1. You’ve been hired to store a sequence of \( n \) books on shelves in a library. The order of the books is fixed by the cataloging system and cannot be changed; each shelf must store a contiguous interval of the given sequence of books. You are given two arrays \( H[1..n] \) and \( T[1..n] \), where \( H[i] \) and \( T[i] \) are respectively the height and thickness of the \( i \)th book in the sequence. All shelves in this library have the same length \( L \); the total thickness of all books on any single shelf cannot exceed \( L \).

(a) Suppose all the books have the same height \( h \) (that is, \( H[i] = h \) for all \( i \)) and the shelves have height larger than \( h \), so each book fits on every shelf. Describe and analyze a greedy algorithm to store the books in as few shelves as possible. [Hint: The algorithm is obvious, but why is it correct?]

Solution: The algorithm is simple: If there are any books left, pack as many books as possible onto the first shelf and then recurse.

```plaintext
SHELFBOOKS(T[1..n], L):
width ← 0 \( \langle \) total width of books on current shelf\( \rangle \)
for \( i ← 1 \) to \( n \)
    width ← width + \( T[i] \)
    if \( width > L \)
        return 1 + SHELFBOOKS(T[1..n], L)
return 1
```

We can express this algorithm non-recursively as follows:

```plaintext
SHELFBOOKS(T[1..n], L):
num ← 0 \( \langle \) number of shelves used\( \rangle \)
width ← L \( \langle \) total width of books on current shelf\( \rangle \)
for \( i ← 1 \) to \( n \)
    if \( width + T[i] > L \)
        num ← num + 1
        width ← 0
    width ← width + \( T[i] \)
return num
```

In either formulation, the algorithm runs in \( O(n) \) time.

Let \( g_1 \) be the maximum number of books that fit in the first shelf. The correctness of the algorithm follows by induction from the following claim:

**Lemma 1.** There is an optimal solution where the first shelf contains the first \( g_1 \) books.

**Proof:** Consider any legal partition \( P \) of the books into shelves, where the first shelf contains fewer than \( g_1 \) books. Let \( P' \) be the partition obtained from \( P \) by moving the first \( g_1 \) books to the first shelf, and leaving all other books as assigned by \( P \). Then \( P' \) uses at most as many shelves at \( P \). In particular, if \( P \) is an optimal partition, then so is \( P' \). \( \square \)

We can now prove the algorithm correct by induction. Assume the greedy algorithm finds a solution for any sequence of fewer than \( n \) books. Lemma 1 implies that there is an optimal solution where the first shelf contains \( g_1 \) books, and the induction hypothesis implies that the algorithm recursively finds the minimum number of shelves for books \( g_1 + 1 \) through \( n \).

**Rubric:** 4 points = 1 for the algorithm + \( \frac{1}{2} \) for time analysis + 2 for Lemma 1 (or equivalent exchange argument) + \( \frac{1}{2} \) for other details. This are not the only correct solution, or even the only correct way of formulating this solution.
(b) Suppose the books have different heights, but you can adjust the height of each shelf to match the tallest book on that shelf. Now your task is to store the books so that the sum of the heights of the shelves is as small as possible. Show that your greedy algorithm from part (a) does not always give the best solution to this problem.

**Solution:** Consider three books with heights 1, 2, 2, all with thickness 1, and let \( L = 2 \). The greedy algorithm puts the first two books on one shelf and the third book on another; the total height of this solution is 4. But if we put the first book on one shelf and the other two books on the another, the total height is only 3.

**Rubric:** 1 point: all or nothing. This is not the only solution.

(c) Describe and analyze an algorithm to find the best assignment of books to shelves as described in part (b).

**Solution:** For any index \( i \), let \( \text{MinTotalH}(i) \) denote the minimum total height required to shelve books \( i \) through \( n \). We also define two helper functions for all indices \( i \leq j \):
- \( \text{MaxH}(i,j) \) denotes the maximum height of books \( i \) through \( j \).
- \( \text{SumT}(i,j) \) denotes the total thickness of books \( i \) through \( j \).

These functions satisfy the following recurrences:

\[
\text{MaxH}(i,j) = \begin{cases} 
0 & \text{if } i > j \\
\max \{ H[i], \text{MaxH}(i+1,j) \} & \text{otherwise}
\end{cases}
\]

\[
\text{SumT}(i,j) = \begin{cases} 
0 & \text{if } i > j \\
T[i] + \text{SumT}(i+1,j) & \text{otherwise}
\end{cases}
\]

\[
\text{MinTotalH}(i) = \begin{cases} 
0 & \text{if } i > n \\
\min \left\{ \begin{array}{l}
\text{MaxH}(i,j) \\
+ \text{MinTotalH}(j+1)
\end{array} \right\} & \text{if } i \leq j \leq n \text{ and } \text{SumT}(i,j) \leq L \\
\text{MinTotalH}(i) & \text{otherwise}
\end{cases}
\]

The following dynamic programming algorithm evaluates these recurrences and computes \( \text{MinTotalH}(1) \) in \( O(n^2) \) time:

```
\text{MinTotalH}(T[1..n], H[1..n], L):
for } j \leftarrow 1 \text{ to } n 
\text{MaxH}[j+1,j] \leftarrow 0 
\text{SumT}[j+1,j] \leftarrow 0 
\text{for } i \leftarrow j \text{ down to } 1 
\text{MaxH}[i,j] \leftarrow \max \{H[i], \text{MaxH}[i+1,j]\} 
\text{SumT}[i,j] \leftarrow T[i] + \text{SumT}[i+1,j] 
\text{MinTotalH}[n+1] \leftarrow 0 
\text{for } i \leftarrow n \text{ down to } 1 
\text{MinTotalH}[i] \leftarrow \infty 
\text{while } j \leq n \text{ and } \text{SumT}[i,j] \leq L 
\text{MinTotalH}[i] \leftarrow \min \{ \text{MinTotalH}[i], \text{MaxH}[i,j] + \text{MinTotalH}[j+1] \}
return \text{MinTotalH}[1]
```

**Rubric:** 5 points: standard dynamic programming rubric. This is not the only correct formulation of this algorithm.
Consider a directed graph $G$, where each edge is colored either red, white, or blue. A walk in $G$ is called a French flag walk if its sequence of edge colors is red, white, blue, red, white, blue, and so on. More formally, a walk $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ is a French flag walk if, for every integer $i$, the edge $v_i \rightarrow v_{i+1}$ is red if $i \mod 3 = 0$, white if $i \mod 3 = 1$, and blue if $i \mod 3 = 2$.

Describe an efficient algorithm to find all vertices in a given edge-colored directed graph $G$ that can be reached from a given vertex $v$ through a French flag walk.

**Solution:** Given the edge-colored graph $G = (V, E)$, we construct a new directed graph $G'$ with vertices $V \times \{R, W, B\}$ and the following edges:

- $(x, B)\rightarrow(y, R)$ for every red edge $x \rightarrow y \in E$;
- $(x, R)\rightarrow(y, W)$ for every white edge $x \rightarrow y \in E$; and
- $(x, W)\rightarrow(y, B)$ for every blue edge $x \rightarrow y \in E$.

Every walk in $G'$ that starts at a “blue” vertex $(x, B)$ corresponds to a French flag walk in $G$ that starts at the corresponding vertex $x$, and vice versa. Thus, our task is to determine which vertices are reachable in $G'$ from vertex $(v, B)$. We can solve this problem by whatever-first search in $O(V' + E') = O(V + E)$ time.

**Rubric:** 10 points; standard graph reduction rubric
- “whatever-first search” is not a problem; it’s an algorithm.
- $-\frac{1}{2}$ for writing “depth” or “breadth” instead of “whatever”
3. **Racetrack** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade. The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

Each car has a position and a velocity, both with integer $x$- and $y$-coordinates. A subset of grid squares is marked as the starting area, and another subset is marked as the finishing area. The initial position of each car is chosen by the player somewhere in the starting area; the initial velocity of each car is always $(0, 0)$. At each step, the player optionally increments or decrements either or both coordinates of the car’s velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car’s new position is then determined by adding the new velocity to the car’s previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position inside the finishing area.

Suppose the racetrack is represented by an $n \times n$ array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the “starting area” is the first column, and the “finishing area” is the last column.

Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack. [Hint: Build a graph. No, not that graph, a different one. What are the vertices? What are the edges? What problem is this?]

**Solution:** First, we claim that the horizontal speed of the car cannot exceed $\sqrt{2n}$. Suppose the car starts in column 0 of the grid, with velocity zero, and accelerates as quickly as possible to the right. After $t$ steps, the car has velocity $(t, 0)$ and lies in column $\sum_{i=1}^t i = t(t + 1)/2$. Because the car cannot go past the right edge of the grid, we must have $t(t + 1)/2 \leq n$, which implies that $t \leq \sqrt{2n}$. Similar arguments imply that the car’s vertical speed cannot exceed $\sqrt{2n}$.

Let $T[1..n, 1..n]$ be the input bitmap. We construct a directed graph $G$ with $O(n^3)$ vertices and $O(n^3)$ edges as follows:

- $G$ has a vertex for each integer vector $(x, y, \Delta x, \Delta y)$ that represents a legal position and velocity for the car. The integer vector $(x, y)$ is a legal position if and only if $T[x, y] = 0$ (and therefore $1 \leq x \leq n$ and $1 \leq y \leq n$). Similarly, as we argued above, $(\Delta x, \Delta y)$ is a legal velocity if and only if $-\sqrt{2n} \leq \Delta x \leq \sqrt{2n}$ and $-\sqrt{2n} \leq \Delta y \leq \sqrt{2n}$.
- $G$ has a directed edge for each legal move by the car. Specifically, $G$ contains the directed edge $(x, y, \Delta x, \Delta y) \rightarrow (x', y', \Delta x', \Delta y')$ if and only if both endpoints are legal vertices and all of the following conditions are satisfied:

  \[
  \begin{align*}
  x' &= x + \Delta x \quad \Delta x' &\in \{\Delta x - 1, \Delta x, \Delta x + 1\} \\
  y' &= y + \Delta y \quad \Delta y' &\in \{\Delta y - 1, \Delta y, \Delta y + 1\}
  \end{align*}
  \]

- $G$ also has an artificial starting vertex $s$, with $O(n)$ outgoing edges to every vertex of the form $(1, y, 0, 0)$ such that the point $(1, y)$ is in the starting area.
- Finally, $G$ also has an artificial target vertex $t$, with $O(n^2)$ incoming edges from every vertex of the form $(n, y, \Delta x, \Delta y)$ such that the point $(n, y)$ is in the finishing area.

By construction, every directed path from $s$ to $t$ in $G$ represents a sequence of steps that begins with the car at the starting line with velocity $(0, 0)$ and ends with the car at the finish line. Moreover, a path of length $\ell$ corresponds to a sequence of $\ell - 2$ steps. Thus, the shortest sequence of steps corresponds to the shortest path from $s$ to $t$ in $G$. We compute this shortest path in $O(V + E) = O(n^3)$ time via breadth-first search.  


Rubric: 10 points: standard graph-reduction rubric.

- 1 for using Dijkstra’s algorithm instead of BFS.
- 1 for only showing $O(n^4)$ time instead of $O(n^3)$ time for this algorithm.
- Max 8 points for an algorithm that actually runs in $O(n^4)$ time (for example, BFS from each start position); scale partial credit.
- 1 for each factor of $n$ in the running time above $n^4$. Thus, an algorithm that runs in $\Theta(n^c)$ time is worth at most $12 - c$ points, for any $c \geq 4$. Scale partial credit.