C-5.2. We are given \( n \) tasks where task \( i \) starts at time \( s_i \) and ends at time \( f_i \). The goal is to find a subset of these tasks so that (i) the tasks are non-conflicting and (ii) the size of the subset is as large as possible.

In the greedy algorithm below, \( S \) denotes the subset of tasks that we are constructing and \( \text{end} \) denotes the latest finish time of a task in \( S \).

Sort the \( n \) tasks according to their finish times. Assume \( f_1 \leq f_2 \leq \cdots \leq f_n \).

Set \( S \leftarrow \{1\} \), \( \text{end} \leftarrow f_1 \).

for \( i = 2 \) to \( n \)

if \( s_i \geq \text{end} \)

add \( i \) to \( S \)

\( \text{end} \leftarrow f_i \)

endif

endfor

return(\( S \))

Running time. The first step takes \( O(n \log n) \) time. The for loop takes \( O(n) \) time. Hence, the entire algorithm runs in \( O(n \log n) \) time.

Correctness. This is a “transformation” argument. Let us now argue that the algorithm is correct. Suppose \( S \) consists of tasks \( i_1, i_2, \ldots, i_k \) sorted according to their finish times. Let \( S^* \) be an optimal solution to the problem, and suppose it consists of tasks \( j_1, j_2, \ldots, j_r \) also sorted according to their finish times. Clearly, \( r \geq k \). We shall now construct a series of optimal solutions \( S^*_1, S^*_2, \ldots, S^*_k \) from \( S^* \) so that the optimal solutions will look more and more like \( S \).

Consider the first task \( j_1 \) in \( S^* \). If \( j_1 = i_1 \) then let \( S^*_1 = S^* \). Otherwise, task \( j_1 \) will have a finish time that is equal to or later than that of task \( i_1 \). We know this because \( i_1 \) has the earliest finish times among all the \( n \) tasks. This means that \( i_1 \) will not be in conflict with the other tasks in \( S^* \). Let \( S^*_1 = S^* - \{j_1\} \cup \{i_1\} \). Notice that \( S^*_1 \) is an optimal solution since it is made up of non-conflicting tasks and its size is the same as that of \( S^* \).

Now, consider task \( j_2 \) in \( S^*_1 \). If \( j_2 = i_2 \) then let \( S^*_2 = S^*_1 \). Otherwise, task \( j_2 \) must have a finish time that is equal to or later than that of task \( i_2 \). We know this because both \( j_2 \) and \( i_2 \) do not conflict with \( i_1 \) but the greedy algorithm chose \( i_2 \), and not \( j_2 \), to add to \( S \). Hence, \( i_2 \) will again not be in conflict with the other tasks in \( S^*_1 \). Let \( S^*_2 = S^*_1 - \{j_2\} \cup \{i_2\} \). Once more, \( S^*_2 \) is an optimal solution. By repeatedly apply this kind of an argument, we can replace \( j_3, \ldots, j_k \) by \( i_3, \ldots, i_k \) respectively so that \( S^*_k = \{i_1, i_2, \ldots, i_k, j_{k+1}, \ldots, j_r\} \) is an optimal solution.

But we note that if \( S^*_k \) has more than \( k \) tasks in it then task \( j_{k+1} \) should have been added to \( S \) by the greedy algorithm. Since the greedy algorithm stopped at task \( i_k \), it
must be the case that all tasks after $i_k$ in the ordering were in conflict with it. Hence, task $j_{k+1}$ does not exist and $S_k^* = S$. That is, $S$ is an optimal solution.

C-5.5. Let the position of $n$ paintings in a long hallway be $x_0, x_1, \ldots, x_{n-1}$. Assume these positions are sorted from left to right. Each guard can protect a painting within 1 unit of distance from his or her position. Our goal is to place guards along this hallway so that (i) all paintings are protected and (ii) the number of guards is as few as possible.

In the greedy algorithm below, $m$ will keep track of the number of guards we are using while $y_i$ will denote the position of guard $i$.

Set $m \leftarrow 1$, $y_1 \leftarrow x_0 + 1$. 
for $i = 1$ to $n - 1$

if $|y_m - x_i| > 1$ /* Guard $m$ cannot protect the painting at $x_i$. */

$\quad m \leftarrow m + 1$ /* Add a new guard. */

$\quad y_m \leftarrow x_i + 1$ /* one unit to the right of the painting at $x_i$. */

endfor

return($m, y_1, \ldots, y_m$)

Running time. If the painting locations are already sorted, the algorithm runs in $O(n)$ time. Otherwise, sort the locations first and run the algorithm above. This will take $O(n \log n)$ time.

Correctness. This is a lower bound argument. By construction, each of the guards at $y_1, \ldots, y_m$ are uniquely protecting some painting. That is, each one is protecting a painting that no other guard can protect. They are the paintings at $y_1 - 1, y_2 - 1, \ldots, y_m - 1$. Since we are always placing the guards one unit to the right of the paintings they are uniquely protecting, the painting at $y_i - 1$ and the painting at $y_{i+1} - 1$ are more than 2 units apart for $i = 1, 2, \ldots, m - 1$. Hence, any placement of guards that protects all paintings must use at least $m$ guards since a guard can protect at most one of those $m$ paintings only. Since the greedy algorithm uses exactly $m$ guards, it must be optimal.