Precept Outline
- Review of Lectures 5 and 6:
  - Comparators and Comparables
  - Elementary sorts
  - Mergesort

Relevant Book Sections
- Book chapters: 2.1, 2.2 and 2.5

A. Review: O/Ω Notation + Elementary Sorts + Mergesort + Comparable/Comparator

Your preceptor will briefly review key points of this week's lectures. They may refer to the warm-up exercise and the code snippet shown below.

Warm-up: Let $f(n) = 3n + 4n \log_2 n + 8\sqrt{n} \log_2 n$. Select all that apply.

( ) $f(n) = O(n)$
( ) $f(n) = \Omega(n)$
( ) $f(n) = O(\sqrt{n} \log n)$
( ) $f(n) = \Omega(\sqrt{n} \log n)$
( ) $f(n) = O(n \log n)$
( ) $f(n) = \Omega(n \log n)$
( ) $f(n) = O(n^2)$
( ) $f(n) = \Omega(n^2)$
( ) $f(n) = O(\log n)$
( ) $f(n) = \Omega(\log n)$
( ) $f(n) = O(2^n)$
( ) $f(n) = \Omega(2^n)$

```
1  public class YourClass implements Comparable<YourClass> {
2    public int compareTo(YourClass that) {
3        // returns int > 0 if this > that
4        // returns int < 0 if this < that
5        // returns 0 otherwise
6    }
7
8    private static class YourComparator implements Comparator<YourClass> {
9            public int compare(YourClass obj1, YourClass obj2) {
10           // returns int > 0 if obj1 > obj2
11           // returns int < 0 if obj1 < obj2
12           // returns 0 otherwise
13       }
14   }
15   public static Comparator<YourClass> yourComparison() {
16       return new YourComparator();
17   }
18   ...
19 }
```

B. Comparable & Comparator

The code snippet below shows the instance variables of a class `Movie`, and partially filled instance methods that should support comparing elements of this class in three ways:

- by alphabetical order of `title` (the default order);
- by release `year`; and
- by `rating` (0-5 stars).

Fill in the blanks numbered 1 to 6.

```
public class Movie implements _____(1)_____ {
    private String title;
    private int    year;
    private int    rating;

    public int compareTo(Movie m) {
        return _____(2)_____;
    }

    public static Comparator<Movie> byYear() {
        return new YearComparator();
    }

    private static class YearComparator implements _____(3)_____ {
        public int compare(Movie m1, Movie m2) {
            return _____(4)_____;
        }
    }

    public static Comparator<Movie> byRating() {
        return new RatingComparator();
    }

    private static class RatingComparator implements _____(5)_____ {
        public int compare(Movie m1, Movie m2) {
            return _____(6)_____;
        }
    }
    ...
}
```

C. Sorting Algorithms

Part 1: Spring'24 Midterm Problem

Given two integer arrays, `a[]` and `b[]`, the *symmetric difference* between `a[]` and `b[]` is the set of elements that appear in exactly one of the arrays. Design an algorithm that receives two *sorted arrays*, each consisting of n *distinct elements*, and outputs the size of their symmetric difference.

For full credit, it must use $\Theta(1)$ extra memory and its running time must be $\Theta(n)$ in the worst case (the arrays `a[]` and `b[]` should not be modified).

Part 2: Sorting Lower Bounds

Imagine you are given unlimited access to call a method (say, via "the cloud") which costs your program *constant time* in order to help sort an array `Comparable[] a`.

Suppose the method is `argmin(Comparable[] a, int i)`, which returns $\arg\min_{i \le k < n}\{\texttt{a[k]}\}$, i.e. the minimum elements in the range $[i, n]$. Can you use it to implement a (comparison-based) sorting algorithm with $O(n)$ running time? If so, how? If not, why not?

Part 3: Finding the Missing Element

Suppose that you are given a sorted array `a[]` with $n - 1$ distinct integers between 0 and $n - 1$. In other words, you are given the array `[0, 1, ..., n - 1]` but with one of the elements missing. Design an algorithm with $\Theta(\log n)$ worst-case running time that outputs the missing element.

For example, if the array is `a[] = [0, 1, 2, 3, 5, 6, 7]`, then $n = 8$ and the missing element is 4.