Exercise 8.1  

Branch and bound.

a) Each partial solution defines a set $I$ of nodes that should be included in the dominating set. We define $\delta_v$ as the number of new nodes that are dominated if we add $v$ to the set $I$ (i.e., $\delta_v$ might count $v$ itself if it is not dominated yet). Of course, $\delta_v = 0$ for every $v \in I$, since these nodes are already in the dominating set. Let also $\delta_{\text{max}} = \max_{v \in V - O} \delta_v$ be the maximum number of nodes that we can dominate with a single new node, and $\bar{D}$ be the set of all nodes that are not dominated yet. To dominate all the nodes in $\bar{D}$, we need at least $|\bar{D}|/\delta_{\text{max}}$ additional nodes. We can therefore use $|I| + |\bar{D}|/\delta_{\text{max}}$ as the lower bound on the number of nodes in the dominating set, if we start from the current partial solution.

We can improve this bound by setting it to $\infty$ if a non-dominated node cannot be dominated by any node in $V - O$. In this case our partial solution is hopeless, and we should not pursue it further!

b) In each step we take a node $v \in V - (I \cup O)$ that dominates the maximum possible number of nodes, i.e. $\delta_v = \delta_{\text{max}}$. We hope that this will lead us to the target as quickly as possible.

c) The decision tree is the following. The sequence of steps is indicated by the numbers in the boxes. We perform exactly 8 branches before an optimal solution is found. Our solution is $I = \{d, f, g\}$. We can stop as soon as we find this solution, since all the remaining lower bounds are greater than 2, and there cannot be a solution with less than 3 nodes.
Alle lower bounds für die noch nicht expandierten Blätter sind auch mehr als 2 ⇒ diese können alle abgeschnitten werden. ⇒ Diese Lösung ist ein Optimum.
Exercise 8.2  Algorithm design: hotel booking (part of an exam in August 2009).

a) As data structure, we can simply use an array in which the \( n \) bookings are sorted according to arrival times. To build the array, we can, for example, use the merge sort. This needs \( O(n \log n) \) steps. To check whether a given booking \((x', y')\) is available, we search the array using binary search for the two entries \((x_1, y_1), (x_2, y_2)\) with arrival times \( x_1 \leq x' \leq x_2 \) directly before and after \( x' \). Since it is not possible to have another booking that overlaps with \((x_1, y_1)\) and \((x_2, y_2)\), it is sufficient to check whether the desired query fits between these two. We can easily test \( y_1 \leq x' \) and \( y' \leq x_2 \). If \((x', y')\) is before or after every other reservation we can of course skip one of the two tests. The running time for a single query is determined by the binary search, and is \( O(\log n) \).

b) As data structure, we can choose an AVL-tree, where we store all the current bookings, using the arrival time as the key once again. Inserting and deleting are simply the corresponding operations on the AVL-tree. For a query \((x, y)\), we have to look for the bookings that are directly before and after \((x, y)\), and then proceed exactly as in a). If every key also contains a reference to its predecessor (the next smaller key), or a reference to its successor (the next greater key) in the tree, we can find both with a single search. The asymptotic running time remains the same in both cases: \( O(\log n) \) (of course, it is the same for the insertion and deletion).