What have we learned?

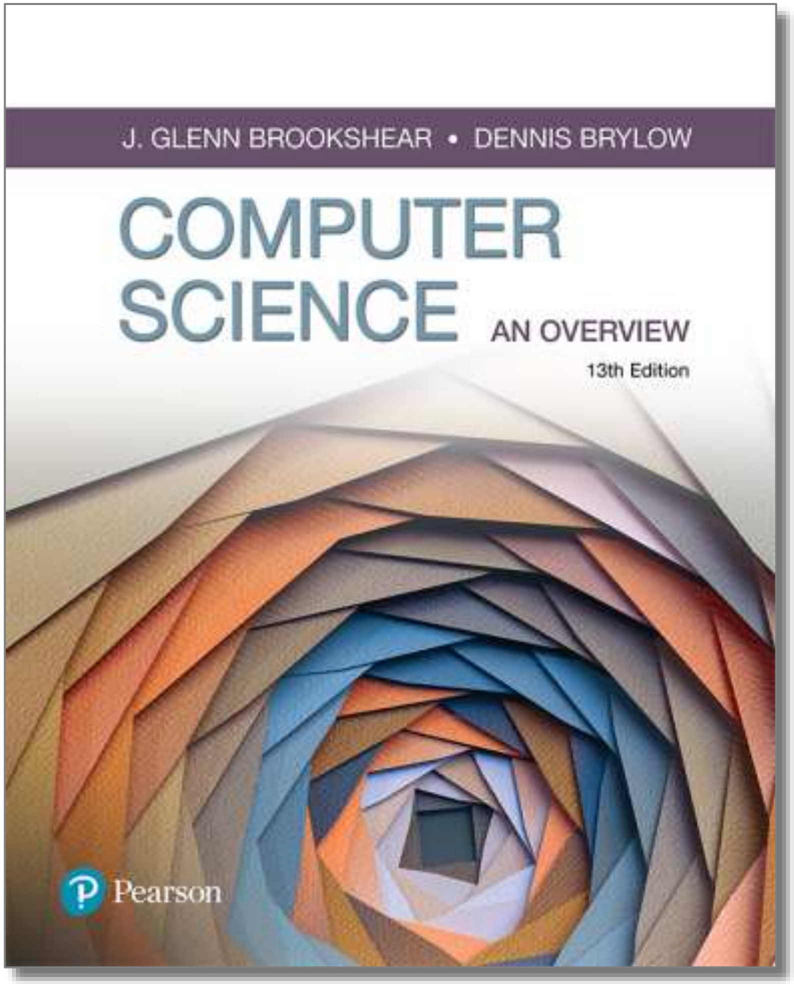
- How does the computer / computer network work?

What are we going to learn?

- How to make computers work?

Computer Science An Overview

13th Edition



Chapter 5

Algorithms

解决问题的方法

- 内功心法， 菜**谱**， 祖**传**秘方
- Algorithm(算法)

Chapter 5: Algorithms

- 5.1 The Concept of an Algorithm
- 5.2 Algorithm Representation
- 5.3 Algorithm Discovery
- 5.4 Iterative Structures
- 5.5 Recursive Structures
- 5.6 Efficiency and Correctness

5.1 The Concept of an Algorithm

- Algorithms from previous chapters
  - Converting from one base to another
  - Correcting errors in data
  - Compression
- Many researchers believe that every activity of the human mind is the result of an algorithm

Algorithm and program

- Algorithm
  - An **ordered** set of **unambiguous**, **executable** steps that defines a **terminating** process to solve the problem
- Program
  - A set of instructions, which describe how computers process data and solve the problem

The Abstract Nature of Algorithms

- There is a difference between an algorithm and its representation.
  - Analogy: difference between a story and a book
- A Program is a representation of an algorithm.
- A Process is the activity of executing an algorithm.

Formal Definition of Algorithm

- An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process
- The steps of an algorithm can be sequenced in different ways
  - Linear (1, 2, 3, …)
  - Parallel (multiple processors)
  - Cause and Effect (circuits)

Formal Definition of Algorithm

- A Terminating Process
  - Culminates with a result
  - Can include systems that run continuously
    - Hospital systems
    - Long Division Algorithm
- A Non-terminating Process
  - Does not produce an answer
  - Chapter 12 : “Non-deterministic Algorithms”

Example: estimation of π

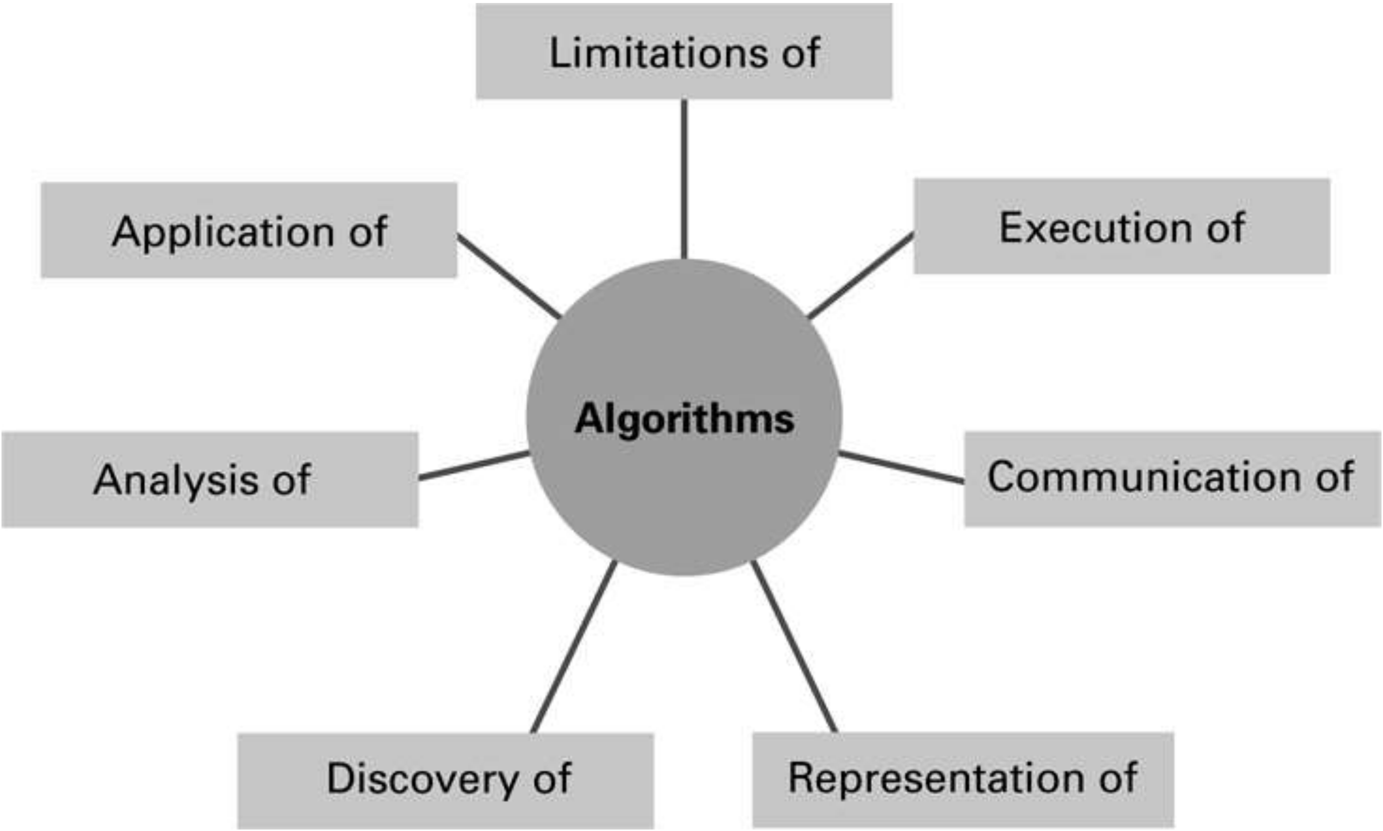
公式 1: $$\frac{\pi}{2}=\frac{2^2}{1\times3}\times\frac{4^2}{3\times5}\times\frac{6^2}{5\times7}\times\frac{8^2}{7\times9}\times\cdots$$

公式 2: $$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots$$

公式 3: $$\frac{\pi}{6}=\frac{1}{\sqrt{3}}\times\left(1-\frac{1}{3\times3}+\frac{1}{3^2\times5}-\frac{1}{3^3\times7}+\cdots\right)$$

- Different algorithms to solve the same problem
- Different efficiency
- Choose the one that can be easily understood and implemented

Central role of algorithms



5.2 Algorithm Representation

- Natural language
- Well-defined Primitives原语
- Flow chart
- Pseudocode伪代码

Figure 5.2 Folding a bird from a square piece of paper

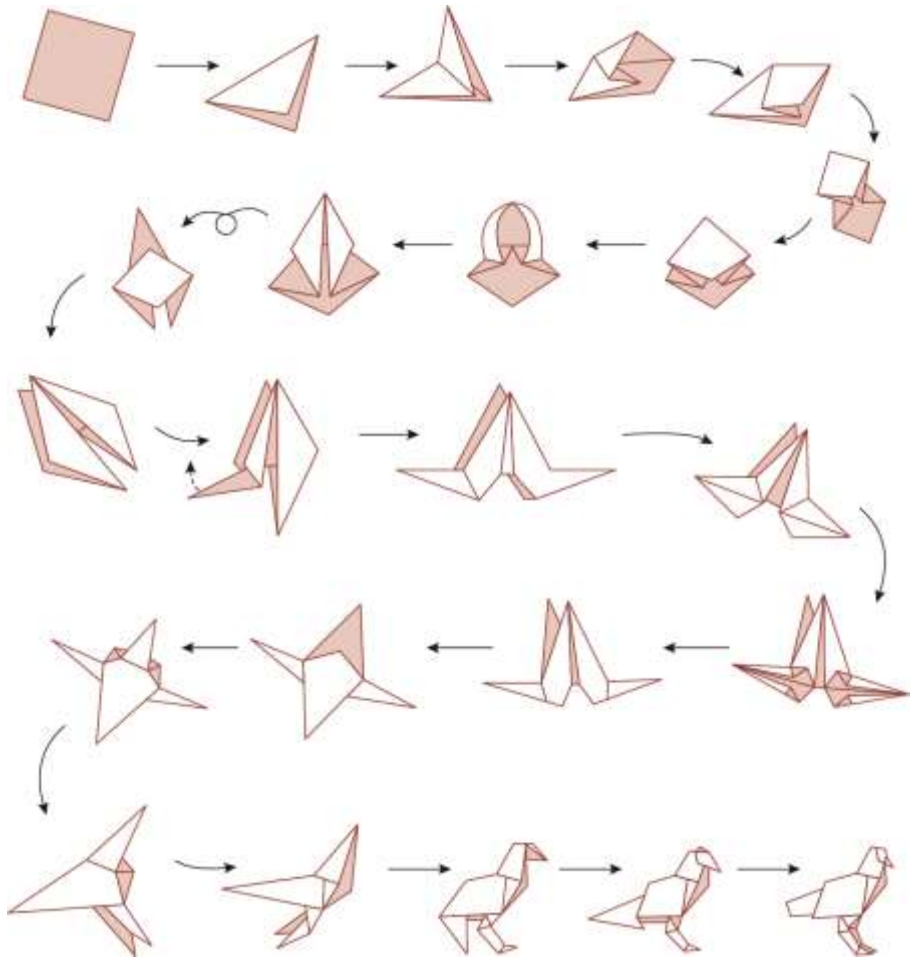
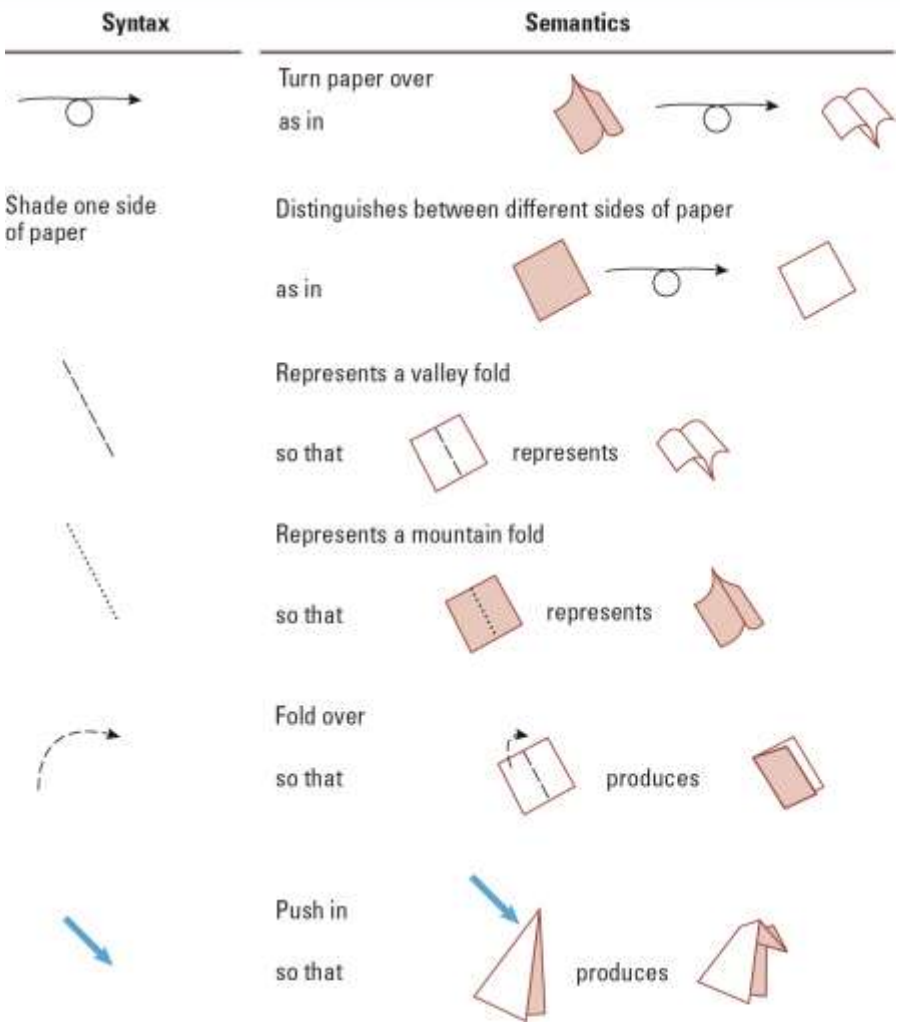
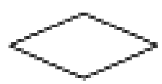
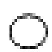


Figure 5.3 Origami primitives

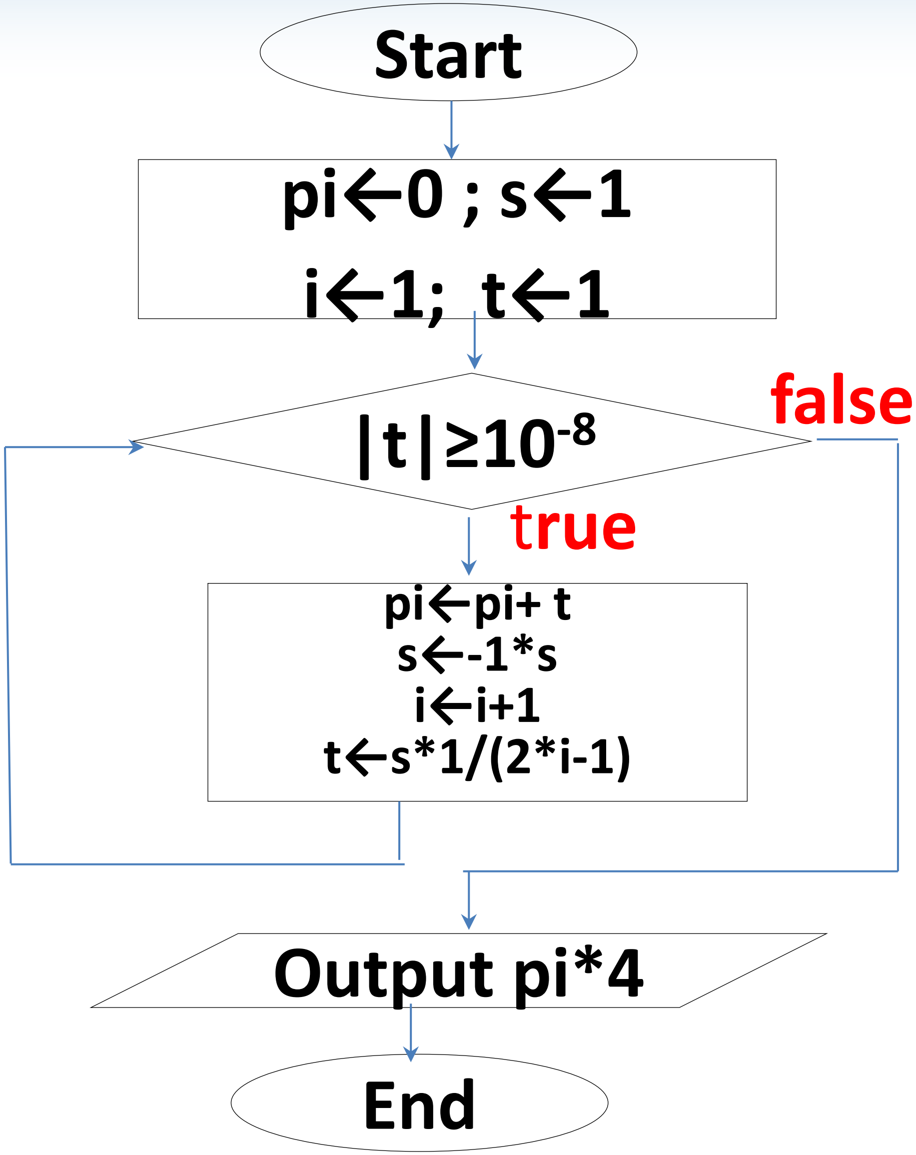


A collection of primitives constitutes a programming language.

Flow chart

符号名称	图形	功能
起止框		表示算法的开始和结束
输入/输出框		表示算法的输入/输出操作
处理框		表示算法中的各种处理操作
判断框		表示算法中的条件判断操作
流程线		表示算法的执行方向
连接点		表示流程图的延续

Computation of Pi



公式 1: $\frac{\pi}{2}=\frac{2^2}{1\times3}\times\frac{4^2}{3\times5}\times\frac{6^2}{5\times7}\times\frac{8^2}{7\times9}\times\cdots$

公式 2: $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots$

公式 3: $\frac{\pi}{6}=\frac{1}{\sqrt{3}}\times(1-\frac{1}{3\times3}+\frac{1}{3^2\times5}-\frac{1}{3^3\times7}+\cdots)$

公式 2: $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots$

Designing a Pseudocode Language

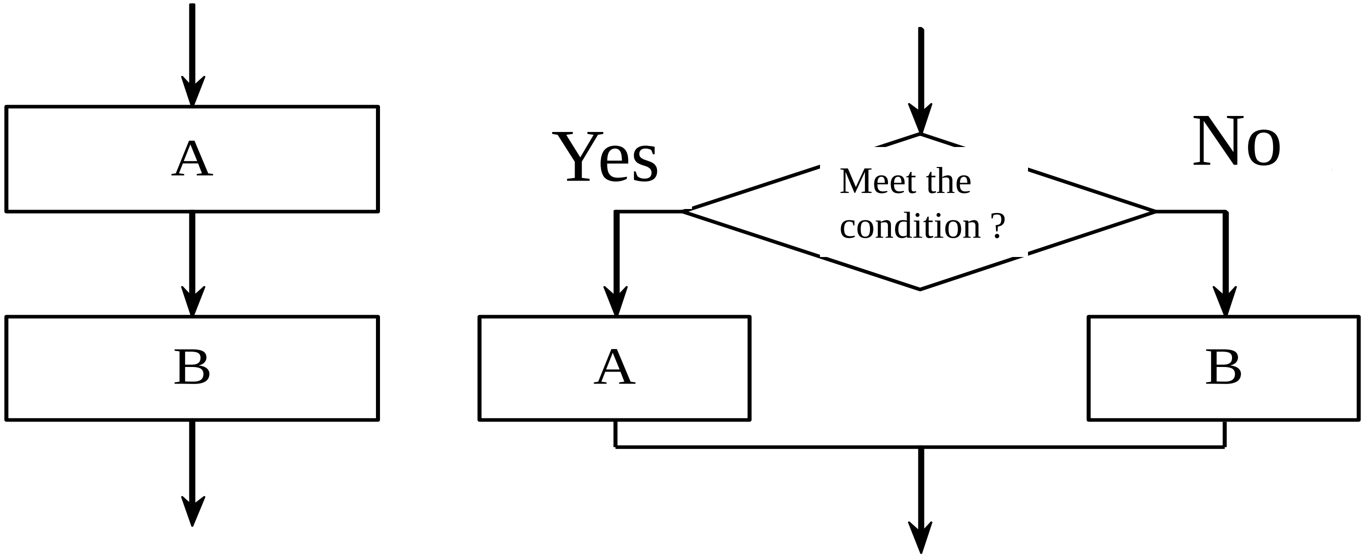
- Choose a common programming language
- Loosen some of the syntax rules
- Allow for some natural language
- Use consistent, concise notation
- We will use a Python-like Pseudocode

How to represent algorithms?

- **Algorithm: operation + control structure**
- **Operation:**
  - Arithmetic: +, -, *, / , etc.
  - Relation: >=, <=, etc.
  - Logic: and, or, not, etc.
  - Data transfer: load, store

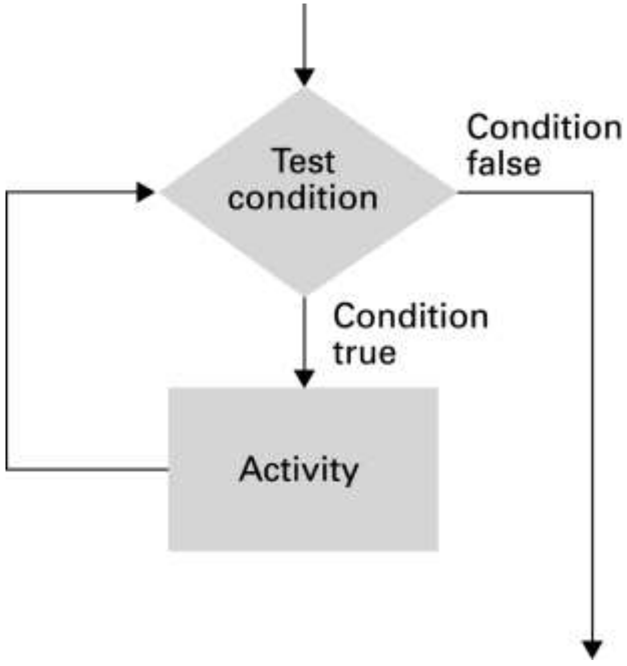
Control structure

- Sequential
- Conditional
- loop

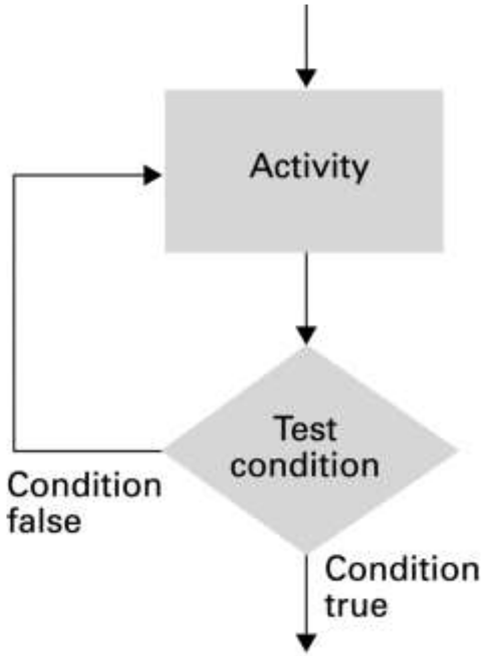


Sequential **Conditional**

Loop structure



While loop structure



Repeat loop structure

Pseudocode Primitives

- Assignment

```
name = expression
```

- example

```
RemainingFunds = CheckingBalance +
                          SavingsBalance
```

Pseudocode Primitives (continued)

- Conditional selection

```
if (condition):
    activity
```

- example

```
if (sales have decreased):
    lower the price by 5%
```

Pseudocode Primitives (continued)

- Conditional selection

```
if (condition):
    activity
else:
    activity
```

- example

```
if (year is leap year):
    daily total = total / 366
else:
    daily total = total / 365
```

Pseudocode Primitives (continued)

- Repeated execution

```
while (condition):
    body
```

- example

```
while (tickets remain to be sold):
    sell a ticket
```

Pseudocode Primitives (continued)

- Indentation shows **nested** conditions

```
if (not raining):
    if (temperature == hot):
        go swimming
    else:
        play golf
else:
    watch television
```

Pseudocode Primitives (continued)

- Define a function

```
def name():
```

- example

```
def ProcessLoan():
```

- Executing a function

```
if (. . .):
    ProcessLoan()
else:
    RejectApplication()
```

Figure 5.4 The procedure Greetings in pseudocode

```
def Greetings():
    Count = 3
    while (Count > 0):
        print('Hello')
        Count = Count - 1
```

Pseudocode Primitives (continued)

- Using parameters

```
def Sort(List):
        .
        .
```

- Executing Sort on different lists

```
Sort(the membership list)
Sort(the wedding guest list)
```

5.3 Algorithm Discovery

- The first step in developing a program
- More of an art than a skill
- A challenging task

Ages of Children Problem

- Person A is charged with the task of determining the ages of B’s three children.
  - B tells A that the product of the children’s ages is 36.
  - A replies that another clue is required.
  - B tells A the sum of the children’s ages.
  - A replies that another clue is needed.
  - B tells A that the oldest child plays the piano.
  - A tells B the ages of the three children.
- How old are the three children?

Figure 5.5

a. Triples whose product is 36

(1,1,36)	(1,6,6)
(1,2,18)	(2,2,9)
(1,3,12)	(2,3,6)
(1,4,9)	(3,3,4)

b. Sums of triples from part (a)

1 + 1 + 36 = 38	1 + 6 + 6 = 13
1 + 2 + 18 = 21	2 + 2 + 9 = 13
1 + 3 + 12 = 16	2 + 3 + 6 = 11
1 + 4 + 9 = 14	3 + 3 + 4 = 10

Chain Separating Problem

- A traveler has a gold chain of seven links.
- He must stay at an isolated hotel for seven nights.
- The rent each night consists of one link from the chain.
- What is the fewest number of links that must be cut so that the traveler can pay the hotel one link of the chain each morning without paying for lodging in advance?

Figure 5.21 Separating the chain using only three cuts

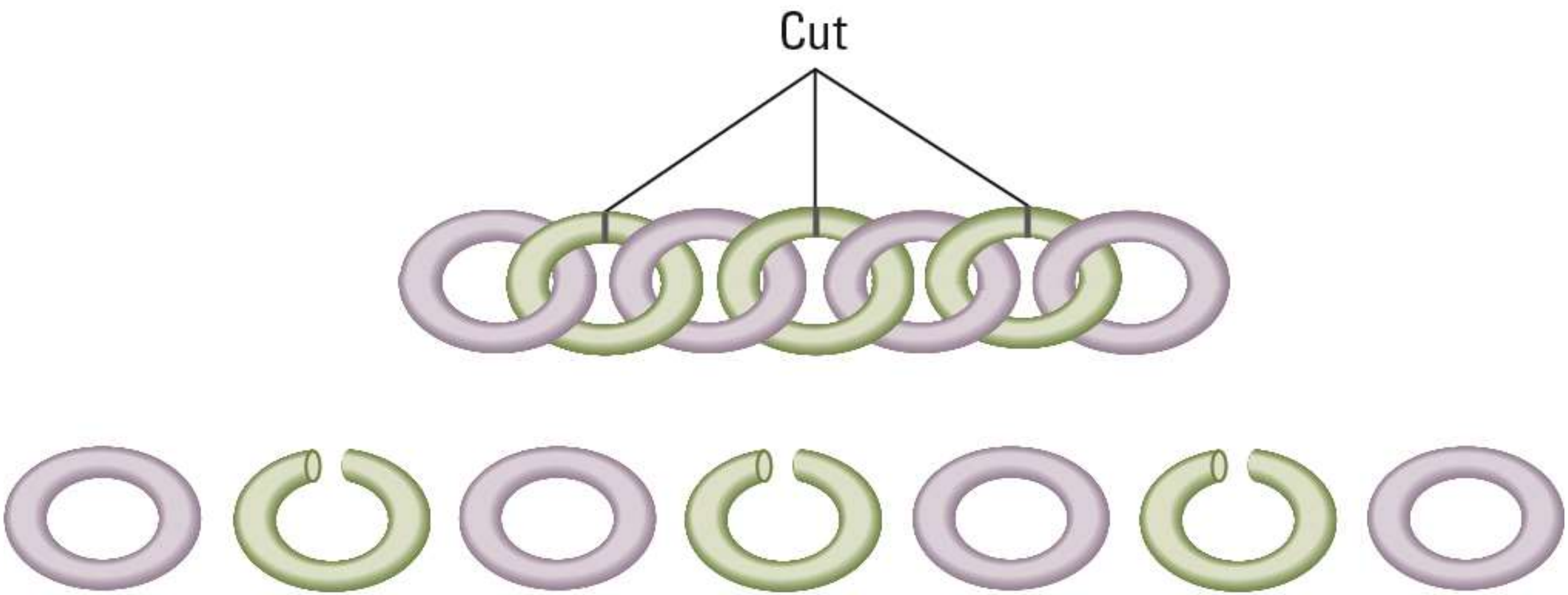
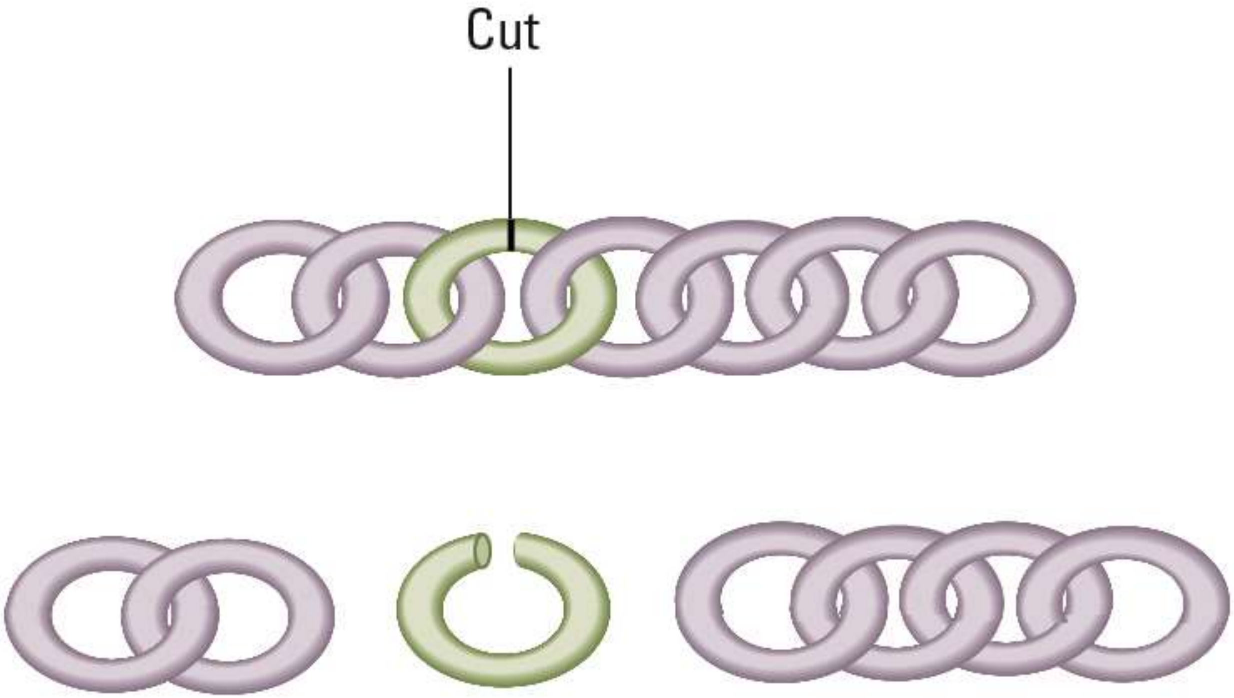


Figure 5.22 Solving the problem with only one cut



5.4 Iterative Structures

- A collection of instructions repeated in a looping manner
- Examples include:
  - Sequential Search Algorithm
  - Insertion Sort Algorithm

Figure 5.8 The while loop structure

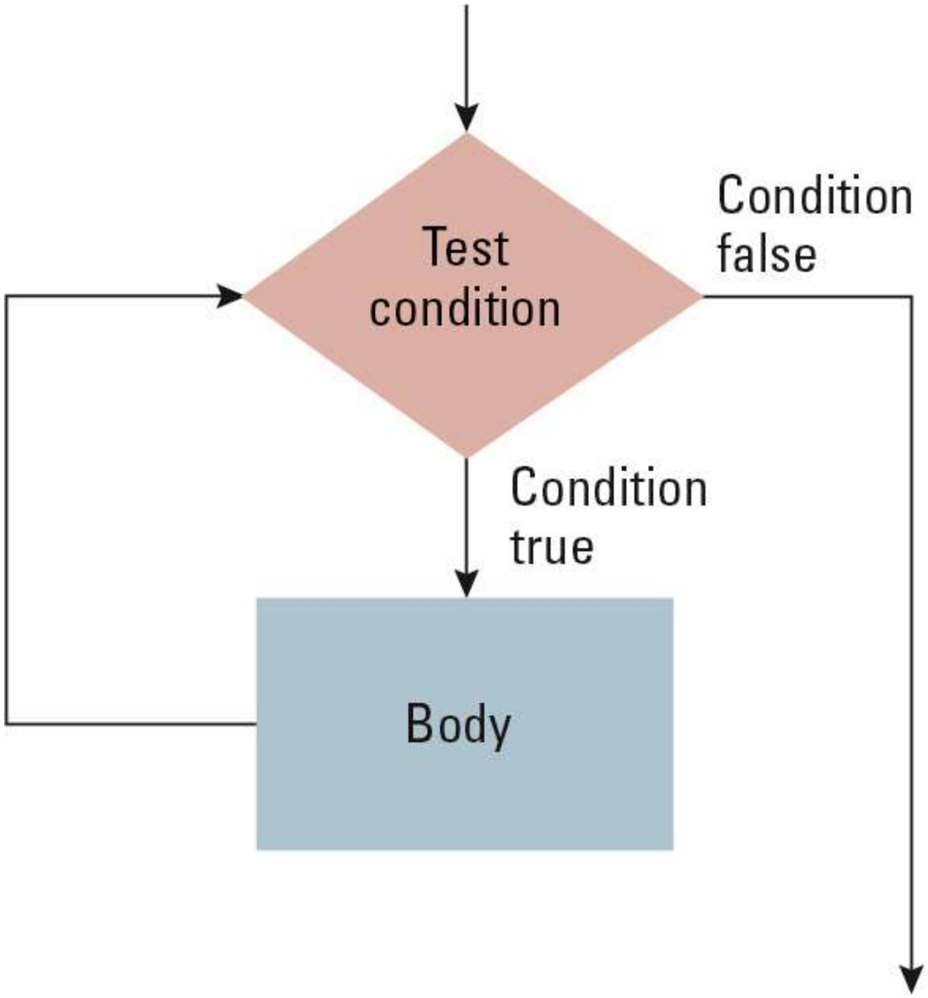
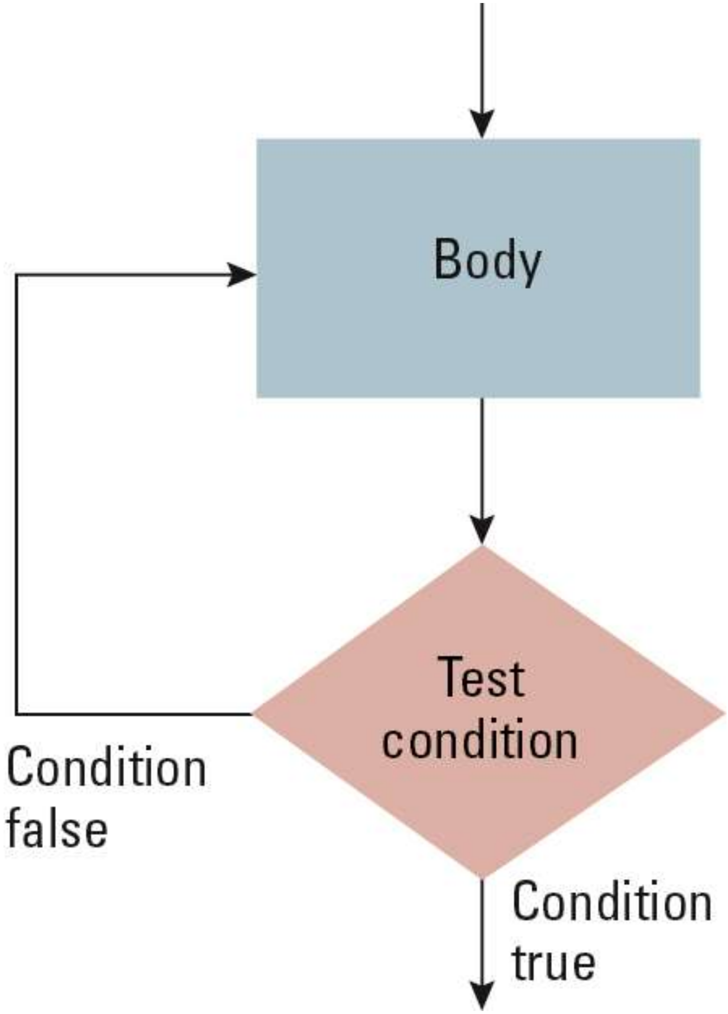


Figure 5.9 The repeat loop structure



Iterative Structures

- Pretest loop:

```
while (condition):
  body
```

- Posttest loop:

```
repeat:
  body
  until(condition)
```

Figure 5.6 The sequential search algorithm in pseudocode


```
def Search (List, TargetValue):
    if (List is empty):
        Declare search a failure
    else:
        Select the first entry in List to be TestEntry
        while (TargetValue > TestEntry and entries remain):
            Select the next entry in List as TestEntry
        if (TargetValue == TestEntry):
            Declare search a success
        else:
            Declare search a failure
```

Figure 5.7 Components of repetitive control

Initialize:	Establish an initial state that will be modified toward the termination condition
Test:	Compare the current state to the termination condition and terminate the repetition if equal
Modify:	Change the state in such a way that it moves toward the termination condition

Sort algorithms

- Insertion sort (插入排序)
- Selection sort (选择排序)
- Quick sort (快速排序)
- Merge sort(归并排序)
- Bubble sort (冒泡排序)

排序算法

插入排序

基本思想：
每次将一个待排序的数据元素，插入到前面已经排好序的数列中的适当位置，使数列依然有序；直到待排序数据元素全部插入完为止。

第一趟：{2、12、}16、30、28、10、16*、20、6、18
第二趟：{2、12、16、}30、28、10、16*、20、6、18
第三趟：{2、12、16、30、}28、10、16*、20、6、18
第四趟：{2、12、16、28、30、}10、16*、20、6、18
第五趟：{2、10、12、16、28、30、}16*、20、6、18
第六趟：{2、10、12、16、16*、28、30、}20、6、18
第七趟：{2、10、12、16、16*、20、28、30、}6、18
第八趟：{2、6、10、12、16、16*、20、28、30、}18
第九趟：{2、6、10、12、16、16*、18、20、28、30}

原始序列{12、2、16、30、28、10、16*、20、6、18}

选择排序

算法基本思想：

对待排序的序列进行**n-1**遍处理：

第1遍处理是从a[1],a[2],……a[n]中选择最小的放在a[1]位置；

第2遍处理是从a[2],a[3],……a[n]中选择最小的放在a[2]位置；

……

第I遍处理是将a[i],a[i+1],……a[n]中最小的数与a[i]交换位置，这样经过第i遍处理后，a[i]是所有的中的第i小。即前i个数就已经排好序了。

N-1遍处理后，剩下的最后一个一定是最大的，不需要再处理了。

排序算法

选择排序

第一趟：{[2、]12、16、30、28、10、16*、20、6、18}
第二趟：{[2、6、]12、16、30、28、10、16*、20、18}
第三趟：{[2、6、10、]12、16、30、28、16*、20、18}
第四趟：{[2、6、10、12、]16、30、28、16*、20、18}
第五趟：{[2、6、10、12、16、]30、28、16*、20、18}
第六趟：{[2、6、10、12、16、16*、]30、28、20、18}
第七趟：{[2、6、10、12、16、16*、18、]30、28、20}
第八趟：{[2、6、10、12、16、16*、18、20、]30、28}
第九趟：{2、6、10、12、16、16*、18、20、28、30}

原始序列{12、2、16、30、28、10、16*、20、6、18}

快速排序

- 高效
- 分治思想：
  - 先保**证**列表的前半部分都小于后半部分，然后分别**对**前半部分和后半部分排序
  - 按此方法**对这**两部分数据分别**进**行快速排序
- 算法的高效与否与列表中数字**间**的比**较**次数有直接的关系
- 前半部分的任何一个数不再跟后半部分的数**进**行比**较**

快速排序

初始

{49 38 65 97 76 13 27 49}

一次划分之后

{27 38 13} **49** {76 97 65 49}

序列左继续排序

{13} **27** {38} 49 {76 97 65 49}

(结束) (结束)

序列右继续排序

{49 65} **76** {97}

(结束)

49 {65}

(结束)

有序序列

{13 27 38 49 49 65 76 97}

排序算法

快速排序

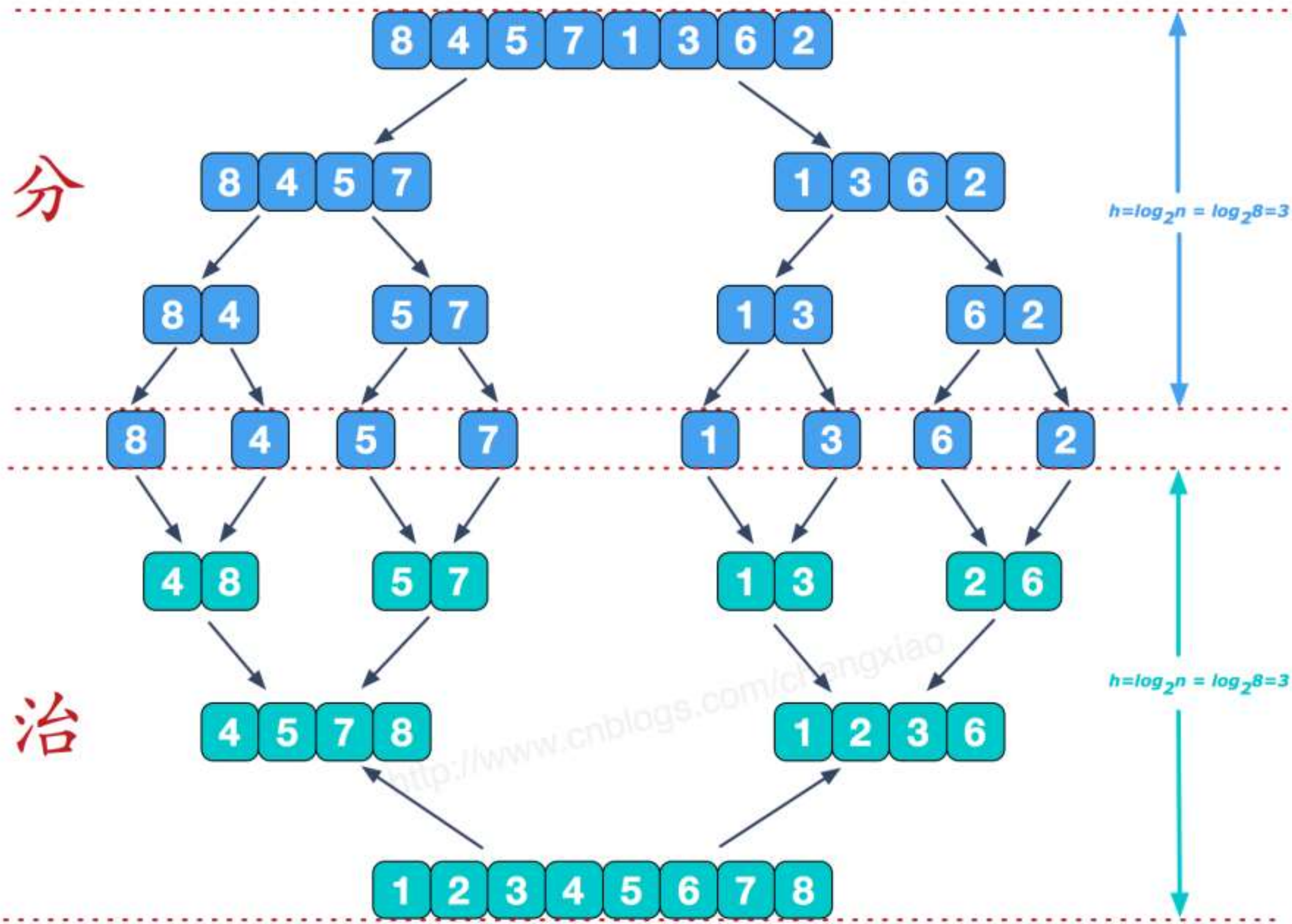
第一趟：{6、2、10、12、28、30、16*、20、16、18} pivot=12
第二趟：{2、6、10、12、28、30、16*、20、16、18} pivot=6
第三趟：{2、6、10、12、18、16、16*、20、28、30} pivot=28
第四趟：{2、6、10、12、16*、16、18、20、28、30} pivot=18
第五趟：{2、6、10、12、16、16*、18、20、28、30}

原始序列{12、2、16、30、28、10、16*、20、6、18}

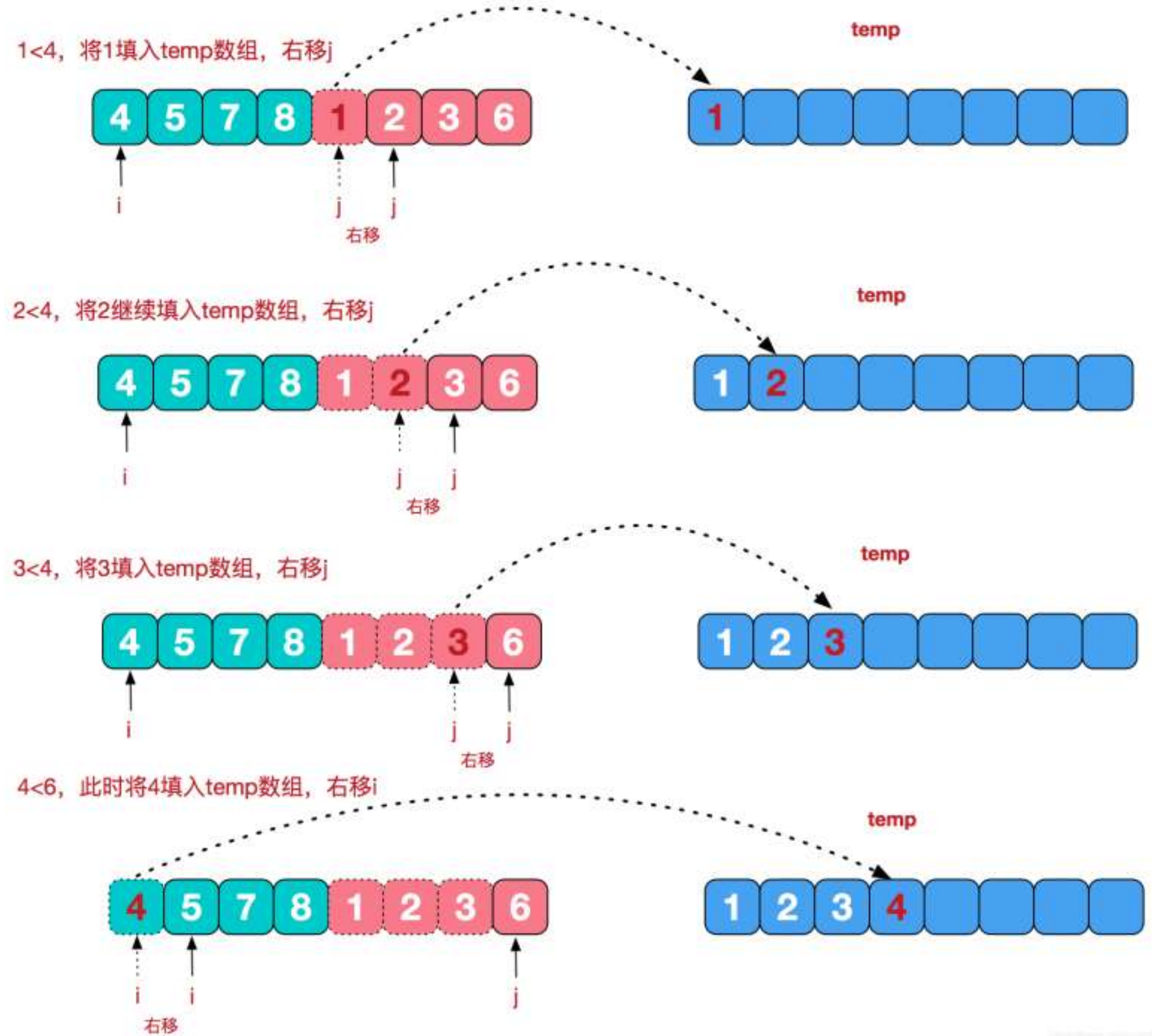
排序算法

归并排序

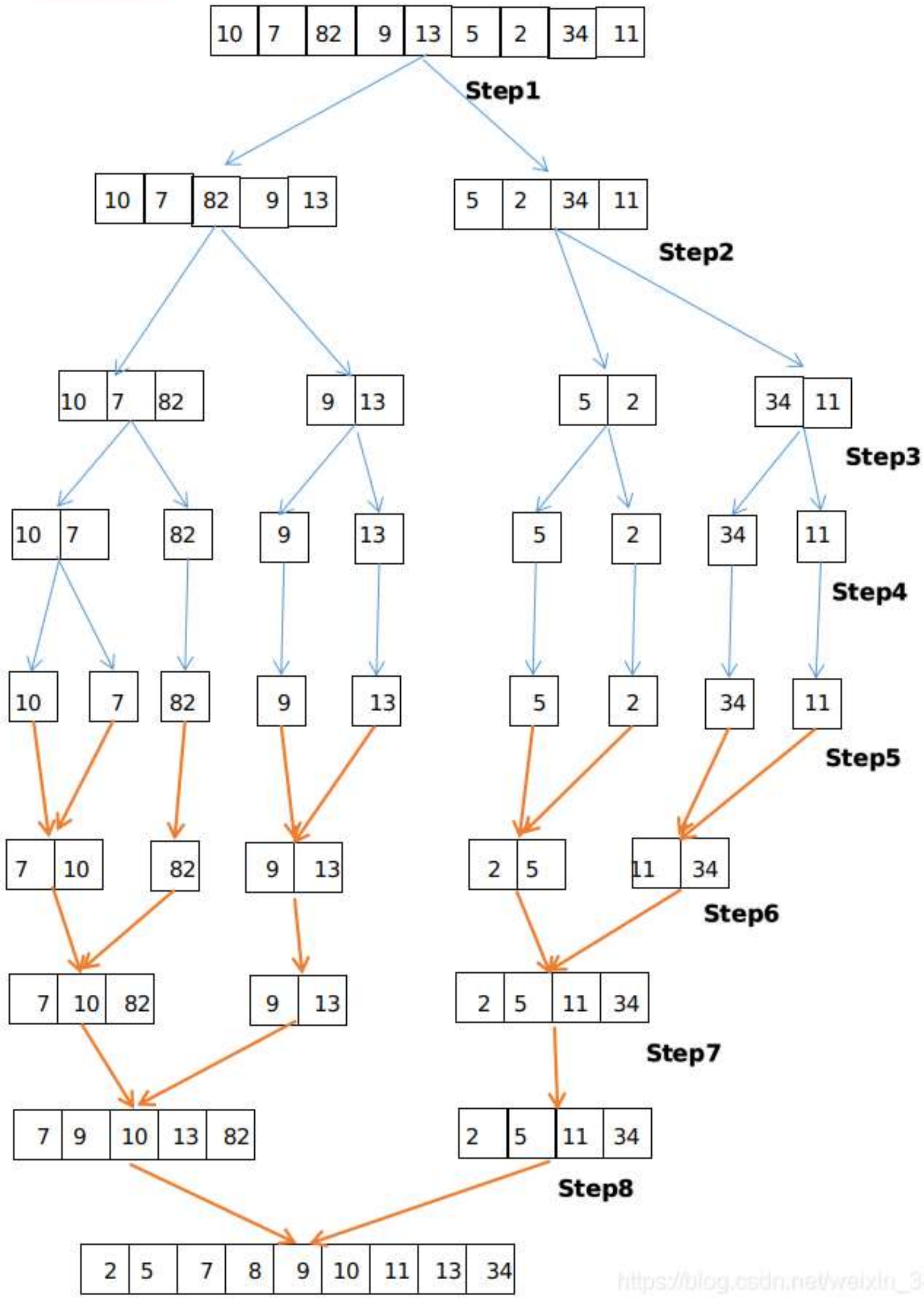
该算法采取分治(Divide and Conquer)的思想。合并算法是将两个(或两个以上)有序表合并成一个新的有序表，即把待排序的序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。



8 4 5 7 1 3 6 2
分
8 4 5 7
1 3 6 2
h=log2n = log28=3
8 4
5 7
1 3
6 2
8
4
5
7
1
3
6
2
4 8
5 7
1 3
2 6
治
4 5 7 8
1 2 3 6
h=log2n = log28=3
1 2 3 4 5 6 7 8


1<4，将1填入temp数组，右移j
temp
4 5 7 8 1 2 3 6
1
i
j
j
右移
2<4，将2继续填入temp数组，右移j
temp
4 5 7 8 1 2 3 6
1 2
i
j
j
右移
3<4，将3填入temp数组，右移j
temp
4 5 7 8 1 2 3 6
1 2 3
i
j
j
右移
4<6，此时将4填入temp数组，右移i
temp
4 5 7 8 1 2 3 6
1 2 3 4
i
i
j
右移

mergeSort


10 7 82 9 13 5 2 34 11
Step1
10 7 82 9 13
5 2 34 11
Step2
10 7 82
9 13
5 2
34 11
Step3
10 7
82
9
13
5
2
34
11
Step4
10
7
82
9
13
5
2
34
11
Step5
7 10
82
9 13
2 5
11 34
Step6
7 10 82
9 13
2 5 11 34
Step7
7 9 10 13 82
2 5 11 34
Step8
2 5 7 8 9 10 11 13 34

排序算法

二路归并排序

第一趟：{[2、12、] [16、30、] [10、28、] [16*、20、] [6、18]}
第二趟：{[2、12、16、30、] [10、16*、20、28、] [6、18]}
第三趟：{[2、10、12、16、16*、20、28、30、] [6、18]}
第四趟：{2、6、10、12、16、16*、18、20、28、30}

原始序列{12、2、16、30、28、10、16*、20、6、18}

冒泡排序

基本思想：（从小到大排序）

将待排序的数据看作竖派排的一列”气泡“，小的数据比较轻，从而要上浮。

共进行**n-1**遍处理，每一遍处理，就是从底向上检查序列，如果相邻的两个数据顺序不对，即轻（小）的在下面，就交换他们的位置。

第一遍处理完后，“最轻”的就浮到上面。

第二遍处理完后，“次轻”的就浮到上面。

共需要**n-1**遍处理即完成排序。

Bubble Sort!

3, 4, 2, 1

3, **2**, **4**, 1

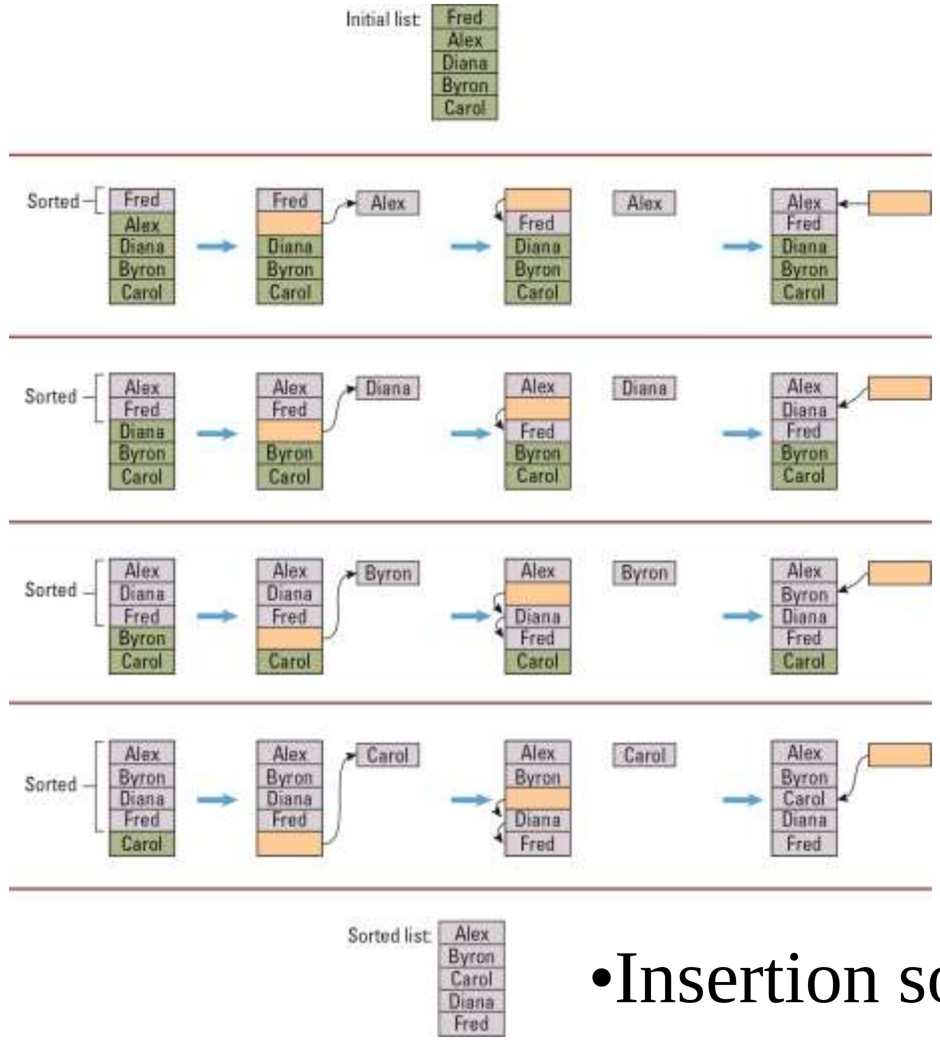
2, **3**, 4, 1

2, 3, **1**, **4**

2, **1**, **3**, 4

1, **2**, 3, 4

Figure 5.10 Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically



•Insertion sort (插入排序)

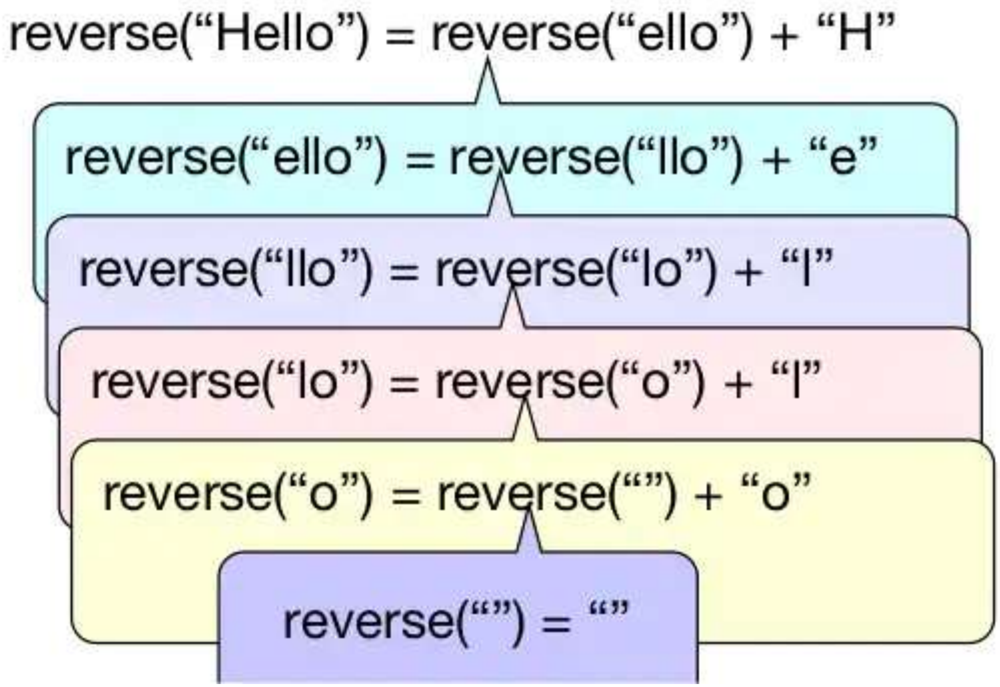
Figure 5.11 The insertion sort algorithm expressed in pseudocode

```
def Sort(List):
    N = 2
    while (N <= length of List):
        主元Pivot = Nth entry in List
        Remove Nth entry leaving a hole in List
        while (there is an Entry above the
                  hole and Entry > Pivot):
            Move Entry down into the hole leaving
            a hole in the list above the Entry
        Move Pivot into the hole
        N = N + 1
```

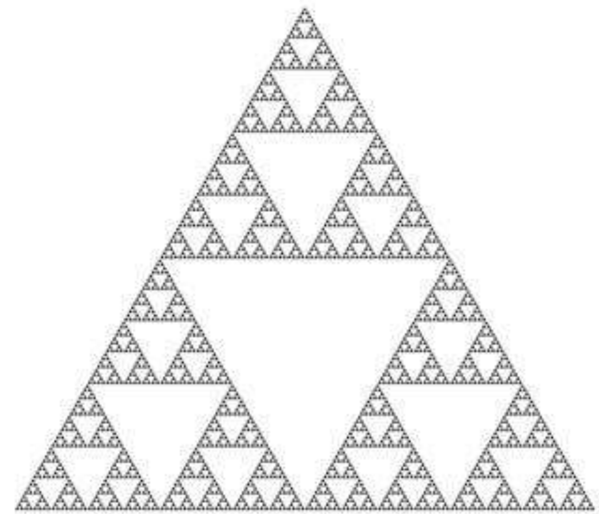
5.5 Recursive Structures递归结构

- The execution of a procedure leads to another execution of the procedure.
- Repeating the set of instructions as a subtask of itself.
- Multiple activations of the procedure are formed, all but one of which are waiting for other activations to complete.

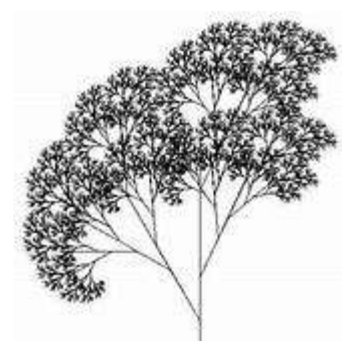
Recursive Algorithm



Properties of all recursive algorithms



- A recursive algorithm solves the large problem by using its solution to a simpler sub-problem
  - divide and conquer approach
- Eventually the sub-problem is simple enough that it can be solved without applying the algorithm to it recursively
  - This is called the ‘base case’



Example: Count Down to Zero

```
>>> def countdown(n):
...     print(n)
...     if n == 0:
...         return             # Terminate recursion
...     else:
...         countdown(n - 1)   # Recursive call
...

>>> countdown(5)
5
4
3
2
1
0
```

Recursion: example

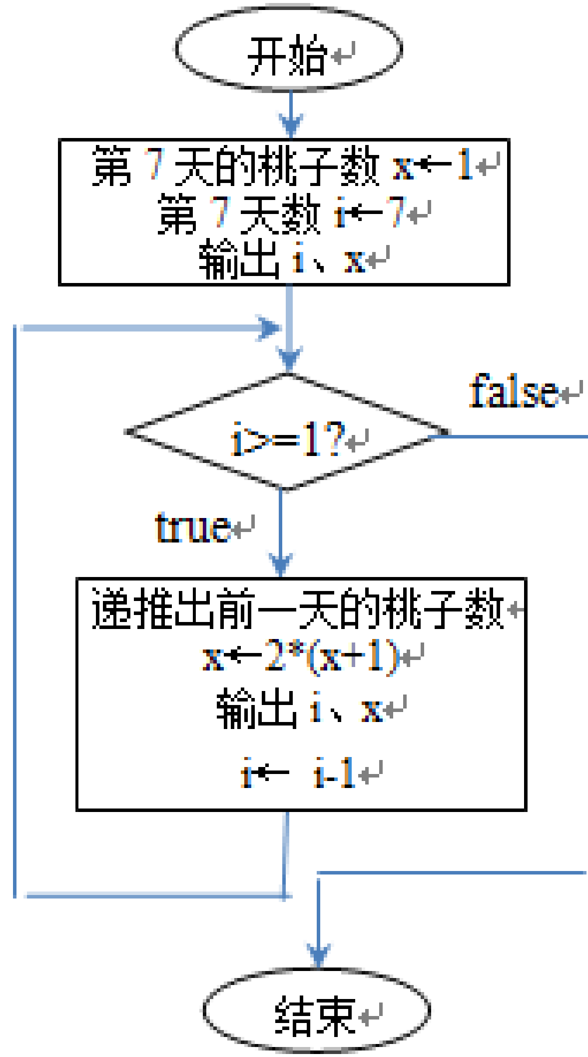
- **例：猴子吃桃问题**

小猴有桃若干，当天吃掉一半多一个；第二天接着吃了剩下的桃子的一半多一个；以后每天都吃尚存桃子的一半零一个，到第7天早上只剩下1个了，问小猴原有多少个桃子？

- 设第n天的桃子为x_n，它是前一天的桃子数的一半少1个，即

$x_{n-1}=(x_n+1)\times 2$ （**递推公式**）

```
第 7 天的桃子数为：1只
第 6 天的桃子数为：4只
第 5 天的桃子数为：10只
第 4 天的桃子数为：22只
第 3 天的桃子数为：46只
第 2 天的桃子数为：94只
第 1 天的桃子数为：190只
```

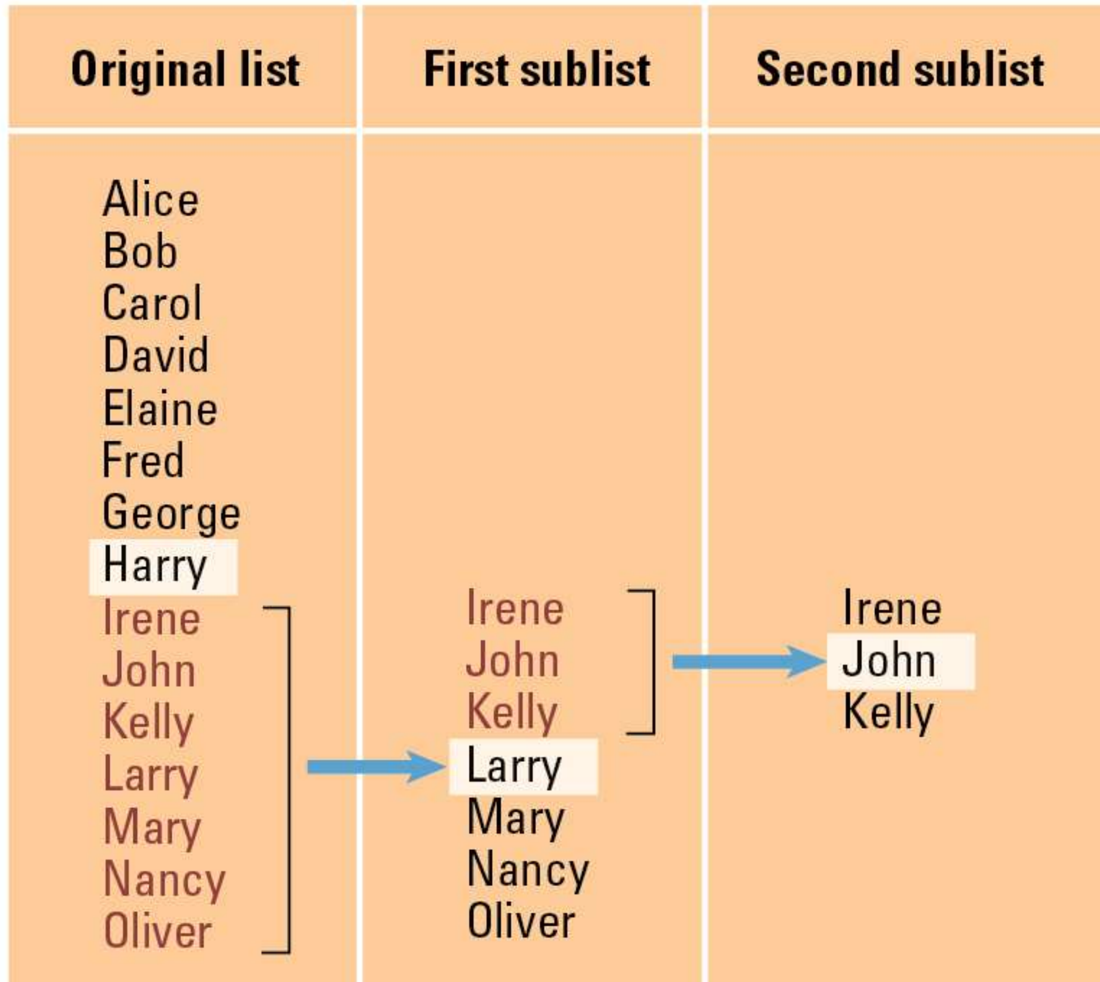


数字炸弹

The Binary Search Algorithm

Figure 5.12 Applying our strategy to search a list for the entry John

The Binary Search Algorithm



二分查找

- 二分查找又称折半查找
- 优点：比较次数少，查找速度快，平均性能好；
- 缺点：要求待查表为有序表，且插入删除困难。因此，折半查找方法适用于不经常变动而查找频繁的有序列表

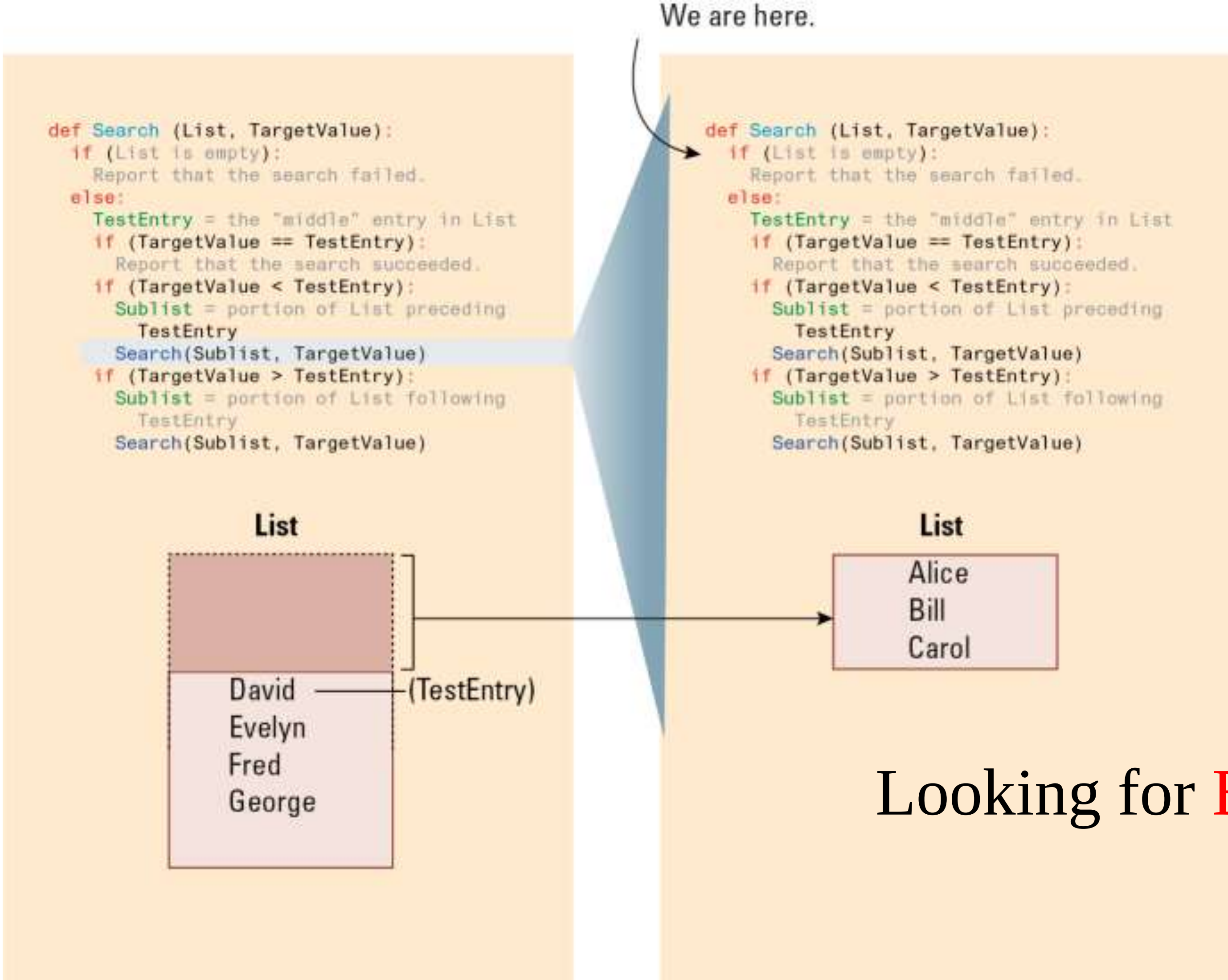
Figure 5.13 A first draft of the binary search technique

```
if (List is empty):
    Report that the search failed
else:
    TestEntry = middle entry in the List
    if (TargetValue == TestEntry):
        Report that the search succeeded
    if (TargetValue < TestEntry):
        Search the portion of List preceding TestEntry for
        TargetValue, and report the result of that search
    if (TargetValue > TestEntry):
        Search the portion of List following TestEntry for
        TargetValue, and report the result of that search
```

Figure 5.14 The binary search algorithm in pseudocode

```
def Search(List, TargetValue):
    if (List is empty):
        Report that the search failed
    else:
        TestEntry = middle entry in the List
        if (TargetValue == TestEntry):
            Report that the search succeeded
        if (TargetValue < TestEntry):
            Sublist = portion of List preceding TestEntry
            Search(Sublist, TargetValue)
        if (TargetValue > TestEntry):
            Sublist = portion of List following TestEntry
            Search(Sublist, TargetValue)
```

Figure 5.15 Recursively Searching

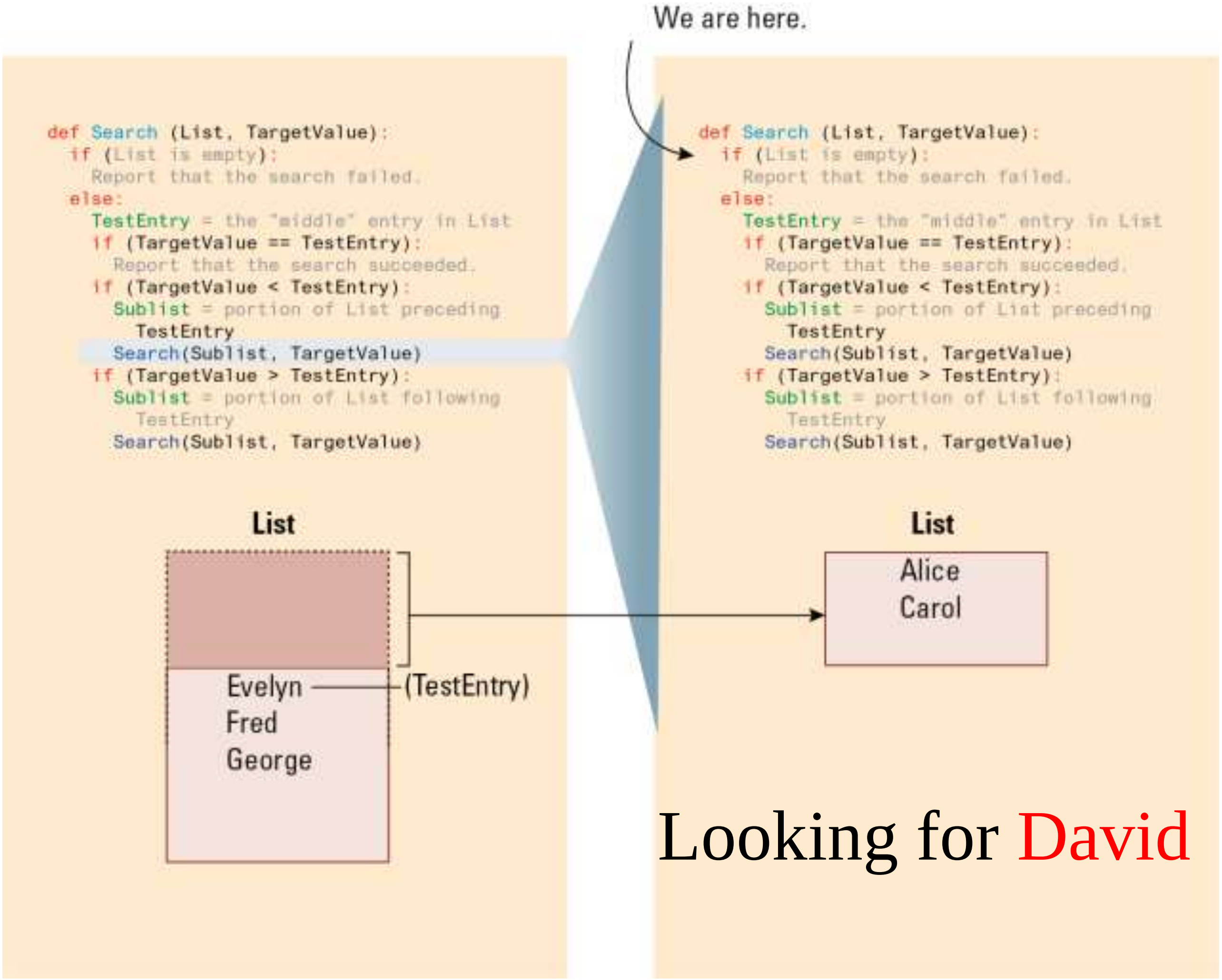


Looking for Bill

Alice
Bill
Carol
David
Evelyn
Fred
George

Figure 5.16 Second Recursive Search

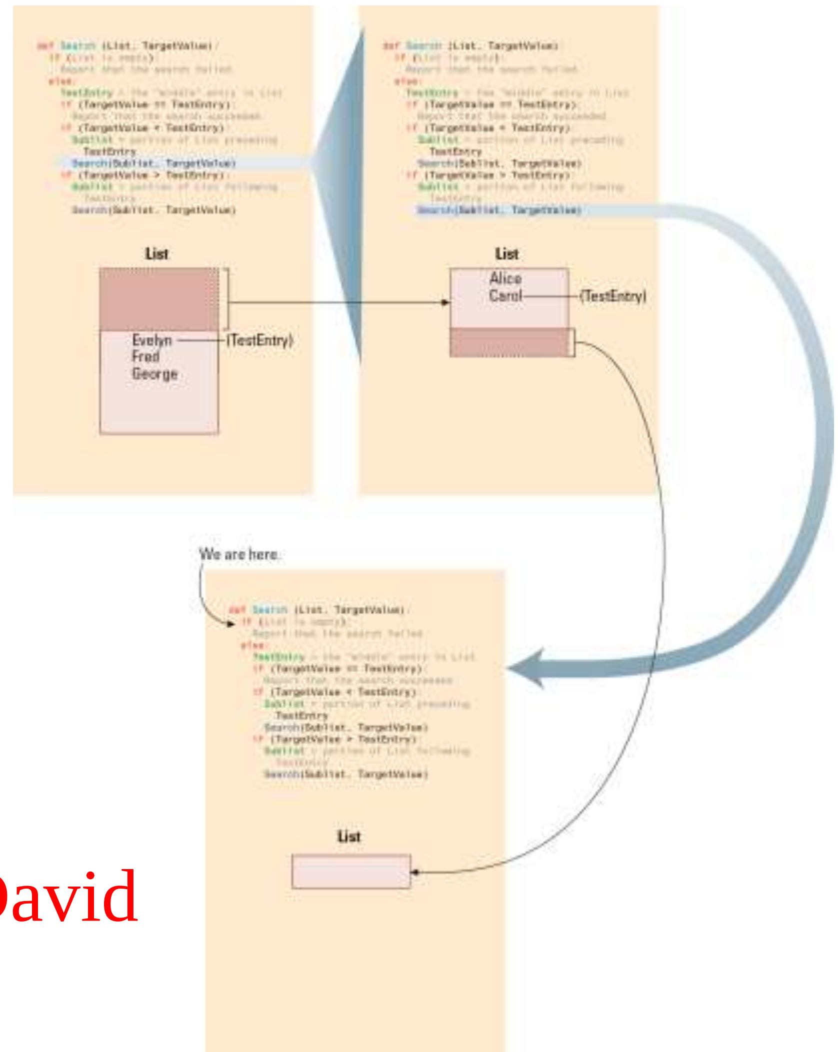
Alice
Carol
Evelyn
Fred
George



Looking for David

Figure 5.17 Second Recursive Search, Second Snapshot

Alice
Carol
Evelyn
Fred
George



Looking for David

Recursive Control

- Requires initialization, modification, and a test for termination (base case)
- Provides the illusion of multiple copies of the function, created dynamically in a telescoping manner
- Only one copy is actually running at a given time, the others are waiting

5.6 Efficiency and Correctness

- The choice between efficient and inefficient algorithms can make the difference between a practical solution and an impractical one
- The correctness of an algorithm is determined by reasoning formally about the algorithm, not by testing its implementation

Algorithm Efficiency

- Measured as number of instructions executed
- Big O notation: Used to represent efficiency classes
  - Example: Insertion sort is in $O(n^2)$
- Best, worst, and average case analysis

算法复杂度

- 同一**问题**可用不同算法解决，而一个算法的**质**量**优**劣将影响到算法乃至程序的效率。
- 算法分析的目的在于**选择**合适算法和改**进**算法。
- 一个算法的**评**价主要从**时间**复**杂**度和空**间**复**杂**度来考**虑**

算法的时间复杂度

- 算法的时间复杂度：指执行算法所需要的计算工作量
  - 时间频度T(n)：一个算法中的语句执行次数称为语句频度或时间频度，n为问题的规模
  - 若有某个辅助函数f(n),使得当n趋近于无穷大时，T(n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=O(f(n)),称O(f(n)) 为算法的渐进时间复杂度，简称时间复杂度
  - 如T(n)=n^2+3n+4与T(n)=4n^2+2n+1它们的频度不同，但时间复杂度相同，都为O(n^2)

Figure 5.18 Applying the insertion sort in a worst-case situation

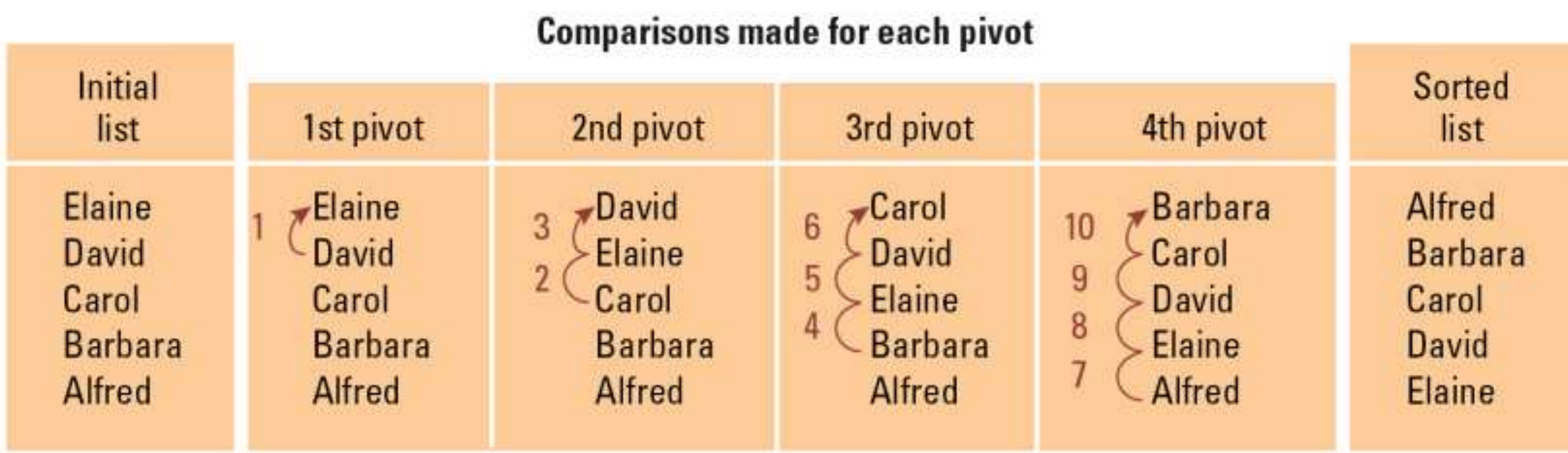
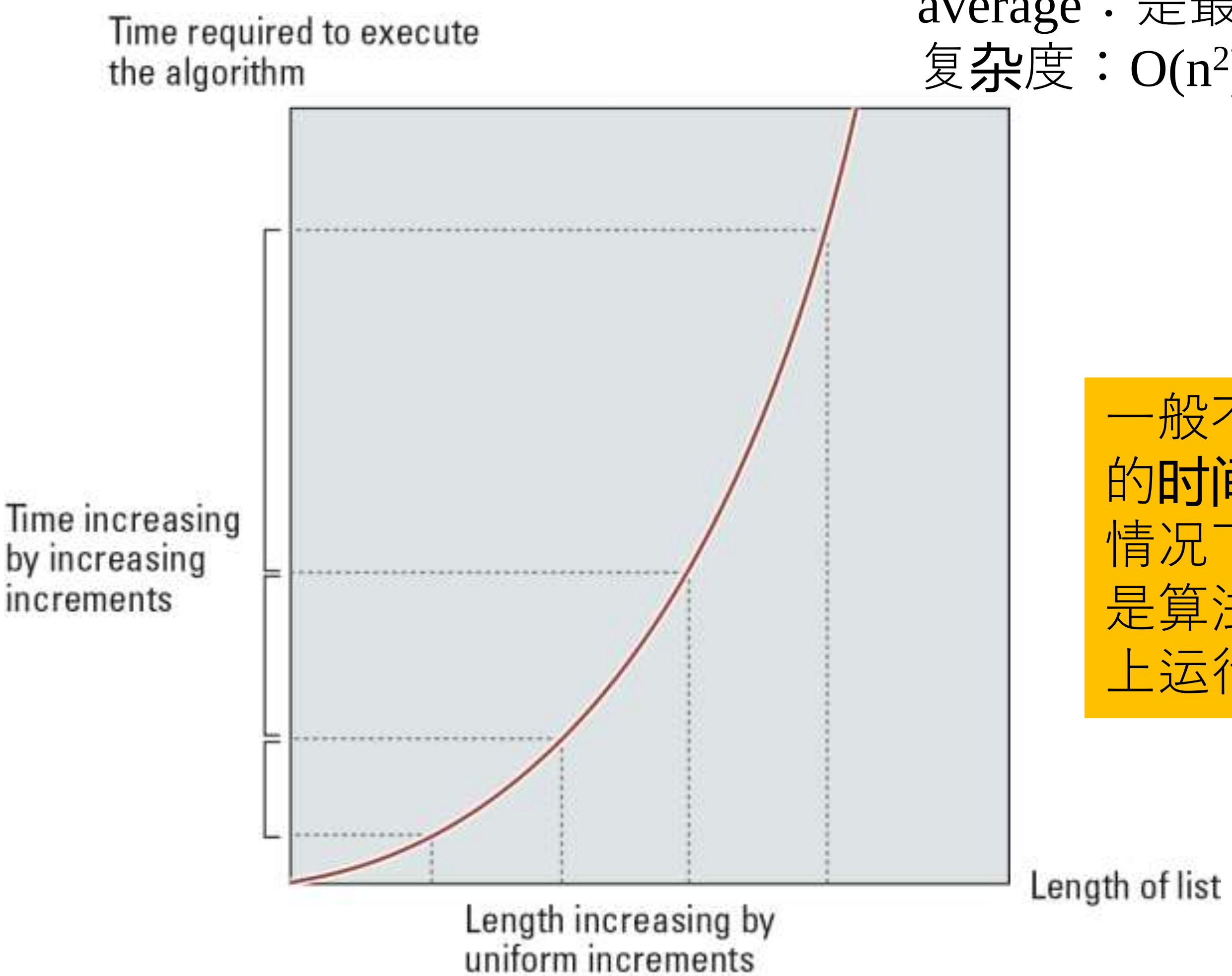


Figure 5.19 Graph of the worst-case analysis of the insertion sort algorithm

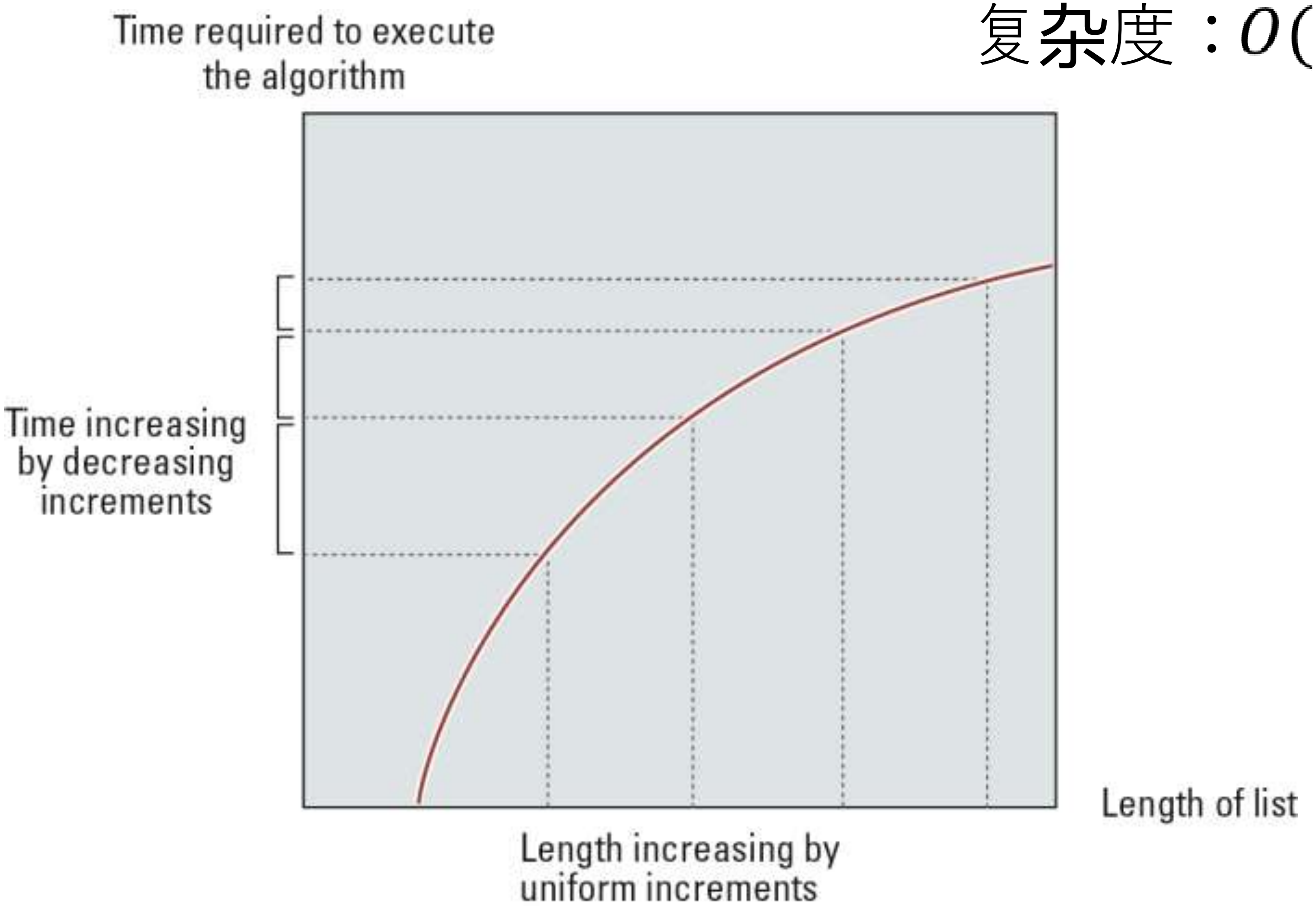
worst：$1+2+3+\ldots+(n-1)=(1/2)(n^2-n)$
average：是最差情况的一半$(1/4)(n^2-n)$
复杂度：$O(n^2)$



一般不特别**说**明，**讨论**的**时间**复**杂**度均是最坏情况下的**时间**复**杂**度，是算法在任何**输**入**实**例上运行**时间**的上界。

Figure 5.20 Graph of the worst-case analysis of the binary search algorithm

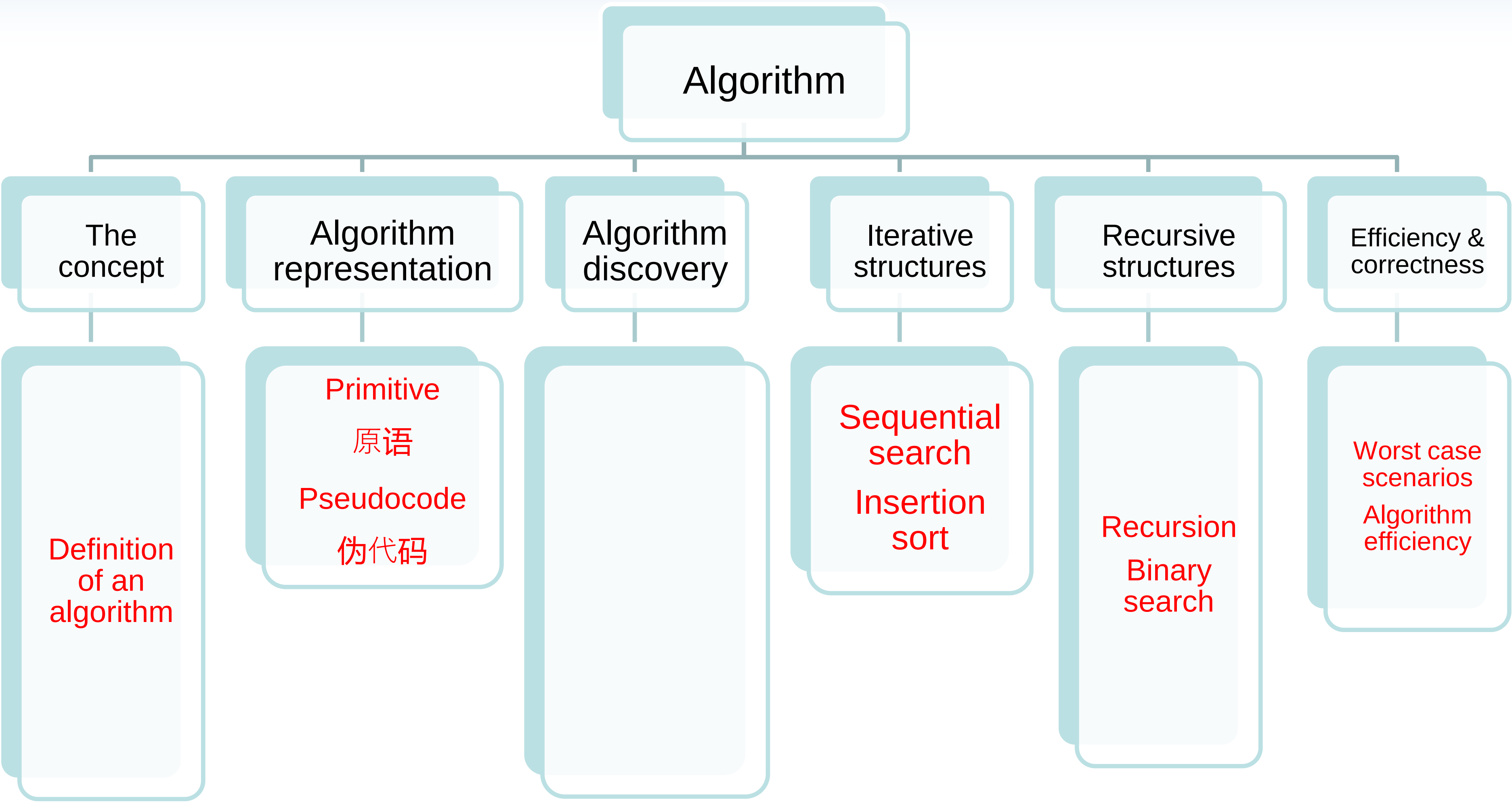
复杂度：$O(log_2 n)$



算法的空间复杂度

- 空间复杂度是指算法在计算机内执行时所需存储空间的度量
  - ·算法程序所占的空间；
  - ·输入的初始数据所占的存储空间；
  - ·算法执行过程中所需要的额外空间。

排序算法	平均时间复杂度	最坏时间复杂度
插入排序	$O(n^2)$	$O(n^2)$
冒泡排序	$O(n^2)$	$O(n^2)$
选择排序	$O(n^2)$	$O(n^2)$
快速排序	$O(nlog_2n)$	$O(n^2)$
归并排序	$O(nlog_2n)$	$O(nlog_2n)$


Algorithm
The concept
Algorithm representation
Algorithm discovery
Iterative structures
Recursive structures
Efficiency & correctness
Definition of an algorithm
Primitive
原语
Pseudocode
伪代码
Sequential search
Insertion sort
Recursion
Binary search
Worst case scenarios
Algorithm efficiency