Solutions to Homework 1

Problems marked with **E** are graded on effort, which means that they are graded subjectively on the perceived effort you put into them, rather than on correctness. For bonus questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions. We strongly encourage you to typeset your solutions in LaTeX.

If you collaborated with someone, you must state their names. You must write your own solution and may not look at any other student's write-up.

0. If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

1. (a) Use the Euclidean algorithm to compute the GCDs of the following pairs of integers. State how many iterations each one takes to compute, and the value of the potential s_i at each stage. Verify that indeed $s_{i+1} \leq \frac{2}{3}s_i$. When stating how many iterations it takes to compute the GCD of a pair of integers, do not count any iteration in which only the order of the operands was swapped.

   You are encouraged to use a computer for this part.

   i. $(42, 165)$
   ii. $(388, 282)$
   iii. $(610, 377)$
   iv. $(3456, 4567)$

Solution:

i. $(165, 42)$

$$\begin{aligned}
\gcd(165, 42) &= \gcd(42, 39) & \frac{s_1}{s_0} &= \frac{42+39}{165+42} & &\approx 0.391 \leq \frac{2}{3} \\
&= \gcd(39, 3) & \frac{s_2}{s_1} &= \frac{39+3}{42+39} & &\approx 0.519 \leq \frac{2}{3} \\
&= \gcd(3, 0) & \frac{s_3}{s_2} &= \frac{3+0}{6+3} & &\approx 0.333 \leq \frac{2}{3} \\
&= 3
\end{aligned}$$

ii. $(388, 282)$

$$\begin{aligned}
\gcd(388, 282) &= \gcd(282, 106) & \frac{s_1}{s_0} &= \frac{203+173}{376+203} & &\approx 0.649 \leq \frac{2}{3} \\
&= \gcd(106, 70) & \frac{s_2}{s_1} &= \frac{173+30}{203+173} & &\approx 0.540 \leq \frac{2}{3} \\
&= \gcd(70, 36) & \frac{s_3}{s_2} &= \frac{30+23}{173+30} & &\approx 0.261 \leq \frac{2}{3} \\
&= \gcd(36, 34) & \frac{s_4}{s_3} &= \frac{23+7}{30+23} & &\approx 0.566 \leq \frac{2}{3} \\
&= \gcd(34, 2) & \frac{s_5}{s_4} &= \frac{7+2}{23+7} & &\approx 0.3 \leq \frac{2}{3}
\end{aligned}$$

$$
\begin{aligned}
&= \gcd(2,0) && \frac{s_6}{s_5} = \frac{2+1}{7+2} && \approx 0.333 \leq \frac{2}{3} \\
&= 2
\end{aligned}
$$

iii. $(610, 377)$

$$
\begin{aligned}
\gcd(610,377) &= \gcd(377,233) && \frac{s_1}{s_0} = \frac{377+233}{610+377} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(233,144) && \frac{s_2}{s_1} = \frac{233+144}{377+233} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(144,89) && \frac{s_3}{s_2} = \frac{144+89}{233+144} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(89,55) && \frac{s_4}{s_3} = \frac{89+55}{144+89} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(55,34) && \frac{s_5}{s_4} = \frac{55+34}{89+55} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(34,21) && \frac{s_6}{s_5} = \frac{34+21}{55+34} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(21,13) && \frac{s_7}{s_6} = \frac{21+13}{34+21} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(13,8) && \frac{s_8}{s_7} = \frac{13+8}{21+13} && \approx 0.618 \leq \frac{2}{3} \\
&= \gcd(8,5) && \frac{s_9}{s_8} = \frac{8+5}{13+8} && \approx 0.619 \leq \frac{2}{3} \\
&= \gcd(5,3) && \frac{s_{10}}{s_9} = \frac{5+3}{8+5} && \approx 0.615 \leq \frac{2}{3} \\
&= \gcd(3,2) && \frac{s_{11}}{s_{10}} = \frac{3+2}{5+3} && \approx 0.625 \leq \frac{2}{3} \\
&= \gcd(2,1) && \frac{s_{12}}{s_{11}} = \frac{2+1}{3+2} && \approx 0.6 \leq \frac{2}{3} \\
&= 1
\end{aligned}
$$

iv. $(3456, 4567)$

$$
\begin{aligned}
\gcd(3456,4567) &= \gcd(4567,3456) \\
&= \gcd(3456,1111) && \frac{s_1}{s_0} = \frac{3456+1111}{4567+3456} && \approx 0.596 \leq \frac{2}{3} \\
&= \gcd(1111,123) && \frac{s_2}{s_1} = \frac{1111+123}{3456+1111} && \approx 0.270 \leq \frac{2}{3} \\
&= \gcd(123,4) && \frac{s_3}{s_2} = \frac{123+4}{1111+123} && \approx 0.103 \leq \frac{2}{3} \\
&= \gcd(4,3) && \frac{s_4}{s_3} = \frac{4+3}{123+4} && \approx 0.055 \leq \frac{2}{3} \\
&= \gcd(3,1) && \frac{s_5}{s_4} = \frac{3+1}{4+3} && \approx 0.571 \leq \frac{2}{3} \\
&= 1
\end{aligned}
$$

E (b) Try to find pairs of inputs (x, y) such that the number of iterations of Euclid(x, y) is "large", that is, as close as possible to the upper bound of $\log_{3/2}(x + y)$ that we derived in lecture. Can you come up with a hypothesis about what kinds of inputs yield the worst-case running time?

Hint 1: Recall the inequalities used when showing the potential function for the Euclidean algorithm. To minimize the difference between the potential function on steps i and $i + 1$, what should q_i be?

Hint 2: Question 1(a)iii is a worst-case example. Can we apply our choice of q_i to derive a relationship between inputs in worst-case examples?

Solution: We can begin by finding pairs of (x, y) such that the pair "works against" the method of Euclidean algorithm. Recall that in class, we defined the potential $s_i = x_i + y_i$ and proved that $s_i \geq 1.5 \cdot s_{i+1}$ by showing:

$$s_i = x_i + y_i = q_i y_i + r_i + y_i \geq 2y_i + r_i \geq 2y_i + r_i - (y_i - r_i)/2 = 1.5 \cdot s_{i+1}.$$

To make s_i as close as possible with $1.5 \cdot s_{i+1}$, we set $q_i = 1$ for each i.
If we require that the last step is

$$x_N = 2, \quad y_N = 1,$$

then the previous step must be

$$y_{N-1} = x_N = 2, \quad x_{N-1} = 1 \cdot y_{N-1} + y_N = 3,$$

and yet previously the step is

$$y_{N-2} = x_{N-1} = 3, \quad x_{N-2} = 1 \cdot y_{N-2} + y_{N-1} = 5,$$

so on and so forth. Following this procedure, we obtain that x_i and y_i are precisely the Fibonacci sequences. Therefore, we can determine with proof that pairs of the Fibonacci sequences yield the worst-case running time.

2. Consider the recurrence

$$T(n) = 3\sqrt[3]{n} \cdot T(n^{\frac{2}{3}}) + O(n).$$

(a) Define $S(n) = T(n)/n$. Using this substitution, write a simplified recurrence for $S(n)$.

(b) Now define $R(n) = S(2^n)$. Using this substitution, write a simplified recurrence for $R(n)$.

(c) Use the Master Theorem and the above recurrence to find the asymptotic complexity (Big-O approximation) of $R(n)$, then use this expression to find the asymptotic complexities of $S(n)$ and finally $T(n)$.

Solution:

(a)

$$S(n) = \frac{T(n)}{n}$$

Expanding the definition of $T(n)$, we have

$$= \frac{3\sqrt[3]{n} \cdot T(n^{\frac{2}{3}}) + O(n)}{n}$$

Simplifying, we obtain

$$= \frac{3T(n^{\frac{2}{3}})}{n^{\frac{2}{3}}} + O(1)$$

Note that $S(n) = T(n)/n$. In terms of $n^{\frac{2}{3}}$: $S(n^{\frac{2}{3}}) = T(n^{\frac{2}{3}})/n^{\frac{2}{3}}$, and thus

$$S(n) = 3S(n^{\frac{2}{3}}) + O(1)$$

(b)

$$\begin{aligned} R(n) &= S(2^n) \\ &= 3S\left((2^n)^{\frac{2}{3}}\right) + O(1) \\ &= 3S(2^{\frac{2n}{3}}) + O(1) \\ &= 3R\left(\frac{2n}{3}\right) + O(1) \end{aligned}$$

(c)

$$R(n) = 3R\left(\frac{2n}{3}\right) + O(1).$$

By the Master Theorem,

$$R(n) = O(n^{\log_{1.5} 3}).$$

Now we use the fact that $R(n) = O(n^{\log_{1.5} 3})$ and $R(n) = S(2^n)$ to get an asymptotic bound on $S(n)$.

$$S(2^n) = O(n^{\log_{1.5} 3})$$

We also have that

$$S(2^m) = O(m^{\log_{1.5} 3}).$$

When we let $m = \log n$,

$$S(2^{\log n}) = O((\log n)^{\log_{1.5} 3}).$$

Since $2^{\log n} = n$, we have that:

$$S(n) = O((\log n)^{\log_{1.5} 3}).$$

Now we use the fact that $S(n) = O((\log n)^{\log_{1.5} 3})$ and $S(n) = T(n)/n$ to derive an asymptotic bound for $T(n)$.

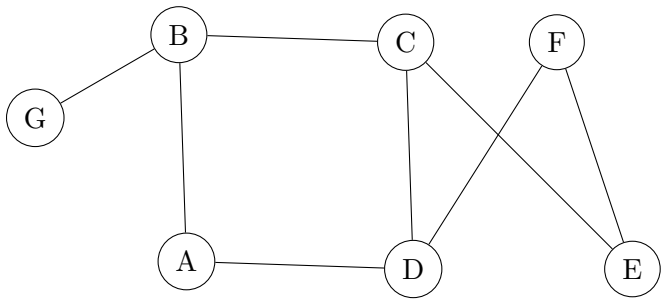
$$\begin{aligned} T(n)/n &= O((\log n)^{\log_{1.5} 3}), \quad \text{so} \\ T(n) &= O(n(\log n)^{\log_{1.5} 3}). \end{aligned}$$

E 3. (a) Determine if the following algorithm terminates. If so, prove this using the potential method as shown in lecture. If not, describe an input that would cause this algorithm not to terminate.

```
Input: A connected graph G
1: function ALGORITHM(G)
2:     Assign every vertex in G the color black
3:     Let S be an empty stack
4:     Choose an arbitrary vertex v ∈ G and assign it the color red
5:     PUSH(S, v)   // Push v onto the stack
6:     while S is not empty do
7:         u ← POP(S)   // Pop a node off the stack and assign it to u
8:         for each neighbor t of u do
9:             if t is colored black then
10:                if u is colored red then
11:                    Assign t the color blue
12:                else
13:                    Assign t the color red
14:                PUSH(S, t)
15:            else if t and u have the same color then
16:                return False
17:    return True
```

Note: If you need a refresher on what stacks are, click here: `https://en.wikibooks.org/wiki/Data_Structures/Stacks_and_Queues`.

(b) Run the algorithm on the following graph and compute the value of the potential function you generated for each iteration of the line 6 (the while loop). For line 4, choose A to be the arbitrary vertex. For line 8, start from the lexicographically smallest node if there is a choice of at least two nodes.



(c) *Food for thought.* The above algorithm determines (or attempts to determine, depending on what you concluded in (a)) if its input graph G has a particular property. What is this property? Note that there are no points awarded for this question, but it is something to think about. Try running the algorithm on many different graphs and see what you get.

Solution:

(a) This algorithm does indeed terminate on all inputs. Intuitively, each vertex starts out colored black and once a vertex is assigned either red or blue, it never becomes black again. Since we only push nodes that started off black (before being assigned either red or blue) onto the stack, each node can only be pushed onto the stack exactly once and hence there can not be more iterations of the while loop than there are vertices in the graph.

More formally, let b_i be the number of black-colored nodes on the ith iteration of the while loop, let s_i be the number of nodes in the stack on the ith iteration, and let our potential function $f(i) = s_i + b_i$. In order to show that the algorithm terminates, we need to show that our potential function f is bounded from below and strictly decreasing as i (the number of iterations) increases. $f(i)$ is bounded from below by 0 since $f(i) = 0$ implies that $s_i = 0$ i.e., that the size of the stack is empty, which is the condition that causes the while loop to terminate. Now we show that $f(i)$ is strictly decreasing i.e. that $f(i+1)$ is strictly less than $f(i)$. Notice that on each iteration we pop a node from the stack, and we only push a node onto the stack if it was originally black and then we assigned it a different color. Thus, if we let p_i be the number of nodes pushed onto the stack during iteration i, we see that $s_{i+1} = s_i - 1 + p_i$ (i.e. the number of nodes in the stack on iteration $i+1$ is the number of nodes there were on the last iteration, minus the one we popped, plus all the ones we pushed) and $b_{i+1} = b_i - p_i$ (i.e. the number of black nodes in the graph on iteration $i+1$ is the number of black nodes there were on iteration i minus all the ones we colored red or blue and pushed onto the stack). From here, we see that:

$$\begin{aligned} f(i+1) &= b_{i+1} + s_{i+1} \\ &= (b_i - p_i) + (s_i - 1 + p_i) \\ &= b_i + s_i - 1 \\ &= f(i) - 1 \end{aligned}$$

and hence, f decreases by one on every iteration and is thus strictly decreasing.

(b) Initially, all vertices in G are black and the stack S is empty.

Iteration	Black Nodes	Red Nodes	Blue Nodes	Stack	Potential
Initial	B,C,D,E,F,G	A	—	A	7
1	C,E,F,G	A	B,D	B,D	6
2	E,G	A,C,F.	B,D	B,C,F	5
3	G	A,C,F	B,D,E	B,C,E	4
4	G	A,C,F	B,D,E	B,C	3
5	G	A,C,F	B,D,E	B	2
6	—	A,C,F,G	B,D,E	G	1
7	—	A,C,F,G	B,D,E	—	0

(c) This algorithm tests if G is bipartite (other answers saying that the algorithm tests that G has no odd-length cycles or that G is two-colorable are also correct as these

are equivalent statements). Notice that the algorithm colors the neighbors of every red node blue and every blue node red. The result is that (if the graph is indeed bipartite), we end up with a graph where the bipartite components are colored red and blue respectively.

4. Suppose *input* is a function which returns a user-specified positive integer. For each of the following problems, do the following tasks:
   - Determine if the following program halts for all possible sequences of (valid) inputs.
   - Either provide a proof of termination for all possible sequences of (valid) inputs or provide a sequence which causes the program to loop.
   - If the program does not halt, explain why the potential method does not apply.

(a)

```
1: x ← input()
2: y ← input()
3: while x > 0 and y > 0 do
4:     z ← input()
5:     if z is even then
6:         x ← x − 1
7:         y ← y + 1
8:     else
9:         y ← y − 1
```

Solution: We claim that the value of $2x + y$ serves as a potential for this program.

Claim: $2x + y$ has a lower bound of 0.

The program terminates when either $x \leq 0$ or $y \leq 0$. Suppose both $x \leq 0$ and $y \leq 0$. Then $2x + y \leq 2(0) + 0 = 0$. So if $2x + y \leq 0$, the program terminates.

Claim: $2x + y$ is strictly decreasing.

Suppose z is even. Then the value of the potential changes to $2(x - 1) + y + 1 = 2x + y - 1 < 2x + y$.

Suppose z is odd. Then the value of the potential changes to $2x + y - 1 < 2x + y$.

In either case, the value of the potential function is bounded from below by 0 and decreases by 1 at every iteration, so the program must terminate.

(b)

```
1: x ← input()
2: y ← input()
3: while x > 0 and y > 0 do
4:     z ← input()
5:     if z is even then
6:         x ← x − 1
7:         y ← y + 1
8:     else
9:         y ← y − 1
10:        x ← x + 1   // This line differs from the program in (a)
```

Solution: The sequence $x \leftarrow 2$, $y \leftarrow 2$, and $z_i \leftarrow i$ (where z_i denotes the i-th input for z) causes the program to loop.

Observe that if the potential method did apply, we could use the function specified by the potential method to show that the program halts for all possible sequences

of valid inputs.

For example, if we take the potential function to be $2x+y$ as with the program from part (a), we can still see that the potential has a lower bound of 0.

However, the value of the potential function is not strictly decreasing over all sequences of inputs. Consider the sequence $x \leftarrow 2$, $y \leftarrow 2$, and $z_i \leftarrow i$ (where z_i denotes the i-th input for z). Then on the first iteration the value of the potential changes from $6 = 2(2) + 2$ to $7 = 2(3) + 1$.

In general for any potential function, the value is the same after any pair of iterations where z is even in one and odd in the other, since the values of x and y are the same before and after those two iterations. Thus, the value does not strictly decrease for any potential function.

(c)

```
1: x ← input()
2: y ← input()
3: while x > 0 or y > 0 do   // This line differs from the program in (a)
4:     z ← input()
5:     if z is even then
6:         x ← x − 1
7:         y ← y + 1
8:     else
9:         y ← y − 1
```

Solution: The sequence $x \leftarrow 1$, $y \leftarrow 1$, and $z_i \leftarrow 1$ for all i causes the program to loop.

Observe that if the potential method did apply, we could use the function specified by the potential method to show that the program halts for all possible sequences of valid inputs.

For example, if we take the potential function to be $2x+y$ as with the program from part (a), by the analysis from the first part of the problem, the value of the potential function is strictly decreasing over all sequences of inputs.

However, the value of the potential function is not bounded from below. In other words, there is an assignment of x and y such that $2x + y$ is arbitrarily small, but at least one of x and y is greater than 0. Regardless of the choice of z the value of $2x+y$ decreases by 1 every iteration. However, if we choose only z even, then y will always be greater than 0 so the loop condition always holds true.

More generally, any valid potential function must decrease in each iteration. For this to hold in the input sequence above, the value of the potential function must decrease when x is constant but the value of y decreases. However, this algorithm has no lower bound on the value of y when x is held constant at a positive value, so there is no lower bound on the value of any such potential function either. Thus, it does not meet the requirements of a valid potential function.

5. Before resuming in-person activities, the University of Michigan has made changes to protocols regarding room usage. Specifically, before a room is to be used for any event which might have multiple in-person attendees, building facility staff is to verify that the configuration of the room supports social distancing. One condition which a room must satisfy before it can be used for an event is that there are no pairs of seats for which the distance between them is strictly less than 6ft.

   Suppose that the seats are sorted by their x-coordinate and y-coordinate.

   (a) Prove that the number of seats which are within 6ft of a given seat *assuming the configuration of the room supports social distancing* is bounded from above by some constant.
   Hint: Consider the following diagram. What can be said about the number of chairs in each of the four square sub-areas?

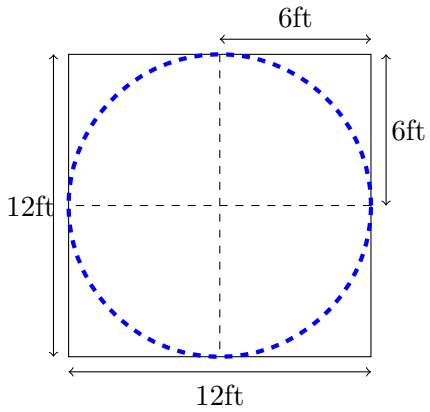


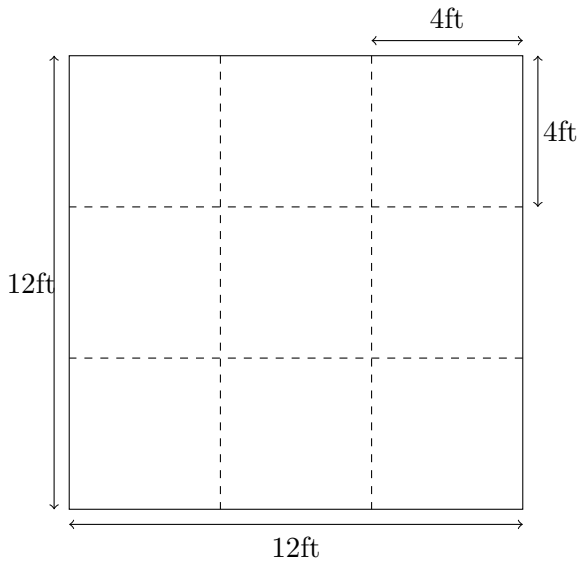
Figure 1: Hint

Solution: For simplicity, we consider a 12ft by 12ft square with the given seat at the center, call it the "large square". If there exists an upper bound on the number of seats in the large square, that bound also works for the 6ft radius circle around the given seat as it is a sub-area of the large square.

We divide the large square into four congruent square sub-areas with side length 6ft, call them "medium squares". Consider a medium square. Let's further divide this medium square up into four even smaller square sub-areas, call them "little squares". The side length of of a little square is 3ft, so the farthest any two points can be in a little square is $\sqrt{3^2 + 3^2} < 6$. We know that the room follows social distancing guidelines, meaning the distance between any two points is at least 6ft. Therefore, at most one point can be located in any little square. Therefore, at most four points can be located inside of a medium square, and therefore a large square can fit at most 16 points. That bound also applies to the circle of radius 6 ft.

So, there is an upper bound of sixteen points.

Alternatively, we could consider dividing the 12×12 square into sub-squares of dimension 4×4.



In each of the 4×4 squares, we can observe that the farthest two points within the square may be are on the corners. By the Pythagorean theorem, we find that $4^2 + 4^2 = 32 < 6^2$, so the distance between the points on the diagonal is no greater than 6ft. If two chairs are placed in the same square, then we know that the room configuration does not support social distancing. By the Pigeonhole Principle, we know that we can place at most 9 chairs before a square has 2 chairs inside it. This allows us to derive a tight upper bound of 9 chairs in a 12ft by 12ft square.

(b) Conclude that, *for any room configuration*, staff can determine if there are seats in the room that are closer than 6ft to each other using $O(n)$ measurements where n is the total number of seats.

Solution:

If only a constant number of points needs to be checked for every point in the room, then and only then will it take $O(n)$ measurements to determine whether the n seats are appropriately spaced.

The points are pre-sorted by x and y coordinates. Therefore for every point (x, y) in the room, we can check for the first 16 points found between lines $x - 6$, $x + 6$, $y - 6$ and $y + 6$ whether they are too close to the point (x, y). (We can filter the points that are outside of this area without doing any measurements.) If any of the points in that space are too close to (x, y), or if there are more than 16 points in that space (by part (a)), then the room does not follow social distancing guidelines.

We showed that a constant number of operations is sufficient for each seat in order to determine whether the room follows social distancing guidelines, therefore the number of measurements necessary is linear in the number of seats in the room.

Optional bonus questions

For bonus questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions. *Only attempt these questions **after** you have finished the rest of the homework.*

1. This question explores the optimality of the 2/3 constant we saw in class in the analysis of Euclid's algorithm.

   Let $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ be a solution to $\varphi^2 = \varphi + 1$. This value is known as the *Golden Ratio*. Show that the number of iterations (not including the base case) made by Euclid's algorithm on (x, y) is at most $\log_\varphi(x + y)$. Note that this is a slight improvement over the bound of $\log_{3/2}(x + y)$ that was shown in class.

   Submit your solution via Gradescope.

   Solution: In class, we declare a potential function $s(x_i, y_i) = x_i + y_i$ and determine $i < log_{1.5}(x + y)$. To find a better bound, we can write a slightly different potential function: $s(x, y) = cx + y$. We claim that $c = \varphi$ works. Slightly adapting the proof from class, we have:

   $$\begin{aligned} s(x_i, y_i) = \varphi x_i + y_i &= \varphi(qy_i + r) + y_i \\ &\geq \varphi(y_i + r) + y_i && \text{since } q \geq 1 \\ &= (\varphi + 1)y_i + \varphi r \\ &= \varphi^2 y_i + \varphi r && \text{since } \varphi^2 = \varphi + 1 \\ &= \varphi(\varphi y_i + r) \\ &= \varphi(\varphi x_{i+1} + y_{i+1}) \\ &= \varphi \cdot s(x_{i+1}, y_{i+1}) \end{aligned}$$

   The potential of the final (base-case) call, where $x > y = 0$, is $\varphi x \geq \varphi$. So, the algorithm runs for at most $\log_\varphi((\varphi x + y)/\varphi) \leq \log_\varphi(x + y)$ iterations, as desired.

   It seems like the choice $c = \varphi$ came out of thin air, so let's see how we arrived at it. For potential function $cx + y$, we want to prove that

   $$cx_i + y_i \geq \varphi(cx_{i+1} + y_{i+1}) = \varphi(cy_i + r).$$

   Using similar steps as the proof from class, a sufficient condition for this is

   $$(c + 1)y_i + cr \geq \varphi c y_i + \varphi r.$$

   The above holds if we satisfy two separate inequalities, $c + 1 \geq \varphi c$ and $c \geq \varphi$. The first inequality is equivalent to $c \leq 1/(\varphi - 1)$. From the fact that $\varphi(\varphi - 1) = \varphi^2 - \varphi = 1$, we have $\frac{1}{\varphi - 1} = \varphi$, so there is exactly one way to satisfy the two inequalities simultaneously: $c = \varphi$. For more details, see `https://arxiv.org/abs/1808.07957`.

2. Compute a tight upper bound on the number of moves in the flipping game described in lecture. That is, find a positive integer t_n such that:

- For all possible starting configurations, and for all possible sequences of legal moves, the flipping game on the $n \times n$ board ends in at most t_n moves.
- There is some starting configuration, and some sequence of legal moves, such that the flipping game on the $n \times n$ board ends in exactly t_n moves.

In order to be awarded bonus points, your answer **must** be supplemented with a formal proof.

Note that we only defined the flipping game for boards with odd dimensions, so n should be odd (though we could attempt to extend to n being even by specifying what are the rules when the number of OSU and UMich chips in a column or row are the same).

Submit your solution via e-mail directly to Professor Volkovich at ilyavol@umich.edu.