Algorithm Design

Computational Problems

An input-output specification

- Given some input satisfying some constraints,
- return some output that guarantees some conditions.

Example: `find_max`

- Given a list of numbers,
- return a number from the list which is greater than or equal to any number in the list.

Algorithms

An algorithm is a sequence of operations that accomplishes a task, or solves a computational problem. We demand that an algorithm be

- correct – if the algorithm's input satisfies the algorithm's assumptions, the algorithm always produces correct output –

and want an algorithm to be

- efficient – the algorithm uses the least amount of resources necessary to accomplish its task.

Today we'll learn how to design algorithms to solve computational problems and reason about the correctness of those algorithms.

Review: Function Design Recipe

1. Examples – name of function, example function calls and return values
2. Header – parameter names and types, return type
3. Description – to put in docstring
4. Body – implement the algorithm that transforms the function's input – parameters – to the functions output (or effect) – the return value
5. Test – call the function with argument values that test edge cases and execute each path of execution in the function

Let's apply this function design recipe and design an algorithm to solve the `find_max` problem.

find_max Examples

```
>>> find_max([1,2,3,6,5,4])
6
```

find_max Header

```
def find_max(L: List[int]) -> int:
```

find_max Description

```
def find_max(L: List[int]) -> int:
    """Takes a list of integers and returns the largest of the integers
    """
```

`find_max` Body

Here's where we design our algorithm. We'll use a top-down approach:

1. Describe solution in English
2. Translate steps that are specific enough to implement in Python statments into Python
3. Rewrite steps still described in English in more detail until each step can be translated into a Python statement

We'll start by writing our English-language version of the algorithm in the body of the function as Python comments.

find_max ALgorithm in English

```
def find_max(L: List[int]) -> int:
    """Takes a list of integers and returns the largest of the integers
    """
    # Store the first element of the list as our current guess
    # Look at the remaining elements of the list, replacing our guess
        with
    # elements of the list that are larger than the current guess.
    # After we look at each element of the list, the guess will contain
    # the largest value in the list.
```

Design Pattern - Exhaustive Search

Our `find_max` algorithm is an instance of a general algorithm design pattern: *exhaustive search*:

Proving Algorithms Correct

Computer scientists prove the correctness of their algorithms. In practice, working programmers only informally reason about the correctness of their algorithms, but the techniques used in formal correctness proofs are useful intellectual tools for all programmers. Here we demonstrate the use of one tool: loop invariants.

A loop invariant expresses a formal property of an algorithm that:

- is true prior to the first iteration of the loop,
- if it is true before an iteration of the loop remains true before the next iteration, and
- upon loop termination gives a useful property that helps show that the algorithm is correct.

A Loop Invariant for `find_max`

At the start of each loop, `max_element` is greater than or equal to each element in `L[0:i]`, where i ranges from `0` to `len(L) - 1`.

- Since `i = 0` prior to the first iteration of the loop, `max_element > L[0:i]`.
- Since `max_element` is assigned to `L[i]` if `L[i]` is greater than `max_element`, `max_element > L[0:i]` remains true before the next iteration.
- Upon loop termination `i` is the last valid index for `L`, so `max_element` is greater than or equal to all the elements of `L`

Most algorithms contain some sort of loop (or recursion, which is equivalent), so reasoning about algorithm correctness using loop invariants is useful.

Design Pattern - Generate and Test

1. Generate a guess of the correct result
2. Test the guess
3. Loop until the correct result is found or the possibilities are exhausted

Newton's Square Root Algorithm