Precept Outline
- Course introduction.
- Review of Lectures 1 and 2.
- Overview of Assignment 1 (Percolation).

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

A. Introduction

We're pretty psyched for you to see what we've got in store, but, before we let you loose, here are just a few words about the format of precept in this course:

Precepts will be a mix of review, problem solving and discussion. Each precept will start with a short review of the lecture contents, followed by solving a mix of exercises in this handout. A lot of the exercises follow the same format as the ones you will find in the midterm and final exams, so precepts will be good practice for those. There are also optional bonus problems in the PDF on the course webpage (but not in the handout), for students who want an extra challenge.

The exercises are meant to be done in pairs. We want to encourage you to talk about the details of algorithms and data structures with a peer so you can help fill in each other's blind spots.

These exercises are not graded. You don't have to hand in any solutions, we won't grade any of your precept work. So you are encouraged to ask questions about the problems. The solutions to each exercise will be released after all precepts are done.

You are not expected to complete all of the problems in each handout. These handouts and the accompanying Ed lessons are intended for practice, you don't have to solve all of them during or after precept. In fact, it's very unlikely you'll go through the whole handout in any precept. Some of the problems are marked as "optional", which means they are outside of the scope of the course and are intended to be bonus challenge problems.

Attendance is mandatory. Your preceptor will keep track of your attendance (except for this first week), and your attendance will be 2.5% of your grade.

B. Review: Analysis and Union-Find

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

C. Analysis

Part 1: Loops

Runtime analysis can be tricky business; even short and simple-looking 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 ($\sim$) and big Theta (Θ, i.e., order-of-growth) notation.

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 = 0; j < n; j++)
        op();
    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 / 2; k++)
            for (int l = 0; l < n; l++)
                for (int m = 0; m < l; m++)
                    op();
```

D. 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]`.

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

Find a number of elements n, a sequence of `union()` operations and integer $0 \leq i, j < n$ such that i and j should 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., 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?

E. Assignment Overview: Percolation

Your preceptor will introduce and give an overview of your first assignment. Please don't hesitate to ask questions!

F. Optional Bonus Problems

Part 1: Useful Identities

We will make *extensive* use (really) of two identities throughout this course: for partial sums of arithmetic and geometric progressions (in special cases).

Let's start by proving these identities: *formally* show that

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

and

$$\sum_{i=0}^{n-1} 2^i = 1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1.$$

Part 2: More Useful Identities

Prove that

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \sim \frac{n^3}{3}$$

and

$$H_n \coloneqq \sum_{i=1}^{n} \frac{1}{i} \sim \ln n.$$

Part 3: Applying Union Find (Challenge)

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 $\Theta(n^2)$.