**4.** Suppose you're running a lightweight consulting business—just you, two associates, and some rented equipment. Your clients are distributed between the East Coast and the West Coast, and this leads to the following question.

Each month, you can either run your business from an office in New York (NY) or from an office in San Francisco (SF). In month $i$, you'll incur an *operating cost* of $N_i$ if you run the business out of NY; you'll incur an operating cost of $S_i$ if you run the business out of SF. (It depends on the distribution of client demands for that month.)

However, if you run the business out of one city in month $i$, and then out of the other city in month $i+1$, then you incur a fixed *moving cost* of $M$ to switch base offices.

Given a sequence of $n$ months, a *plan* is a sequence of $n$ locations—each one equal to either NY or SF—such that the $i^{\text{th}}$ location indicates the city in which you will be based in the $i^{\text{th}}$ month. The *cost* of a plan is the sum of the operating costs for each of the $n$ months, plus a moving cost of $M$ for each time you switch cities. The plan can begin in either city.

**The problem.** Given a value for the moving cost $M$, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, find a plan of minimum cost. (Such a plan will be called *optimal*.)

**Example.** Suppose $n = 4$, $M = 10$, and the operating costs are given by the following table.

| | Month 1 | Month 2 | Month 3 | Month 4 |
|---|---|---|---|---|
| NY | 1 | 3 | 20 | 30 |
| SF | 50 | 20 | 2 | 4 |

Then the plan of minimum cost would be the sequence of locations

$$[NY, NY, SF, SF],$$

with a total cost of $1 + 3 + 2 + 4 + 10 = 20$, where the final term of 10 arises because you change locations once.

**(a)** Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For $i$ = 1 to $n$
  If $N_i < S_i$ then
    Output "NY in Month $i$"
  Else
    Output "SF in Month $i$"
End
```

In your example, say what the correct answer is and also what the algorithm above finds.

**(b)** Give an example of an instance in which every optimal plan must move (i.e., change locations) at least three times.

Provide a brief explanation, saying why your example has this property.

**(c)** Give an efficient algorithm that takes values for $n$, $M$, and sequences of operating costs $N_1, \ldots, N_n$ and $S_1, \ldots, S_n$, and returns the *cost* of an optimal plan.

**6.** In a word processor, the goal of “pretty-printing” is to take text with a ragged right margin, like this,

```
Call me Ishmael.
Some years ago,
never mind how long precisely,
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.
```

and turn it into text whose right margin is as "even" as possible, like this.

```
Call me Ishmael. Some years ago, never
mind how long precisely, having little
or no money in my purse, and nothing
particular to interest me on shore, I
thought I would sail about a little
and see the watery part of the world.
```

To make this precise enough for us to start thinking about how to write a pretty-printer for text, we need to figure out what it means for the right margins to be "even." So suppose our text consists of a sequence of *words*, $W = \{w_1, w_2, \ldots, w_n\}$, where $w_i$ consists of $c_i$ characters. We have a maximum line length of $L$. We will assume we have a fixed-width font and ignore issues of punctuation or hyphenation.

A *formatting* of $W$ consists of a partition of the words in $W$ into *lines*. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have

$$\left[\sum_{i=j}^{k-1}(c_i + 1)\right] + c_k \leq L.$$

We will call an assignment of words to a line *valid* if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the *slack* of the line—that is, the number of spaces left at the right margin.

Give an efficient algorithm to find a partition of a set of words $W$ into valid lines, so that the sum of the *squares* of the slacks of all lines (including the last line) is minimized.

**9.** You're helping to run a high-performance computing system capable of processing several terabytes of data per day. For each of $n$ days, you're presented with a quantity of data; on day $i$, you're presented with $x_i$ terabytes. For each terabyte you process, you receive a fixed revenue, but any unprocessed data becomes unavailable at the end of the day (i.e., you can't work on it in any future day).

You can't always process everything each day because you're constrained by the capabilities of your computing system, which can only process a fixed number of terabytes in a given day. In fact, it's running some one-of-a-kind software that, while very sophisticated, is not totally reliable, and so the amount of data you can process goes down with each day that passes since the most recent reboot of the system. On the first day after a reboot, you can process $s_1$ terabytes, on the second day after a reboot, you can process $s_2$ terabytes, and so on, up to $s_n$; we assume $s_1 > s_2 > s_3 > \cdots > s_n > 0$. (Of course, on day $i$ you can only process up to $x_i$ terabytes, regardless of how fast your system is.) To get the system back to peak performance, you can choose to reboot it; but on any day you choose to reboot the system, you can't process any data at all.

**The problem.** Given the amounts of available data $x_1, x_2, \ldots, x_n$ for the next $n$ days, and given the profile of your system as expressed by $s_1, s_2, \ldots, s_n$ (and starting from a freshly rebooted system on day 1), choose the days on which you're going to reboot so as to maximize the total amount of data you process.

**Example.** Suppose $n = 4$, and the values of $x_i$ and $s_i$ are given by the following table.

| | **Day 1** | **Day 2** | **Day 3** | **Day 4** |
|---|---|---|---|---|
| $x$ | 10 | 1 | 7 | 7 |
| $s$ | 8 | 4 | 2 | 1 |

The best solution would be to reboot on day 2 only; this way, you process 8 terabytes on day 1, then 0 on day 2, then 7 on day 3, then 4 on day 4, for a total of 19. (Note that if you didn't reboot at all, you'd process $8 + 1 + 2 + 1 = 12$; and other rebooting strategies give you less than 19 as well.)

**(a)** Give an example of an instance with the following properties.
- There is a "surplus" of data in the sense that $x_i > s_1$ for every $i$.
- The optimal solution reboots the system at least twice.

In addition to the example, you should say what the optimal solution is. You do not need to provide a proof that it is optimal.

**(b)** Give an efficient algorithm that takes values for $x_1, x_2, \ldots, x_n$ and $s_1, s_2, \ldots, s_n$ and returns the total *number* of terabytes processed by an optimal solution.