(a) The algorithm goes like this:

- Organize all processes in a sequence S by non-decreasing order of their finish times
- While some process in S is still not covered: Insert a status_check right at the finish time of the first uncovered process in S

First, the algorithm won't stop until all the processes are covered; and it terminates since in every iteration at least one uncovered process gets covered. So it does compute a valid set of status_checks that covers all sensitive processes.

Second, we will show that the set of status_checks computed by the algorithm has the minimum possible size, via a "staying ahead" type of argument. Let us call the set of status_checks computed by the above algorithm as C. Take any other set C' that also covers all processes, and we will show by induction that in the interval up to the k^{th} status_check by C, there will also be at least k status_checks by C'.

- Base case: C' has to put a status_check no later than the first status_check in C, since a sensitive process ends at that time.
- Inductive step: Suppose there are at least k status_checks in C' up to the time of the k^{th} status_check in C. We already know that the process causing the insertion of the $(k+1)$-th status_check in C is not covered by all the k status_checks ahead, according to our algorithm, so C' has to put a status_check between the k^{th} and the $(k+1)^{\text{st}}$ status_checks in C to cover that process. Therefore C' has at least $k+1$ status_checks up to the time of the $(k+1)^{\text{st}}$ status_check in C.

So we know that the algorithm is correct.

The running time is $O(n \log n)$ to sort the start and finish times of all the processes, and then $O(n)$ to insert all the status_checks. To do this in linear time, we keep an array that records which processes have been covered so far, and a queue of all processes whose start times we've seen since the last status_check. When we first see a finish time of some uncovered process, we insert a status_check and mark all processes currently in the queue as covered. In this way, we do constant work per process over the course of this part of the algorithm.

(b) The claim is true.

Let us use C to denote the solution provided by our algorithm. The question can be rephrased as asking whether $|C| = k^*$. The question already argues that $k^* \leq |C|$, so what we need to do is to prove $|C| \leq k^*$. As k^* is the *largest* size of a set of sensitive processes with no two ever running at the same time, it is enough to find $|C|$ such "disjoint" processes to show $|C| \leq k^*$. Our algorithm from part (a) actually provides such a set of disjoint processes: the processes whose finish times cause the insertions of the status_checks are disjoint, since each is not covered by all the previous status_checks.

[1] ex559.176.225