Linear Search Efficiency

One thing that we can tell from this short experiment is that if a name is not in the directory, we'll have to look through all 1,000 names to find that out. As a result, (assuming that the time to compare each name is constant), the **running time** of linear search **grows proportionally** with the size of the directory we're searching. If we had a directory of 10,000 names it would be roughly **ten times slower** than our version with 1,000 names.

However, that is the **worst case**. Most of the time you will find the name you are looking for and won't need to compare all one thousand names.

In the **best case**, the name you'll decide that you only need to look up **Kimberly Thompson**, the first name on the list, and your program will seem to run like lightning. **On average**, though, assuming the names are randomly distributed throughout the directory, you'll have to search through **500** names (or half of the total array) to find the one you want.

In Computer Science we have a particular terminology for discussing the runtime efficiency of algorithms. We say that **linear search** is an `O(n)` algorithm, (which mean **on the order of** `n`), because the time to find your element increases in a linear manner as the number of elements in the array (`n`) increases. This is known as **Big-O notation**.

Even more specifically, we can say that our implementation:

- Has a **worst case** of `O(n)`. We will always have to search through the entire directory if a name is not found.
- Has a **best case** of `O(1)` meaning that if we always search for the first name on the list, we'll find it in **constant time**.
- Has an **average case** of `(n/2)`, meaning that on average, we'll find the name by searching half of the the array. Some will be more, some less, but it will average out.