CHAPTER 22

# The Cost of Computing

**22.1** Algorithms Operating on Collections
**22.2** Algorithm Performance Categories
22.2.1 Polynomial Algorithms
22.2.2 O($2^N$)
22.2.3 O(N!)
**22.3** Unreasonable Algorithms
**22.4** Tractability of Problems
22.4.1 Definitions
22.4.2 Closed versus Open Problems
22.4.3 More Definitions
**22.5** N-P Complete Problems
22.5.1 Yet More Definitions
22.5.2 Illustrations
22.5.3 The Opportunity

## Introduction

A book about computing would be incomplete without summarizing what computer programs can and cannot do. As the chapters progressed, we discussed a number of data collections and a number of operations on those data collections. In Chapter 16 we also discussed Big O, the basic technique for evaluating the computing cost of an algorithm.

In this chapter, first we will summarize the computing cost of the classic operation types on the several data collections. Then we will discuss in general the subject of computability, and will conclude with a discussion of a very special class of problem—the **N-P Complete** problem.

## 22.1 Algorithms Operating on Collections

Table 22.1 illustrates the Big O performance of the most common operations we have discussed on the common data collections. In summary, any traversal, mapping, folding, or filtering is O(N) because all the data items have to be touched to perform these operations. Inserting (the critical component of building) into unstructured collections is usually O(1) because it does not matter where in the collection the new data are placed. Searching is O(N) unless the structure facilitates a binary search, and sorting is O(N log N) unless the structure cannot support random access, in which case insertion sort must be used with O($N^2$). The X symbols indicate that the operation is not defined for this collection.

**Table 22.1 Performance of operations on collections**

| Operation | Array | Sorted Array | Structure | Linked List | Binary Tree | BST | N-Ary Tree | Graph |
|---|---|---|---|---|---|---|---|---|
| Traverse | N | N | X | N | N | N | N | X |
| Insert | 1 | N | 1 | 1 | 1 | Log N | X | X |
| Map | N | N | N | N | N | N | N | X |
| Filter | N | N | N | N | X | N | X | X |
| Fold | N | N | N | N | N | N | N | X |
| Search | N | Log N | X | N | N | Log N | N | N |
| Sort | N Log N | N Log N | X | $N^2$ | X | X | X | X |

## 22.2 Algorithm Performance Categories

As we discussed in Chapter 16 while presenting the Big O discussion, the performance of algorithms really does not matter for small amounts of data. However, in the information age in which we now live, we are confronted daily with enormous amounts of data, and computer scientists are busy polishing algorithms to process these massive amounts of data efficiently. For example, it is only in recent years that the telephone company has made national telephone directories searchable. Before, you had to know the area code before you could ask for a person's phone number because the national listing was too cumbersome to search.

Even with the best algorithms available, however, there are cases where exact solutions are not computable in reasonable amounts of time.

### 22.2.1 Polynomial Algorithms

We tend to think of algorithms where the Big O has N to a constant power as being "OK." However, consider some algorithm that is O($N^5$). With 256 items of data and a processor, the computer performs one operation in a microsecond ($10^{-6}$ sec); this algorithm therefore would take about 1.1 million seconds or 12.7 days, to complete. Probably something to avoid, but it could be worse.

### 22.2.2 O($2^N$)

You probably heard the story of the peasant who did a good deed for the king. In return, the king, not being very bright, offered to do anything the peasant wanted. The peasant, being a bit too greedy for his own good, suggested that they take a chessboard and put a grain of rice on the first square (see Figure 22.1). Each day for the next 63 days, they would put on the next square double the number of grains as on the previous square. If one were to chart the number of grains of rice involved, it would look like the one shown in Table 22.2.

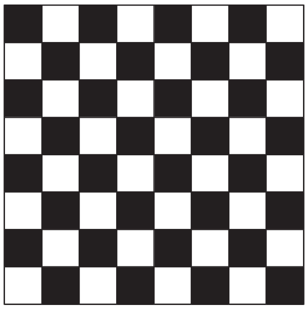

**Figure 22.1** *The king and the peasant*

**Table 22.2 $2^N$ performance**

| Day | Grains |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 4 |
| 4 | 8 |
| 5 | 16 |
| 6 | 32 |
| . | |
| . | |
| 63 | 9,223,000,000,000,000,000 |
| 64 | 18,450,000,000,000,000,000 |

If you could produce a grain of rice every nanosecond ($10^{-9}$ sec), it would take 585 years just to produce the grain for the last square.

(Of course, the king cut off the peasant's head when he realized…)

The Towers of Hanoi problem is also O($2^N$). Consider the child's puzzle toy shown in Figure 22.2. The objective is to move all the rings from peg A to peg C. However, they can only be moved one ring at a time, and you cannot put a larger ring on top of a smaller ring. Figure 22.3 illustrates the solution with three rings. When you realize what is happening, you notice that for each ring you add, you have to move all the smaller rings twice to get the new ring from A to B and then B to C—O($2^N$). So with 64 rings and moving one ring every microsecond, it would take 584,000 years to finish the puzzle.

But it could be worse…

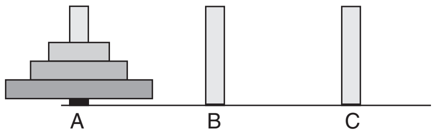

**Figure 22.2** *The Towers of Hanoi*

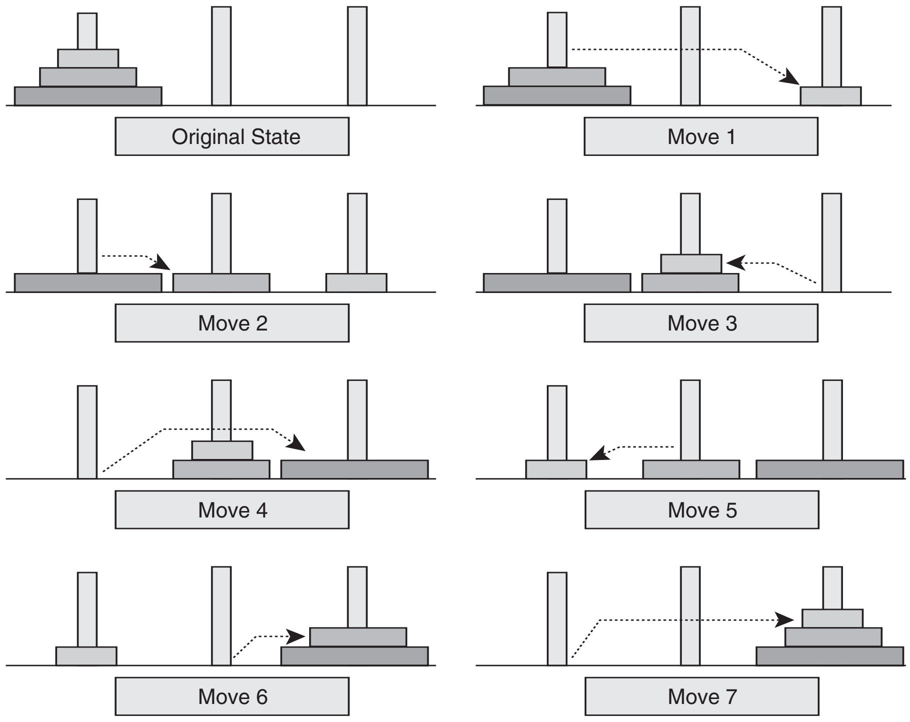

**Figure 22.3** *Towers of Hanoi solution*

### 22.2.3 O(N!)

This is the sad story of our bathroom floor. Our interior decorator came across this new kind of tile with random patterns, as shown in Figure 22.4. The general idea is that you want to match the patterns at the tile edges. So we measured the bathroom floor and discovered we would need only 25 tiles to do the job. We bought a box of tiles and expected that we would have the bathroom retiled by dinner time. Fortunately, we decided not to spread the adhesive until we had the pattern figured out.

We picked the first tile at random, placed it in one corner, and went through the other 24 tiles looking for a match to the first edge. This repeated with the second, third, and fourth tiles down to the last one, and wouldn't you know it, the last one didn't fit! I had to start again with a new configuration. How many combinations are there to try? Yes, N!, and 25! works out to be 15,500,000,000,000,000,000,000,000 (roughly). So if I could try one combination every microsecond, I would be working at tiling the floor for 470 billion years!

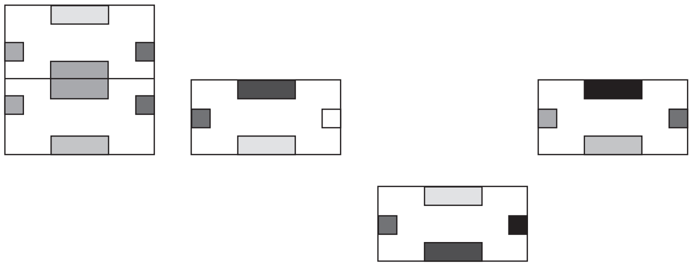

**Figure 22.4** *Random tile patterns*

## 22.3 Unreasonable Algorithms

Although the computation time of polynomial algorithms (where the Big O has N raised to a constant power) can get quite nasty, we class them as **reasonable** algorithms. However, where N gets into the exponent in some form, we refer to the algorithms as exponential algorithms and classify them as **unreasonable**. While we don't really care about the runtime performance for very small amounts of data, the worst of the exponential algorithms take an enormous amount of time even on a really fast computer, as shown in Figure 22.5.

Would a faster computer help, or a large cluster of computers, or both? All these approaches merely improve the performance by linear amounts; if you really need to solve the kind of problem where these algorithms are invoked, you have to use sophisticated approximation techniques.

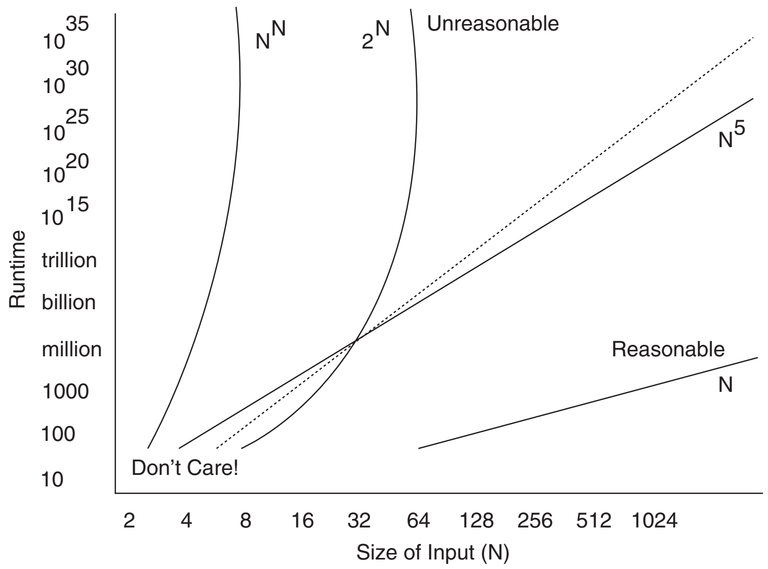

**Figure 22.5** *Algorithm categories*

## 22.4 Tractability of Problems

Since algorithms are really only used to solve problems, we also need some way to assess the advisability of even starting to solve a problem whose solution might end up being unreasonable. We turn from the performance of algorithms to assessing the tractability of problems.

### 22.4.1 Definitions

First we consider two definitions:

- The **upper bound** of a problem is the Big O of the best algorithm that has ever been found to solve the problem completely
- The **lower bound** of a problem is the Big O of the best proof mathematicians can develop that the solution to this class of problem must be at least this complexity

For example, a trivial lower bound would be O(1) because if you were smart enough, you could always write down the right answer immediately.

### 22.4.2 Closed versus Open Problems

There are some classes of problem where the upper and lower bounds have the same Big O value. For example, mathematicians are satisfied that the best sorting algorithm will be O(NlogN). Since we already have sorting algorithms of O(NlogN), this indicates that sorting is a closed problem—there is no room for improving the state of the art. This does not mean that there will never be better sorting algorithms, because there will always be linear improvements, such as the progress from Merge Sort to Quick Sort. But the improvement will not affect the Big O.

There are other problems where the upper and lower bounds have not yet been met—they are still open for better algorithms or better proofs.

### 22.4.3 More Definitions

This leads us to the following definitions:

- A problem is said to be **tractable** if its upper and lower bounds are at worst polynomial
- A problem is said to be **intractable** if its upper and lower bounds are both exponential

## 22.5 N-P Complete Problems

These definitions leave an obvious hole. What do we do with problems whose upper bound is exponential (there is not yet a polynomial algorithm that solves the problem completely) yet whose lower bound is polynomial (the mathematicians cannot prove that the solution has to be exponential). This is in fact an enormous class of problems—perhaps larger than any other class of problems—referred to as N-P Complete problems.

### 22.5.1 Yet More Definitions

To understand this description of problems, we need a few more definitions:

- A solution is said to be **deterministic** if at each step of the way, there exists some logic indicating which choice to make without looking forward to the rest of the solution.
- A solution is said to be **non-deterministic** if no logic exists indicating which choice to make without looking forward to the rest of the solution.
- Perhaps the most remarkable attribute of this class of problem is that all these problems can be expressed in the same notation—statements in propositional calculus. This means that the problems are deemed to be **complete**, that is, transformable from one to another.

The phrase "N-P Complete" therefore describes that set of problems that is **N**on-Deterministic, should have a **P**olynomial solution, and is a **Complete** set.

### 22.5.2 Illustrations

The world of science and engineering is overflowing with N-P Complete problems that are being solved approximately every day:

- **Traveling salesman problem—**compute the shortest route for a traveling salesman to visit all of the cities in his territory
- **Map coloring problem**—find the algorithm that will color a given map with arbitrary boundaries using only four different colors with no two adjacent regions the same color[1]
- **2-D arrangement problem—**cut out a random selection of parts from a sheet of metal with minimal waste
- **Bin packing problem**—arrange odd-shaped items as densely as possible in a given space
- **Planning problem—**schedule classes for the most efficient use of classroom space and teacher hours
- **Scheduling problem—**determine schedules for airline service or delivery trucks, minimizing the distance traveled to accommodate a specific passenger or cargo demand
- **Clique problem—**find the largest set of vertices in an undirected graph in which there is an edge between every pair of vertices

All of these problems and many more of their cousins are approximated every day of the week thousands of times. The inefficiency of the solutions to these problems is a permanent drain on the national economy.

### 22.5.3 The Opportunity

If you can lower the upper bound of one of the previous problems, or develop a proof that raises its lower bound, you will be famous overnight. They will name buildings after you.

## Chapter Summary

*In this chapter we have seen that there is a significant number of practical, everyday problems that are physically impossible to solve exactly; particularly, the class of problem described as N-P Complete. Algorithms beyond the scope of this text are used to develop and refine approximate solutions to these problems.*

[1] This is easily done if you restrict the number of regions that can intersect at a point to four.