# MSTs & Sorting

**Due date:** Monday August 17, 2020 at 11:59 pm PDT

**Instructions:** Submit your responses to the "EX4: MSTs & Sorting" assignment on Gradescope here: https://www.gradescope.com/courses/141341/assignments/588907. Make sure to log in to your Gradescope account using your UW email to access our course.

These problems are meant to be done **individually**. If you do want to discuss problems with a partner or group, make sure that you're writing your answers individually later on. Check our course's collaboration policy if you have questions.

## 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.

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 then sorted by time.

## 2. Stable Sorts

Consider the following Java class:

```
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. Choosing an Input

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. Note that there's no restriction on the 5 card objects being

### 2.2. Choosing an Ordering Relation

Now, suppose we want to modify the `compareTo` method of `PlayingCard` so that Insertion sort and Selection sort will result in the same output for **any** input list of `PlayingCards`. If that's possible, write Java or pseudocode that would replace the body of the 'compareTo' method above to achieve the desired effect. If it's not possible, describe in 2-3 sentences why it cannot be done. *Hint:* recall that Insertion sort is a stable sort, but Selection sort is not.

## 3. Using Sorting Algorithms

You're given an array where each element is an (age, name) pair representing guests at a socially distant Zoom dinner party. You have been asked to print each guest at the party in ascending order of their ages, but if multiple guests have the same age, only the one who appears first in the original array should be printed. For example, if the input array is

> [(23, Noah), (2000, Gandalf), (50, Frodo), (47, Emma), (23, Sophia), (83200, Superman), (23, Alice), (47, Madison), (47, Mike), (150, Dumbledore)]

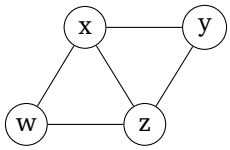
the output should be

> (23, Noah) (47, Emma) (50, Frodo) (150, Dumbledore) (2000, Gandalf) (83200, Superman)

Describe a solution that takes $O(1)$ space in addition to the provided input array. Your algorithm may modify the input array. This party has some very old guests and your solution should still work correctly for parties that have even older guests, so your algorithm can't assume the maximum age of the partygoers. Give the worst-case tight Big-Oh time complexity of your solution.

# 4. Minimum Spanning Trees

Consider the graph below, which has four vertices ($w$, $x$, $y$, $z$) and five undirected edges.




## 4.1.

Label the edges in the graph with non-negative weights 1, 2, 3, 4, 5 such that the shortest path from $w$ to $y$ does not entirely consist of edges that also fall within a minimum spanning tree of the graph. You should use each weight on exactly one edge.

## 4.2.

Label the edges in the graph with non-negative weights 1, 2, 3, 4, 5 such that all of the edges in the shortest path tree starting from $w$ are also edges in a minimum spanning tree of the graph. You should use each weight on exactly one edge.

## 4.3.

Label the edges in the graph with weights 1, 2, 3, 4, -5 such that Dijkstra's algorithm starting from $w$ would compute an incorrect shortest path from $w$ to $y$. You should use each weight on exactly one edge.