1. (2 points) Suppose you are a freelance programmer. There are two jobs available each week, a low-stress job and a high-stress job. You have three options each week: take the low-stress job, take the high-stress job, or take a rest.

In week $i$, the low-stress job earns you $\ell_i > 0$ dollars, the high-stress job earns you $h_i > 0$ dollars, and taking a rest earns you 0 dollars. The catch, however, is that in order to take on a high-stress job in week $i$, it is required that you take a rest in week $(i - 1)$; you need a full week of prep time to get ready for the crushing stress level.

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

(Formally, a *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 ok to choose a high-stress job in week 1. The value of the plan is the total amount of money you can make in these $n$ weeks.)

**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 greedy algorithm work for 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 array out of bounds error, 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) It is near the end of the semester. Suppose you are taking \( n \) courses, each with a final project that still has to be done. Your goal is to figure out how best to divide up your time to maximize your average grade on these \( n \) projects.

You have a total of \( H \) hours to spend on these \( n \) projects. Suppose \( H \geq 0 \) is an integer, and you can only spend an integer number of hours on each project.

You have come up with a set of functions \( (f_i)_{i=1}^n \) for each of your projects: if you spend \( 0 \leq h \leq H \) hours on project \( i \), you will receive a grade of \( f_i(h) \) on that project. For simplicity, you can assume that each function \( f_i \) is nondecreasing (i.e., if \( h < h' \), then \( f_i(h) \leq f_i(h') \)). You can also assume \( f_i(0) = 0 \) for all \( i \).

**Goal:** Given the functions \( (f_i)_{i=1}^n \), decide how many hours to spend on each project so that your average grade is as high as possible. Your algorithm should run in time \( O(nH^2) \). Prove the correctness of your algorithm and analyze its running time.

3. (2 points) Simulate the Bellman-Ford algorithm on the following graph to compute the length of the shortest path from \( s = 1 \) to every node. Show your work.

An implementation of Bellman-Ford is provided. (You can work with other implementations.)

---

**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 \( G \).

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