1. ***Flappy Bird*** is a popular mobile game written by Nguyễn Hà Đông, originally released in May 2013. The game features a bird named "Faby", who flies to the right at constant speed. Whenever the player taps the screen, Faby is given a fixed upward velocity; between taps, Faby falls due to gravity. Faby flies through a landscape of pipes until it touches either a pipe or the ground, at which point the game is over. Your task, should you choose to accept it, is to develop an algorithm to play Flappy Bird automatically.

   Well, okay, not Flappy Bird exactly, but the following drastically simplified variant, which I will call ***Flappy Pixel***. Instead of a bird, Faby is a single point, specified by three integers: horizontal position x (in pixels), vertical position y (in pixels), and vertical speed y' (in pixels per frame). Faby's environment is described by two arrays $Hi[1..n]$ and $Lo[1..n]$, where for each index i, we have $0 < Lo[i] < Hi[i] < h$ for some fixed height value h. The game is described by the following piece of pseudocode:

```
FlappyPixel(Hi[1..n], Lo[1..n]):
  y ← ⌈h/2⌉
  y' ← 0
  for x ← 1 to n
      if the player taps the screen
          y' ← 10              ⟨⟨flap⟩⟩
      else
          y' ← y' − 1          ⟨⟨fall⟩⟩
      y ← y + y'
      if y < Lo[x] or y > Hi[x]
          return False         ⟨⟨player loses⟩⟩
  return True                  ⟨⟨player wins⟩⟩
```

   Notice that in each iteration of the main loop, the player has the option of tapping the screen.

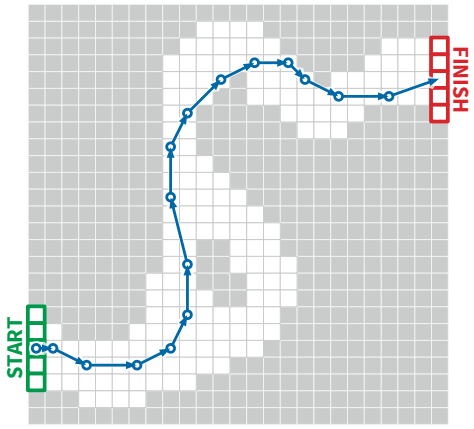
   Describe and analyze an algorithm to determine the minimum number of times that the player must tap the screen to win Flappy Pixel, given the integer h and the arrays $Hi[1..n]$ and $Lo[1..n]$ as input. If the game cannot be won at all, your algorithm should return ∞. Describe the running time of your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

[Problem 2 is on the back.]

2. ***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.[1] 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 changes each component of the velocity by at most 1. 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.[2] The race ends when the first car reaches a position inside the finishing area.

velocity	position
$(0,0)$	$(1,5)$
$(1,0)$	$(2,5)$
$(2,-1)$	$(4,4)$
$(3,0)$	$(7,4)$
$(2,1)$	$(9,5)$
$(1,2)$	$(10,7)$
$(0,3)$	$(10,10)$
$(-1,4)$	$(9,14)$
$(0,3)$	$(9,17)$
$(1,2)$	$(10,19)$
$(2,2)$	$(12,21)$
$(2,1)$	$(14,22)$
$(2,0)$	$(16,22)$
$(1,-1)$	$(17,21)$
$(2,-1)$	$(19,20)$
$(3,0)$	$(22,20)$
$(3,1)$	$(25,21)$



A 16-step Racetrack run, on a 25 × 25 track. This is *not* the shortest run on this track.

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

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: Your initial analysis can be improved.]

[1] The actual game is a bit more complicated than the version described here. See http://harmmade.com/vectorracer/ for an excellent online version.

[2] However, it is not necessary for the entire line segment between the old position and the new position to lie inside the track. Sometimes Speed Racer has to push the A button.

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.

```
FlappyPixel2(Points[1..n]):
  score ← 0
  y ← ⌈h/2⌉
  y' ← 0
  for x ← 1 to n
      if the player taps the screen
          y' ← 10          ⟨⟨flap⟩⟩
      else
          y' ← y' − 1      ⟨⟨fall⟩⟩
      y ← y + y'
      if y < 1 or y > h
          return −∞        ⟨⟨fail⟩⟩
      score ← score + Points[x, y]
  return score
```

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

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?]