1. **Are you comfortable with graphs?**

(a) Suppose $Y[0] = Y[n] = 0$ and $Y[i] > 0$ for every other index i; that is, the endpoints of the path are strictly below every other point on the path. Prove that under these conditions, Alice and Bob can meet.

Solution: First, we define a point (x, y) on the path to be a ***stop*** if $y = Y[i]$ for some index i. Every corner of the path is a stop; every point on the path at the same y-coordinate as a corner is also a stop. We call two stops *adjacent* if the line segment between them is entirely contained in the path and contains no other stop, or equivalently, if they are adjacent in x-coordinate order.

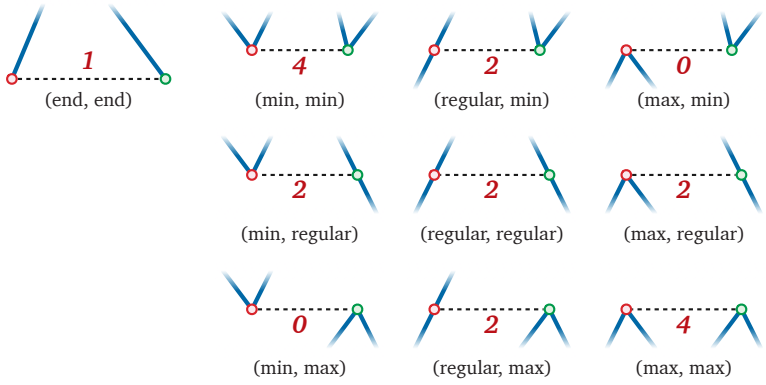
Now we define an undirected graph $G = (V, E)$ as follows:

- The vertices are all triples (x_A, x_B, y) such that the points (x_A, y) and (x_B, y) are stops. For each vertex (x_A, x_B, y), we call the corresponding pair of stops (x_A, y) and (x_B, y) a *vertex configuration*.
- Vertices (x_A, x_B, y) and (x'_A, x'_B, y') are adjacent if and only if (1) stops (x_A, y) and (x'_A, y') are adjacent and (2) stops (x_B, y) and (x'_B, y') are adjacent.

In each vertex configuration, Alice is either

- at one of the endpoints, so she can only move up,
- at a local minimum, so she can only move up,
- at a local maximum, so she can only move down, or
- at a *regular* point, where she can move either up or down.

Bob has the same four options. Because the endpoints of the path lie strictly below every other corner, either both Alice and Bob are at endpoints, or neither of them is at an endpoint; otherwise, any combination is possible. This leaves us with exactly ten types of vertex configurations, shown below.



- If Alice and Bob are at endpoints, they can only move inward. Thus, the four (end, end) vertices each have degree 1.
- If Alice and Bob are both at local minima, they have four options: as they both move upward, Alice can move left or right, and independently, Bob can move left or right. Thus, every (min, min) vertex has degree 4. Similarly, every (max, max) vertex has degree 4.
- If Alice is at an local maximum and Bob is at a local minimum, or vice versa, neither of them can move at all. Thus, every (min, max) and (max, min) vertex has degree 0.

- If Alice is at a local minimum and Bob is at a regular point, then Bob can only move down, but Alice has two options. Thus, every (min, regular) vertex has degree 2. Symmetrically, every (max, regular), (regular, min), and (regular, max) vertex has degree 2.
- In the only remaining case, Alice and Bob are both at regular points, so they can either both move right or both move left. Thus, every (regular, regular) vertex has degree 2.

We conclude that the graph G has exactly four vertices of degree 1, and every other vertex has even degree. Let's call the degree-1 vertices LL, LR, RL, and RR, where the letters indicate which endpoint(s) Alice and Bob occupy.

Finally, let H be the component of G that contains the vertex LR, which corresponds with Alice and Bob's starting configuration. By the Handshake Lemma, H contains at least one of the other degree-1 vertices.[1]

- If H contains RR, then Alice and Bob can meet by following a path in H from LR to RR.
- If H contains LL, then Alice and Bob can meet by following a path in H from LR to LL.
- If H contains RL, then Alice and Bob can *exchange their positions* by following a path in H from LR to RL. Because Alice and Bob move continuously, Alice and Bob must meet at some intermediate time.

In all cases, Alice and Bob can meet. ■

[1] In fact, H contains *all* of the other degree-1 vertices.

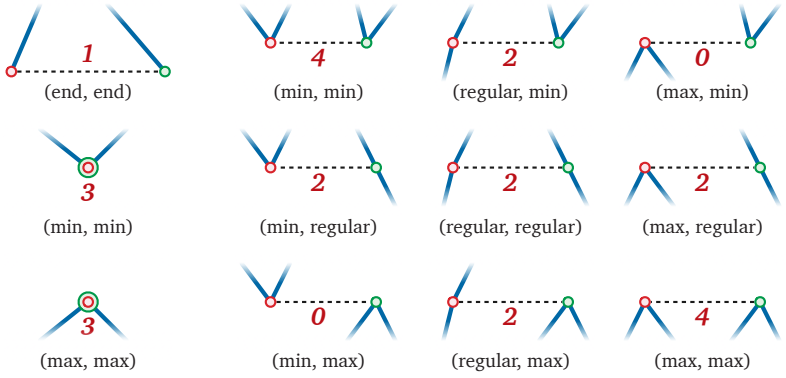
Solution (smaller graph): We define a graph $G = (V, E)$ as follows:

- The vertices are all triples (x_A, x_B, y) that meet the following conditions:
  - $x_A \le x_B$
  - Both points (x_A, y) and (x_B, y) lie on the path
  - At least one of the points (x_A, y) and (x_B, y) is a corner of the path

  Each vertex represents an "interesting" pair of positions for Alice and Bob. For each vertex (x_A, x_B, y), we call the corresponding pair of points (x_A, y) and (x_B, y) a *vertex configuration*.
- Two vertices (x_A, x_B, y) and (x'_A, x'_B, y') are connected by an undirected edge if and only if Alice and Bob can move from one vertex configuration to the other without passing through any other vertex configurations. Geometrically, this means that the path contains the line segments $(x_A, y)(x'_A, y')$ and $(x_B, y)(x'_B, y')$.

Now consider the various types of vertex configurations.

- If Alice is at an endpoint, then Bob must also be at an endpoint. There are exactly three such vertices (because $x_A \le x_B$), each with degree 1.
- If Alice and Bob are at the same local maximum or local minimum of the path, they have three choices: Move together left, move together right, or Alice moves left and Bob moves right. Thus, these $n-2$ vertices each have degree 3.
- In every other vertex configuration, Alice is either at a local minimum, at a local maximum, or neither; Bob has a similar set of choices. Straightforward analysis of all nine cases implies that every other vertex of G has degree 0, 2, or 4; as illustrated below. (The central case (regular, regular) includes configurations where Alice and Bob are on the same corner of the path, but that corner is neither a local maximum nor a local minimum.)



In short, the start vertex s has degree 1, and every other odd-degree vertex in G corresponds to a configuration where Alice and Bob are at the same location.

Let H be the component of G that contains the start vertex s. The Handshake Lemma implies that H contains at least one other odd-degree vertex t.[2] Alice and Bob can meet by following the path from s to t in H. ■

Rubric: 5 points = 1 for vertices + 1 for edges + 2 for degree analysis + 1 for remaining details. These are not the only correct solutions. This is more detail than necessary for full credit.

[2] In fact, H contains *every* odd-degree vertex in G.

(b) If the endpoints of the path are *not* below every other vertex, Alice and Bob might still be able to meet, or they might not. Describe an algorithm to decide whether Alice and Bob can meet, without either breaking east-west eye contact or stepping off the path, given the arrays $X[0..n]$ and $Y[0..n]$ as input.

Solution: We construct the graph G described in our first solution to part (a) essentially by brute force. For each index i, there is at most one stop $(x, Y[i])$ on each segment of the path, so there are $O(n^2)$ stops altogether, and we can compute the set of stops in $O(n^2)$ time. We sort the stops by their x-coordinate in $O(n^2 \log n)$ time, so that we can find the (at most two) neighbors of each stop along the path in $O(1)$ time.

Next, we enumerate the vertex configurations $(x_A, x_B, Y[i])$ for each index i, by extracting the $O(n)$ stops $O(x, Y[i])$ by brute force, and then listing all pairs of those stops by brute force. Finally, for each of the $O(n^3)$ vertex configurations, we can find the (at most four) neighboring configurations in $O(1)$ time using the sorted list of stops. Altogether we need $\boldsymbol{O(n^3)}$ ***time*** to construct G.

Let s be the starting configuration, and let T be the set of configurations where Alice and Bob are together. We need to know whether there is a path in G from s to any vertex in T. To solve this problem, we mark every vertex reachable from s, using whatever-first search from s, and then check by brute force whether any vertex in T is marked. The resulting algorithm runs in $O(V + E) = \boldsymbol{O(n^3)}$ ***time***. ■

Rubric: 5 points = 2 for building the graph + 1 for the graph problem ("reachability") + 1 for the graph algorithm ("whatever-first search") + 1 for running time as a function of n.

+2 extra credit for $O(n^2)$ time, by considering only configurations where either Alice or Bob is on a corner of the path, as in the second solution to part (a). However, constructing this smaller graph in only $O(n^2)$ time requires a bit more care.

+10 extra credit for $O(n)$ time. Yes, there is a linear-time algorithm, which does *not* build a configuration graph. However, describing the algorithm is tricky, and proving the algorithm correct is even trickier.

2. **Are you comfortable with recursion?**

(a) Describe an algorithm to solve the Baguenaudier puzzle.

Solution: The solution consists of two mutually recursive algorithms.

- RingsOff(n) prints a reduced sequence of moves that changes the n-ring Baguenaudier puzzle from state $\mathtt{1}^n$ to state $\mathtt{0}^n$ (taking all n rings off).
- RingsOn(n) prints a reduced sequence of moves that changes the n-ring Baguenaudier puzzle from state $\mathtt{0}^n$ to state $\mathtt{1}^n$ (putting all n rings on).

```
⟨⟨Transform w1^n into w0^n⟩⟩
RingsOff(n):
  if n = 1
      print 1
  else if n > 1
      RingsOff(n − 2)
      print n
      RingsOn(n − 2)
      RingsOff(n − 1)
```

```
⟨⟨Transform w0^n into w1^n⟩⟩
RingsOn(n):
  if n = 1
      print 1
  else if n > 1
      RingsOn(n − 1)
      RingsOff(n − 2)
      print n
      RingsOn(n − 2)
```

Alternatively, RingsOn(n) could just print the output of RingsOff(n) in reverse order.

To show that this algorithm is correct, we prove by induction that for any non-negative integer n and any bitstring w, RingsOff(n) (prints a sequence of moves that) changes $w\mathtt{1}^n$ into $w\mathtt{0}^n$, and RingsOn(n) changes $w\mathtt{0}^n$ into $w\mathtt{1}^n$.

Let n be an arbitrary non-negative integer and let w be an arbitrary bit string. Assume for any non-negative integer $k < n$ and any bitstring x that RingsOff(k) changes $x\mathtt{1}^k$ into $x\mathtt{0}^k$, and that RingsOn(k) changes $x\mathtt{0}^k$ into $x\mathtt{1}^k$. There are three cases to consider:

- If $n = 0$, both algorithms correctly do nothing.
- If $n = 1$, both algorithms correctly toggle the 1st bit.
- Suppose $n > 2$. First we prove RingsOff(n) correct:
  - RingsOff($n-2$) changes $w\mathtt{1}^n$ into $w\mathtt{1}\mathtt{0}^{n-2}$, by the inductive hypothesis.
  - Toggling the nth bit changes $w\mathtt{1}\mathtt{0}^{n-2}$ into $w\mathtt{0}\mathtt{1}\mathtt{0}^{n-2}$.
  - RingsOn($n-2$) changes $w\mathtt{0}\mathtt{1}\mathtt{0}^{n-2}$ into $w\mathtt{0}\mathtt{1}^{n-1}$, by the inductive hypothesis.
  - RingsOf($n-1$) changes $w\mathtt{0}\mathtt{1}^{n-1}$ into $w\mathtt{0}^n$, by the inductive hypothesis.

A symmetric induction argument shows that RingsOff(n) is correct. ■

Solution (clever[3]): In part (c), we proved that the configuration graph G_n of the n-ring puzzle is a simple path through all 2^n possible configurations, with endpoints $\mathtt{0}^n$ and $\mathtt{1}\mathtt{0}^{n-1}$. We can solve the puzzle *backwards in time* by starting at $\mathtt{0}^n$ and moving along this path to $\mathtt{1}^n$. In other words, starting with all rings off, repeatedly make the only legal move that is not the reverse of the previous move, until all rings are on. Reversing the resulting sequence of moves is straightforward. ■

Rubric: 5 points. No time analysis is necessary because that's part (b). The clever solution only works if you actually proved in part (c) that the configuration graph is a path.

[3]This is not a complement.

(b) *Exactly* how many moves does your algorithm perform, as a function of n? Prove your answer is correct.

Solution (via Wikipedia): Let $T(n)$ be the number of moves performed by either RingsOn(n) or RingsOff(n). This function obeys the following recurrence:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ T(n-1) + 2T(n-2) + 1 & \text{otherwise} \end{cases}$$

Wikipedia[4] claims that

$$T(n) = \begin{cases} \dfrac{2^{n+1}-2}{3} & \text{if } n \text{ is even} \\ \dfrac{2^{n+1}-1}{3} & \text{if } n \text{ is odd} \end{cases}$$

Let's prove by induction on n that Wikipedia is correct.

Let n be an arbitrary non-negative integer. Assume for all non-negative integers $k < n$ that

$$T(k) = \begin{cases} \dfrac{2^{k+1}-2}{3} & \text{if } k \text{ is even} \\ \dfrac{2^{k+1}-1}{3} & \text{if } k \text{ is odd} \end{cases}$$

There are four cases to consider:

- If $n = 0$, then $T(n) = 0 = \frac{2^{0+1}-2}{3}$. ✓
- If $n = 1$, then $T(n) = 1 = \frac{2^{1+1}-1}{3}$. ✓
- Suppose $n \geq 2$ and n is even. Then

$$\begin{aligned} T(n) &= T(n-1) + 2T(n-2) + 1 \\ &= \frac{2^n - 1}{3} + 2 \cdot \frac{2^{n-1} - 2}{3} + 1 && \text{[induction hypothesis]} \\ &= \frac{2^{n+1} - 2}{3} \ \checkmark \end{aligned}$$

- Finally, suppose $n \geq 2$ and n is odd. Then

$$\begin{aligned} T(n) &= T(n-1) + 2T(n-2) + 1 \\ &= \frac{2^n - 2}{3} + 2 \cdot \frac{2^{n-1} - 1}{3} + 1 && \text{[induction hypothesis]} \\ &= \frac{2^{n+1} - 1}{3} \ \checkmark \end{aligned}$$

In all cases, Wikipedia's claimed solution is correct. ■

[4] https://en.wikipedia.org/wiki/Baguenaudier

Solution (via annihilators): Let $T(n)$ be the number of moves performed by either RingsOn(n) or RingsOff(n). This function obeys the following recurrence:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ T(n-1) + 2T(n-2) + 1 & \text{otherwise} \end{cases}$$

We solve this recurrence using the annihilator method, described in the recurrences lecture notes. The operator $(\mathbf{E}^2 - \mathbf{E} - 2)(\mathbf{E} - 1) = (\mathbf{E} - 2)(\mathbf{E} + 1)(\mathbf{E} - 1)$ annihilates this recurrence. Thus, the function $T(n)$ has closed form $T(n) = \alpha 2^n + \beta + \gamma(-1)^n$ for some constants α, β, and γ. Small cases give us a system of three linear equations:

$$\begin{aligned} T(0) &= 0 = \alpha + \beta + \gamma \\ T(1) &= 1 = 2\alpha + \beta - \gamma \\ T(2) &= 4 = 4\alpha + \beta + \gamma \end{aligned}$$

Solving this system gives us the constants $\alpha = 2/3$, $\beta = -1/2$, and $\gamma = -1/6$. We conclude that

$$T(n) = \frac{2}{3}2^n - \frac{1}{2} - \frac{1}{6}(-1)^n = \frac{2^{n+2} - 3 - (-1)^n}{6} = \begin{cases} \dfrac{2^{n+1} - 2}{3} & \text{if } n \text{ is even} \\ \dfrac{2^{n+1} - 1}{3} & \text{if } n \text{ is odd} \end{cases}$$

■

Rubric: 5 points = 2 for the closed-form solution + 3 for the proof. The closed forms

$$T(n) = \left\lfloor \frac{2^{n+1} - 1}{3} \right\rfloor = \left\lceil \frac{2^{n+2} - 2}{3} \right\rceil$$

are also correct.

(c) ***[Extra credit]*** Prove that for any non-negative integer n, there is *exactly one* reduced sequence of moves that solves the n-ring Baguenaudier puzzle.

Solution: Fix an arbitrary non-negative integer n. Let G_n be the graph whose vertices correspond to the 2^n configurations of the n-ring puzzle, and whose edges correspond to transitions. A reduced sequence of moves that solves the puzzle corresponds to a walk in G_n that never backtracks.

Every vertex in G_n has degree at least 1, because we can always flip the last bit (that is, move the last ring). On the other hand, every vertex in G_n has degree at most 2, because there are at most two legal moves from any configuration: Flip the rightmost bit, and flip the bit just before the rightmost `1`. In fact, exactly two vertices in G_n have degree 1:

- $\texttt{0}^n$ — because the rightmost `1` doesn't exist.
- $\texttt{10}^{n-1}$ — because there is no bit just before the rightmost `1`.

All other vertices in G_n have degree 2. Thus, every component of G_n is either a path or a cycle. Moreover, because $\texttt{0}^n$ and $\texttt{10}^{n-1}$ are the only vertices with degree 1, they must be the endpoints of the only path component of G_n; all other components of G_n are cycles.

The algorithm in part (b) implies that the starting configuration $\texttt{1}^n$ lies in the came component of G_n as the ending configuration $\texttt{0}^n$. But we just argued that this component is a simple path. So the only reduced walk from $\texttt{1}^n$ to $\texttt{0}^n$ lies on this path.

In fact, G_n is a simple path through all 2^n configurations of the puzzle, with endpoints $\texttt{0}^n$ and $\texttt{10}^{n-1}$. Consider the following pair of mutually recursive algorithms:

```
⟨⟨Transform w0^n into w10^(n-1)⟩⟩
OnlyLastRingOn(n):
  if n > 1
      OnlyLastRingOn(n − 1)
      print n
      OnlyLastRingOff(n − 1)
```

```
⟨⟨Transform w10^(n-1) into w0^n⟩⟩
OnlyLastRingOff(n):
  if n > 1
      OnlyLastRingOff(n − 1)
      print n
      OnlyLastRingOn(n − 1)
```

In fact, these algorithms print exactly the same sequence of moves, which is also the sequence of moves for solving the Tower of Hanoi problem! Each of these algorithms performs a reduced sequence of exactly $2^n - 1$ moves, and therefore traverses a path through every vertex of G_n. We conclude that G_n consists entirely of this path. ■

3. **Are you comfortable describing algorithms?**

(a) Describe an algorithm to sort an arbitrary stack of n pancakes using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

Solution:

```
⟨⟨Sort n pancakes⟩⟩
FLIPSORT(n):
  for i ← 1 to n − 2
      k ← position of the ith largest pancake
      flip the top k pancakes
      flip the top n − i + 1 pancakes
  if the top 2 pancakes are out of order
      flip the top 2 pancakes
```

For any non-negative integer i, the i largest pancakes are at the bottom of the stack in sorted order after i iterations of the main loop.

The algorithm performs $\sum_{i=1}^{n-2} 2 + 1 = \mathbf{2n - 3}$ flips in the worst case. ■

Rubric: 5 points: 3 for algorithm + 2 for exact analysis.

- Watch for "repeat this process" Deadly Sin!
- −½ for $2n \pm O(1)$ flips but more than $2n - 3$.
- −1 for $O(n)$ flips but more than $2n - 3$.
- +5 extra credit for a self-contained description, analysis, and justification of an algorithm that performs $\leq cn$ flips for some $c < 2$.
- A journal paper (and an A+ in the class) for an algorithm that performs $\leq cn$ flips for some $c < 18/11$.
- A PhD thesis (and an A+ in the class) for matching upper and lower bounds—both an algorithm A and a proof that no algorithm uses fewer flips in the worst case than A.

(b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of n pancakes, so that the burned side of every pancake is facing down, using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

Solution:

```
⟨⟨Sort n burned pancakes⟩⟩
BurnedFlipsort(n):
  for i ← 1 to n−1
      k ← position of the ith largest pancake
      flip the top k pancakes
      if the top pancake has its burned side down
          flip the top pancake
      flip the top n−i+1 pancakes
  if the top pancake has its burned side up
      flip the top pancake
```

For any non-negative integer i, the i largest pancakes are at the bottom of the stack, burned side down and in sorted order, after i iterations of the main loop.

The algorithm performs $\sum_{i=1}^{n-1} 3+1 = \mathbf{3n-2}$ flips in the worst case. ■

Rubric: 5 points: 3 for algorithm + 2 for exact analysis.

- Watch for "repeat this process" Deadly Sin!
- −½ for $3n \pm O(1)$ flips but more than $3n-2$.
- −1 for $O(n)$ flips but more than $3n-2$.
- +5 extra credit for a self-contained description, analysis, and justification of an algorithm that performs $\leq cn$ flips for some $c < 3$.
- A journal paper (and an A+ in the class) for an algorithm that performs $\leq cn$ flips for some $c < 3/2$.
- A PhD thesis (and an A+ in the class) for matching upper and lower bounds—both an algorithm A and a proof that no algorithm uses fewer flips in the worst case than A.