Chapter 16

16.1-3 The obvious solution using greedy-activity-selector to find a max-size set of S1 of compatible activities from S for the first lecture hall, then using it again to find a max-size set S2 of compatible activities from S-S1 from the second hall, and so on until all activities are assigned, requires $\Theta(n^2)$ time in the worst case.

There is a faster algorithm with complexity depending on the time needed to sort activities by time. ($O(n \log n)$ in general, $O(n)$ for small integers.) The general idea is to go through the activities in order of start time, assigning each to any hall that is available at that time.

Think of the start and finish times of activities as *events* kept in order by time. At any time $t$, there are two lists of lecture halls – the ones that are busy at time $t$ and the ones that are available. Process events in order. If time $t$ is a start time, assign that activity to a free hall, and move the hall from the available list to the busy list. If time $t$ is a finish time, move the hall for that activity from the busy list to the available list. By picking a hall that has already had an activity assigned to it, rather than one that has never been used, we will use as few lecture halls as possible. (Maintain the available list as a stack.)

16.1-4 shortest duration

```
xxxxxx xxxxxxxxxxx xxxxx
xxx xxx
```

least overlap

```
xxx xxx xxx
xxx xxx xxx
```

16.3-1 Prove that a binary tree that is not full cannot correspond to an optimal prefix code.

Let $T$ be a binary tree that is not full. $T$ represents a binary prefix code for a file composed of characters from alphabet $C$, where for $c \in C$, $f(c)$ is the number of occurrences of $c$ in the file. The cost of tree $T$, or the number of bits in the encoding, is $\sum_{c \in C} d_T(c) \cdot f(c)$, where $d_T(c)$ is the depth of character $c$ in tree $T$.

Let $N$ be a node of greatest depth that has exactly one child. If $N$ is the root of $T$, $N$ can be removed and the depth of each node reduced by one, yielding a tree representing the same alphabet with a lower cost. This means the original code was not optimal.

Otherwise, let $M$ be the parent of $N$, let $T1$ be the (possibly non-existent) sibling of $N$, and let $T2$ be the subtree rooted at the child of $N$. Replace $M$ by $N$, making the
children of \( N \) the roots of subtrees \( T_1 \) and \( T_2 \). If \( T_1 \) is empty, repeat the process. We have a new prefix code of lower cost, so the original was not optimal.

16.3-2

\[
\begin{array}{c}
o \\
/ \ \ \ \ \ \ \ \ \ / \\
21 \ o \\
/ \ \\
13 \ o \\
/ \\
8 \ o \\
/ \\
5 \ o \\
/ \\
3 \ o \\
/ \\
2 \ o \\
/ \\
1 \ o \\
/ \\
1 \ 1
\end{array}
\]

16-1

1. \( c = n \)

\[
\text{while } c \geq 25 \\
\text{output 1 quarter} \\
c = c - 25
\]

\[
\text{while } c \geq 10 \\
\text{output 1 dime} \\
c = c - 10
\]

\[
\text{while } c \geq 5 \\
\text{output 1 nickel} \\
c = c - 5
\]

\[
\text{while } c \geq 1 \\
\text{output 1 penny} \\
c = c - 1
\]

2. Suppose the solution is not optimal. Then some subset of some denomination should be replaced. Replacing by smaller denomination increases the number of coins used. And replacing by a larger denomination is impossible. What about combinations? There are none – the decomposition is unique.

3. one cent, ten cents, and twenty-five cents. Change for forty cents can be made in two ways: a greedy strategy gives one quarter, one dime, and five pennies, for a total of seven coins, but using four dimes results in a smaller number of coins.