In the ticket tracking system described in the video we keep track of date, priority and description for each ticket. Tickets are then placed in a ticket\_box where they are sorted by date with the newest tickets at the front.

Answer the following questions about this system.

## Tradeoff Analysis

The following questions compare the three options for storing tickets discussed in the video. For your reference:

**Option 1: Prioritized Bags** - Three bags of tickets (one each for "high", "medium", and "low" priority). Once a ticket is filled out it is tossed into the appropriate bag.

**Option 2: Single box, prioritized first by priority, then by date** - One box. High priority tickets go in the back. Low priority go in the front. Within a priority group tickets are sorted by date.

**Option 3: Three boxes, one for each priority level** - Each box contains tickets of a single priority level, sorted by date.

A screenshot of a cell phone

Description automatically generated

### Narrowing our options

After considering the question above, I decided to rule out option 1. I can't imagine having to scan through every single ticket to find the oldest one.

Whatever system we use, it should have some date-based organization to it.

The following questions weigh the pros and cons of options 2 and 3.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

### Making a choice

After considering the questions above, I decided to go with option 3. It may take up more space in the vehicle, but there's plenty of space in a car and the improved insert and lookup speeds made that an acceptable tradeoff for me.

This sort of thinking is very similar to the tradeoff analysis you do when writing software. Usually there isn't one best-in-every-way solution, but often there is one that is clearly superior for the requirements of the problem.

If you have another solution or just want to discuss these tradeoffs: go to your Student Hub and discuss!

# Performance Considerations

Until now, our reasons for using one data structure over another have mostly had to do with convenience and readability. But there's another reason to use dictionaries and sets over lists. Speed!

In the rest of this lesson you'll see how fast dictionaries and sets can be.

Note that when we talk about a data structure being "fast", we'll be talking about how long it takes to perform a **membership test** with Python's in keyword.

Here's a brief reminder of how membership testing works...

...on lists

> my\_list = [1,2,3]

> 1 **in** my\_list

**True**

> 4 **in** my\_list

**False**

...on dictionaries

> my\_dictionary = {1: 'one', 2: 'two', 3: 'three'}

> 1 **in** my\_dictionary

**True**

> 'one' **in** my\_dictionary

**False**

...on sets

> my\_set = set([1, 2, 3])

> 1 **in** my\_set

**True**

> 'one' **in** my\_set

**False**

In the next section you'll explore how the size of a list impacts its speed.

# How Lists Work

In the previous section you saw two graphs that looked something like these:

A close up of a logo

Description automatically generated

A close up of a logo

Description automatically generated

We repeated the second experiment on 4 different computers, each of which had a different processor. The results of this experiment are shown below.

A screenshot of a cell phone

Description automatically generated

## How do lists work?

The time it takes to test for membership in a list "scales linearly" with the size of the list. That just means if you double the size of the list you double the amount of time it takes to test for membership of an element (on average).

**Note:** As you learn more, you might see this algorithm described as "big O of N", or mathematically as \mathcal{O}(n)O(*n*). We aren't going to get too deep into the topic at this time, but this notation is all about the [**analysis of algorithms**](https://en.wikipedia.org/wiki/Analysis_of_algorithms) in relation to their efficiency.

The reason it takes longer to membership test a big list has to do with what the computer is doing behind the scenes. When you write:

**if** -1 **in** my\_list

The computer will go through every single element of my\_list in order until it finds -1. If it makes it through the entire list without finding -1 then the result of the membership test will be False.

# How Sets and Dictionaries Work

Underlying Python sets and dictionaries is something called a **hash table**. We aren't going to discuss how these work in this Nanodegree because you don't need to know how they work in order to use them. But if you're interested in learning more you should take a look at:

* [**Udacity videos on hash tables**](https://classroom.udacity.com/courses/cs101/lessons/48682650/concepts/486996020923). The link is to the first video in a series of videos about how hash tables work.
* [**Wikipedia article on hash tables**](https://en.wikipedia.org/wiki/Hash_table).

# Choosing Good Data Structures

#### Lists

Lists are good when:

1. Your data is ordered (and indexing based on position-in-list makes sense).
2. You need to keep track of duplicates.
3. Your data is **mutable** (dictionary keys and elements in a set **must** be immutable).

#### Sets

Sets are good for:

1. **Fast** membership testing.
2. Removing duplicates from a sequence.
3. Computing mathematical operations like intersection, union, and difference.

#### Dictionaries

Dictionaries are good when:

1. You want to associate keys with values.
2. You want fast key-based lookups.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated