1. Describe and analyze an algorithm to determine the minimum number of times that the player must tap the screen to win Flappy Pixel, given the integer h and the arrays $Hi[1..n]$ and $Lo[1..n]$ as input. If the game cannot be won at all, your algorithm should return ∞. Describe the running time of your algorithm as a function of n and h.

Solution (graph reduction): We reduce the problem to computing shortest paths in a dag $G = (V, E)$ defined as follows:

- V is the set of all triples (x, y, y') that correspond to possible positions and vertical speeds for Flappy, plus one additional vertex t. The values x, y, and y' that we actually care about fall within the following ranges:

$$1 \le x \le n \qquad 1 \le Lo[x] \le y \le Hi[x] \le h \qquad -\sqrt{2h} \le y' \le 10.$$

  The first two bounds follow directly from the problem statement. The lower bound $y' \ge -\sqrt{2h}$ is a consequence of the following observation: If Flappy started at height h with speed 0 and then accelerated downward (by not flapping) for $\sqrt{2h}$ iterations, his height would be $h - \sqrt{2h}(\sqrt{2h}+1)/2 < 0$, which is impossible. There are $O(nh^{3/2})$ vertices.

- E is the set of all possible transitions for Flappy. These are all pairs of the forms

$$(x, y, y') \to (x+1, y+10, 10) \qquad \text{or} \qquad (x, y, y') \to (x+1, y+y'-1, y'-1)$$

  such that both triples in each pair are actually vertices. Since every vertex has at most two outgoing edges, there are $O(nh^{3/2})$ edges altogether. Edges of the first type (flaps) have weight 1; edges of the second type (nonflaps) have weight 0.

G must be a dag, because every edge in G has the form $(x, \cdot, \cdot) \to (x+1, \cdot, \cdot)$.

Now we need to compute the shortest path from the vertex $(0, \lceil h/2 \rceil, 0)$ to any vertex of the form (n, y, y'). Because G is a dag, we can compute these shortest paths in $O(V + E) = \boldsymbol{O(nh^{3/2})}$ ***time*** via dynamic programming/depth-first search.[a] ■

[a]Because every edge in G has weight 0 or 1, we can also achieve this running time using a variant of breadth-first search/Dijkstra's algorithm that maintains two separate queues containing only weight-0 edges and only weight-1 edges, and Pulls from the 1-queue only when the 0-queue is empty.

Solution (dynamic programming): Let $Flaps(x, y, y')$ denote the minimum number of times the player must tap the screen to win, starting at position (x, y) with vertical speed y'. This function satisfies the following recurrence (with the sentinel values $Lo[0] = Hi[0] = \lceil h/2 \rceil$):

$$Flaps(x, y, y') = \begin{cases} \infty & y < Lo[x] \text{ or } y > Hi[x] \\ 0 & \text{if } x = n \\ \min \begin{Bmatrix} 1 + Flaps(x+1, y+10, 10) \\ Flaps(x+1, y+y'-1, y'-1) \end{Bmatrix} & \text{otherwise} \end{cases}$$

Our goal is to compute $\mathit{Flaps}(0, \lceil h/2 \rceil, 0)$.

As we argued in the previous solution, the values x, y, and y' that we care about fall within the following ranges:

$$0 \le x \le n \qquad 1 \le y \le h \qquad \mathit{miny}' \le y' \le 10,$$

where $\mathit{miny}' = \lceil -\sqrt{2h} \rceil$. Thus, we can memoize *Flaps* into a three-dimensional array $\mathit{Flaps}[0\,..\,n, 1\,..\,h, \mathit{miny}'\,..\,10]$. Because each entry $\mathit{Flaps}[x, y, y']$ depends only on entries of the form $\mathit{Flaps}[x+1, \cdot, \cdot]$, we can fill the array by considering x in decreasing order in the outermost loop, and filling each two-dimensional slice $\mathit{Flaps}[x, \cdot, \cdot]$ in any order we like in two inner loops. The resulting algorithm runs in ***$O(nh^{3/2})$ time.***

Here is standalone pseudocode for the dynamic programming algorithm. The body of the innermost loop is more complicated than the recurrence so that we don't have to memoize $\mathit{Flaps}[x, y, y']$ when $y \le 0$ or $y > h$.

```
MinimumFlaps($Hi[1..n], Lo[1..n]$):
  $miny' \gets \lceil -\sqrt{2h} \rceil$
  for $y \gets 0$ to $h$
      for $y' \gets miny'$ to 10
          if $y < Lo[n]$ or $y > Hi[n]$
              $Flaps[n, y, y'] \gets \infty$
          else
              $Flaps[n, y, y'] \gets 0$
  for $x \gets n-1$ down to 0
      for $y \gets 0$ to $h$
          for $y' \gets miny'$ to 10
              if $y < Lo[x]$ or $y > Hi[x]$
                  $Flaps[x, y, y'] \gets \infty$
              else
                  if $y + 10 > h$
                      $yes \gets \infty$
                  else
                      $yes \gets 1 + Flaps[x+1, y+10, 10]$
                  if $y + y' - 1 < 1$ or $y + y' - 1 > h$
                      $no \gets \infty$
                  else
                      $no \gets Flaps[x+1, y+y'-1, y'-1]$
                  $Flaps[x, y, y'] \gets \min\{yes, no\}$
  return $Flaps[0, \lceil h/2 \rceil, 0]$
```

■

2. 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.

   Assume 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 line" consists of all `0` bits in column 1, and the "finishing line" consists of all `0` bits in column n.

Solution (graph reduction): First, as in the previous problem, we claim that the horizontal speed of the car cannot exceed $\sqrt{2n}$. Suppose the car starts in column 0, 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 $(x, y, \Delta x, \Delta y) \to (x', y', \Delta x', \Delta y')$ if and only if both endpoints are legal vertices and all of the following conditions are satisfied:

$$\begin{aligned} x' &= x + \Delta x \qquad & \Delta x' &\in \{\Delta x - 1, \Delta x, \Delta x + 1\} \\ y' &= y + \Delta y \qquad & \Delta y' &\in \{\Delta y - 1, \Delta y, \Delta y + 1\} \end{aligned}$$

- 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 $T[1, y] = 0$.
- 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 $T[n, y] = 0$.

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 ℓ 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 can compute this shortest path in $O(V + E) = \boldsymbol{O(n^3)}$ ***time*** via ***breadth-first search***. ■

To think about later:

3. Consider the following variant of Flappy Pixel. The mechanics of the game are unchanged, but now the environment is specified by an array $Points[1..n, 1..h]$ of integers, which could be positive, negative, or zero. If Faby falls off the top or bottom edge of the environment, the game immediately ends and the player gets nothing. Otherwise, at each frame, the player earns $Points[x, y]$ points, where (x, y) is Faby's current position. The game ends when Faby reaches the right end of the environment.

   Describe and analyze an algorithm to determine the maximum possible score that a player can earn in this game.

   Solution (graph reduction): We construct the same dag G defined in the first solution to problem 1, but now we assign a weight of $Points[x, y]$ to each edge of the form $(x-1,\cdot,\cdot)\to(x, y, y')$. Now we need to compute the *longest* path from vertex $(0, \lceil h/2 \rceil, 0)$ to any vertex of the form $(n,\cdot,\cdot)$. We can do this in $O(V+E) = \boldsymbol{O(nh^{3/2})}$ ***time*** using the algorithm described in class and in the textbook. ■

   Solution (dynamic programming): Let $MaxScore(x, y, y')$ denote the maximum score that the player can earn if they start at position (x, y) with vertical speed y'. This function satisfies the following recurrence (with sentinel value $Points[0, \lceil h/2 \rceil] = 0$):

$$MaxScore(x, y, y') = \begin{cases} 0 & \text{if } x > n \\ \infty & y < 1 \text{ or } y > h \\ Points[x, y] + \max\left\{\begin{matrix} MaxScore(x+1, y+10, 10) \\ MaxScore(x+1, y+y'-1, y'-1) \end{matrix}\right\} & \text{otherwise} \end{cases}$$

   We need to compute $MaxScore(0, \lceil h/2 \rceil, 0)$.

   As in the previous solution, we can memoize $MaxScore$ into an array $MaxScore[0..n, 1..h, \lceil -\sqrt{2h} \rceil..10]$, which we can fill the array by considering x in decreasing order in the outermost loop, and filling each two-dimensional slice $MaxScore[x,\cdot,\cdot]$ in any order we like in two inner loops. The resulting algorithm runs in $\boldsymbol{O(nh^{3/2})}$ ***time.*** ■

4. We can also consider a similar variant of Racetrack. Instead of bits, the “track” is described by an array $Points[1..n, 1..n]$ of *numbers*, which could be positive, negative, or zero. Whenever the car lands on a grid cell (i, j), the player receives $Points[i, j]$ points. Forbidden grid cells are indicated by $Points[i, j] = -\infty$.

   Describe and analyze an algorithm to find the largest possible score that a player can earn by moving a car from column 1 (the starting line) to column n (the finish line).

   [Hint: Wait, what if all the point values are positive?]

Solution (graph reduction): 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 exactly as we did for problem 2.

- 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 now a legal position if and only if $1 \le x \le n$ and $1 \le y \le n$. Similarly, as we argued in the solution for problem 2, $(\Delta x, \Delta y)$ is a legal velocity if and only if $-\sqrt{2n} \le \Delta x \le \sqrt{2n}$ and $-\sqrt{2n} \le \Delta y \le \sqrt{2n}$.
- G has a directed edge for each legal move by the car. Specifically, G contains the directed edge $(x, y, \Delta x, \Delta y) \to (x', y', \Delta x', \Delta y')$ if and only if both endpoints are legal vertices and all of the following conditions are satisfied:

  $$\begin{array}{ll} x' = x + \Delta x & \Delta x' \in \{\Delta x - 1, \Delta x, \Delta x + 1\} \\ y' = y + \Delta y & \Delta y' \in \{\Delta y - 1, \Delta y, \Delta y + 1\} \end{array}$$

  Each edge $(x, y, \Delta x, \Delta y) \to (x', y', \Delta x', \Delta y')$ has weight $-Points[x', y']$, the *negation* of the point value for the ending position.
- G also has an artificial starting vertex s, with an outgoing edge $s \to (1, y, 0, 0)$ with weight $-Points[1, y]$ for all $1 \le y \le n$.
- Finally, G also has an artificial target vertex t, with $O(n^2)$ incoming edges with weight 0 from every vertex of the form $(n, y, \Delta x, \Delta y)$ such that $T[n, y] = 0$.

By construction, every directed ***walk*** 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 walk of length L corresponds to a sequence of steps that earn the player $-L$ points. Thus, the *maximum* achievable score corresponds to the ***shortest walk*** from s to t in G.

Moreover, a negative cycle in G corresponds to a sequence of moves that begins and ends at the same location and velocity, and that earns a positive total number of points. Just as a negative cycle in G implies that shortest paths in G don't exist, a *positive* cycle of moves implies that there is no maximum achievable score—we can always get more points by going around the cycle one more time.

We can either compute shortest walk from s to t in G, or correctly report that no shortest walk exists, in $O(VE) = \boldsymbol{O(n^6)}$ ***time*** using Bellman-Ford. ■