university
of
tyumen

school
of advanced
studies

Information Technology - advanced

Lecture 5

Low/high-level, interpreted, compiled. Syntax structures.

Fabio Grazioso - *April 2018*

lecture summary

summary

- Object Oriented Programming (from last lecture)
- low level languages
  - machine language
  - assembly language
- high level language
  - main control structures
    - for cycle
    - if-then structure
- Algorithms examples
  - search
  - sort

Object Oriented Programming

Object-oriented Programming

- object-oriented programming is not primarily concerned with the details of program operation. Instead, it deals with the **overall organization of the program**.
- Most individual program statements in C++ are similar to statements in procedural languages, and many are identical to statements in C. Indeed, an entire member function in a C++ program may be very similar to a procedural function in C.
- It is only when you look at the larger context that you can determine whether a statement or a function is part of a procedural C program or an object-oriented C++ program.

Lafore p15

Characteristics of Object-Oriented Languages

- An **object** has the same relationship to a **class** that a variable has to a data type. An object is said to be an instance of a class, in the same way my 1954 Chevrolet is an instance of a vehicle.
- The data items within a class are called data members (or sometimes member data). There can be any number of data members in a class, just as there can be any number of data items in a structure. The data member somedata follows the keyword private, so it can be accessed from within the class, but not from outside.

- Member Functions
- Member functions are functions that are included within a class. In some object-oriented languages member functions are called methods; some writers use this term in C++ as well.
- However, when member functions are small, it is common to compress their definitions this way to save space.
- Functions Are Public, Data Is Private
- Usually the data within a class is private and the functions are public. This is a result of the way classes are used. The data is hidden so it will be safe from accidental manipulation, while the functions that operate on the data are public so they can be accessed from outside the class. However, there is no rule that says data must be private and functions public; in some circumstances you may find you'll need to use private functions and public data.

Lafore p217

Data hiding

- The body of the class contains two unfamiliar keywords: private and public. A key feature of object-oriented programming is data hiding.
- Data hiding, means hiding data from parts of the program that don't need to access it. More specifically, one class's data is hidden from other classes. Data hiding is designed to protect well-intentioned programmers from honest mistakes.

Lafore p217

Examples

- Defining the Class
- Here's the definition (sometimes called a specifier) for the class smallobj, copied from the SMALLOBJ listing:

```
class smallobj                  //define a class
    {
    private:
       int somedata;       //class data
    public:
       void setdata(int d)     //member function to set data
    { somedata = d; }
void showdata()                 //member function to display data
{ cout << "\nData is " << somedata; }


  };
```

Lafore p218

Examples

Using the Class

- Now that the class is defined, let's see how main() makes use of it. We'll see how objects are defined, and, once defined, how their member functions are accessed.

Defining Objects

- The first statement in main()

```
smallobj s1, s2;
```

- defines two objects, s1 and s2, of class smallobj. Remember that the definition of the class smallobj does not create any objects. It only describes how they will look when they are created, just as a structure definition describes how a structure will look but doesn't create any structure variables. It is objects that participate in program operations. Defining an object is similar to defining a variable of any data type: Space is set aside for it in memory.
- Defining objects in this way means creating them. This is also called instantiating them. The term instantiating arises because an instance of the class is created. An object is an instance (that is, a specific example) of a class. Objects are sometimes called instance variables.

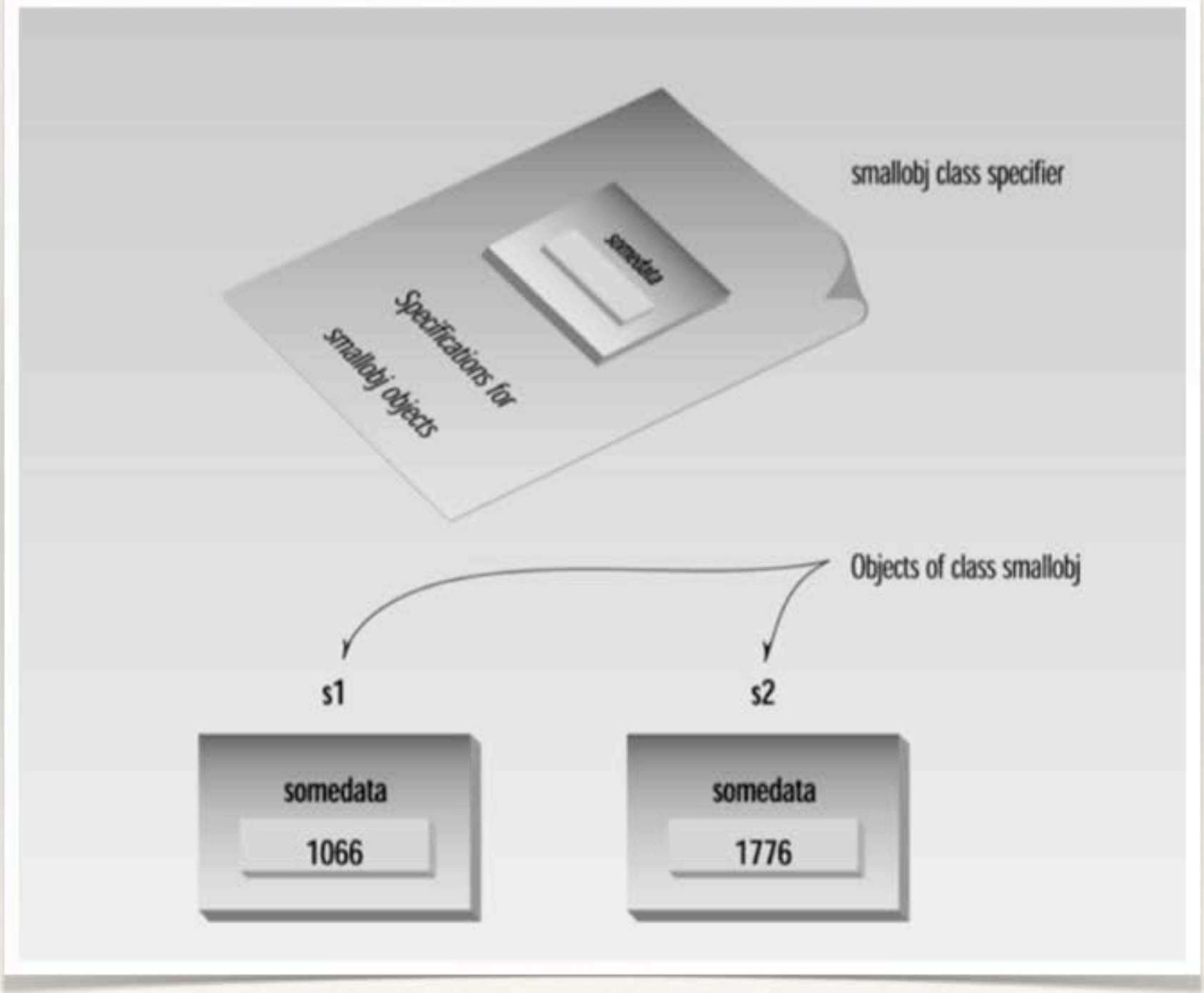
Calling Member Functions

- The next two statements in main() call the member function setdata():

```
s1.setdata(1066);
s2.setdata(1776);
```

Lafore p217

Examples



Lafore p222

low-level

low level code

- it is CPU-specific
- the elementary instructions of the CPU are used
- PROs:
  - fast
  - small memory usage
- CONs:
  - difficult to use
  - non-portable

assembly

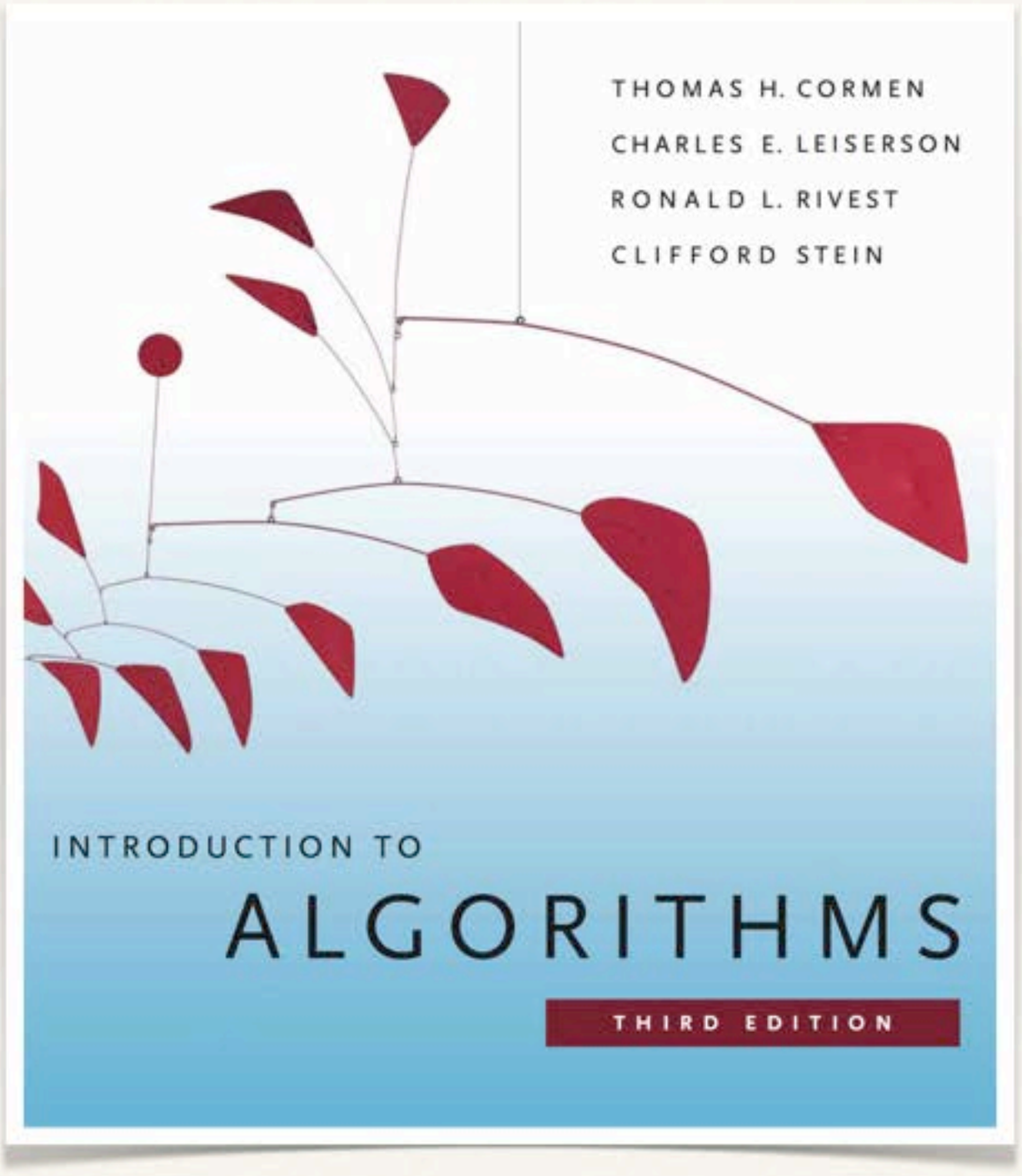
- the first "abstraction" from machine code is to use mnemonic codes.
- in this case, a "translator" will turn mnemonic code into machine code.
- this translator is called "assembler".

examples

- arithmetic shift left
- read memory location

algorithms

reference book



sort algorithm

the sorting problem:

Input

A sequence of n numbers $(a_1; a_2; \ldots; a_n)$.

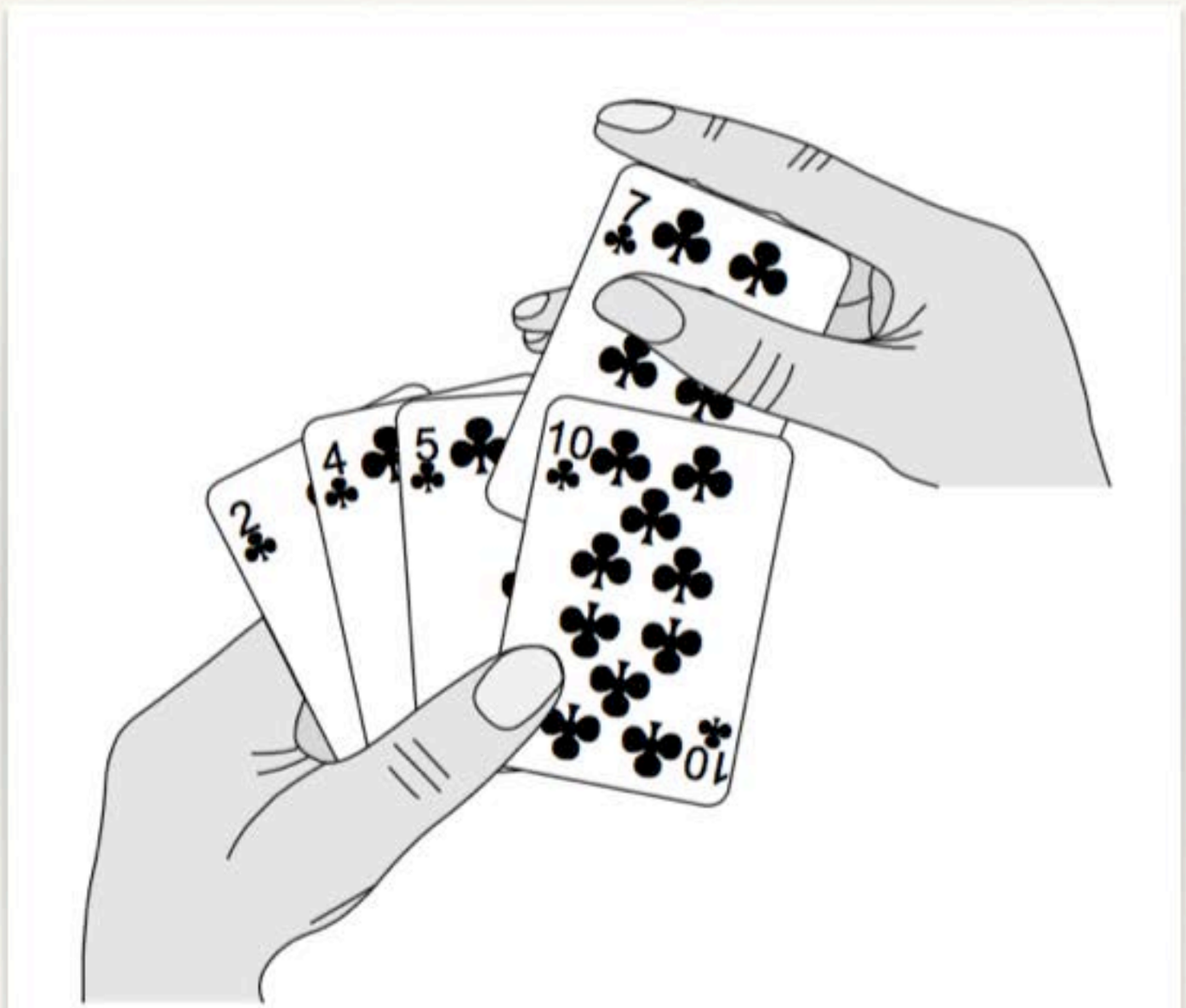
Output

A permutation (reordering) $(a^*_1; a^*_2; \ldots; a^*_n)$ of the input sequence such that:

$$a^*_1 \leq a^*_2 \leq \ldots \leq a^*_n.$$

Cormen p5

insertion sort



Cormen p17

insertion sort

```
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1 .. j − 1].
4       i = j − 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i − 1
8       A[i + 1] = key
```

Cormen p18

insertion sort

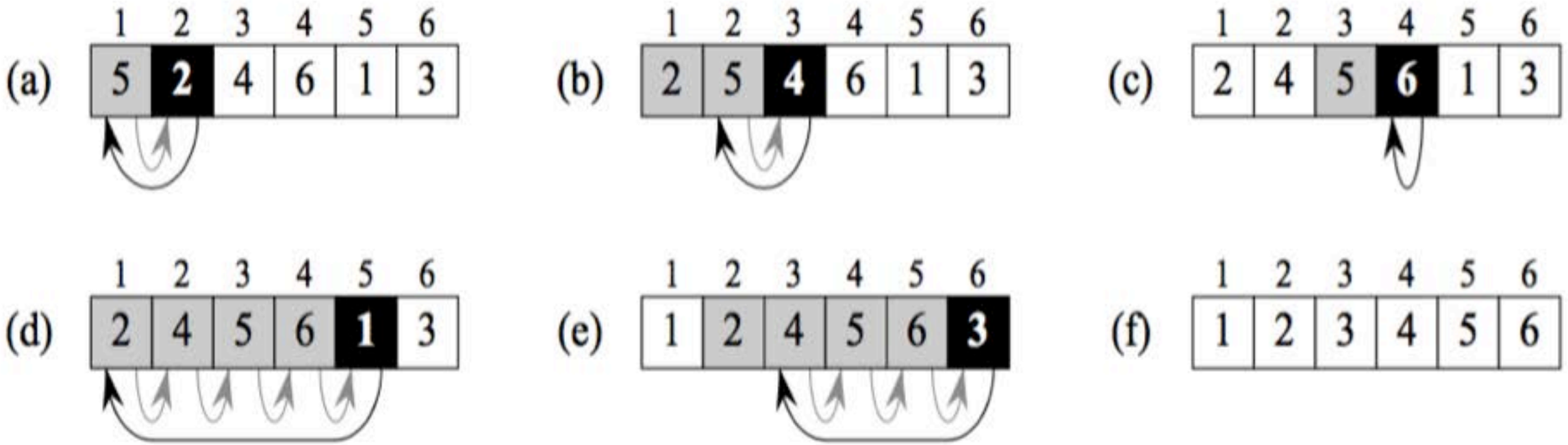


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

Cormen p18

Hard problems

- What is an efficient algorithm?
- One measure of efficiency is speed, i.e., how long an algorithm takes to produce its result.
- Since the speed is most often a function of the input length, we want that this function is linear, or polynomial.
- For sure, not exponential!
- There are some problems, for which no efficient solution is known.

data structure

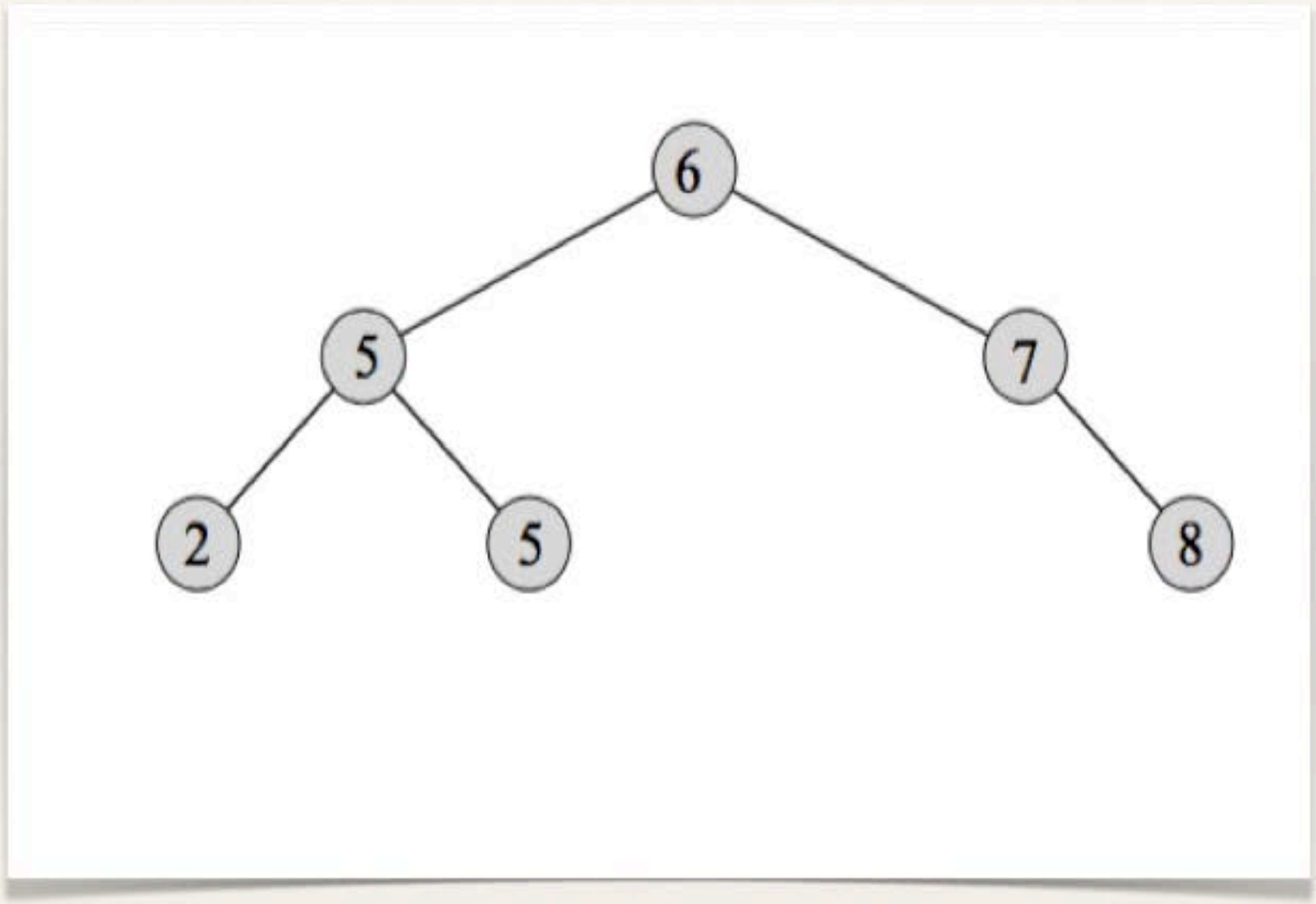
- The indexed sequence is an example of data structures.
- A data structure is a way to store and organize data in order to facilitate access and modifications.
- No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them.

Cormen p9

search algorithm

- a search algorithm is any algorithm which solves the search problem, namely, to retrieve information stored within some data structure (e.g. a list).

binary search tree



Cormen p287