The Unquenched Need for Speed

Step 1

Throughout this text, three basic operations for storing data have been on our minds: *find*, *insert*, and *remove*. We have discussed various data structures that can be used to implement these three functions, and we have analyzed their time complexities in order to describe their performance:

- In an *unsorted* Array List and a Linked List, the *worst*-case time complexity to find an element is **O(*n*)**
- In a Randomized Search Tree and a well-structured Skip List, the *average*-case time complexity to find an element is **O(log *n*)**
- In a *sorted* Array List and a balanced Binary Search Tree, the *worst*-case time complexity to find an element is **O(log *n*)**

O(log *n*) is pretty fast, but as we have said before, us computer scientists can never be satisfied: we want even *faster* data structures. With an array, if we knew the specific index we wanted to access, we could theoretically access our element of interest in O(1) time. Formally, if we were looking for a key *k* in an array *a* **and** if we had a way of knowing that key *k* would be at index *i*, we could find *k* with a single O(1) array access operation: *a*[*i*].

Hashing is a way of making the idea above a reality. Given an element *k*, **Hashing** would help us figure out where we would expect *k* to appear in an array. In this section, we will discuss good design methods that allow us to achieve an average-case time complexity of **O(1)** for finding, inserting, and removing elements. Specifically, the data structure we will discuss is the **Hash Table**.

Step 2

If you do a simple Google search for "**Hashing** definition computer science," you may notice that you will get a variety of definitions. For example,

- "**Hashing** is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string." (http://searchsqlserver.techtarget.com/definition/hashing),
- "**Hashing** is one way to enable security during the process of message transmission when the message is intended for a particular recipient only... **Hashing** is also a method of sorting key values in a database table in an efficient manner.", (https://www.techopedia.com/definition/14316/hashing)
- "**Hashing** involves applying a hashing algorithm to a data item, known as the hashing key, to create a hash value." (https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_data_structures/Hash_tab...),
- and, perhaps our favorite, "**Hashing** is one of the great ideas of computing and every programmer should know something about it." (http://www.i-programmer.info/babbages-bag/479-hashing.html)

Technically, all the definitions above are correct in their own contexts. However, for our purposes, we will stick most closely with the third definition. For our purposes, we will describe **Hashing** as follows:

- Given a key *k*, use a **hash function** to compute a number (called a **hash value**) that represents *k* (we will cover **hash functions** and their design in more detail in the next section of the text). This process of using a **hash function** to compute a *hash value* for some key *k* is called **Hashing**
- Then, we can use the *hash value* of *k* to compute an index in some array in which we will store *k*, which is the idea behind a **Hash Table**

There are many design choices that need to be made in order to design a fast **Hash Table**, so throughout this chapter, we will discuss the following design topics:

- Designing a good **hash function**
- Deciding on the size of the **Hash Table**

- Deciding on the **collision resolution strategy** (i.e., what should be done when multiple keys map to the same location in a **Hash Table**)

These topics (as well as the notion of what a **Hash Table** really is) are probably unclear so far, so we will chip away at these topics one-by-one. In the next section, we will discuss the first topic: **hash functions**.