Precept Outline
- Course Introduction
- Review of Lectures 1 and 2
- Problem Solving

Relevant Book Sections
- 1.4 (Analysis) and 1.5 (Union-Find)
- 1.1 and 1.2 (Java review)

A. Review: Analysis and Union-Find

Your preceptor will briefly review key points of this week's lectures.

B. Analysis

Part 1: Loops

Runtime analysis can be tricky; even short and simple code can be hard to analyze correctly. The key to mastering this skill is practice, practice, practice. That's what we'll do in this part.

Determine the number of times the function `op()` is called *asymptotically*, as a function of n, using **both** tilde notation ($\sim$) and big Theta notation (Θ, i.e. order of growth).

1.

```
for (int i = 10; i < n + 5; i += 2)
    op();
```

2.

```
for (int i = 1; i <= n * n * n; i *= 2)
    op();
```

3.

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < 100; j++)
        op();
```

4.

```
for (int i = 0; i * i < n; i++)
    for (int j = 1; j < n; j *= 3)
        op();
```

5.

```
for (int i = 0; i < n; i++)
    for (int j = 1; j < n; j *= 2)
        op();
```

6.

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < 100; j++)
        for (int k = 0; k < n; k++)
            for (int l = k; l < n; l++)
                op();
```

C. Union-Find

Part 1: Find the Bug!

Consider the following (incorrect) implementation of `union()` in the *quick-find* data structure. Recall that the length-n `leader[]` array is initialized with `leader[i] = i` for all i, and that `find(i)` returns `leader[i]`.

```
1 public void union(int p, int q) {
2     for (int i = 0; i < leader.length; i++)
3         if (leader[i] == leader[p])
4             leader[i] = leader[q];
5 }
```

Find a number of elements n, a sequence of `union()` operations, an integer $0 \leq i < n$, and an integer $0 \leq j < n$, such that elements i and j belong to the same set but `find(i)` and `find(j)` return different values.

Part 2: Fall'22 Midterm Question

For the items below, assume we initialize a union-find data structure with n elements. Then, we perform the following sequence of `union()` operations: `union(0, 1)`, `union(0, 2)`, `union(0, 3)`, ..., `union(0, n - 1)`.

(a) How many total connected components (i.e., disjoint sets) does the resulting data structure contain?

(b) Assume that the data structure implementation is quick-find. How many array updates are made by these `union()` operations, as a function of n, in tilde notation? (Recall that our quick-find implementation of `union(p,q)` never changes `leader[q]`.)

(c) Assume that the data structure implementation is quick-union, and that we call `find(0)` after the sequence of operations above. How many array accesses would `find(0)` make as a function of n in Θ notation? (Recall that our quick-union implementation of `union(p,q)` operation never changes `parent[q]`.)

(d) Assume that the data structure implementation is weighted quick-union, and that we call `find(0)` after the sequence of operations above. How many array accesses, as a function of n in Θ notation, would `find(0)` make? (Recall that our weighted quick-union implementation of `union(p,q)` operation changes `parent[q]` if both trees have the same size.)

D. Optional Bonus Problems

Part 1: Union-Find with "Special" Leaders

Describe a modification of our weighted quick-union data structure that ensures the leader of every set is its *minimum* element.

Part 2: An Application of Union-Find

Suppose you are given a sequence of n positive integers. Define a "group" as a contiguous subsequence of elements. The length of the group is the number of elements in it, and its value is its smallest element. (E.g., the sequence $(5, 3, 4, 1)$ has a single group of length 4 and value 1; two groups of length 3, with values 3 and 1; etc.)

For each $\ell = 1, \ldots, n$, determine the maximum value among all groups of length ℓ. The runtime of your algorithm should be asymptotically smaller than $\Theta(n^2)$. *Hint: consider using the minimum-representative implementation above.*