CHAPTER ONE

Introduction

But the age of chivalry is gone. That of sophisters, oeconomists, and calculators, has succeeded; and the glory of Europe is extinguished for ever.

Edmund Burke (1729–1797), *Reflections on the Revolution in France*

1.1 Modern Finance: A Brief History

Modern finance began in the 1950s [659, 666]. The breakthroughs of Markowitz, Treynor, Sharpe, Lintner (1916–1984), and Mossin led to the Capital Asset Pricing Model in the 1960s, which became the quantitative model for measuring risk. Another important influence of research on investment practice in the 1960s was the Samuelson–Fama efficient markets hypothesis, which roughly says that security prices reflect information fully and immediately. The most important development in terms of practical impact, however, was the Black–Scholes model for option pricing in the 1970s. This theoretical framework was instantly adopted by practitioners. Option pricing theory is one of the pillars of finance and has wide-ranging applications [622, 658]. The theory of option pricing can be traced to Louis Bachelier's Ph.D. thesis in 1900, "Mathematical Theory of Speculation." Bachelier (1870–1946) developed much of the mathematics underlying modern economic theories on efficient markets, random-walk models, Brownian motion [ahead of Einstein (1879–1955) by 5 years], and martingales [277, 342, 658, 776].[1]

1.2 Financial Engineering and Computation

Today, the wide varieties of financial instruments dazzle even the knowledgeable. Individuals and corporations can trade, in addition to stocks and bonds, options, futures, stock index options, and countless others. When it comes to diversification, one has thousands of mutual funds and **exchange-traded funds** to choose from. Corporations and local governments increasingly use complex derivative securities to manage their financial risks or even to speculate. **Derivative securities** are financial instruments whose values depend on those of other assets. All are the fruits of **financial engineering**, which means structuring financial instruments to target investor preferences or to take advantage of arbitrage opportunities [646].

The innovations in the financial markets are paralleled by equally explosive progress in computer technology. In fact, one cannot think of modern financial systems without computers: automated trading, efficient bookkeeping, timely clearing and settlements, real-time data feed, online trading, day trading, large-scale databases, and tracking and monitoring of market conditions [647, 866]. These applications deal with *information*. Structural changes and increasing volatility in financial markets since the 1970s as well as the trend toward greater complexity in financial product design call for *quantitative* techniques. Today, most investment houses use sophisticated models and software on which their traders depend. Here, computers are used to model the behavior of financial securities and key indicators, price financial instruments, and find combinations of financial assets to achieve results consistent with risk exposures. The confidence in such models in turn leads to more financial innovations and deeper markets [659, 661]. These topics are the focus of **financial computation**.

One must keep in mind that every computation is based on input and assumptions made by the model. However, input might not be accurate enough or complete, and the assumptions are, at best, approximations.[2] Computer programs are also subject to errors ("bugs"). These factors easily defeat any computation. Despite these difficulties, the computer's capability of calculating with fine details and trying out vast numbers of scenarios is a tremendous advantage. Harnessing this power and a good understanding of the model's limitations should steer us clear of blind trust in numbers.

1.3 Financial Markets

A society improves its welfare through investments. Business owners need outside capital for investments because even projects of moderate sizes are beyond the reach of most wealthy individuals. Governments also need funds for public investments. Much of that money is channeled through the financial markets from savers to borrowers. In so doing, the financial markets provide a link between saving and investment,[3] and between the present and the future. As a consequence, savers can earn higher returns from their savings instead of hoarding them, borrowers can execute their investment plans to earn future profits, and both are better off. The economy also benefits by acquiring better productive capabilities as a result. Financial markets therefore facilitate real investments by acting as the sources of information.

A financial market typically takes its name from the borrower's side of the market: the government bond market, the municipal bond market, the mortgage market, the corporate bond market, the stock market, the commodity market, the foreign exchange (forex) market,[4] the futures market, and so on [95, 750]. Within financial markets, there are two basic types of financial instruments: **debt** and **equity**. Debt instruments are loans with a promise to repay the funds with interest, whereas equity securities are shares of stock in a company. As an example, Fig. 1.1 traces the U.S. markets of debt securities between 1985 and 1999. Financial markets are often divided into **money markets**, which concentrate on short-term debt instruments, and **capital markets**, which trade in long-term debt (bonds) and equity instruments (stocks) [767, 799, 828].

Outstanding U.S. Debt Market Securities (U.S. $ billions)

Year	*Municipal*	*Treasury*	*Agency MBSs*	*U.S. corporate*	*Fed agencies*	*Money market*	*Asset-backed*	*Total*
1985	859.5	1,360.2	372.1	719.8	293.9	847.0	2.4	4,454.9
1986	920.4	1,564.3	534.4	952.6	307.4	877.0	3.3	5,159.4
1987	1,010.4	1,724.7	672.1	1,061.9	341.4	979.8	5.1	5,795.4
1988	1,082.3	1,821.3	749.9	1,181.2	381.5	1,108.5	6.8	6,331.5
1989	1,135.2	1,945.4	876.3	1,277.1	411.8	1,192.3	59.5	6,897.6
1990	1,184.4	2,195.8	1,024.4	1,333.7	434.7	1,156.8	102.2	7,432.0
1991	1,272.2	2,471.6	1,160.5	1,440.0	442.8	1,054.3	133.6	7,975.0
1992	1,302.8	2,754.1	1,273.5	1,542.7	484.0	994.2	156.9	8,508.2
1993	1,377.5	2,989.5	1,349.6	1,662.1	570.7	971.8	179.0	9,100.2
1994	1,341.7	3,126.0	1,441.9	1,746.6	738.9	1,034.7	205.0	9,634.8
1995	1,293.5	3,307.2	1,570.4	1,912.6	844.6	1,177.2	297.9	10,403.5
1996	1,296.0	3,459.0	1,715.0	2,055.9	925.8	1,393.8	390.5	11,235.0
1997	1,367.5	3,456.8	1,825.8	2,213.6	1,022.6	1,692.8	518.1	12,097.2
1998	1,464.3	3,355.5	2,018.4	2,462.0	1,296.5	1,978.0	632.7	13,207.4
1999	1,532.5	3,281.0	2,292.0	3,022.9	1,616.5	2,338.2	746.3	14,829.4

Figure 1.1: U.S. debt markets 1985–1999. The Bond Market Association estimates. Sources: Federal Home Loan Mortgage Corporation, Federal National Mortgage Association, Federal Reserve System, Government National Mortgage Association, Securities Data Company, and U.S. Treasury. MBS, mortgage-backed security.

Borrowers and savers can trade directly with each other through the financial markets or direct loans. However, minimum-size requirements, transactions costs, and costly evaluation of the assets in question often prohibit direct trades. Such impediments are remedied by **financial intermediaries**. These are financial institutions that act as middlemen to transfer funds from lenders to borrowers; unlike most firms, they hold only financial assets [660]. Banks, savings banks, savings and loan associations, credit unions, pension funds, insurance companies, mutual funds, and money market funds are prominent examples. Financial intermediaries can lower the minimum investment as well as other costs for savers.

Financial markets can be divided further into primary markets and secondary markets. The **primary market** is often merely a fictional, not a physical, location. Governments and corporations initially sell securities – debt or equity – in the primary market. Such sales can be done by means of public offerings or private placements. A syndicate of investment banks underwrites the debt and the equity by buying them from the issuing entities and then reselling them to the public. Sometimes the investment bankers work on a best-effort basis to avoid the risk of not being able to sell all the securities. Subsequently people trade those instruments in the **secondary markets**, such as the New York Stock Exchange. Existing securities are exchanged in the secondary market.

The existence of the secondary market makes securities more attractive to investors by making them tradable after their purchases. It is the very idea that created the secondary market in mortgages in 1970 by asset securitization [54]. **Securitization** converts assets into traded securities with the assets pledged as collaterals, and these assets can often be removed from the balance sheet of the bank. In so doing,

financial intermediaries transform illiquid assets into liquid liabilities [843]. By making mortgages more attractive to investors, the secondary market also makes them more affordable to home buyers. In addition to mortgages, auto loans, credit card receivables, senior bank loans, and leases have all been securitized [330]. Securitization has fundamentally changed the credit market by making the capital market a major supplier of credit, a role traditionally held exclusively by the banking system.

1.4 Computer Technology

Computer hardware has been progressing at an exponential rate. Measured by the widely accepted integer Standard Performance Evaluation Corporation (SPEC) benchmarks, the workstations improved their performance by 49% per year between 1987 and 1997. The memory technology is equally impressive. The dynamic random-access memory (DRAM) has quadrupled its capacity every 3 years since 1977. Relative performance per unit cost of technologies from vacuum tube to transistor to integrated circuit to very-large-scale-integrated (VLSI) circuit is a factor of 2,400,000 between 1951 and 1995 [717].

Some milestones in the industry include the IBM/360 **mainframe**, followed by Digital's **minicomputers**. (Digital was acquired by Compaq in 1998.) The year 1963 saw the first **supercomputer**, built by Cray (1926–1996) at the Control Data Corporation. Apple II of 1977 is generally considered to be the first **personal computer**. It was overtaken by the IBM Personal Computer in 1981, powered by Intel microprocessors and Microsoft's disk operating system (DOS) [638, 717]. The 1980s also witnessed the emergence of the so-called **massively parallel computers**, some of which had more than 65,000 processors [487]. Parallel computers have also been applied to database applications [247, 263] and pricing complex financial instruments [528, 794, 891]. Because commodity components offer the best performance/cost ratio, personal computers connected by fast networks have been uprooting niche parallel machines from most of their traditional markets [24, 200].

On the software side, high-level programming languages dominate [726]. Although they are easier to program with than low-level languages, it remains difficult to design and maintain complex software systems. In fact, in the 1960s, the software cost of the IBM/360 system already dominated its hardware cost [872]. The current trend has been to use the **object-oriented principles** to encapsulate as much information as possible into the so-called **objects** [101, 466]. This makes software easier to maintain and develop. Object-oriented software development systems are widely available [178].

The revolution fostered by the **graphical user interface** (**GUI**) brought computers to the masses. The omnipotence of personal computers armed with easy-to-use interfaces enabled employees to have access to information and to bypass several layers of management [140]. It also paved the way for the **client/server** concept [736].

Client/server systems consist of components that are logically distributed rather than centralized (see Fig. 1.2). Separate components therefore can be optimized based on their functions, boosting the overall performance/cost ratio. For instance, the **three-tier** client/server architecture contains three parts: user interface, computing (application) server, and data server [310]. Because the user interface demands fewer resources, it can run on lightly configured computers. Best of all, it can potentially be made *platform independent*, thus offering maximum availability of the

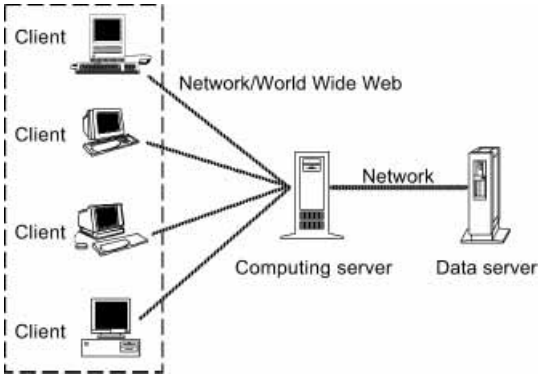

Figure 1.2: Client/server architecture. In a typical three-tier client/server architecture, client machines are connected to the computing server, which in turn is connected to the data server. As the bulk of the computation is with the computing server and the bulk of the data access is with the data server, the client computer can be lightly equipped.

server applications, thanks to Internet-induced developments in the mid-1990s. The server machines, on the other hand, can be powerful **multiprocessors** for the computing servers and machines with high disk throughputs for the data servers. The typical **World Wide Web** (**WWW**) architecture, for instance, is a three-tier client/server system consisting of the **browser**, **Web server**, and **database server**. The object-oriented methodology and client/server architecture can be profitably combined for financial computation [626, 867].

Database management systems are the backbone of information systems [497, 871]. With products from Computer Associates, IBM, Informix, Microsoft, Oracle, and Sybase, the database scenery is dominated by the **relational database** model invented by Codd at IBM in 1970 [216]. In a relational database, data are organized as two-dimensional tables. Consider the following table for storing daily interest rate data.

Attribute	Null?	Type
maturity	NOT NULL	CHAR(10)
ratedate	NOT NULL	DATE
rate	—	DECIMAL(15,8)

Name the table yieldcurve. The structured query language (SQL)[5] statement below can be used to retrieve the two-year U.S. Treasury yield as of December 1, 1994,

```
SELECT rate FROM yieldcurve
        WHERE maturity = '2YR' AND ratedate = '1994-12-01'
```

SQL can also be embedded into general-purpose programming languages. The advancement in the capability of low-cost personal computers and the release of truly multitasking operating systems for them (IBM's OS/2, Microsoft's Windows NT, and Linux) brought client/server database systems to the masses [1, 182, 688, 888]. However, by 1996, the relational database market started to be affected by the Internet momentum [311].

Prototyped in 1991 by Berners-Lee, the WWW is a global information system that provides easy access to Internet resources [63]. It quickly sparked a revolution in the use of the Internet for communications, information, and businesses [655]. A personal computer with access to the WWW – typically through a graphical browser from Microsoft or Netscape (part of America Online) – opens up a window to a world that can be described only as awesome: shopping, stock and bond quotes, online stock trading, up-to-date and historical financial data, financial analysis software, online versions of major newspapers and magazines, academic research results, journal archives and preprints, to mention just a few. The WWW can also form the information network *within* corporations, or **intranet** [733]. The surge of the WWW was one of the major reasons behind the Internet's growing from fewer than 500,000 hosts to more than 10 million between 1990 and 1996 [63, 655] (that number stood at 93 million as of July 2000). In 1998, 100 million people were using the Internet [852]. Even software development strategies were fundamentally changed [488]. These amazing developments are currently reshaping the business and the financial worlds [13, 338, 498, 831].

NOTES

1. Bachelier remained obscure until approximately 1960 when his major work was translated into English. His career problem seems to stem from some technical errors and the topic of his dissertation [637]. "The topic is somewhat remote from those our candidates are in the habit of treating," wrote his advisor, Poincaré (1854–1912) [277]. This is not the first time that ideas in economics have influenced other sciences [426, 660], the most celebrated being Malthus's simultaneous influence on Darwin and Wallace in 1838 [648].
2. Two Nobel laureates in economics, Merton and Scholes, helped found the hedge fund company, Long-Term Capital Management (LTCM). The firm's tools were "computers and powerful mathematics, not intuition nor inside information" [869]. The company underwent a U.S.$3.6 billion forced bailout by 14 commercial and investment banks in September 1998.
3. Distinction is often made between *real* and *financial* investments. What economists mean by investment is the sort that produces real capital formation such as plants, land, and machinery [778]. Investments in this book will be of the financial kind as opposed to the real kind mentioned above. They involve only papers such as stocks and bonds [797].
4. The forex market is the world's largest financial market, in which an estimated U.S.$1.5 trillion was traded in April 1998 [51]. Players are the major commercial and investment banks, with their traders connected by computers, telephones, and other telecommunication equipment [767].
5. The most widely used database language, SQL [315] is derived from SEQUEL (for **S**tructured **E**nglish **QUE**ry **L**anguage), which was designed and implemented at IBM.

CHAPTER TWO

Analysis of Algorithms

In computer science there is no history of critical experiments that decide between the validity of various theories, as there are in physical sciences.

Juris Hartmanis [421]

Algorithms are precise procedures that can be turned into computer programs. A classical example is Euclid's algorithm, which specifies the exact steps toward computing the greatest common divisor. Problems such as the greatest common divisor are therefore said to be **computable**, whereas those that do not admit algorithms are **uncomputable**. A computable problem may have complexity so high that no efficient algorithms exist. In this case, it is said to be **intractable**. The difficulty of pricing certain financial instruments may be linked to their intrinsic complexity [169].

The hardest part of software implementation is developing the algorithm [264]. Algorithms in this book are expressed in an informal style called a **pseudocode**. A pseudocode conveys the algorithmic ideas without getting tied up in syntax. Pseudocode programs are specified in sufficient detail as to make their coding in a programming language straightforward. This chapter outlines the conventions used in pseudocode programs.

2.1 Complexity

Precisely predicting the performance of a program is difficult. It depends on such diverse factors as the machine it runs on, the programming language it is written in, the compiler used to generate the binary code, the workload of the computer, and so on. Although the actual running time is the only valid criterion for performance [717], we need measures of complexity that are machine independent in order to have a grip on the expected performance.

We start with a set of basic operations that are assumed to take one unit of time. Logical comparisons ($\leq$, $=$, $\geq$, and so on) and arithmetic operations of finite precision ($+$, $-$, $\times$, $/$, exponentiation, logarithm, and so on) are among them. The total number of these operations is then used as the total work done by an algorithm, called its **computational complexity**. Similarly, the **space complexity** is the amount of memory space used by an algorithm. The purpose here is to concentrate on the abstract complexity of an algorithm instead of its implementation, which involves so many details that we can never fully take them into account. Complexity serves

Algorithm for searching an element:

input: $x, n, A_i\ (1 \le i \le n)$;
integer k;
for ($k = 1$ to n)
    if [$x = A_k$] return k;
return not-found;

Figure 2.1: Sequential search algorithm.

as a good guide to an algorithm's actual running time. Because space complexity is seldom an issue in this book, the term complexity is used to refer exclusively to computational complexity.

The complexity of an algorithm is expressed as a function of the size of its input. Consider the search algorithm in Fig. 2.1. It looks for a given element by comparing it sequentially with every element in an array of length n. Apparently the worst-case complexity is n comparisons, which occurs when the matching element is the last element of the array or when there is no match. There are other operations to be sure. The for loop, for example, uses a loop variable k that has to be incremented for each execution of the loop and compared against the loop bound n. We do not need to count them because we care about the **asymptotic** growth rate, not the exact number of operations; the derivation of the latter can be quite involved, and its effects on real-world performance cannot be pinpointed anyway [37, 227]. The complexity from maintaining the loop is therefore subsumed by the complexity of the body of the loop.

2.2 Analysis of Algorithms

We are interested in worst-case measures. It is true that worst cases may not occur in practice. But an average-case analysis must assume a distribution on the input, whose validity is hard to certify. To further suppress unnecessary details, we are concerned with the rate of growth of the complexity only as the input gets larger, ignoring constant factors and small inputs. The focus is on the asymptotic growth rate, as mentioned in Section 2.1.

Let $\boldsymbol{R}$ denote the set of real numbers, $\boldsymbol{R}^+$ the set of positive real numbers, and $\boldsymbol{N} = \{0, 1, 2, \ldots, \}$. The following definition lays out the notation needed to formulate complexity.

DEFINITION 2.2.1 We say that $g = O(f)$ if $g(n) \le cf(n)$ for some nonnegative c and sufficiently large n, where $f, g : \boldsymbol{N} \to \boldsymbol{R}^+$.

EXAMPLE 2.2.2 The base of a logarithm is not important for asymptotic analysis because

$$\log_a x = \frac{\log_e x}{\log_e a} = O(\log_e x),$$

where $e = 2.71828\ldots$. We abbreviate $\log_e x$ as $\ln x$.

EXAMPLE 2.2.3 Let $f(n) = n^3$ and $g(n) = 3.5 \times n^2 + \ln n + \sin n$. Clearly, $g = O(f)$ because $g(n)$ is less than n^3 for sufficiently large n. On the other hand, $f \ne O(g)$.

Denote the input size by N. An algorithm runs in **logarithmic time** if its complexity is $O(\log N)$. An algorithm runs in **linear time** if its complexity is $O(N)$. The sequential search algorithm in Fig. 2.1, for example, has a complexity of $O(N)$

because it has $N = n+2$ inputs and carries out $O(n)$ operations. A complexity of $O(N \log N)$ typifies sorting and various divide-and-conquer types of algorithms. An algorithm runs in **quadratic time** if its complexity is $O(N^2)$. Many elementary matrix computations such as matrix–vector multiplication have this complexity. An algorithm runs in **cubic time** if its complexity is $O(N^3)$. Typical examples are matrix–matrix multiplication and solving simultaneous linear equations. An algorithm runs in **exponential time** if its complexity is $O(2^N)$. Problems that *require* exponential time are clearly intractable. It is possible for an exponential-time algorithm to perform well on "typical" inputs, however. The foundations for computational complexity were laid in the 1960s [710].

➤ **Exercise 2.2.1** Show that $f + g = O(f)$ if $g = O(f)$.

➤ **Exercise 2.2.2** Prove the following relations: (1) $\sum_{i=1}^{n} i = O(n^2)$, (2) $\sum_{i=1}^{n} i^2 = O(n^3)$, (3) $\sum_{i=0}^{\log_2 n} 2^i = O(n)$, (4) $\sum_{i=0}^{\alpha \log_2 n} 2^i = O(n^\alpha)$, (5) $n \sum_{i=0}^{n} i^{-1} = O(n \ln n)$.

2.3 Description of Algorithms

Universally accepted mathematical symbols are respected. Therefore $+$, $-$, $\times$, $/$, $<$, $>$, $\leq$, $\geq$, and $=$ mean addition, subtraction, and so on. The symbol $:=$ denotes assignment. For example, $a := b$ assigns the value of b to the variable a. The statement `return` a says that a is returned by the algorithm.

The construct

`for` $(i = a$ `to` $b)$ $\{ \cdots \}$

means that the statements enclosed in braces ({ and }) are executed $b - a + 1$ times, with i equal to $a, a+1, \ldots, b$, in that order. The construct

`for` $(i = a$ `down to` $b)$ $\{ \cdots \}$

means the statements enclosed in braces are executed $a - b + 1$ times, with i equal to $a, a-1, \ldots, b$, in that order. The construct

`while` $[S]$ $\{ \cdots \}$

executes the statements enclosed in braces until the condition S is violated. For example, `while` $[a = b]$ $\{ \cdots \}$ runs until a is not equal to b. The construct

`if` $[S]$ $\{ T_1 \}$ `else` $\{ T_2 \}$

executes T_1 if the expression S is true and T_2 if the expression S is false. The statement `break` causes the current `for` loop to exit. The enclosing brackets can be dropped if there is only a single statement within.

The construct $a[n]$ allocates an array of n elements $a[0], \ldots, a[n-1]$. The construct $a[n][m]$ allocates the following $n \times m$ array (note that the indices start from *zero*, not *one*):

$$\begin{matrix} a[0][0] & \cdots & a[0][m-1] \\ \vdots & \ddots & \vdots \\ a[n-1][0] & \cdots & a[n-1][m-1] \end{matrix}.$$

Although the zero-based indexing scheme is more convenient in many cases, the one-based indexing scheme may be preferred in others. So we use $a[1..n][1..m]$

to denote an array with the following $n \times m$ elements,

$$a[1][1], a[1][2], \ldots, a[n][m-1], a[n][m].$$

Symbols such as $a[\,]$ and $a[\,][\,]$ are used to reference the entire array. Anything following // is treated as comment.

2.4 Software Implementation

Implementation turns an algorithm into a computer program on specific computer platforms. Design, coding, debugging, and module testing are all integral parts of implementation.[1] A key to a productive software project is the reuse of code, either from previous projects or commercial products [650]. The current trend toward object-oriented programming and standardization promises to promote software reuse.

The choice of algorithms in software projects has to be viewed within the context of a larger system. The overall system design might limit the choices to only a few alternatives [791]. This constraint usually arises from the requirements of other parts of the system and very often reflects the fact that most pieces of code are written for an existing system [714].

I now correct a common misconception about the importance of performance. People tend to think that a reduction of the running time from, say, 10 s to 5 s is not as significant as that from 10 min to 5 min. This view rests on the observation that a 5-s difference is not as critical as a 5-min difference. This is wrong. A 5-s difference can be easily turned into a 5-min difference if there are 60 such tasks to perform. A significant reduction in the running time for an important problem is always desirable.

Finally, a word of caution on the term **recursion**. Computer science usually reserves the word for the way of attacking a problem by solving smaller instances of the same problem. Take sorting a list of numbers as an example. One recursive strategy is to sort the first half of the list and the second half of the list separately before merging them. Note that the two sorting subproblems are indeed smaller in size than the original problem. Consistent with most books in finance, however, in this book the term "recursion" is used loosely to mean "iteration." Adhering to the strict computer science usage will usually result in problem formulations that lead to highly inefficient pricing algorithms.

NOTE

1. Software errors can be costly. For example, they were responsible for the crash of the maiden flight of the Ariane 5 that was launched on June 4, 1996, at a cost of half a billion U.S. dollars [606].

> Probably only a person with some mathematical knowledge would think of beginning with 0 instead of with 1.
>
> Bertrand Russell (1872–1970), *Introduction to Mathematical Philosophy*