R12.4 Suppose that in an instance of the coins-in-a-line game, the coins have the following values in a line:

$$V = (9, 1, 7, 3, 2, 8, 9, 3).$$

What is the maximum that the first player, Alice, can win, assuming that the second player, Bob, plays optimally?

Solution. Assuming that both play optimally, the different sub-games can be expressed by the following quantity $F[i, j]$, which represents the current player's total value, minus the other player's total value, on the game where the coins start at the i-th coin and go up to the j-th coin (included). (This is the recurrence described in the lecture notes in class.) The recurrence is

$$F[i, i] = V[i] \qquad \text{and} \qquad F[i, j] = \max\left(V[i] - F[i+1, j], V[j] - F[i, j-1]\right) \text{ if } i < j.$$

$F[i, j]$ is undefined if $i > j$. The table for the set of values V given in this problem is

	1	2	3	4	5	6	7	8
1	9	8	3	12	4	6	3	12
2	—	1	6	-3	5	3	6	-3
3	—	—	7	4	6	2	7	4
4	—	—	—	3	1	7	2	3
5	—	—	—	—	2	6	3	0
6	—	—	—	—	—	8	1	2
7	—	—	—	—	—	—	9	6
8	—	—	—	—	—	—	—	3

If both play optimally and Alice wins x and Bob wins y, then

$$x + y = \sum_{i=1}^{8} V[i] = 42 \qquad \text{and} \qquad x - y = F[1, 8] = 12,$$

which means Alice wins $x = 27$ and Bob $y = 15$. ●

R12.8 Show the longest common subsequence table L for the strings "skullandbones" and "lullabybabies". What is a longest common subsequence between these strings?

Solution. We fill out the table of longest common subsequences for prefixes of the two strings:

	ε	L	U	L	L	A	B	Y	B	A	B	I	E	S
ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	0	0	0	0	0	0	0	1
U	0	0	1	1	1	1	1	1	1	1	1	1	1	1
L	0	1	1	2	2	2	2	2	2	2	2	2	2	2
L	0	1	1	2	3	3	3	3	3	3	3	3	3	3
A	0	1	1	2	3	4	4	4	4	4	4	4	4	4
N	0	1	1	2	3	4	4	4	4	4	4	4	4	4
D	0	1	1	2	3	4	4	4	4	4	4	4	4	4
B	0	1	1	2	3	4	5	5	5	5	5	5	5	5
O	0	1	1	2	3	4	5	5	5	5	5	5	5	5
N	0	1	1	2	3	4	5	5	5	5	5	5	5	5
E	0	1	1	2	3	4	5	5	5	5	5	5	6	6
S	0	1	1	2	3	4	5	5	5	5	5	5	6	7

The longest common subsequence has length 7 and is "ullabes". The backtracking solution in the table shows the following match:

l**ullab**ybabi**es**   sk**ulla**nd**bones**. •

C12.3 Show that, in the coins-a-line game, a greedy-denial strategy of having the first player, Alice, always choose the available coin that minimizes the maximum value of the coin available to Bob will not necessarily result in an optimal solution for her.

Solution. In the game

$$100, 101, 100, 1,$$

if Alice plays according to the greedy-denial strategy, she will choose 1 (to deny Bob coin 101), then Bob will choose 100, so Alice will have a value of 102, whereas Bob will have a value of 200. On the other hand, if Alice plays optimally, she will choose 100 first, and Bob will choose 101 (if he plays optimally). Alice will earn 200 and Bob 102. This shows that the greedy-denial strategy is not always optimal. •

A12.1 Suppose that the coins of Combinatoria come in the denominations $d_1, \ldots d_k$. where $d_1 = 1$ and the other values d_i values for a set of distinct integers greater than 1. Given an integer, $n > 0$, the problem of making change is to find the fewest number of Combinatorian coins whose values sum to n.

(a) Give an instance of the making-change problem for which it is suboptimal to use the standard greedy algorithm, which repeatedly chooses a highest-valued coin whose value is at most n until the sum of chosen coins equal n.

Solution. Suppose that the denominations are $d_1 = 1, d_2 = 3, d_3 = 4$. Then to make change for $n = 6$, the greedy strategy will return three coins, namely $(4, 1, 1)$, even though this could have been done with only two coins of denomination 3. •

(b) Describe an efficient algorithm for solving the problem of making change. What is the running time of your algorithm?

Solution. Here is Python code for solving this problem:

```
def make_change(denoms, value):
    # fill table
    num_coins = [value + 1 for _ in range(value + 1)]
    num_coins[0] = 0
    best_denom = [0 for _ in range(value + 1)]
    for v in range(1, value + 1):
        for d in denoms:
            if d <= v and num_coins[v - d] + 1 < num_coins[v]:
                num_coins[v] = num_coins[v - d] + 1
                best_denom[v] = d

    # backtrack
    change = []
    while value > 0:
        change.append(best_denom[value])
        value -= best_denom[value]
    return change
```

This algorithm has runtime $O(nk)$ where k is the number of denominations and n the value for which we're making change •