EX6: Sorting

Due date: Friday May 27, 2022 at 11:59 pm
Latest turn-in date: Monday May 30, 2022 at 11:59 pm

Instructions:

High-level collaboration is allowed, but exercises are to be completed and submitted individually.
Submit your responses **digitally** through text box submission in the "EX6: Sorting" assignment on Gradescope here: https://www.gradescope.com/courses/379339/assignments/1938578/.
Make sure to log in to your Gradescope account using your UW email to access our course.

1. Sorting Design Decisions

For each of the following scenarios, choose the best sorting algorithm(s) from the list below. Then, justify your choice by describing *what* properties of that sorting algorithm make it the best choice, and *why* those properties are useful in this particular scenario.
Length: 2-3 sentences.

Insertion sort, Selection sort, Heap sort, Merge sort, Quick sort

(a) **Game Programming** You are a game programmer in the 1980s who is working on displaying a sorted list of enemy names that a player has encountered during their gameplay. Since it is a game, you want to display the names of the enemies as fast as possible, but because it is the 1980s, your customers are used to and will be okay with occasional slow loading times. Additionally, the game is intended to run normally on consoles that don't have much memory available.

(b) **Computer Files** Imagine that you are sorting a **small set** of computer files by their file name. You realize, however, that each computer file is huge and takes up a lot of disk space, so you do not want to copy excessively when sorting. In fact, even just moving and rearranging these large files is expensive, so you don't want to move them often.
(Hint: You may find it useful to refer to visuals of the sorting algorithms. Lecture slides and https://visualgo.net/en/sorting are good resources for remembering these general ideas.)

(c) **NASA Probe** Imagine that you are a NASA software engineer. You're assigned a task to sort data you receive from a probe on Mars, in which each piece of data includes time and temperature. The sensors on this probe capture very large amounts of data. The data is already given to you in sorted order of earliest to latest time, but you want to sort them by temperature, where ties in temperature are ordered by time.

2. Stable Sorts

Consider the following Java class:

```java
public class PlayingCard {
    public String suit;
    public int rank;

    public PlayingCard(String suit, int rank) {
        this.suit = suit;
        this.rank = rank;
    }

    public int compareTo(PlayingCard other) {
        return suit.compareTo(other.suit);
    }
}
```

This class is intended to represent an arbitrary French Playing Card, so we restrict `suit` to be one of ”spades”, ”clubs”, ”hearts”, or ”diamonds”, and restrict `rank` to a value between 1 and 13 (inclusive). Beyond these restrictions, you do not have to be familiar with French playing cards to solve this problem.

Note: this problem makes no assumptions about how these cards are being used – in particular, do not assume that all `PlayingCard` objects in the problem need to be from the same deck of cards.

In this problem, assume that PlayingCard objects are indistinguishable from one another if the values of their fields are equal. (That is, assume that we will never use == or care about any object references themselves when we examine the difference between sorting algorithm results).

2.1. Same output

Suppose we want to run a sorting algorithm on a list of 5 PlayingCard objects that uses the `compareTo` method to compare any two cards (note that it simply uses the underlying compareTo of the String `suit` field).

Give a group of 5 PlayingCard objects where, if they were placed in a list **in any order** and fed into a sorting algorithm, Insertion sort and Selection sort would **always** result in the same output.
(*Hint:* recall that Insertion sort is a stable sort, but Selection sort is not.)

Indicate the 5 cards in your group as calls to the PlayingCard constructor, in the format
`PlayingCard(”spades”, 1)`.

2.2. Different outputs

Now, give an ordered list of 5 PlayingCard objects where, if they were fed into a sorting algorithm, Insertion sort and Selection sort would result **different** outputs.

Indicate the 5 cards in your group as calls to the PlayingCard constructor, in the format
`PlayingCard(”spades”, 1)`.

3. Time complexity in special cases

Suppose we run a sorting algorithm on an array of integers that every number is the same, for example, {1,1,1,1}. Write down the **tight** asymptotic running time for each of the following sorting algorithms:

(a) Insertion sort.

(b) Selection sort.

(c) Heap sort.

(d) Merge sort.

(e) Quick sort.