Prerequisites from Logic and Analysis

We begin with logic.

This book does not require any prior knowledge of formal logic.

First we expand upon some points made in the introduction to the book. All of our notions of computability (for real numbers, continuous functions, and beyond) are based on the notion of a recursive function a: $\mathbb{N} \to \mathbb{N}$ or a: $\mathbb{N}^q \to \mathbb{N}$. ($\mathbb{N}$ denotes the set of non-negative integers.) On the other hand, this book does not require a detailed knowledge of recursion theory. For reasons to be explained below, an intuitive understanding of that theory will suffice.

We continue for now to consider only functions from $\mathbb{N}$ to $\mathbb{N}$. Intuitively, a recursive function is simply a "computable" function. More precisely, a recursive function is a function which is computable by a Turing machine. The weight of fifty years experience leads to the conclusion that "recursive function" is the correct definition of the intuitive notion of a computable function. The definition is as solid as the definition of a group or a vector space. By now, the theory of recursive functions is highly developed.

However, as we have said, this book does not require a detailed knowledge of that theory. We avoid the need for heavy technical prerequisites in two ways.

1. Whenever we prove that some process is computable, we actually give the algorithm which produces the computation. As we shall see, some of these algorithms are quite intricate. But each of them is built up from scratch, so that the book is self-contained.
2. To prove that certain processes are not computable, we shall find that it suffices to know one basic result from recusive function theory—the existence of a recursively enumerable nonrecursive set. This we now discuss.

Imagine a computer which has been programmed to produce the values of a function a from nonnegative integers to nonnegative integers. We set the program in motion, and have the computer list the values $a(0), a(1), a(2), \ldots$ in order. This set A of values, a subset of the natural numbers, is an example of a "recursively enumerable set". If we take a general all purpose computer—e.g. a Turing machine —and consider the class of all such programs, we obtain the class of all recursively enumerable sets.

Suppose now that we have a recursively enumerable set A. Can we find an effective procedure which, for arbitrary n, determines whether or not $n \in A$? If $n \in A$, then