Almost every class day of the semester will have a corresponding daily exercise. These daily exercises are designed to help guide you through the course material and provide regular opportunities to reflect on your work.

Each daily exercise consists of three parts:

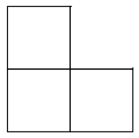
1. **Submitting your answer.** Each daily exercise is due **before class** on its corresponding day, and late submissions will not be accepted. Your solution must be uploaded to Gradescope before the deadline. Please try to keep you solution on a **single page**.
2. **Seeing the solution.** At the beginning of class on the day of the deadline, we will go over the solution to the exercise together.
3. **Reflection.** After seeing the in-class solution, students must fill out a short questionnaire, reflecting on ways to improve their solution. You are encouraged to complete the reflection on your own submission shortly after class while the solution is still fresh in your mind. Each reflection is due before the next class period.

Note that not all the daily exercises are available at the beginning of the semester. As a result, this PDF will be updated regularly to append more daily exercises as the semester goes on; be sure to check back regularly.

General tips for solving problems in this class.

- Start early! Sometimes problems take a lot of brainstorming before a solution comes to mind.
- Treat your solution similar to an essay for an English class: Draft; Edit; Rewrite. Think about clarity as you write your answers; one of your goals is to effectively communicate your solution.
- Homework questions are not always intended to be straightforward—you will need to think and struggle through some of the questions. This is where a lot of your learning in this class will occur. If you get stuck, write-up what you have been able to complete and where you are stuck. For example, "Here is where I got stuck. I can't see how to finish the argument because of x."
- If you're totally stuck on a question, find someone (a classmate or your instructor) with whom to chat about it.

Daily Exercise 1. Tim Urness challenges you to a game. He gives you a board upon which is drawn a grid of squares with 2^n rows and 2^n columns (i.e. there are $(2^n)^2$ squares in the grid). He also gives you a bag containing a large number of L-shaped pieces like the one shown below. (Each square is the size of a space on the grid.)



The challenge is this: Tim will first select a single square on the grid and mark it as *unusable*. Your goal is to place the L-shaped tiles in such a way that they cover the entire grid with no overlaps and leaving only the unusable square uncovered. If you can do so, you win; otherwise, Tim wins.

Prove using induction that you can win the game for any n and for any unusable square that Tim selects.

Daily Exercise 2. Do exercise 2-10 from the textbook.

Daily Exercise 3. Do exercise 2-2 from the textbook.

Daily Exercise 4. Do exercise 3-4 from the textbook.

Daily Exercise 5. Do exercise 3-12 from the textbook.

Daily Exercise 6. Do exercise 4-2 from the textbook.

Daily Exercise 7. Do exercise 4-14 from the textbook.

Daily Exercise 8. Do exercise 5-2 from the textbook.

Daily Exercise 9. Suppose we have a weighted, directed graph $G = (V, E)$, except instead of the edges having weights, the *vertices* have weights. In other words, the weight function is defined to be $w : V \to \mathbb{Z}_{\geq 0}$. We'd like to efficiently find short paths from some start vertex $s \in V$ to some target vertex $t \in V$ in such graphs. Since the vertices have the weights, the weight of a path is defined to be the sum of all the vertex weights along the path.

Describe an efficient algorithm that will find the length of the shortest paths in such vertex-weighted graphs. Give a brief argument why your algorithm is correct and what it's runtime complexity is.

Hint: Instead of writing a whole new algorithm, think of a way to modify the graph G so that you can use the normal Dijkstra's algorithm on it.

Daily Exercise 10. We've now seen weighted graph problems for finding the minimum spanning tree (Prim's algorithm and Kruskal's algorithm) and for shortest paths (Dijkstra's algorithm). For all of these algorithms, we assumed that edge weights were *strictly positive*.

What if we allowed negative edge weights?

(a) Discuss the effects of negative edge weights on the MST algorithms.

(b) Discuss the effects of negative edge weights on the shortest path algorithm.

For each part above, you must provide either an argument that the algorithm(s) still are correct in the presence of negative edge weights or a counterexample that demonstrates that they fail on at least one input.

Daily Exercise 11. Do exercise 10-12 from the textbook.