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.

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

```java
  public class YourClass implements Comparable<YourClass> {
    public int compareTo(YourClass that) {
        // returns int > 0 if this > that
        // returns int < 0 if this < that
        // returns 0 otherwise
    }

    private static class YourComparator implements Comparator<YourClass> {
            public int compare(YourClass obj1, YourClass obj2) {
            // returns int > 0 if obj1 > obj2
            // returns int < 0 if obj1 < obj2
            // returns 0 otherwise
        }
    }
    public static Comparator<YourClass> yourComparison() {
        return new YourComparator();
    }
    ...
}
```

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). A solution with $O(n \log n)$ runtime and $O(n)$ extra memory that does not satisfy the full credit performance requirements receives partial (at least half) credit.

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.

(a) Suppose the method is `sum(int[] a, int i, int j)`, which, given two indices $0 \leq i \leq j < n$, returns the sum $\sum_{k=i}^{j}$ `a[k]`. Can you use it to implement a (comparison-based) sorting algorithm with $O(n)$ running time? If so, how? If not, why not?

(b) Suppose the method is `min(int[] a, int i)`, which returns $\min_{i \le k < n}\{$`a[k]`$\}$. 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: Equality of Histograms

The *histogram* of an array `s[]` of samples is the set of pairs (i, f_i), where f_i is the number of indices j such that the j^{th} sample `s[j]` has value i. (That is, $f_i = |\{j : s[j] = i\}|$.)

Let `a[]` and `b[]` be integer arrays representing sample sequences. Design an algorithm with $O(n \log n)$ worst-case running time that identifies whether the histograms of `a` and `b` are equal (i.e., if, for all i, the frequency of i is the same in `a` and `b`).

D. Assignment Overview: Autocomplete

Your preceptor will introduce and give an overview of your third assignment. Don't hesitate to ask questions! Summary of the assignment:

- Implement a `Term` class, which stores a word (as a string) and a numeric weight, and also implements comparators for comparing terms in natural order, in decreasing order of weight, and lexicographically based on the first r characters.
- Create a data type `Autocomplete` that initializes with given arrays of terms and weights, and supports methods to return the weight of a term, the top matching term, and the top k matching terms in descending order of weight.
- Implement a `BinarySearchDeluxe` class, which should use binary search to find the first and last index of a given key in a sorted array (these are important primitives to the `Autocomplete` class).

E. Optional Bonus Problems

Part 1: Three-way Mergesort

(Two-way) Mergesort is quite a simple algorithm to describe: to sort n elements, divide the array in half, (recursively) sort each then merge the two halves together. In this exercise, we will study a variant of it: in three-way Mergesort, we divide an array of length n into 3 subarrays of length $\frac{n}{3}$, sort each of them and then perform a 3-way merge.

Given 3 **sorted** subarrays of size $\frac{n}{3}$, how many comparisons are needed (in the worst case) to merge them to a sorted array of size n? Provide your answer in tilde notation.

What is the order of growth of the number of compares in 3-way Mergesort as a function of the array size n? (Here we're counting the total number, including all recursive calls.)

Given a choice, would you choose 3-way or 2-way mergesort? Justify your answer.

Part 2: (Challenge) Counting Inversions

In an array h of n numbers, an *inversion* is a pair of elements that isn't sorted; that is, two indices i and j such that $i < j$ and $h[i] > h[j]$.

Describe an algorithm to compute the total number of inversions of an array of length n in time $\Theta(n \log n)$. *Hint: think about how you can modify mergesort to achieve this.*