1 *Introduction*

Imagine Alice, Bob, and two piles of ten rocks. Alice and Bob are bored one Saturday afternoon so they play the following game. In each turn a player may either take one rock from a single pile, or one rock from both piles. Once the rocks are taken, they are removed from play; the player that takes the last rock wins the game. Alice moves first.

It is not immediately clear what the winning strategy is, or even if there is one. Does the first player (or the second) always have an advantage? Bob tries to analyze the game and realizes that there are too many variants in the game with two piles of ten rocks (which we will refer to as the *10+10 game*). Using a reductionist approach, he first tries to find a strategy for the simpler 2+2 game. He quickly sees that the second player—himself, in this case—wins any 2+2 game, so he decides to write the "winning recipe":

> If Alice takes one rock from each pile, I will take the remaining rocks and win. If Alice takes one rock, I will take one rock from the same pile. As a result, there will be only one pile and it will have two rocks in it, so Alice's only choice will be to take one of them. I will take the remaining rock to win the game.

Inspired by this analysis, Bob makes a leap of faith: the second player (i.e., himself) wins in any $n+n$ game, for $n \geq 2$. Of course, every hypothesis must be confirmed by experiment, so Bob plays a few rounds with Alice. It turns out that sometimes he wins and sometimes he loses. Bob tries to come up with a simple recipe for the 3+3 game, but there are a large number of different game sequences to consider, and the recipe quickly gets too complicated. There is simply no hope of writing a recipe for the 10+10 game because the number of different strategies that Alice can take is enormous.

Meanwhile, Alice quickly realizes that she will always lose the 2+2 game, but she does not lose hope of finding a winning strategy for the 3+3 game.

Moreover, she took Algorithms 101 and she understands that recipes written in the style that Bob uses will not help very much: recipe-style instructions are not a sufficiently expressive language for describing algorithms. Instead, she begins by drawing the following table filled with the symbols $\uparrow$, $\leftarrow$, $\nwarrow$, and $*$. The entry in position (i, j) (i.e., the ith row and the jth column) describes the moves that Alice will make in the $i + j$ game, with i and j rocks in piles A and B respectively. A $\leftarrow$ indicates that she should take one stone from pile B. A $\uparrow$ indicates that she should take one stone from pile A. A $\nwarrow$ indicates that she should take one stone from each pile, and $*$ indicates that she should not bother playing the game because she will definitely lose against an opponent who has a clue.

	0	1	2	3	4	5	6	7	8	9	10
0	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$
1	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$
2	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$
3	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$
4	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$
5	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$
6	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$
7	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$
8	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$
9	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$
10	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$	$\leftarrow$	$*$

For example, if she is faced with the 3+3 game, she finds a $\nwarrow$ in the third row and third column, indicating that she should take a rock from each pile. This makes Bob take the first move in a 2+2 game, which is marked with a $*$. No matter what he does, Alice wins. Suppose Bob takes a rock from pile B—this leads to the 2+1 game. Alice again consults the table by reading the entry at (2,1), seeing that she should also take a rock from pile B leaving two rocks in A. However, if Bob had instead taken a rock from pile A, Alice would consult entry (1,2) to find $\uparrow$. She again should also take a rock from pile A, leaving two rocks in pile B.

Impressed by the table, Bob learns how to use it to win the 10+10 game. However, Bob does not know how to construct a similar table for the 20+20 game. The problem is not that Bob is stupid, but that he has not studied algorithms. Even if, through sheer luck, Bob figured how to always win the

20+20 game, he could neither say with confidence that it would work no matter what Alice did, nor would he even be able to write down the recipe for the general $n + n$ game. More embarrassing to Bob is that the a general 10+10+10 game with three piles would turn into an impossible conundrum for him.

There are two things Bob could do to remedy his situation. First, he could take a class in algorithms to learn how to solve problems like the rock puzzle. Second, he could memorize a suitably large table that Alice gives him and use that to play the game. Leading questions notwithstanding, what would you do as a biologist?

Of course, the answer we expect to hear from most rational people is "Why in the world do I care about a game with two nerdy people and a bunch of rocks? I'm interested in biology, and this game has nothing to do with me." This is not actually true: the rock game is in fact the ubiquitous *sequence alignment* problem in disguise. Although it is not immediately clear what DNA sequence alignment and the rock game have in common, the computational idea used to solve both problems is the same. The fact that Bob was not able to find the strategy for the game indicates that he does not understand how alignment algorithms work either. He might disagree if he uses alignment algorithms or BLAST[1] on a daily basis, but we argue that since he failed to come up with a strategy for the 10+10 rock game, he will also fail when confronted with a new flavor of alignment problem or a particularly complex similarity analysis. More troubling to Bob, he may find it difficult to compete with the scads of new biologists who think algorithmically about biological problems.[2]

Many biologists are comfortable using algorithms like BLAST without really understanding how the underlying algorithm works. This is not substantially different from a diligent robot following Alice's winning strategy table, but it does have an important consequence. BLAST solves a particular problem only approximately and it has certain systematic weaknesses. We're not picking on BLAST here: the reason that BLAST has these limitations is, in part, because of the particular problem that it solves. Users who do not know how BLAST works might misapply the algorithm or misinterpret the results it returns. Biologists sometimes use bioinformatics tools simply as computational protocols in quite the same way that an uninformed mathematician

1. BLAST is a database search tool—a Google for biological sequences—that will be introduced later in this book.
2. These "new biologists" have probably already found another even more elegant solution of the rocks problem that does not require the construction of a table.

might use experimental protocols without any background in biochemistry or molecular biology. In either case, important observations might be missed or incorrect conclusions drawn. Besides, intellectually interesting work can quickly become mere drudgery if one does not really understand it.

Many recent bioinformatics books cater to this sort of protocol-centric practical approach to bioinformatics. They focus on parameter settings, specific features of application, and other details without revealing the ideas behind the algorithms. This trend often follows the tradition of biology books of presenting material as a collection of facts and discoveries. In contrast, introductory books in algorithms usually focus on ideas rather than on the details of computational recipes.

Since bioinformatics is a computational science, a bioinformatics textbook should strive to present the principles that drive an algorithm's design, rather than list a stamp collection of the algorithms themselves. We hope that describing the intellectual content of bioinformatics will help retain your interest in the subject. In this book we attempt to show that a handful of algorithmic ideas can be used to solve a large number of bioinformatics problems. We feel that focusing on ideas has more intellectual value and represents a better long-term investment: protocols change quickly, but the computational ideas don't seem to.

We pursued a goal of presenting both the foundations of algorithms and the important results in bioinformatics under the same cover. A more thorough approach for a student would be to take an Introduction to Algorithms course followed by a Bioinformatics course, but this is often an unrealistic expectation in view of the heavy course load biologists have to take. To make bioinformatics ideas accessible to biologists we appeal to the innate algorithmic intuition of the student and try to avoid tedious proofs. The technical details are hidden unless they are absolutely necessary.[3]

This book covers both new and old areas in computational biology. Some topics, to our knowledge, have never been discussed in a textbook before, while others are relatively old-fashioned and describe some experimental approaches that are rarely used these days. The reason for including older topics is twofold. First, some of them still remain the best examples for introducing algorithmic ideas. Second, our goal is to show the progression of ideas in the field, with the implicit warning that hot areas in bioinformatics seem to come and go with alarming speed.

3. In some places we hide important computational and biological details and try to simplify the presentation. We will unavoidably be blamed later for "trivializing" bioinformatics.

One observation gained from teaching bioinformatics classes is that the interest of computer science students, who usually know little of biology, fades quickly when the students are introduced to biology without links to computational issues. The same happens to biologists if they are presented with seemingly unnecessary formalism with no links to real biological problems. To hold a student's interest, it is necessary to introduce biology and algorithms simultaneously. Our rather eclectic table of contents is a demonstration that attempts to reach this goal result in a somewhat interleaved organization of the material. However, we have tried to maintain a consistent algorithmic theme (e.g., graph algorithms) throughout each chapter.

Molecular biology and computer science are complex fields whose terminology and nomenclature can be formidable to the outsider. Bioinformatics merges the two fields, and adds a healthy dose of statistics, combinatorics, and other branches of mathematics. Like modern biologists who have to master the dense language of mathematics and computer science, mathematicians and computer scientists working in bioinformatics have to learn the language of biology. Although the question of who faces the bigger challenge is a topic hotly debated over pints of beer, this is not the first "invasion" of foreigners into biology; seventy years ago a horde of physicists infested biology labs, ultimately to revolutionize the field by deciphering the mystery of DNA.

Two influential scientists are credited with crossing the barrier between physics and biology: Max Delbrück and Erwin Schrödinger. Trained as physicists, their entrances into the field of biology were remarkably different. Delbrück, trained as an atomic physicist by Niels Bohr, quickly became an expert in genetics; in 1945 he was already teaching genetics to other biologists.[4] Schrödinger, on the other hand, never turned into a "certified" geneticist and remained somewhat of a biological dilettante. However, his book *What Is Life?*, published in 1944, was influential to an entire generation of physicists and biologists. Both James Watson (a biology student who wanted to be a naturalist) and Francis Crick (a physicist who worked on magnetic mines) switched careers to DNA science after reading Shrödinger's book. Another Nobel laureate, Sydney Brenner, even admitted to stealing a copy from the public library in Johannesburg, South Africa.

Like Delbrück and Schrödinger, there is great variety in the biological background of today's computer scientists-turned-bioinformaticians. Some of them have become experts in biology—though very few put on lab coats

4. Delbrück founded the famous phage genetics courses at Cold Spring Harbor Laboratory.

and perform experiments—while others remain biological dilettantes. Although there exists an opinion that every bioinformatician should be an expert in *both* biology and computer science, we are not sure that this is feasible. First, it takes a lot of work just to master one of the two, so perhaps understanding two in equal amounts is a bit much. Second, it is good to recall that the first pioneers of DNA science were, in fact, self-proclaimed dilettantes. James Watson knew almost no organic or physical chemistry before he started working on the double helix; Francis Crick, being a physicist, knew very little biology. Neither saw any need to know about (let alone memorize) the chemical structure of the four nucleotide bases when they discovered the structure of DNA.[5] When asked by Erwin Chargaff how they could possibly expect to resolve the structure of DNA without knowing the structures of its constituents, they responded that they could always look up the structures in a book if the need arose. Of course, they *understood* the physical principles behind a compound's structure.

The reality is that even the most biologically oriented bioinformaticians are experts only in some specific area of biology. Like Delbrück, who probably would never have passed an exam in biology in the 1930s (when zoology and botany remained the core of the mainstream biological curriculum), a typical modern-day bioinformatician is unlikely to pass the sequence of organic chemistry, biochemistry, and structural biochemistry classes that a "real" biologist has to take. The question of how much biology a good computer scientist–turned–bioinformatician has to know seems to be best answered with "enough to deeply understand the biological problem and to turn it into an adequate computational problem." This book provides a very brief introduction to biology. We do not claim that this is the best approach. Fortunately, an interested reader can use Watson's approach and look up the biological details in the books when the need arises, or read pages 1 through 1294 of Alberts and colleagues' (including Watson) book *Molecular Biology of the Cell* (3).

This book is what we, as computer scientists, believe that a modern biologist ought to know about computer science if he or she would be a successful researcher.

5. Accordingly, we do not present anywhere in this book the chemical structures of either nucleotides or amino acids. No algorithm in this book requires knowledge of their structure.