1. (2 points) Suppose you are a freelance programmer who needs to decide which job to take in each week. The set of possible jobs is divided into low-stress and high-stress ones. The basic question, each week, is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job in week $i$, you get a revenue of $\ell_i > 0$; if you select a high-stress job, you get a revenue of $h_i > 0$. The catch, however, is that in order to take on a high-stress job in week $i$, it is required that you take no job (of either type) in week $i - 1$; you need a full week of prep time to get ready for the crushing stress level. On the other hand, you can take a low-stress job in week $i$ even if you have done a job (of either type) in week $i - 1$.

Given a sequence of $n$ weeks, a (valid) plan is specified by a choice of “low-stress,” “high-stress,” or “none” for each of the $n$ weeks, with the property that if “high-stress” is chosen for week $i > 1$, then “none” has to be chosen for week $i - 1$. (It is okay to choose a high-stress job in week 1.) The value of the plan is the sum of the revenue you get in each week: $\ell_i$ if you choose “low-stress” in week $i$, $h_i$ if you choose “high-stress”, and 0 if you choose “none”.

**The problem.** Given $n > 0$ and $\ell_1, \ldots, \ell_n$, $h_1, \ldots, h_n > 0$, find a plan of maximum value.

**Example.** Suppose $n = 4$, and the values of $\ell_i$ and $h_i$ are given by the following table.

<table>
<thead>
<tr>
<th>week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell$</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$h$</td>
<td>5</td>
<td>50</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Then the plan of maximum value would be to choose no job in week 1, a high-stress job in week 2, and low-stress jobs in weeks 3 and 4. The value of this plan would be $0 + 50 + 10 + 10 = 70$.

(a) Does the following algorithm correctly solve this problem? Justify your answer.

```plaintext
for $i = 1$ to $n$ do
  if $h_{i+1} > \ell_i + \ell_{i+1}$ then
    Output “choose no job in week $i$” ;
    Output “choose a high-stress job in week $i + 1$” ;
    Continue with iteration $i + 2$ ;
  else
    Output “choose a low-stress job in week $i$” ;
    Continue with iteration $i + 1$ ;
```

(To avoid problems with overflowing array bounds, we define $h_i = \ell_i = 0$ when $i > n$.)

(b) Give an algorithm that outputs the value of the optimal plan. Your algorithm should run in time $O(n)$. Prove the correctness of your algorithm and analyze its running time.
2. (2 points) 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, we need to define 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

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

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 remaining spaces at the right margin.

Give an 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. Your algorithm should take as input \( n > 0 \) and \( c_1, \ldots, c_n \), and output the minimum-possible sum of the squares of the slacks. Your algorithm should run in time \( O(n^2) \). Prove the correctness of your algorithm and analyze its running time.

3. (2 points) Suppose it is near the end of the semester and you are taking \( n \) courses, each with a final project that still has to be done. Each project will be graded on the following scale: it will receive an integer score on a scale of 0 to 100, higher numbers being better grades. Your goal is to maximize your average grade on the \( n \) projects. You have a total of \( H \) hours in which to work on the \( n \) projects cumulatively, and you want to decide how to divide up this
time. For simplicity, assume $H > 0$ is an integer, and you will spend an integer number of hours on each project.

To figure out how best to divide up your time, you have come up with a set of functions $(f_i)_{i=1}^n$ for each of your $n$ courses; if you spend $h < H$ hours on the project for course $i$, you will get a grade of $f_i(h)$. (You may assume that each function $f_i$ are nondecreasing: if $h < h'$, then $f_i(h) \leq f_i(h')$. You can also assume $f_i(0) = 0$ for all $i$.)

**The problem.** Given these functions $(f_i)_{i=1}^n$, decide how many hours to spend on each project (in integer values only) so that your average grade, as computed according to the $f_i$, is as large as possible. Your algorithm should run in time $O(n^3 H^3)$. Prove the correctness of your algorithm and analyze its running time.

4. Use the Bellman-Ford algorithm to decide the length of the shortest path from $s = 1$ to every node in the graph. Show your work.

An implementation of the Bellman-Ford algorithm is given below. (You can work with other implementations if you prefer.)

---

**Algorithm 1: Bellman-Ford Algorithm**

- **Input**: An $n$-node directed graph $G$ with edge weights $w$ and a source node $s$.
- **Output**: The length of the shortest path from $s$ to every node in $G$.

```plaintext
for $u = 1$ to $n$ do
  $d[u] \leftarrow +\infty$;
  $d[s] \leftarrow 0$;
for $i = 1$ to $n$ do
  change $\leftarrow 0$;
  for each edge $e = (u,v) \in E$ with weight $w_e$ do
    if $d[u] + w_e < d[v]$ then
      $d[v] \leftarrow d[u] + w_e$;
      change $\leftarrow 1$;
  if change $= 0$ then break;
  if change $= 1$ and $i = n$ then
    return error “$G$ contains a negative cycle”;
return $d$
```

---