# Introduction to Data Structures & Algorithms.

Every CS cirriculum in the world covers DS & A. It is a fundamental subject and key to the pervasiveness of CS.

We interact with DS constantly:

- opening a file
- logging into a website
- searching the web

Without specialized algorithms to go with the improved data structures, we would not benefit nearly as much.

Analogy: swords / bows & arrows

~~One of the most basic algorithms.~~

What is a <u>data structure</u>? A collection of data, relationships between them, and functions that can be applied to the data.

Basic examples: strings, integers, lists.

What is an <u>algorithm</u>? A finite sequence of rigorous instructions.

Basic examples: we'll go through one soon, but one arises when solving a computational problem.

What is a <u>computational problem</u>? A task with a precise input and output that asks for a solution in terms of an algorithm.

**Example:** Searching through sorted lists of integers.

<u>Data structure:</u> sorted-list.

It's like a list, but it is always sorted in ascending order. The only entries are ints. Some functions associated to this might be:

- get(i) : return the $i^{\underline{th}}$ entry.
- add(x): insert the integer x.
- len() : return the length of the list.

<u>Computational Problem:</u>

<u>Search</u>

Input: sorted-list L, integer x.

Output: $\begin{cases} \text{integer } i & \text{if } x \text{ in } L, \\ \text{null} & \text{if } x \text{ not in } L, \end{cases}$

where $\underbrace{L[i]}_{\text{"get(i)"}} == x$.

<u>Algorithm 1:</u>

<u>Simple Search</u> (Python code)

```
1. for i in range(len(L)):
2.     if L[i] == x:
3.         return i
4. return None
```

Does Alg 1 solve the computational problem of Search?

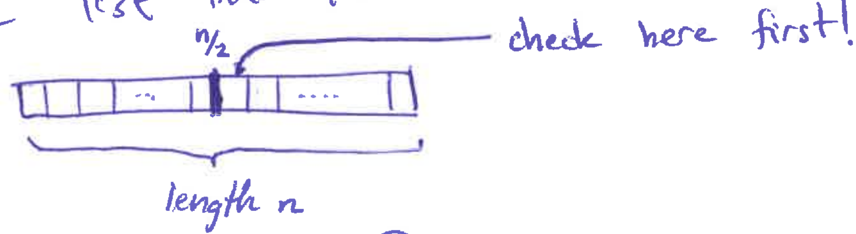
It does! This "check" requires a proof or a correctness proof as it's sometimes called. A correctness proof is a logical argument that convinces its readers that the algorithm solves (correctly!) the computational problem. This can be very challenging, which is why we like to try to keep algorithms ~~so~~ short — and hence computational problems simple and specific.

Is Alg1 a <u>fast algorithm</u>? What does this even mean?

- If $L = [3]$ and $x = 3$, then it's essentially instantaneous.
- If $L = [3, 4, \ldots, 10^{100}]$ and $x = 3$, then same story.
- What about $L = [1, 2, \ldots, 10]$ and $x = 10^{100} + 1$?
  $L = [1, 2, \ldots, 10^{100}]$ and $x = 10^{100} + 1$?
- What about $L = [3, 4, \ldots, 10^{100}]$ and $x = 1$?

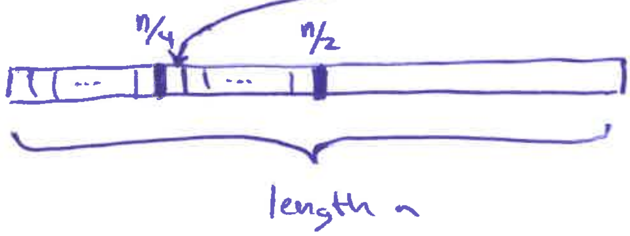
Notice in the last example we did not use the fact that we have a sorted-list. It <u>should</u> have been instantaneous, but it was not. Can we develop a <u>better</u> algorithm?

Let's divide the list into two:



Depending on the outcome, we shift down or up the list. For illustrating, let's assume our $x$ is smaller.

Next step: check here next!




Continue this process until we find a match or no such match exists.

(This is the algorithm. Code is not required to describe algorithms, but it can be very useful.)

## Algorithm 2:

Binary Search (Pseudo code) "←" means "assign".

```
low ← 0
high ← length(L) − 1
while low ≤ high
    mid ← (low+high)/2
    guess ← L[mid]
    if x = guess then
        return mid
    endif
    if x < guess then
        high ← mid − 1
    else
        low ← mid + 1
    end if
end while
return null
```