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* that is 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 following recursive construction.

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

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. We will formally define exactly what a function is in the next chapter.