Discrete Mathematics

Daniel Gonzalez Cedre

University of Notre Dame
Spring of 2023

Chapter 5

Complexity Theory

5.1 Recursion

Idea 5.1 (Recurrence Relation)**.** A *recurrence relation* is a sequence $\langle x_i \rangle_{i \in \mathbb{N}}$ over some set of values (for our purposes, usually $\mathbb{N}$ or $\mathbb{Z}$) that is useful for *modeling* some physical phenomenon or process (such as an algorithm). Even if the phenomena being modeled would not be immediately described as "recursive" in nature, recurrence relations can often be used to give a recursive *interpretation* amenable to formal analysis using induction.

Definition 5.1 (Fibonacci Sequence)**.**
We define the Fibonacci sequence $\langle \mathcal{F}_i \rangle_{i \in \mathbb{N}}$ by the recursive construction

$$\begin{aligned} \mathcal{F}_0 &:= 0 \\ \mathcal{F}_1 &:= 1 \\ \mathcal{F}_n &:= \mathcal{F}_{n-1} + \mathcal{F}_{n-2} \qquad \text{if } n \in \mathbb{N} \setminus \{0, 1\} \end{aligned}$$

Definition 5.2 (Simple Search)**.**

```python
# recursively searches through the list L for the element x
def simple_search(L: list, x) -> bool:
    if len(L) == 0:
        return False
    return (L[0] == x) or search(L[1:], x)
```

Algorithm 5.1: Simple Search

Definition 5.3 (Bubble Sort)**.**

```python
# recursively propagate the largest element to the end of the list
def bubble(L: list) -> list:
    if L[0] > L[1]:
        L = [L[1], L[0]] + L[2:]  # swap the first two elements of the list
    if len(L) == 2:
        return L
    else:
        return [L[0]] + bubble(L[1:])

# recursively sorts the list L
def bubble_sort(L: list) -> list:
    if len(L) <= 1:
        return L
    else:
        L = bubble(L)
        return bubble_sort(L[:-1]) + [L[-1]]
```

Algorithm 5.2: Bubble Sort

Definition 5.4 (Merge Sort).

```python
# recursively merge the sorted lists L and R into one sorted list
def merge_sort(L: list, R: list) -> list:
    raise NotImplementedError

# recursively sorts the list L
def merge_sort(L: list) -> list:
    n = len(L)
    if n <= 1:
        return L
    else:
        left = merge_sort(L[:n // 2])
        right = merge_sort(L[n // 2:])
        return merge(left, right)
```

Algorithm 5.3: Merge Sort

Intuition 5.1 (Function). We use the notation $f : X \to Y$ to denote that f is a function whose *domain* is X and whose *codomain* is Y, meaning that f takes inputs from X and produces values in Y. When we take an element $x \in X$ and *apply* f to x, we denote the produced result by the notation $f(x)$, which satisfies $f(x) \in Y$. The defining characteristic of functions is, intuitively, that they *must* produce *exactly one* output for any valid input. However, we need a set-theoretic representation of this idea.

The simplest intuition here for the set-theoretic implementation is to think of the graph of a function, which (for the traditional real-valued functions you have encountered in grade school and calculus) consists of ordered pairs (x, y), where the x and y were typically real numbers. Drawing on your prior experience with functions, we know:

I. They should be defined on the entirety of their *domain*, meaning that we should have (x, y) ordered pairs for every value of x from the domain. In fact, in grade school you were probably told that the set of input values where the function was defined was, by definition, the domain of the function.

II. They should pass the *vertical line test*, meaning that there is no more than one if (x, y_1) and (x, y_2) are both points on the graph of the function, then $y_1 = y_2$. This corresponds precisely to the idea that a function's output is *solely* determined by its input—that the same input can not lead to two different outputs.

Definition 5.5 (Unique Existential Quantification).
If φ is a *wff*, we use the notation $\exists! x\big(\varphi(x)\big)$ to denote that *there exists a unique* x that satisfies φ. Formally,

$$\exists! x\big(\varphi(x)\big) \;:\Leftrightarrow\; \exists x\Big(\varphi(x) \land \forall y\big(\varphi(y) \;\Rightarrow\; y = x\big)\Big)$$

Definition 5.6 (Function).
We write $f : X \to Y$ to mean that f is a function with domain X and co-domain Y *iff*

I. $f \subseteq X \times Y$

II. $(\forall x \in X)(\exists! y \in Y)\big((x, y) \in f\big)$

5.2 Asymptotic Analysis

Definition 5.7 (Landau Notation).
Let $X \subseteq \mathbb{R}$ and $Y \subseteq \mathbb{R}$ and consider two functions $f : X \to Y$ and $g : X \to Y$. We define here what it means for g to *asymptotically dominate* f, which is usually read as "f is big-oh of g," with the following Landau notation:

$$f \in \mathcal{O}(g) \;:\Leftrightarrow\; (\exists n \in \mathbb{R})(\exists C \in \mathbb{R})(\forall x \in X)\big(n \leqslant x \;\Rightarrow\; |f(x)| \leqslant C|g(x)|\big).$$

Intuitively, this is saying that a constant multiple g lies permanently above f *everywhere after some point.*

For our purposes here, we only consider functions that map numbers to numbers, so X and Y could be $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, or any combination of subsets of those. If we wanted to, we could generalize even further by noticing that the only impositions we actually have to make on our domain and codomain are a definition for *scalar multiplication* on Y, a notion of *absolute value* $|\cdot|$ on Y, and *partial orders* $\leqslant$ on both X and Y.

Example 5.1.
Let's show that $f \in \mathcal{O}(g)$, $g \in \mathcal{O}(f)$, and $h \notin \mathcal{O}(f)$ for the functions defined below:

$$\begin{aligned} f &: \mathbb{R} \to \mathbb{R} \quad \text{defined by} \quad f(x) \coloneqq x \\ g &: \mathbb{R} \to \mathbb{R} \quad \text{defined by} \quad g(x) \coloneqq 2x + 5 \\ h &: \mathbb{R} \to \mathbb{R} \quad \text{defined by} \quad h(x) \coloneqq x^2 \end{aligned}$$

Proof of $f \in \mathcal{O}(g)$.

Q.E.D.

Proof 2.

Q.E.D.

Proof 3.

Q.E.D.