How can we tell if a solution is good?

First, we should clarify: what do we mean by “good”?

Then, we should ask: how will we measure “goodness”?

Empirical data (i.e., observations), but it can be messy and incomplete.

A theoretical model (i.e., math) abstracts over the messiness, but it is lossy.

In CS 42, we’ll call this abstraction a cost metric. A cost metric:

• corresponds to one “step”: one unit of work
• highlights the essence of the work (e.g., multiplications, comparisons, function calls...)
• serves as a proxy for an empirical measurement

Good science requires good empirical data and good theoretical models. With both in hand, we can make predictions and communicate with other scientists.

In CS 42, we will learn two techniques for building and communicating theoretical models: asymptotic analysis (e.g., Big O) and exact formulae (e.g., recurrence relations).

Asymptotic analysis

Key question: How does the cost of a problem scale (when we give larger and larger inputs to the problem)?

Informally, we say that the cost of a problem is “in Big O of $f$” if $f$ is a function that describes a reasonable upper bound on (an abstraction of) a problem’s difficulty or a solution’s performance, for reasonably large input sizes.

In other words, Big O is all about scalability: the performance in the limit.

Recurrence relation

A recurrence relation is a mathematical function that is defined in terms of itself.

In Computer Science, we use recurrence relations to describe a theoretical model of the cost of a recursive function.

There is a straightforward procedure for translating recursive functions to recurrence relations:

1. Choose a cost metric.
2. Define what we mean by “the size of the input” (e.g., the value of the input or the size of a list)
3. For each base case, define the cost of computing the result for that base case.
4. For each recursive call, define the cost of computing the result of this iteration of the recursive call plus the cost the recursive call on a smaller input size.
Worst-case inputs

A **worst-case inputs** for an algorithm are the “pathological” ones: the inputs that would be the most expensive to compute.

We usually don’t think of worst-case inputs in terms of size. Instead, the size is fixed, and we talk about other qualities of the input that make it bad.

For example, the worst-case input to a sorting algorithm might be a list of size N that is in reverse order from the order we desire.

When reasoning about a problem, we often like to ask: “What’s the cost for the worst-case input to this problem” because it gives us a sense of how hard the problem might be.

Use it or lose it

**Use it or lose it** is a recursive problem-solving technique.

Before we write code for a use-it-or-lose-it solution, we should fill out the following template:

- What is / are the base case(s)
- For the recursive case(s):
  - What is “it”? a piece of the problem
  - Lose-it solution: How could we solve a smaller version without “it”?
  - Use-it solution: How could we solve a smaller version with “it”?
- How would we combine the solutions to solve the full problem?

No class next Tuesday. Enjoy Fall Break!