Learning Goals

During this lab, you will:

• approach problems with consideration for runtime complexity
• apply sorting as a tool for solving some real problems
• consider tradeoffs between different solutions to problems
• get a feel for the sort of problem solving that this course will focus on

Problem 0: General Approaches

When faced with these sorts of algorithmic problems, there are some general considerations to take:

• Is there a known algorithm that solves this problem, or a related problem?
• Which kinds of data structures fit the operations we need to do?
• Do we have bounds on our input or on our runtime requirement?
• Does the problem give us additional information that might help us do better?
• Ignoring efficiency considerations, what is the very first working solution you can come up with?

Problem 1: Unique Character Strings

*Given a string of length $n$, determine whether it contains all unique characters—or, whether any characters are used more than once.*

Food for Thought:

• If you do it as fast as possible, how much extra space is needed?
• If you do it with no extra space, how quickly can you do it?
• What if you aren’t worried about destroying the input string?

Problem 2: Anagram Strings

*Given two strings, determine whether they are anagrams of each other.*

Example:

```
TOMMARVOLORIDDLE ←→ IAMLORDVOLDEMORT
```

The above are anagrams of each other!

*(Sorry if we spoiled book 2 of Harry Potter for you...)*

• If you do it as fast as possible, how much extra space is needed?
• If you do it with no extra space, how quickly can you do it?
• What if you aren’t worried about destroying the input strings?*
Problem 3: Dutch National Flag Problem

Given \( n \) balls of three colors—say red, white, and blue—arranged randomly in a line, sort them as quickly as possible into contiguous groups of red, white, and blue, with the groups in that order.

\[
\begin{array}{cccccc}
\text{\textcolor{red}{r}} & \text{\textcolor{white}{w}} & \text{\textcolor{blue}{b}} & \text{\textcolor{white}{w}} & \text{\textcolor{red}{r}} & \text{\textcolor{blue}{b}} \\
\text{\textcolor{white}{w}} & \text{\textcolor{white}{w}} & \text{\textcolor{red}{r}} & \text{\textcolor{white}{w}} & \text{\textcolor{blue}{b}} & \text{\textcolor{blue}{b}} \\
\end{array}
\]

- How fast can generic sorts do this?
- What additional information do we have, beyond what generic sorts assume?
- Can we sort it in-place in \( O(n) \) time?

Problem 4: Row/Column-sorted Matrix Membership

Given an \( m \) by \( n \) matrix of integers, where each row and each column of the matrix is sorted, determine whether an integer \( k \) exists somewhere in the matrix.

- How could we do this if we didn’t know the rows and columns were sorted?
- How does knowing the rows and columns are sorted let us do this faster?
- What is our optimal runtime?