. This takes $n - k$ comparisons.

5(b) Any algorithm to solve the problem must perform at least $n - k$ comparisons. Consider the graph $G = (V, E)$ where $V = \{1, 2, \dots, n\}$ and $E = \{(i, j) \mid \text{the algorithm compares } A[i] \text{ and } A[j]\}$. If an algorithm performs less than $n - k$ comparisons then the number of connected components in the graph is greater than k . If the algorithm outputs $A[i]$, we can decrease the value of $A[j]$ for all j not in the same connected component as i by some large constant without changing the outcome of any comparison. Since there are at least k components that don't contain i , we can produce an input in which $A[i]$ is not one of the k smallest elements, and yet the algorithm (based on the unchanged outcomes of the comparisons) will output $A[i]$.