CS 124 Homework 6: Spring 2024

Anusha Murali

Collaborators: None

No. of late days used on previous psets: 5
No. of late days used after including this pset: 5

Homework is due Wednesday Apr 10 at 11:59pm ET. Please remember to select pages when you submit on Gradescope. Each problem with incorrectly selected pages will lose 5 points.

Try to make your answers as clear and concise as possible; style may count in your grades. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan your papers to turn them in.

Collaboration Policy: You may collaborate on this (and all problem sets) only with other students currently enrolled in the class, and of course you may talk to the Teaching Staff or use Ed. You may also consult the recommended books for the course and course notes linked from the timetable. You may not use generative AI or large language models, or search the web for solutions, or post the questions on chat forums. Furthermore, you must follow the "one-hour rule" on collaboration. You may not write anything that you will submit within one hour of collaborating with other students or using notes from such sources. That is, whatever you submit must first have been in your head alone, or notes produced by you alone, for an hour. Subject to that, you can collaborate with other students, e.g. in brainstorming and thinking through approaches to problem-solving.

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential-time algorithms (unless specifically asked for) will receive no or little credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of pseudocode will receive little or no credit. Again, try to make your answers clear and concise.

Problems

1. (a) **(10 points)** Prove that 12403180369 is composite by finding a witness in the form of a positive integer $a < 12403180369$ such that $a^{12403180369-1} \not\equiv 1 \pmod{12403180369}$. Give any information necessary to show that your witness in fact witnesses. (Note: do not use a factor as a witness! Sure, 12403180369 is small enough that you can exhaustively find a factor, but this question is about your knowledge of the primality-testing algorithm.)

   Solution: We shall use Fermat's little theorem to show that 12403180369 is composite. Fermat's little theorem states that for a prime number p and a co-prime integer a (i.e: $(a, p) = 1$), the following equation holds:

$$a^{p-1} \equiv 1 \pmod{p}$$

   When the above equation doesn't hold, we know for certain that p is not a prime and the value of a, for which the above equation failed is known as a Fermat witness for the compositeness of p. However, $a^{p-1} \equiv 1 \pmod{p}$ does not necessarily guarantee that p is a prime as the equation holds for certain composite numbers (such as Carmichael numbers).

I wrote a small Python function to evaluate $a^{p-1} \equiv 1 \pmod{p}$ for a randomly selected a, where $(a, p) = 1$.

For $a = 41$, I obtained the following results:

$$\begin{aligned} a^{n-1} \pmod{n} &= 41^{12403180369-1} \pmod{12403180369} \\ &= 41^{12403180368} \pmod{12403180369} \\ &= 8674949151 \neq 1. \end{aligned}$$

Therefore, using Fermat's little theorem, since $a = 41$ is relatively prime to $n = 12403180369$, we have proved that n =12403180369 is composite, and the witness for its compositeness is $a = 41$. The witness, witnesses 8674949151 (and not 1) as the value of $a^{n-1} \pmod{n}$ = $41^{12403180368} \pmod{12403180369}$. □

The code for finding whether a given n is prime using Fermat's little theorem is shown below:

```
def fermatPrime(n):
    a = random.randrange(3, n)
    while (n%a == 0):                        # We want (a, n) = 1
        a = random.randrange(3, n)
    if (modPower(a, n-1, n) == 1):
        return True
    print("Fermat witness for compositeness is = ",a)
    return False
```

The following function, `modPower(u, v, p)` (used above) computes $u^v \pmod{p}$ efficiently.

```
def modPower(u, v, p):
    answer = 1
    u = u % p
    while (v > 0):
        if (v & 1):                          # If we is odd
            answer = (answer *u)%p;
        v = v >> 1                           # Integer division of v by 2
        u = (u * u) % p
    return answer
```

(b) **(10 points)** The number 63973 is a Carmichael number. Prove that it is composite by finding a witness in the form of a nontrivial square root of 1.

Solution: Since $n = 63973$ is a Carmichael number, we cannot find its compositeness using Fermat's little theorem.
We will use Rabin's testing to find its compositeness. Rabin testing says that if n is a prime, the only solutions to $a^2 \equiv 1 \pmod{n}$ are $a \in \{\pm 1\}$.
Following is a brief description of Rabin primality test.

Rabin Primality Test

Since n is odd, $n-1$ is even. Therefore, we can represent $n-1$ as $n-1 = 2^i \cdot u$, where u is odd. Suppose we choose a random base a and compute a^{n-1} by first computing a^u and then repeatedly squaring. Along the way, we will check to see for the values, $a^u, a^{2u}, a^{4u}, \ldots$ and verify whether they have the following property:

$$\begin{aligned} a^{2^{i-1}u} &\not\equiv \pm 1 \pmod{n} \\ a^{2^i u} &\equiv 1 \pmod{n}. \end{aligned}$$

If we see the above property, we have encountered a non-trivial square root of $1 \pmod{n}$. Rabin primality test, presented in Theorem 2 of Lecture 17, says that only composite numbers have non-trivial square roots of $1 \pmod{n}$. For a random $2 \le a < n-1$ (a is relatively prime to n), we check whether $a^u \equiv 1 \pmod{n}$ or $a^{2^j \cdot u} \equiv 1 \pmod{n}, 0 \le j \le i-1$. If we found an a which doesn't satisfy these properties, then we have found a witness for the compositeness of n, so n is not a prime number.

Following is a description of how Rabin primality test finds the compositeness of the given n.

Let $n = 63973$. Hence $n-1 = 63972 = 2^2 \cdot 15993$. Let $a = 3$ (3 is relatively prime to n) and we run Rabin Testing for primality as follows:

(1) $a^{2^2 \cdot 15993} = 1 \pmod{n}$
(2) $a^{2^1 \cdot 15993} = 1 \pmod{n}$
(3) $a^{15993} = 19683 \pmod{n}$

Since $a^{2^{i-1} \cdot u} \neq \pm 1 \pmod{n}$ and $a^{2^i \cdot u} = 1 \pmod{n}$, where $a = 3, i = 1, u = 15993$, we have found a non-trivial square root of 1 modulo n. In our case, the non-trivial square root of 1 modulo n is $3^{15993} \pmod{63973} = 19683$. Therefore, 63973 is a composite number and $a = 3$ is the witness for the compositeness of 63973. □

(You can find the answers by whatever method you like, but we recommend writing code. You don't need to submit your code.)

2. Consider the task of generating a uniformly random number from 1 to N together with its prime factorization. One naive approach would be to simply generate a number from 1 to N and then factor it, but this would not be computationally efficient. In this problem we will explore an alternative approach. The algorithm is as follows:

Algorithm 1

```
 1: Input: $N$
 2: $m \leftarrow N$
 3: $i \leftarrow 1$
 4: while $m > 1$ do
 5:     Let $s_i$ be a uniformly random number from 1 to $m$
 6:     $m \leftarrow s_i$
 7:     $i \leftarrow i + 1$
 8: end while
 9: Remove all the non-prime $s_i$'s and let $r$ be the product of all prime $s_i$'s generated above.
10: if $r \le N$ then
11:     With probability $r/N$, return $r$ together with the list of all prime $s_i$'s generated above. Otherwise,
        restart the algorithm (i.e. go back to line 2).
12: end if
```

Algorithm 1 runs in expected polynomial time in the number of bits needed to describe N (you need not prove this, but in particular line 9 can be implemented in polynomial time, and optional part (e) below shows why the number of iterations of the whole algorithm till it outputs something is also polynomial). Your goal is to prove that the algorithm outputs a random number in $\{1, \ldots, N\}$.

(a) (**5 points**) Prove that the probability that exactly t of the s_i's are equal to N is given by $\frac{1}{N^t}(1 - 1/N)$.

Solution: The probability that the value N occurs on line number 5 on the very first iteration of the **while** loop is $\frac{1}{N}$ as the numbers from 1 to N are generated uniformly and randomly.
If N appears on the very first iteration of the **while** loop, since $m = N$, on line number 5, the number N can be generated again on the second iteration of the **while** loop with the probability $\frac{1}{N}$. Assume that this occurs consecutively for t iterations of the **while** loop, where each time the number N is generated. Since each occurrence is independent, the overall probability that N appears on t consecutive iterations of the **while** loop is $\frac{1}{N^t}$.
For N to be generated exactly t times, the next (i.e: $(t+1)$-st) iteration of the while loop should generate a number other than N, which occurs with the probability $\left(1 - \frac{1}{N}\right)$. Since this number is less than N, the **while** loop will never generate N again. Therefore, the overall probability that exactly t of the s_i's are equal to N is,

$$\frac{1}{N^t}\left(1 - \frac{1}{N}\right).$$

□

(b) (**10 points**) Let $t_2, \dots, t_N$ be any nonnegative integers. Generalize your proof in the previous part to derive an expression for the probability that exactly t_a of the s_i's are equal to a for all $a \in \{2, \dots, N\}$. (In particular you should get a very simple expression for the probability that there is at least one i such that $s_i = a$.)

Solution: From the results from Part (a), inductively, we find that the probability that exactly t_a of the s_i's are equal to a for an arbitrary a, $a \in \{2, \dots, N\}$ is

$$\frac{1}{a^{t_a}}\left(1 - \frac{1}{a}\right).$$

For the base case, when $a = 2$, we find the probability that exactly t_2 of the s_i's are equal to 2 is $\frac{1}{2^{t_2}}\left(1 - \frac{1}{2}\right) = \frac{1}{2 \cdot 2^{t_2}}$.
For an arbitrary n, when selected uniformly and randomly, we have proved in Part (a) that n occurs with probability $\frac{1}{n^{t_n}}\left(1 - \frac{1}{n}\right)$.
Since the integers $\{2, \dots, N\}$ are generated randomly and uniformly, the occurrences of each s_i is independent. Therefore, the probability that exactly t_a of the s_i's are equal to a for all $a \in \{2, \dots, N\}$ is equal to,

$$\frac{1}{2^{t_2}}\left(1 - \frac{1}{2}\right) \cdot \frac{1}{3^{t_3}}\left(1 - \frac{1}{3}\right) \cdots \frac{1}{(N-1)^{t_{N-1}}}\left(1 - \frac{1}{N-1}\right) \cdot \frac{1}{N^{t_N}}\left(1 - \frac{1}{N}\right) = \prod_{2 \le a \le N}\left(\frac{1}{a^{t_a}}\right)\left(1 - \frac{1}{a}\right).$$

$\square$

(c) (**10 points**) Let r be a particular number with prime factorization $r = 2^{t_2} \cdot 3^{t_3} \cdots p^{t_p}$. Prove that the probability that the r computed in line 9 of the algorithm equals this particular choice of r is given by

$$\frac{1}{r} \cdot \prod_{2 \le p \le N \text{ prime}}\left(1 - \frac{1}{p}\right) \tag{1}$$

Solution: From the results of Part (b), we know that the probability that exactly t_a of the s_i's are equal to a for all $a \in \{2, \dots, N\}$ is,

$$\prod_{2 \le a \le N}\left(\frac{1}{a^{t_a}}\right)\left(1 - \frac{1}{a}\right).$$

Since, we have removed all the non-prime s_i's in line number 9, the probability where each of the s_i's are prime numbers is,

$$\prod_{2 \le p \le N \text{ prime}}\left(\frac{1}{p^{t_p}}\right)\left(1 - \frac{1}{p}\right).$$

Therefore, the probability that the r computed in line 9 of the algorithm equals a particular choice r with prime factorization $r = 2^{t_2} \cdot 3^{t_3} \cdots p^{t_p}$ is,

$$\begin{aligned} Pr\left(r = 2^{t_2} \cdot 3^{t_3} \cdots p^{t_p}\right) &= \prod_{2 \le p \le N \text{ prime}}\left(\frac{1}{p^{t_p}}\right)\left(1 - \frac{1}{p}\right) \\ &= \left(\frac{1}{2^{t_2} \cdot 3^{t_3} \cdots p^{t_p}}\right) \prod_{2 \le p \le N \text{ prime}}\left(1 - \frac{1}{p}\right) \\ &= \left(\frac{1}{r}\right) \cdot \prod_{2 \le p \le N \text{ prime}}\left(1 - \frac{1}{p}\right). \quad \square \end{aligned}$$

(d) (**5 points**) Conclude that if this algorithm returns something, it must be a uniformly random number from 1 to N together with the list of its prime factors.

Solution: We note from line 11, with probability r/N, the above algorithm returns r together with the list of all prime s_i's generated. Otherwise, it restarts the algorithm and goes back to line 2.

Hence, the probability the algorithm returning $r \leq N$ is,

$$\begin{aligned} P &= \left(\frac{r}{N}\right)\left(\frac{1}{r}\right) \cdot \prod_{2 \leq p \leq N \text{ prime}} \left(1 - \frac{1}{p}\right) \\ &= \left(\frac{1}{N}\right) \cdot \prod_{2 \leq p \leq N \text{ prime}} \left(1 - \frac{1}{p}\right). \end{aligned}$$

Therefore, we find that the probability of returning an r is the same for all $r \leq N$, irrespective of the value of r. When we don't output an r, we restart the algorithm. Hence all $r \leq N$ returned by the above algorithm are uniformly random numbers from 1 to N together with the list of its prime factors. □

(e) (**0 points, optional**)[1] Prove that the expected number of times the algorithm restarts is $O(\log n)$. You may find Mertens' third theorem useful.

[1]We won't grade this problem, but may use it for recommendations/TF hiring.

3. Given an undirected graph G, a *nice* 2*-coloring* is an assignment of colors from $\{\text{red}, \text{blue}\}$ to each of the vertices such that for any triangle in the graph (i.e. three vertices which all have edges between each other), the vertices in the triangle are not all assigned the same color.

   Let G be a graph for which there exists a *proper* 3*-coloring*, that is, an assignment of colors from $\{\text{red}, \text{blue}, \text{green}\}$ to each of the vertices such that no two vertices connected by an edge have the same color.

   (a) (**10 points**) Prove that G has a nice 2-coloring. (In fact G has many nice 2-colorings. You don't have to prove this, but understanding why this is so might help with the next part.)

   Solution: Consider a graph G with *proper* 3*-coloring*. Let us partition the vertices of G into three parts, namely V_R, V_B and V_G, each corresponding to vertices colored using red, blue and green respectively. Therefore, G, is a tripartite graph with respect to the three colors, which means there are no intra-edges between the vertices belonging to the same partition (color).
   Now, WLOG, consider any arbitrary coloring of G using only red and blue colors. Let us say that all the vertices in V_R of the above tripartite graph are assigned red color and all the vertices in V_B are assigned blue color. This color assignment makes any triangle in G to have at least one vertex of red color (the vertex that belongs to V_R), and at least one vertex of blue color (the vertex that belongs to V_B). Hence, there will be no monochromatic triangles after this color assignment. Therefore, given G has a *proper* 3*-coloring*, G has a nice 2-coloring. $\square$

   Alternate solution: We shall prove this using contradiction, so let us claim that G does not have a *nice* 2*-coloring*.
   Since the graph G has a *proper* 3*-coloring*, let us color its vertices with 3 different colors, say Blue, Green, and Red using the given *proper* 3*-coloring*. Now, pick one of the three colors, say Red, and change all Red vertices in G with one of the other two colors, say Blue. So, the vertices of G are either Blue or Green now.
   Let us claim that at least one triangle in G has all of its vertices of the same color, Blue after the above color change - violating the *nice* 2*-coloring* rule.
   Assume that the above triangle has the edges, $(u, v), (v, w), (w, u)$, with Blue vertices u, v, w. Then one of the following four must have occurred:

   - All three vertices were previously colored Red and are now colored Blue: This contradicts the fact that G was colored first using its *proper* 3*-coloring*, because every two vertices connecting an edge in a *proper* 3*-coloring* are of different colors.
   - Two of the three vertices were previously colored Red and are now colored Blue and the third one was already Blue: WLOG, let us assume that vertices u and v were previously colored Red and w was Blue: Again this contradicts the fact that G was colored first using its *proper* 3*-coloring*, because every two vertices connecting an edge in a *proper* 3*-coloring* are different. So only one of u or v was previously colored Red, and the other one, w, must have been Green because it is also connected to a Blue vertex.
   - Only one of the three vertices was previously colored Red and is now colored Blue: Since all three vertices of the triangle are now Blue, then two of the vertices must have been already Blue. This again contradicts the fact that G was colored first using its *proper* 3*-coloring*, because every two vertices connecting an edge (in this triangle) in a *proper* 3*-coloring* are of different colors.

- None of the three vertices, u, v, w were previously colored Red: This implies that all three vertices of this triangle were previously colored Blue, which contradicts the fact that G was colored first using its *proper* 3*-coloring*, because every two vertices connecting an edge in a *proper* 3*-coloring* are of different colors.

Therefore, it must be that at least one of the vertices in this triangle is not Blue. So, it follows that G has a *nice* 2*-coloring*.

(b) (**15 points**) Give a randomized algorithm for finding this coloring that has expected runtime polynomial in the number of vertices of G. Determine the expected runtime of this algorithm (that is, analyze the expected amount of time that this algorithm takes before finding a nice 2-coloring.)

(Hint 1: Consider an algorithm that picks an arbitrary starting coloring and while there exists a monochromatic triangle (i.e., a triangle where all vertices have the same color), picks a random vertex in the monochromatic triangle and flips its color. Now try an analysis similar to the random 2SAT algorithm.)

(Hint 2: You may find it helpful to partition the vertices of G into three parts, corresponding to the vertices that are assigned red, blue, and green respectively in the proper 3-coloring. Try keeping track of how far your 2*-coloring is from this assignment over the course of your random walk.)*

Solution: Consider a graph G with *proper* 3*-coloring*. Let C_3 be a valid 3-coloring of G using colors red, blue and green. As suggested in the hint, let us partition the vertices of G into three parts, namely V_R, V_B and V_G, each corresponding to vertices colored using red, blue and green respectively. Therefore, G, is a tripartite graph with respect to the three colors, which means there are no edges between the vertices belonging to the same partition (color).

We provide an analysis similar to the randomized algorithm for 2SAT.

Let C_2 be an arbitrary *nice 2-coloring* of G with colors, say red and blue. We will update C_2 by randomly selecting a monochromatic triangle and flipping the color of one of its randomly selected vertices. We will repeat this process until there is no monochromatic triangle exists in G, which is our goal state.

The idea behind the above strategy is as follows. Since we are starting with an arbitrary *nice 2-coloring* C_2 of G, assuming that C_2 uses only red and blue, when the algorithm reaches a state where all the vertices in V_R are colored red and all the vertices in V_B are colored blue, it is obvious that there will be no monochromatic triangles at this state. This is true, regardless of the color of the vertices in the partition, V_G.

Let C_2' be an intermediate state during the above process of repeatedly selecting a random monochromatic triangle and flipping one of its randomly selected vertices. We count the number of color mismatches between C_2' and their respective partitions V_R and V_B. Let this count to be the *distance to the goal state*. The color mismatches occur when a vertex $v \in V_R$, but it's color is now blue, or a vertex $v \in V_B$, but it's color is now red. The distance will be zero in the goal state.

On the next step, our algorithm randomly selects one of the remaining monochromatic triangles. Since C_2 contains only red and blue vertices, the selected monochromatic triangle can be either RRR or BBB. If the selected vertex belongs to V_G, then flipping its color (to either red or blue) will not change the distance to the goal state. On the other hand, if the monochromatic triangle is RRR, and if our algorithm flips the vertex that belongs to V_B, the distance will decrease by 1. This is because the color was flipped from red to blue, and the number of color-mismatched vertices decreases by 1. However, if our algorithm flips the vertex that belongs to V_R, then the distance will increase by 1 as the number of color mismatches increases.

Similarly, if the monochromatic triangle is BBB, and if our algorithm flips the vertex that belongs to V_R, the distance will decrease by 1. This is because the number of color-mismatched vertices decreases by 1. However, if our algorithm flips the vertex that belongs to V_B, then the distance will increase by 1 as the number of color mismatches increases.

In other words, since we have three equally likely possibilities, the distance will be incremented with probability $\frac{1}{3}$, or decremented with probability $\frac{1}{3}$, or remain the same with probability $\frac{1}{3}$.

Therefore, we can represent this problem as an unbiased random walk on a number line with n vertices, where the distance to goal remains the same with probability $\frac{1}{3}$, decreases with probability $\frac{1}{3}$ or increases with probability $\frac{1}{3}$. Hence the expected number of steps to reach the goal state is equal to the expected number of steps to reach the goal from the start state C_2. In the worst-case, the start state could be the very first vertex on the opposite end of the number line. This is analogous to the analysis for 2SAT provided in Lecture 18. Hence, we find the following recurrence relation:

$$\begin{aligned} T(i) &= \frac{T(i)}{3} + \frac{T(i-1)}{3} + \frac{T(i+1)}{3} \\ \frac{2T(i)}{3} &= \frac{T(i-1)}{3} + \frac{T(i+1)}{3} \\ T(i) &= \frac{T(i-1)}{2} + \frac{T(i+1)}{2}. \end{aligned}$$

The solution to the above recurrence was found to be $T(i) = n^2 - i^2$ in Lemma 1 for the analysis of 2SAT in Lecture 18. Therefore, the expected amount of time that the above algorithm takes before finding a nice *nice 2-coloring* of G is $O(n^2)$. □

For a random start state of C_2, where some of the triangles were already non-monochromatic, the above algorithm may perform better than n^2 steps.

4. Suppose we have a random walk with boundaries 0 and n, starting at position i. As mentioned in class, this can model a gambling game, where we start with i dollars and quit when we lose it all or reach n dollars. Let W_t be our winnings after t games, where W_t is defined only until we hit a boundary (at which point we stop). Note that W_t is negative if we are at a position j with $j < i$; that is, we have lost money. If the probability of winning and the probability of losing 1 dollar each game are 1/2, then with probability 1/2, $W_{t+1} = W_t + 1$ and with probability 1/2, $W_{t+1} = W_t - 1$. Hence

$$E[W_{t+1}] = \frac{1}{2}E[W_t + 1] + \frac{1}{2}E[W_t - 1] = E[W_t],$$

where we have used the linearity of expectations at the last step. Therefore when the walk reaches a boundary, the expected winnings is $E[W_0] = 0$. We can use this to calculate the probability that we finish with 0 dollars. Let this probability be p_0. Then with probability p_0 we lose i dollars, and with probability $1 - p_0$ we gain $n - i$ dollars. Hence

$$p_0(-i) + (1 - p_0)(n - i) = 0,$$

from which we find $p_0 = (n - i)/n$.

Gossip Girl encounters this game in one of her classes at Constance Billard High School. Unfortunately, because it is high school, the game is not fair; instead, the probability of losing a dollar each game is 2/3, and the probability of winning a dollar each game is 1/3. Each student starts with i dollars on day 0 and plays a game on each day and gets to graduate from high school if they reach n dollars before going bankrupt. Bankrupt students don't get to play this game and stay in school for ever.

We wish to understand the probability that a student graduates (by extending the random walk analysis). In each of the following parts, including the optional one, you may assume the result from previous parts even if you did not prove it.

(a) **(5 points)** Show that $E[2^{W_{t+1}}] = E[2^{W_t}]$.

Solution: We know that in every round, $\mathbb{P}[\text{Loss}] = \frac{2}{3}$ and $\mathbb{P}[\text{Win}] = \frac{1}{3}$. Using this, we have the following.

$$\begin{aligned}\mathbb{E}\left[2^{W_{t+1}}\right] &= \mathbb{P}[\text{Loss}] \cdot \mathbb{E}\left[2^{W_t - 1}\right] + \mathbb{P}[\text{Win}] \cdot \mathbb{E}\left[2^{W_t + 1}\right] \\ &= \frac{2}{3} \cdot \mathbb{E}\left[2^{W_t - 1}\right] + \frac{1}{3} \cdot \mathbb{E}\left[2^{W_t + 1}\right] \\ &= \frac{1}{3} \cdot \mathbb{E}\left[2^{W_t}\right] + \frac{2}{3} \cdot \mathbb{E}\left[2^{W_t}\right] \\ &= \mathbb{E}\left[2^{W_t}\right]\end{aligned}$$

This proves the statement. □

(b) **(5 points)** Use this to determine the probability of finishing with 0 dollars and the probability of finishing with n dollars when starting at position i.

Solution: Following the same steps as in the statement prior to the Gossip Girl statement, and assuming p_i is the probability of losing when we start at i, we have the following.

$$2^0 = 1 = p_i 2^{-i} + (1 - p_i) 2^{n-i}$$

Solving, we find thee probability of finishing with 0 dollars is,

$$p_i = \frac{2^{n-i}-1}{2^{n-i}-2^{-i}}.$$

Therefore, the probability of winning with n dollars is given by

$$1-p_i = \frac{1-2^{-i}}{2^{n-i}-2^{-i}}.$$

□

(c) **(10 points)** Now suppose the initial number of dollars that each student has is a random number generated as follows: Every student is given n envelopes and each (independently) has 1 dollar with probability 1/2 and 0 dollars with probability 1/2. On day 0 the student opens all the envelopes to get their initial i dollars. Show that the probability of finishing with n dollars in this sequence of games is at least $\Omega(c^n)$ for some constant $c > 1/2$. (So while graduation is still exponentially unlikely, it is better than the probability of graduating on day 0.)

Solution: Let P denote the required probability (over the randomness of both the initialization and the process of moving back and forth during the game) of graduating/winning. Using the law of total probabilities and our result from Part (b), we have the following.

$$\begin{aligned}
P &= \sum_{i=1}^{n} \binom{n}{i} \frac{1}{2^i} \cdot \frac{1-2^{-i}}{2^{n-i}-2^{-i}} \\
&= \sum_{i=1}^{n} \binom{n}{i} \cdot \frac{1-2^{-i}}{2^n-1} \\
&= \frac{1}{2^n-1} \sum_{i=1}^{n} \binom{n}{i} \cdot (1-2^{-i}) \\
&= \frac{1}{2^n-1} \left[(2^n-1) - \sum_{i=1}^{n} \binom{n}{i} \cdot \frac{1}{2^i} \right] \\
&= \frac{1}{2^n-1} \left[(2^n-1) - \left(\frac{3}{2}\right)^n + 1 \right] \\
&= \frac{2^n-(1.5)^n}{2^n-1} \\
&> \frac{(1.5)^{n-1}}{2^n-1} \\
&\geq \frac{1}{4} \cdot \left(\frac{3}{4}\right)^n && \text{(For large enough } n.) \\
&\geq c^n && \text{(For large enough } n \text{ and } c > 0.5.)
\end{aligned}$$

When $n \geq 4$, $c = 0.51$ satisfies the above. □

(d) **(0 points, optional)** For every positive integer t, show that the probability that a student is neither bankrupt nor has graduated by the end of $10 \cdot t \cdot n$ days is at most $(23/30)^t$. [Hint: Suppose we change the rules so that students can play games even after they graduate or go bankrupt. What is the expected number of dollars they have after $10n$ days. What is the probability they have a positive number of dollars?]

(e) **5 points)** Show that the probability that a student graduates in $O(n^2)$ days is at least $\Omega(c^n)$ (for the same constant c as in Part (c)).

Solution: For $1 \le i \le n$, let $T(i)$ denote the expected number of days taken to reach n dollars first before losing all the money. Then we have the following.

$$\begin{aligned} T(0) &= 0 && \text{(Because we have already lost.)} \\ T(n) &= 0 && \text{(Because we have already won.)} \\ T(i) &= 1 + \frac{2T(i-1)}{3} + \frac{T(i+1)}{3} \end{aligned}$$

We assume that $T(i) \in O(n^2)$ by solving the above recurrence like it was discussed in class. The recurrence is a small variation of the standard recurrence with uniform weights for random walks over a line, which has the same solution, which is why we assumed it here, as well. So, we don't exactly know the solution right now. Now, we have the following.

$$\begin{aligned} \mathbb{E}[T] &= \sum_{i=1}^{n-1} \binom{n}{i} \frac{1}{2^i} \cdot (1-p_i) \cdot T(i) \\ &= O(n^2) \cdot \sum_{i=1}^{n-1} \binom{n}{i} \frac{1}{2^i} \cdot (1-p_i) \\ &= O(n^2 c^n) && \text{(Using Part (c))} \end{aligned}$$

Using Markov's inequality, by setting $t = O(n^2)$, we get the required claim. □

(f) **(10 points)**: For some $c > 1/2$ give an expected $O(n^{124} c^{-n})$-time (therefore, expected $o(2^n)$-time!) algorithm for solving 3SAT. You may assume that the 3SAT formula is satisfiable.

Solution: We use the following algorithm, which is similar to what was described in class, but with a caveat that we only initialize the assignment once. This seems to be a the first choice given the parameters in the Gossip Girl problem, which will be explained after stating the algorithm.

```
i. Set all variables independently, uniformly at random.
ii. Repeat the following $k$ times.
    A. If a clause is not satisfied, pick a variable randomly within the clause and flip its sign in all the clauses.
    B. If the formula is not satisfied, return the assignment.
iii. Return that the clause is not satisfiable.
```

When a clause is not satisfied when the formula is satisfiable, it must be the case that at least one of the assignments in the clause is incorrect. When uniformly picking a variable in the clause to flip, we are increasing the number of "correctly" assigned variables by 1 with probability at least $1/3$, and decreasing by 1 with probability at most $2/3$. This is similar to this Gossip Girl problem in that, given a satisfying assignment, we are either moving towards it or away from it by 1 step in each iteration. The difference is that here, we are allowing ourselves to go bankrupt, i.e., we are okay to be in a state where we are in a configuration that does not intersect at all with any satisfying assignment. In this case, our recursion changes for the 0 dollar (or all incorrect assignments) situation. In other words, we use the following recursion for the $T(i)$ function, which denotes the expected running time when we have i variables correctly assigned.

$$\begin{aligned} T(n) &= 0 \\ T(0) &= 1 + T(1) \\ T(i) &= 1 + \frac{2T(i-1)}{3} + \frac{T(i+1)}{3} \end{aligned}$$

Note that the probability distribution for all i for the initial assignment is the same as in the Gossip Girl problem for each i. We use a very similar analysis as mentioned in the lecture notes for the 3SAT algorithm for this, and also determine the value of k accordingly. It will again be a matter of solving the above recurrence, computing the expected running times, and using Markov's inequality. The expected running time will therefore be of the order of $\text{poly}(n)(3/4)^{-n}$ as indicated in the problem. $\square$