Chapter 5: Algorithms

Computer Science: An Overview
Tenth Edition

by
J. Glenn Brookshear

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

Algorithm discovery

- Art

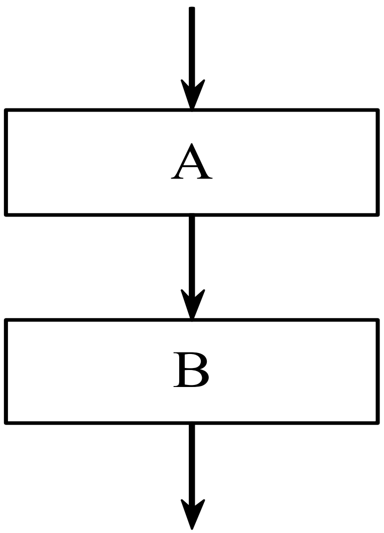
  = analysis + knowledge + experiment + inspiration (sometimes)

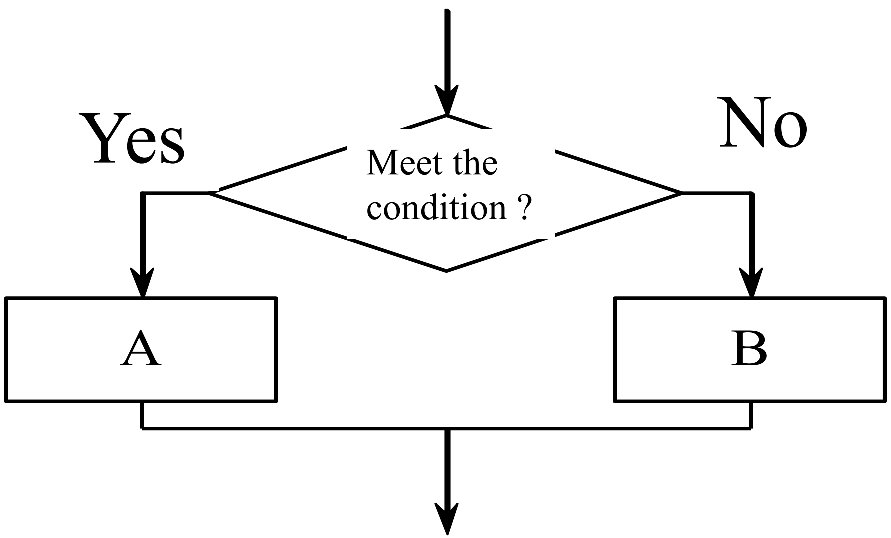
Two elements of algorithm

- **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

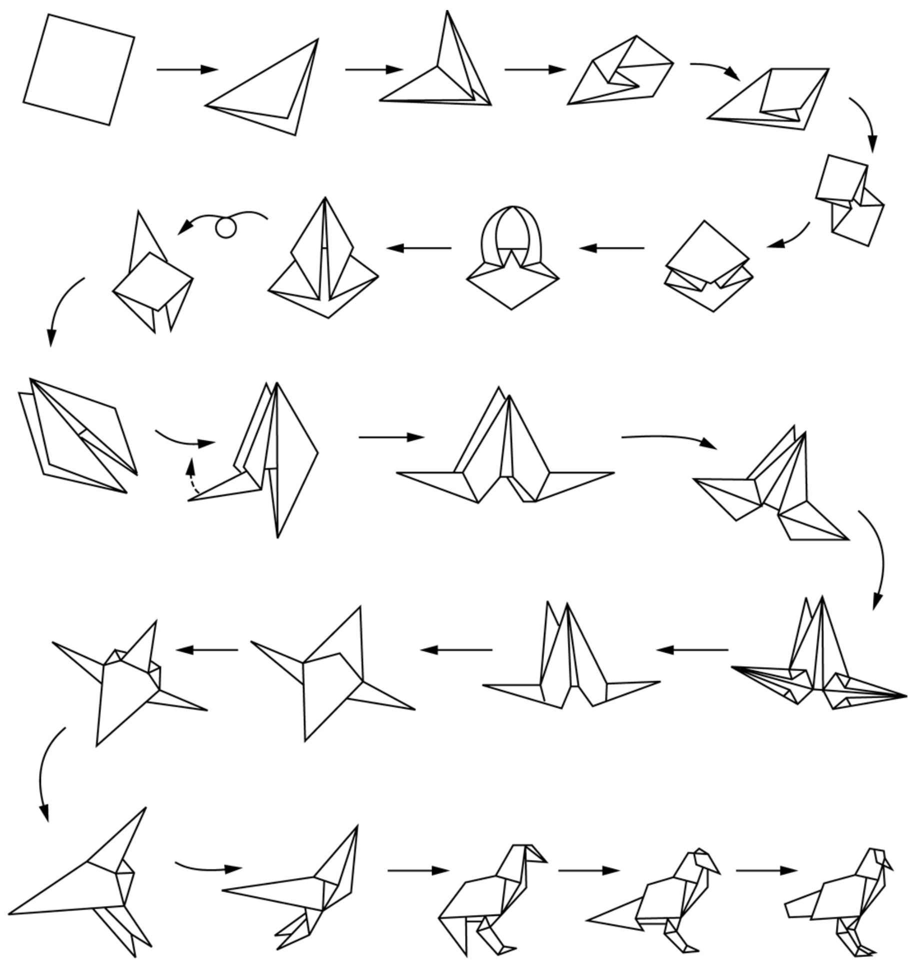
Algorithm representation

- Requires well-defined primitives（原语）
- A collection of primitives constitutes a programming language.

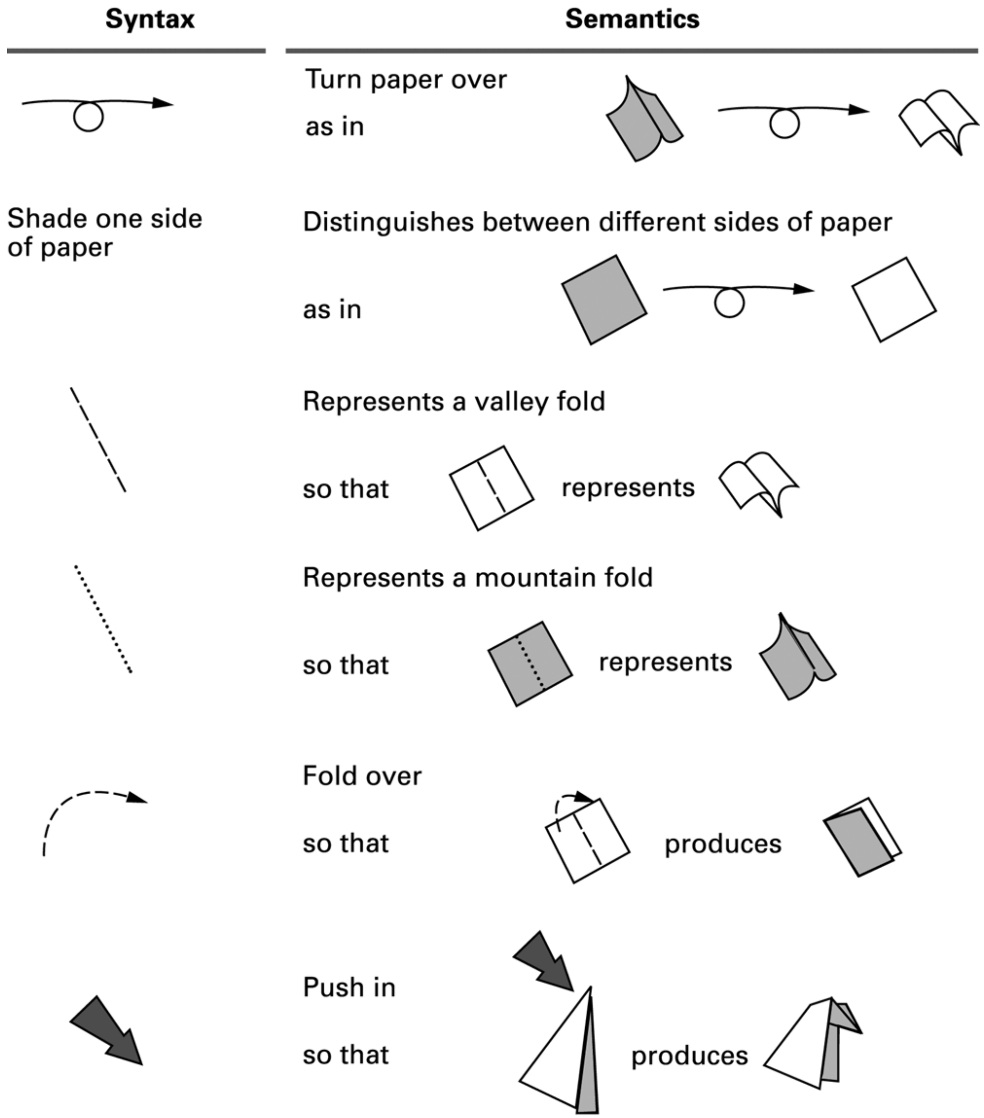
Algorithm representation

- Natural language
- Traditional flow chart
- N-S flow chart
- Pseudo code

Figure 5.2 **Folding a bird from a square piece of paper**



Natural language: Origami primitives



What is an algorithm?

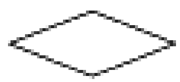
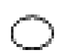
Example: estimation of π

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

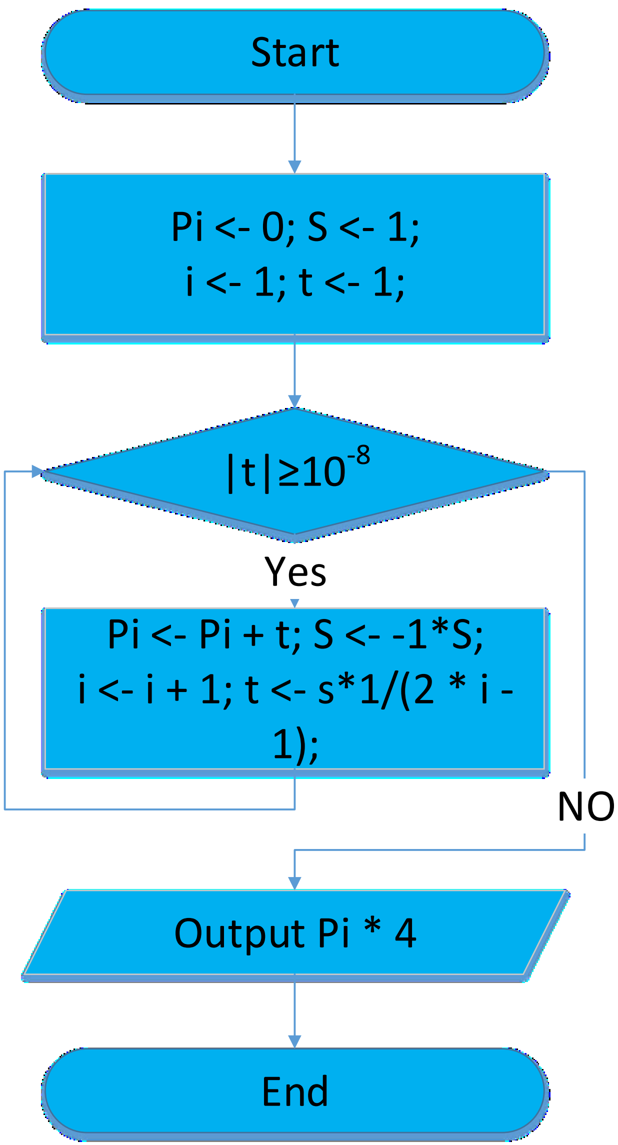
公式 2：$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$

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

Traditional 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



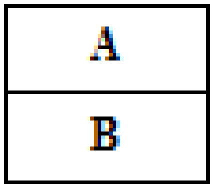
公式 2: $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots+$

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

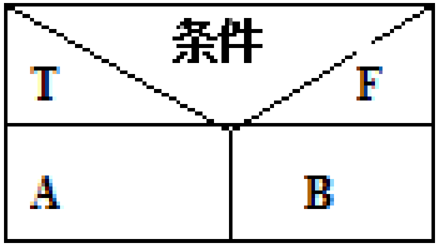
N-S flow chart

- **Proposed by I.Nassi and B.Shneideman**
- **For** structured programming

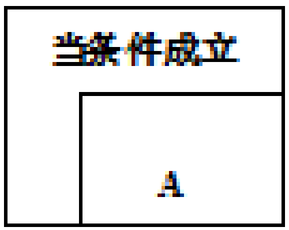
结构化程序设计（structured programming）的主要观点是采用自顶向下、逐步求精及模块化的程序设计方法；使用三种基本控制结构构造程序，任何程序都可由顺序、选择、循环三种基本控制结构构造。结构化程序设计主要强调的是程序的易读性。



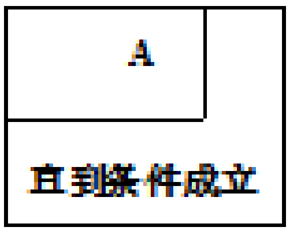
(a)



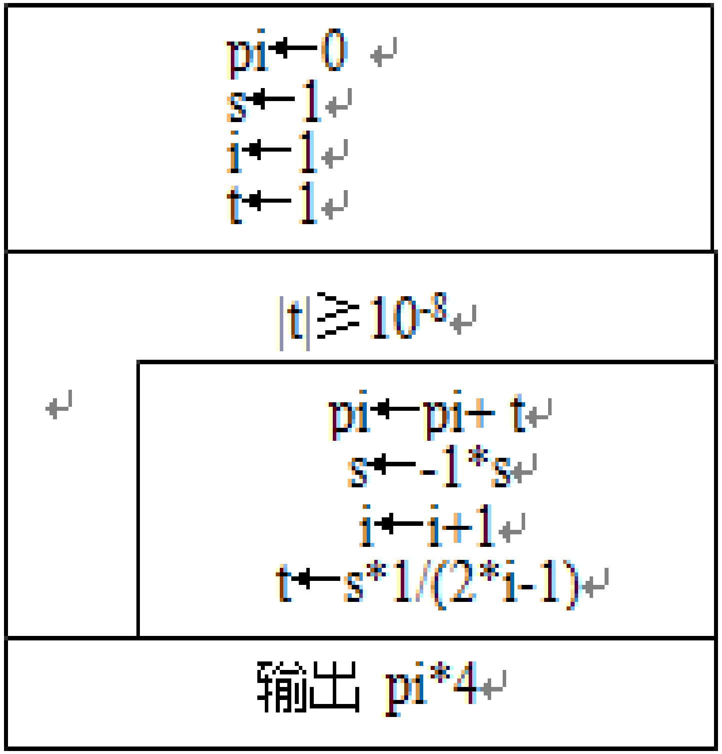
(b)



(c)



(d)



3 control structures for N-S flow chart

Algorithm representation

- Natural language
- Traditional flow chart
- N-S flow chart
- Pseudo code

Pseudocode Primitives

- Assignment

  name ← *expression*

- Conditional selection

  if *condition* **then** *action*

Pseudocode Primitives (continued)

- Repeated execution

  while *condition* **do** *activity*

- Procedure

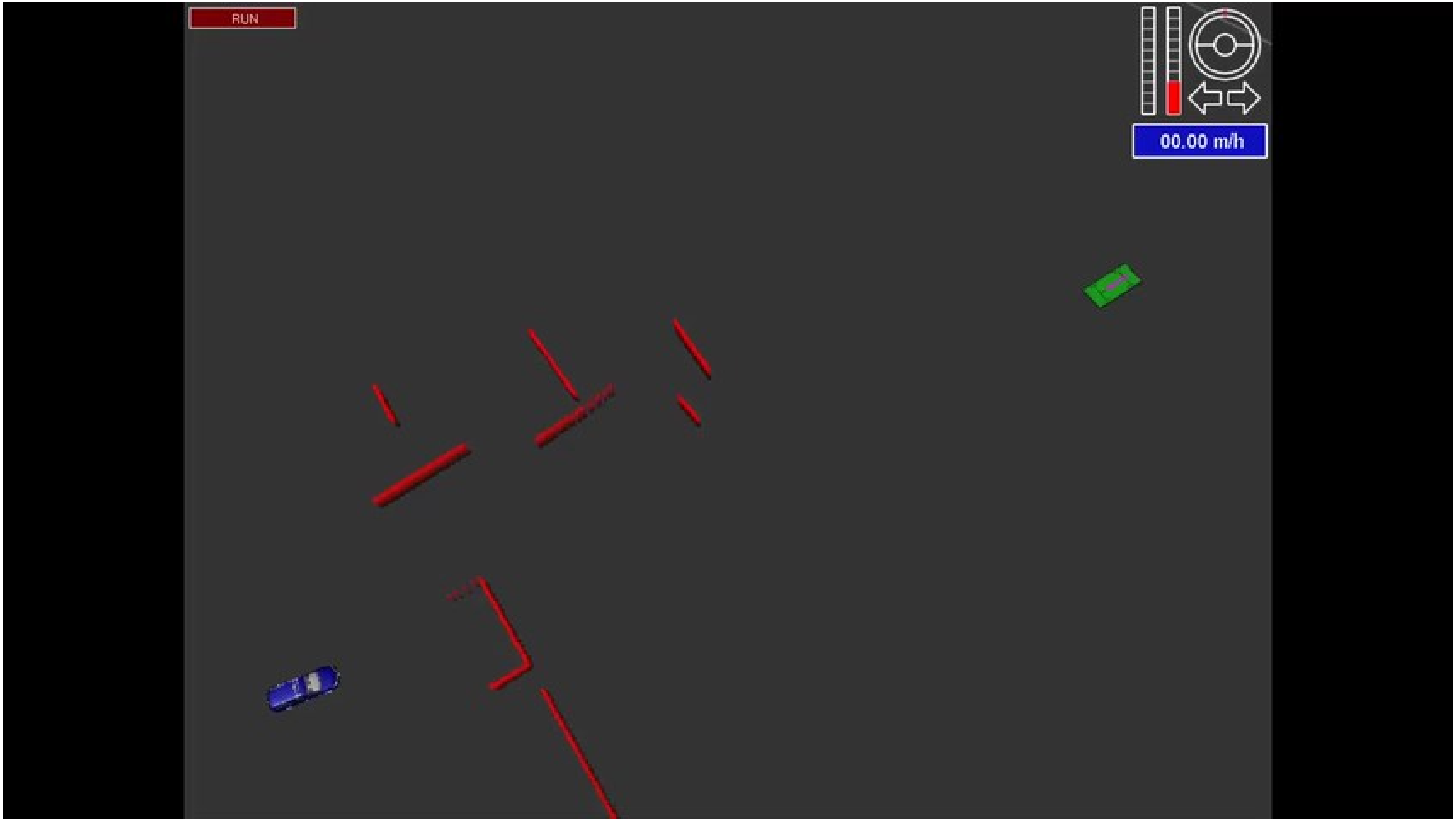
  procedure *name* (*generic names*)

Figure 5.4 **The procedure Greetings in pseudocode**

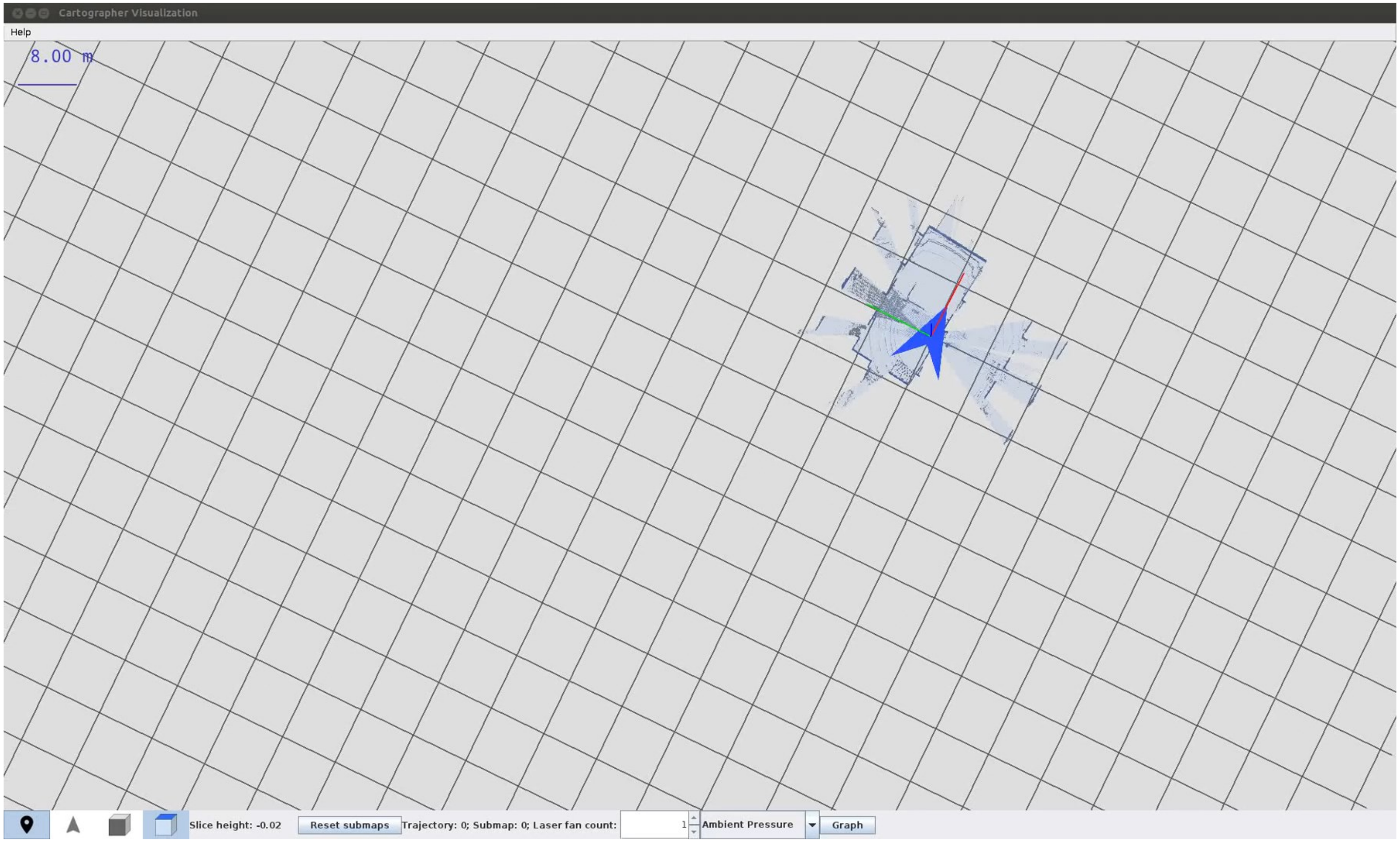
```
procedure Greetings
Count ← 3;
while (Count > 0) do
    (print the message "Hello" and
        Count ← Count −1)
```

Real code?

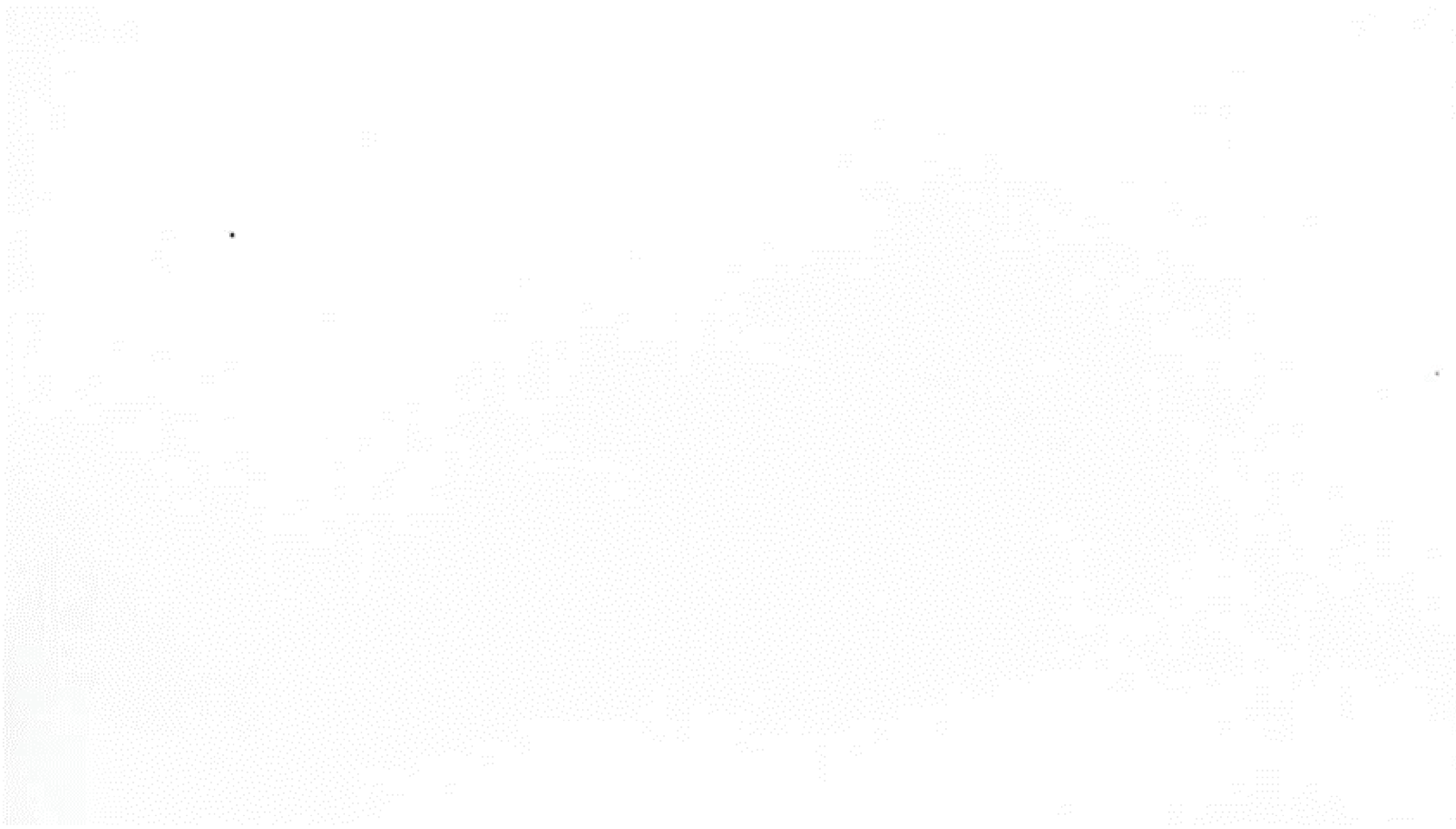
What algorithms can do?

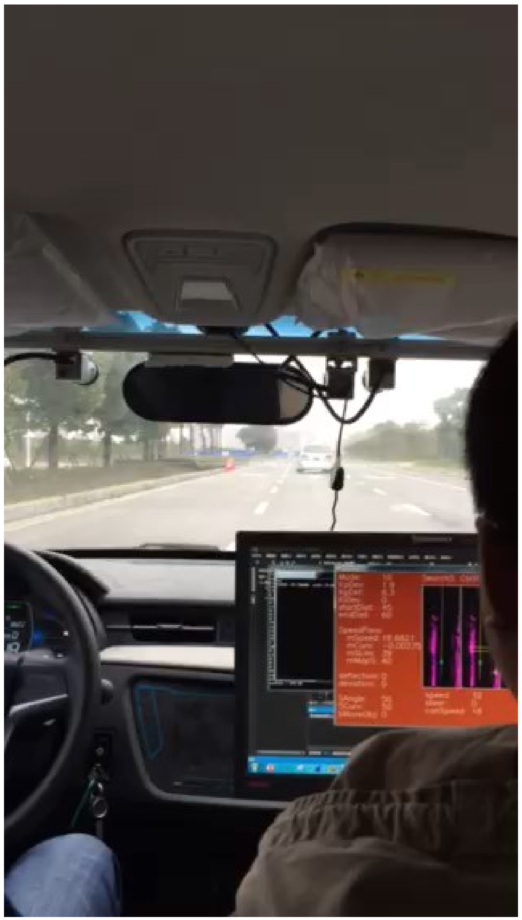


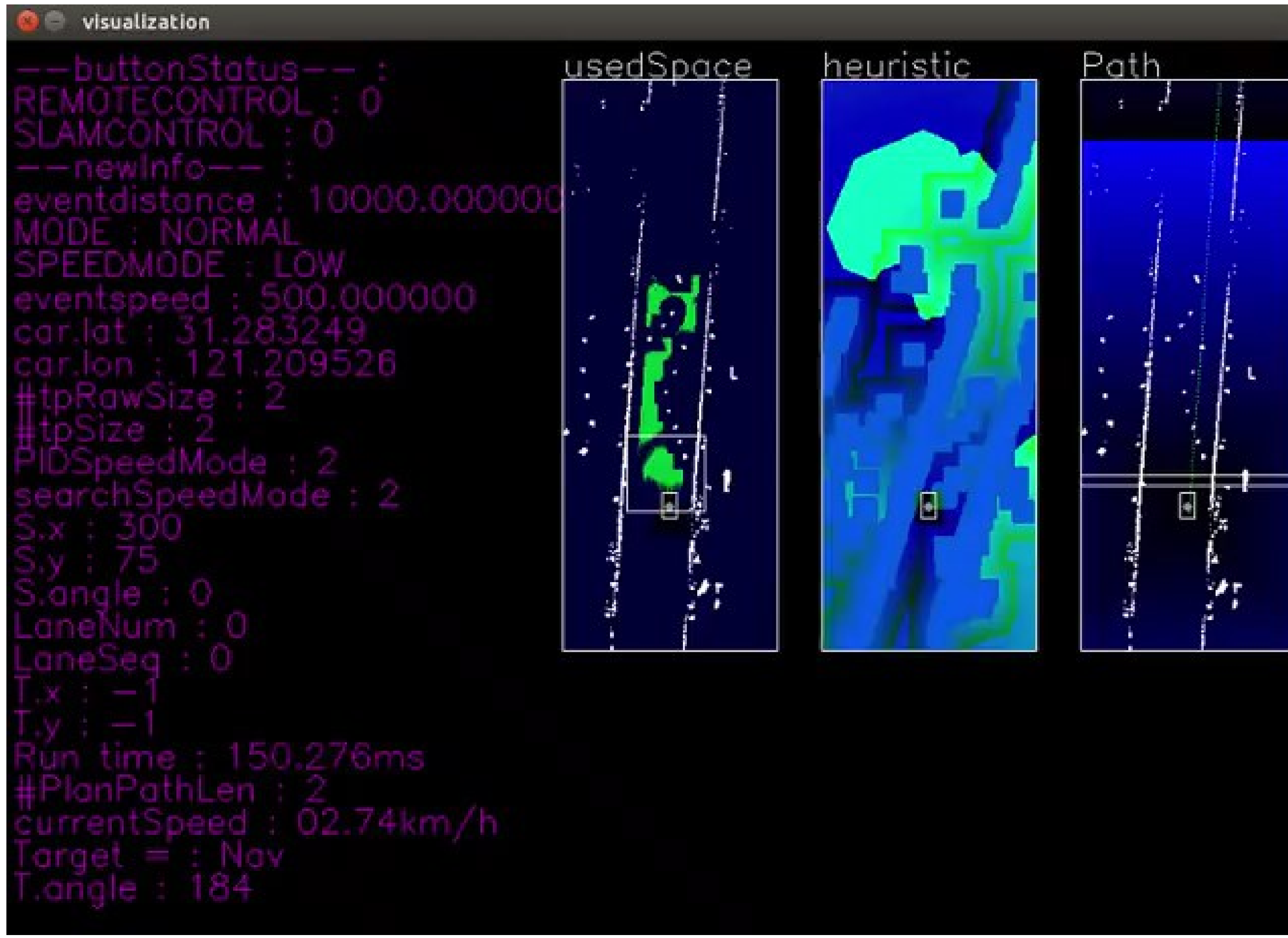
What algorithms can do?



What algorithms can do?






visualization
--buttonStatus-- :
REMOTECONTROL : 0
SLAMCONTROL : 0
--newInfo-- :
eventdistance : 10000.000000
MODE : NORMAL
SPEEDMODE : LOW
eventspeed : 500.000000
car.lat : 31.283249
car.lon : 121.209526
#tpRawSize : 2
#tpSize : 2
PIDSpeedMode : 2
searchSpeedMode : 2
S.x : 300
S.y : 75
S.angle : 0
LaneNum : 0
LaneSeq : 0
T.x : -1
T.y : -1
Run time : 150.276ms
#PlanPathLen : 2
currentSpeed : 02.74km/h
Target = : Nav
T.angle : 184
usedSpace
heuristic
Path

Algorithm discovery

- Art

  = analysis + knowledge + experiment + inspiration (sometimes)

Polya's Problem Solving Steps

- 1. Understand the problem.
- 2. Devise a plan for solving the problem.
- 3. Carry out the plan.
- 4. Evaluate the solution for accuracy and its potential as a tool for solving other problems.

ACM ICPC (International Collegiate Programming Contest)

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

Search algorithms

- Sequential search (顺序查找)
- Binary search (二分查找)

Sequential search algorithm in pseudocode


```
procedure Search (List, TargetValue)
if (List empty)
    then
        (Declare search a failure)
    else
        (Select the first entry in List to be TestEntry;
         while (TargetValue > TestEntry and
                        there remain entries to be considered)
              do (Select the next entry in List as TestEntry.);
         if (TargetValue = TestEntry)
              then (Declare search a success.)
              else (Declare search a failure.)
        ) end if
```

Recursion

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

Recursion vs Iteration

- Are they equivalent?
- Yes, but recursion solution is usually more elegant!
- Which one is faster?
- Yes, Iteration.

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

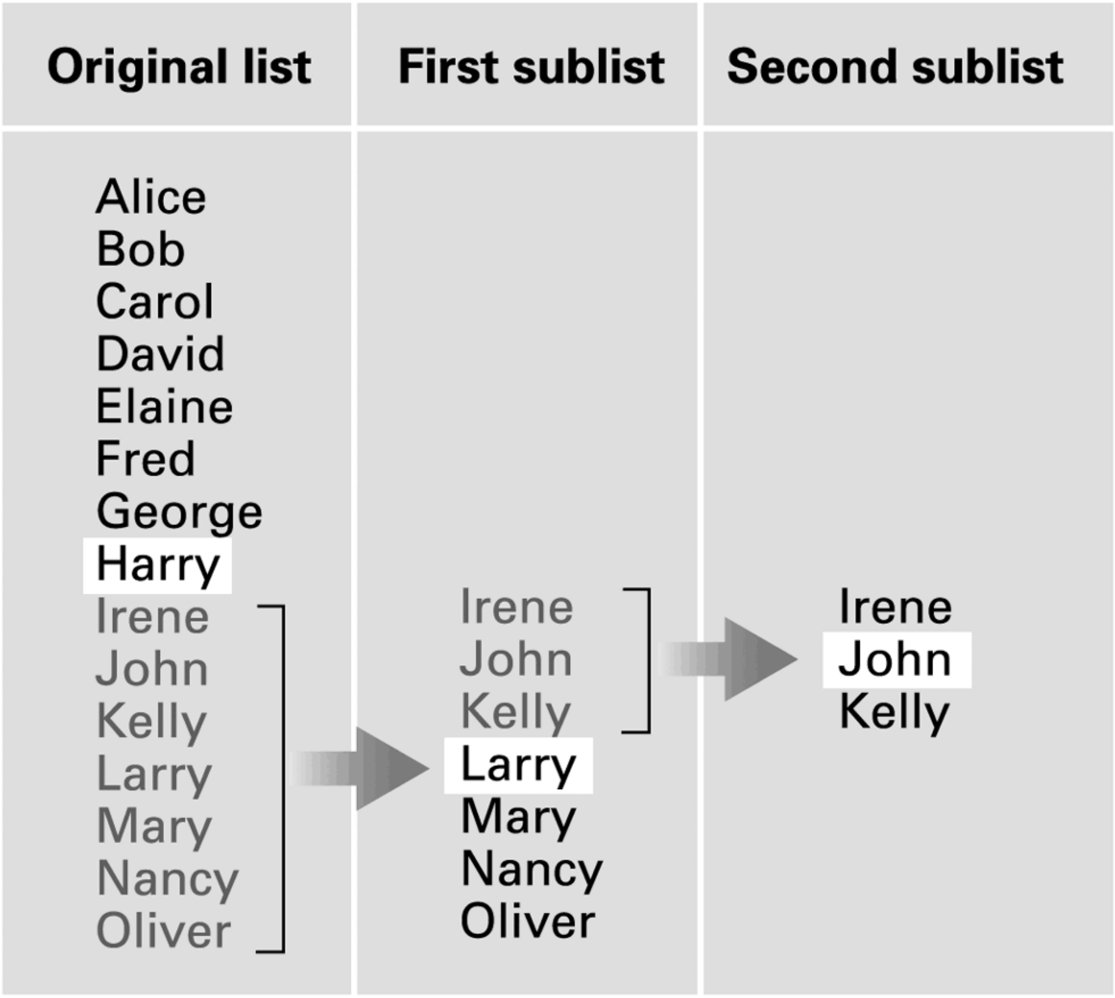


Figure 5.14 **The binary search algorithm in pseudocode**

```
procedure Search (List, TargetValue)
if (List empty)
  then
     (Report that the search failed.)
   else
     [Select the "middle" entry in List to be the TestEntry;
      Execute the block of instructions below that is
          associated with the appropriate case.
              case 1: TargetValue = TestEntry
                          (Report that the search succeeded.)
              case 2: TargetValue < TestEntry
                          (Apply the procedure Search to see if TargetValue
                                 is in the portion of the List preceding TestEntry,
                                 and report the result of that search.)
              case 3: TargetValue > TestEntry
                         (Apply the procedure Search to see if TargetValue
                                is in the portion of List following TestEntry,
                                and report the result of that search.)
     ] end if
```

- Efficiency?

Getting a Foot in the Door

- Try working the problem backwards
- Solve an easier related problem
  - Relax some of the problem constraints
  - Solve pieces of the problem first (bottom up methodology)
- Stepwise refinement: Divide the problem into smaller problems (top-down methodology)

Sort algorithm

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

Insertion sort

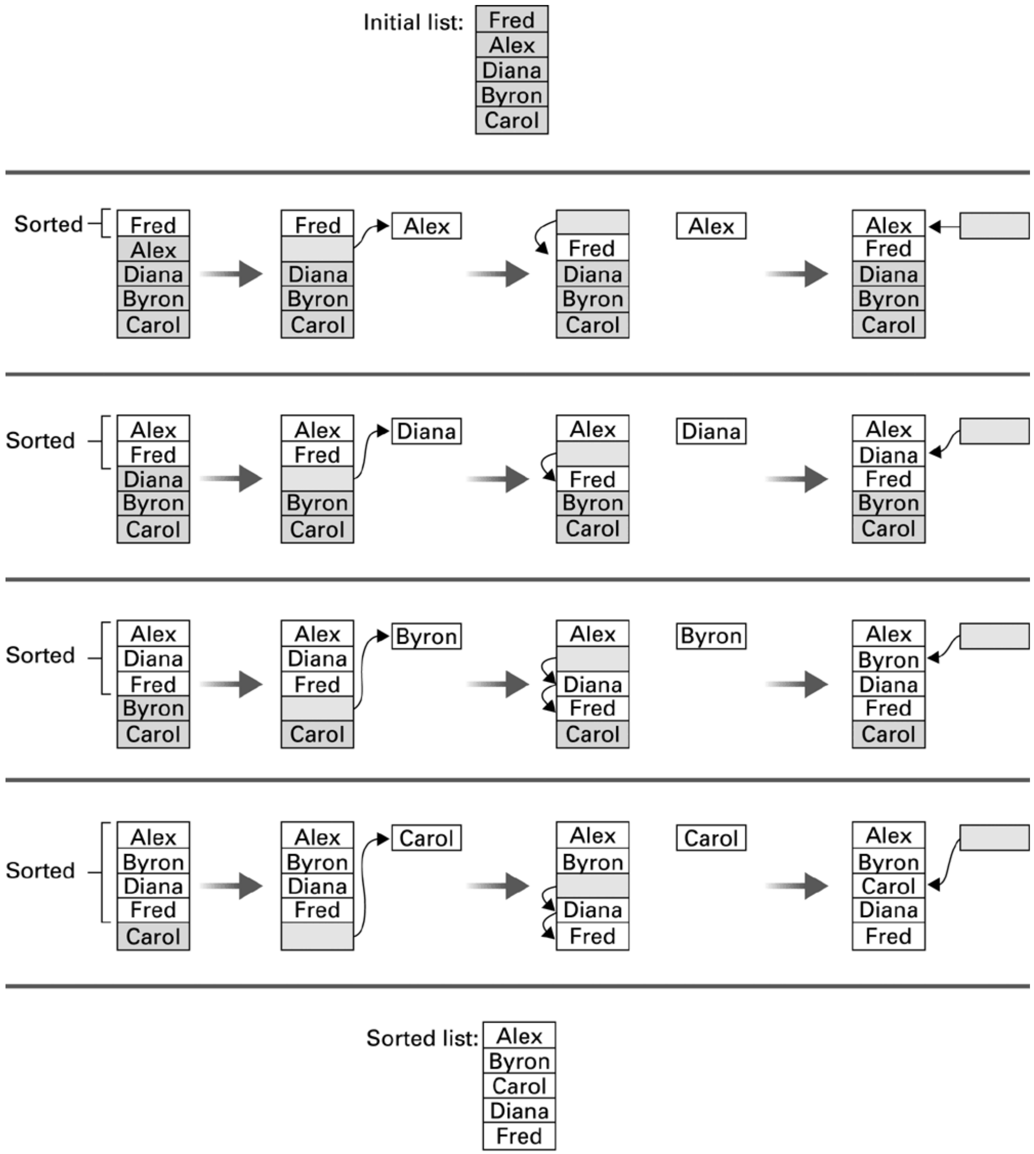


Figure 5.11 **The insertion sort algorithm expressed in pseudocode**

```
procedure Sort (List)
N ← 2;
while (the value of N does not exceed the length of List) do
      (Select the Nth entry in List as the pivot entry;
       Move the pivot entry to a temporary location leaving a hole in List;
        while (there is a name above the hole and that name is greater than the pivot) do
              (move the name above the hole down into the hole leaving a hole above the name)
        Move the pivot entry into the hole in List;
        N ← N + 1
      )
```

Please think

- Reminding merge count， How to implement the merge sort (binary/divide and conquer)?

Figure 12.10 **The hierarchy of problems generated by the merge sort algorithm**

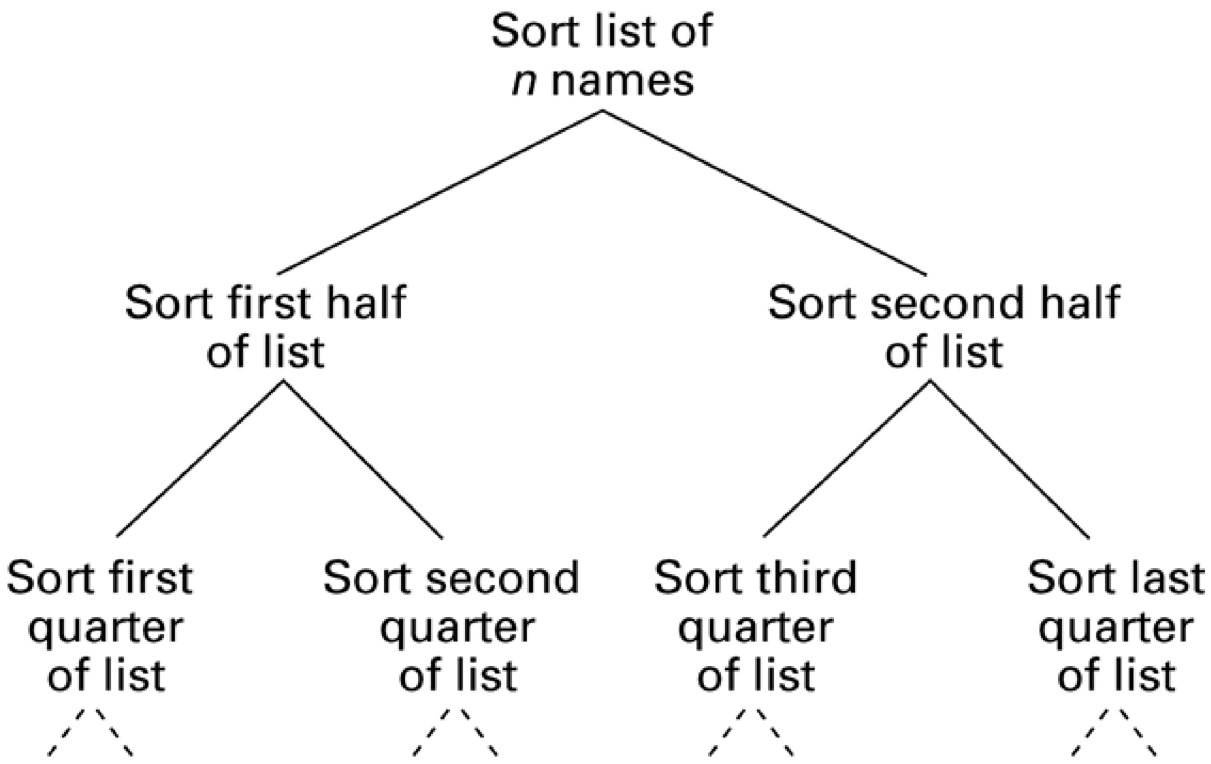


Figure 12.9 **The merge sort algorithm implemented as a procedure MergeSort**

```
procedure MergeSort (List)

if (List has more than one entry)
  then (Apply the procedure MergeSort to sort the first half of List;
        Apply the procedure MergeSort to sort the second half of List;
        Apply the procedure MergeLists to merge the first and second
            halves of List to produce a sorted version of List
       )
```

Figure 12.8 **A procedure MergeLists for merging two lists**

```
procedure MergeLists (InputListA, InputListB, OutputList)

if (both input lists are empty) then (Stop, with OutputList empty)
if (InputListA is empty)
   then (Declare it to be exhausted)
   else (Declare its first entry to be its current entry)
if (InputListB is empty)
   then (Declare it to be exhausted)
   else (Declare its first entry to be its current entry)
while (neither input list is exhausted) do
      (Put the “smaller” current entry in OutputList;
       if (that current entry is the last entry in its corresponding input list)
          then (Declare that input list to be exhausted)
          else (Declare the next entry in that input list to be the list’s current entry )
      )

Starting with the current entry in the input list that is not exhausted,
      copy the remaining entries to OutputList.
```

Complexity Analysis

- O(nlogn)

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
  - Big Omega
  - Big Theta

the worst-case analysis of the sequential search algorithm

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

worst：lgn

复杂度：O(lgn)

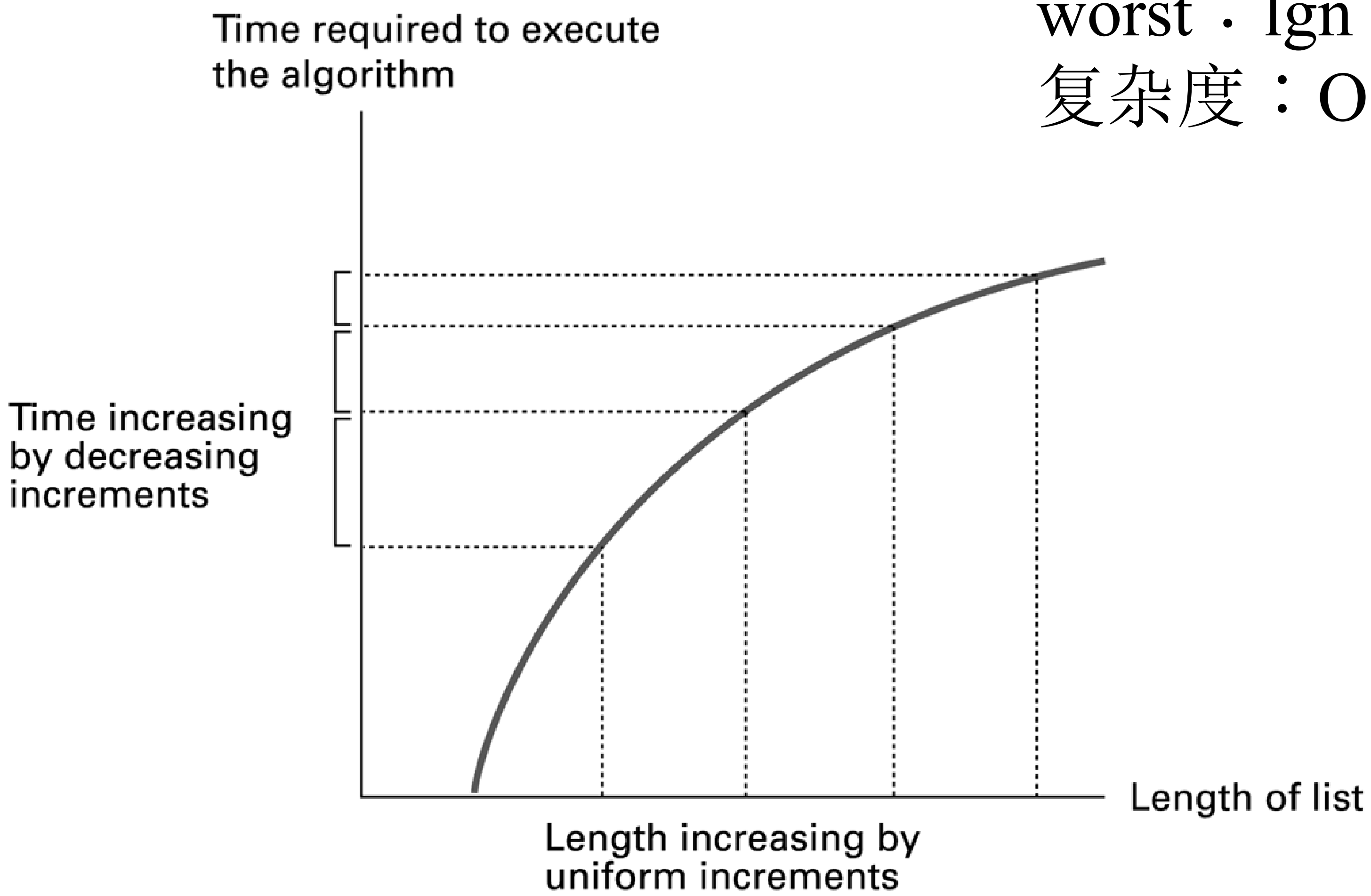
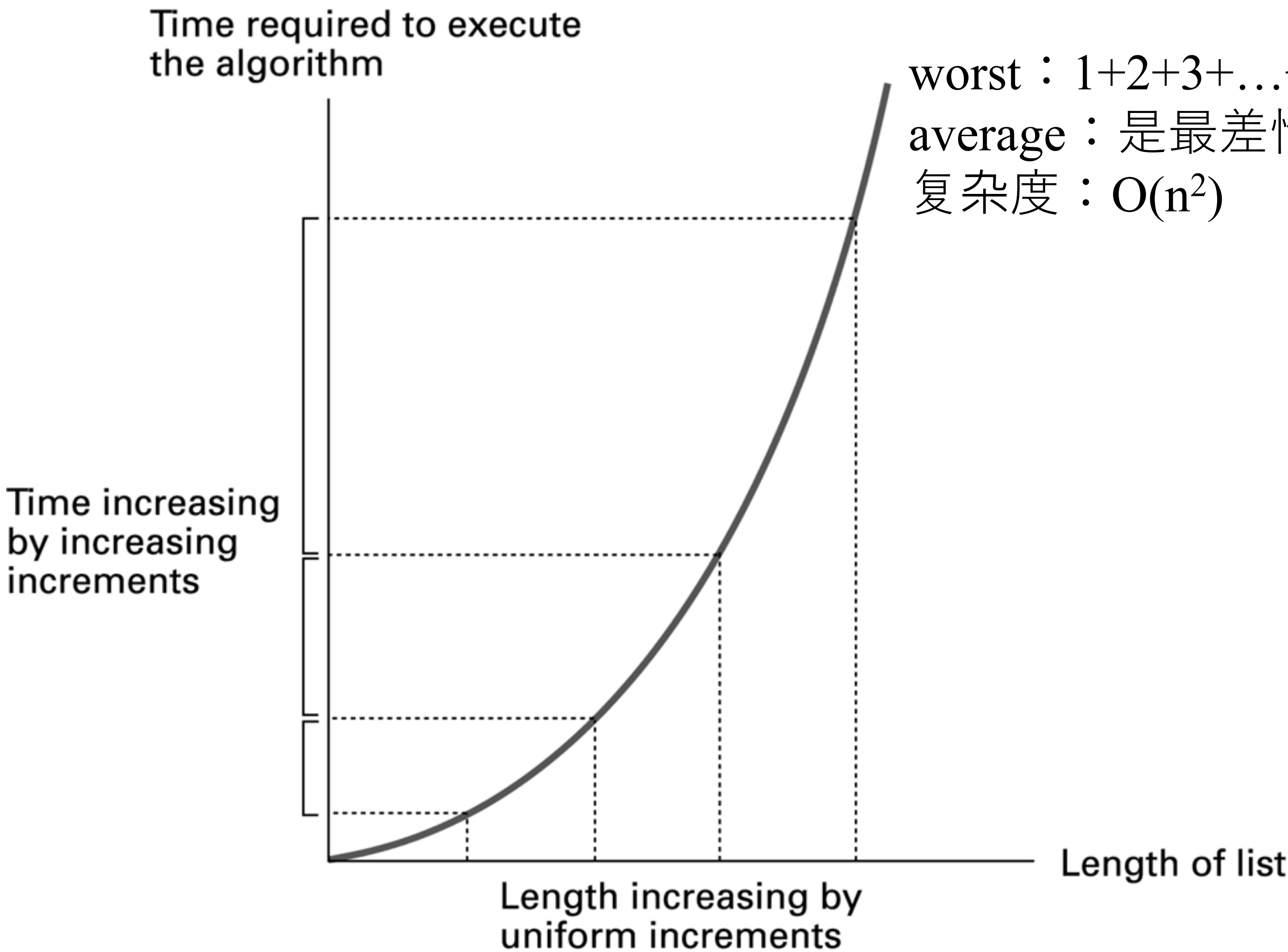


Figure 5.19 **Graph of the worst-case analysis of the insertion sort algorithm**



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：$O(n^2)$

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

Comparisons made for each pivot

Initial list	1st pivot	2nd pivot	3rd pivot	4th pivot	Sorted list
Elaine	1 Elaine	3 David	6 Carol	10 Barbara	Alfred
David	David	2 Elaine	5 David	9 Carol	Barbara
Carol	Carol	Carol	4 Elaine	8 David	Carol
Barbara	Barbara	Barbara	Barbara	7 Elaine	David
Alfred	Alfred	Alfred	Alfred	Alfred	Elaine

- Questions?

Prime Factorization

- 36
- 129
- 523423
- 234578432803423

Prime Factorization?

Prime Factorization of 2034324308234234821 33??

No efficient solution is known!!

It took more than 50 years of computer time to factorize a 200 digit number!

It is assumed that there is no easy solution for prime factorization. In fact, most of the cryptography used today is based on the assumption that prime factorization is a hard problem and that it cannot be done efficiently.

Fundamental Questions

- What can a computer do?
- What can a computer do with limited resources?

Functions

- **Function:** A correspondence between a collection of possible input values and a collection of possible output values so that each possible input is assigned a single output

Functions (continued)

- **Computing a function:** Determining the output value associated with a given set of input values
- **Noncomputable function:** A function that cannot be computed by any algorithm

Figure 12.1 **An attempt to display the function that converts measurements in yards into meters**

Yards (input)	Meters (output)
1	0.9144
2	1.8288
3	2.7432
4	3.6576
5	4.5720
.	.
.	.
.	.

Figure 12.2 **The components of a Turing machine**

- Inputs at each step
  - State
  - Value at current tape position
- Actions at each step
  - Write a value at current tape position
  - Move read/write head
  - Change state

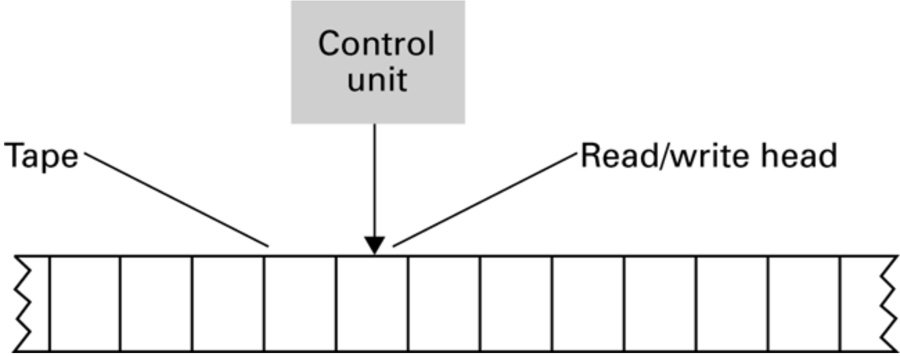


Figure 12.3 **A Turing machine for incrementing a value**

*1 0 1 *
↑
current state

Current state	Current cell content	Value to write	Direction to move	New state to enter
START	*	*	Left	ADD
ADD	0	1	Right	RETURN
ADD	1	0	Left	CARRY
ADD	*	*	Right	HALT
CARRY	0	1	Right	RETURN
CARRY	1	0	Left	CARRY
CARRY	*	1	Left	OVERFLOW
OVERFLOW	(Ignored)	*	Right	RETURN
RETURN	0	0	Right	RETURN
RETURN	1	1	Right	RETURN
RETURN	*	*	No move	HALT

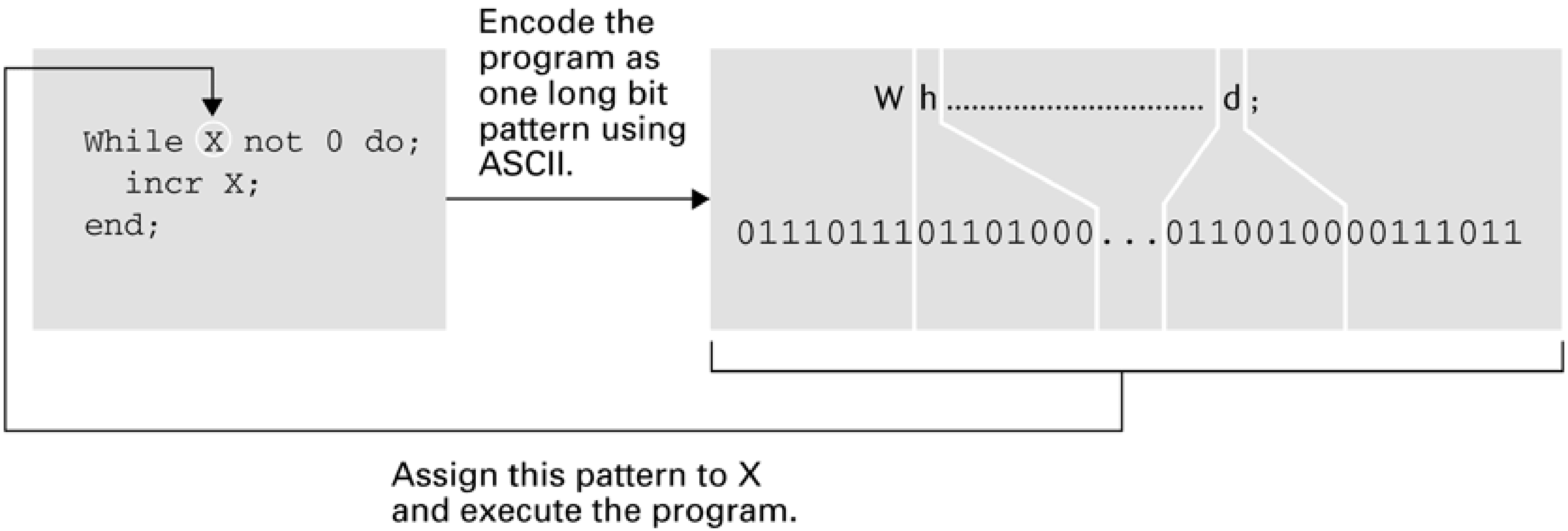
Church-Turing Thesis

- The functions that are computable by a Turing machine are exactly the functions that can be computed by any algorithmic means.

The Halting Problem

- Given the encoded version of any program, return 1 if the program is self-terminating, or 0 if the program is not.
- E.g.
  - While(true);
  - Print(“hello”);

Figure 12.6 **Testing a program for self-termination**



- ```
  bool God_algo(char* program, char* input)
  {
      if(<program> halts on <input>)
          return true;
      return false;
  }
  ```

http://blog.csdn.net/niushuai666/article/details/7260957
  ```

- ```
  bool Satan_algo(char* program)
  {
      if( God_algo(program, program) )
      {
              while(1);    // loop forever!
              return false;// can never get here!
      }
      else
              return true;
  }
  ```

http://blog.csdn.net/niushuai666/article/details/7260957
  ```

- Satan_algo(Satan_algo) 能够停机 => 它不能停机
- Satan_algo(Satan_algo) 不能停机 => 它能够停机

http://blog.csdn.net/niushuai666/article/details/7260957

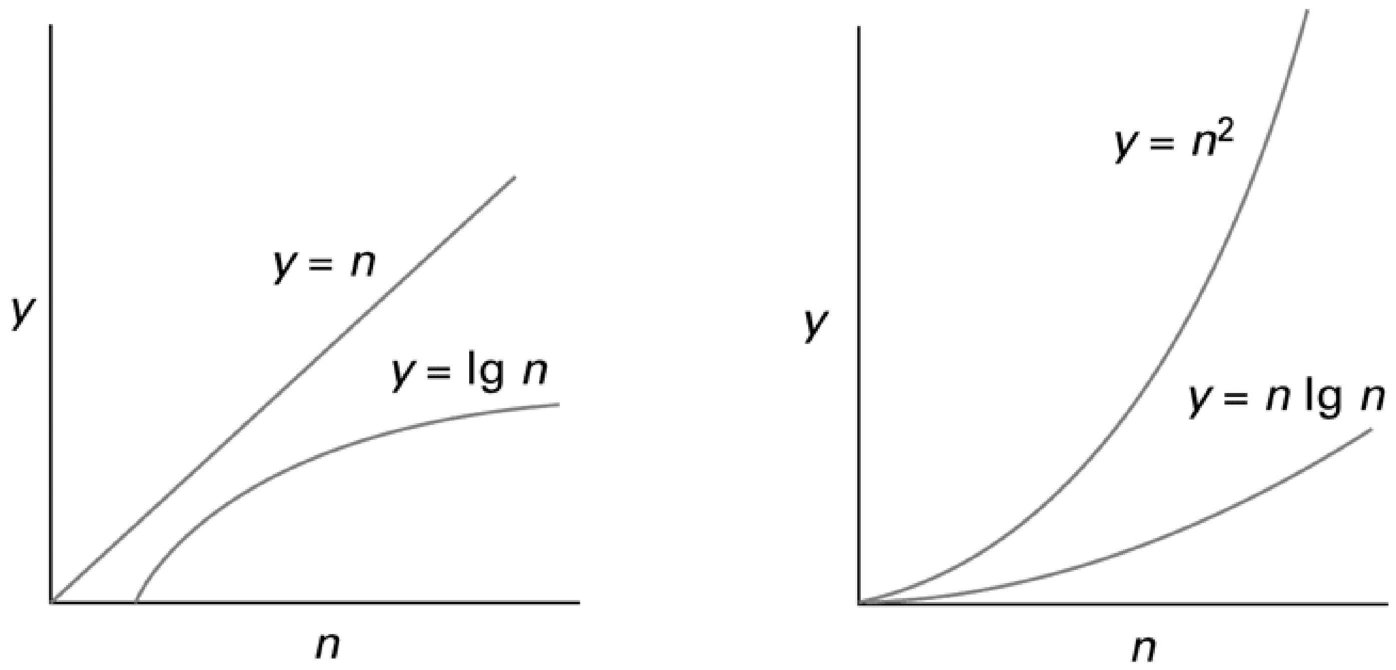
Algorithm

- Solvable problem
- Non-solvable problem

Complexity of Problems

- **Time Complexity:** The number of instruction executions required
  - Unless otherwise noted, “complexity” means “time complexity.”
- A problem is in class $\Theta(f(n))$ if **the best algorithm to solve it is in class $\Theta(f(n))$.**
- A problem is in class $O(f(n))$ if **it can be solved by an algorithm in $\Theta(f(n))$.**

Figure 12.11 **Graphs of the mathematical expressions *n*, lg *n*, *n* lg n, and n^2**



a. *n* versus lg *n*

b. n^2 versus *n* lg *n*

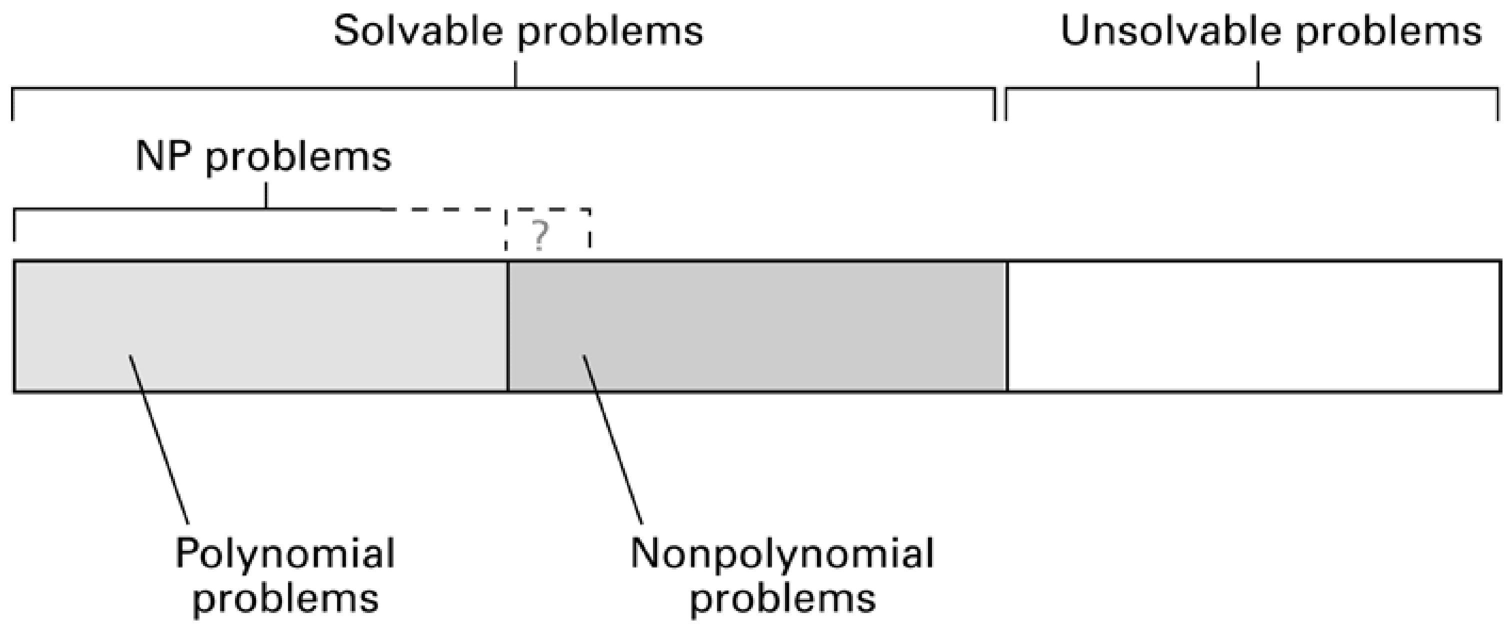
Traveling salesman problem

P versus NP

- **Class P:** All problems in any class $\Theta(f(n))$, where f(n) is a **polynomial** (***Quick Solvable***)
- **Class NP:** All problems that can be *verified* by a nondeterministic algorithm in polynomial time (***Quick Checkable***)

  Nondeterministic algorithm = an "algorithm" whose steps may not be uniquely and completely determined by the process state
- Whether the class NP is bigger than class P is currently unknown.

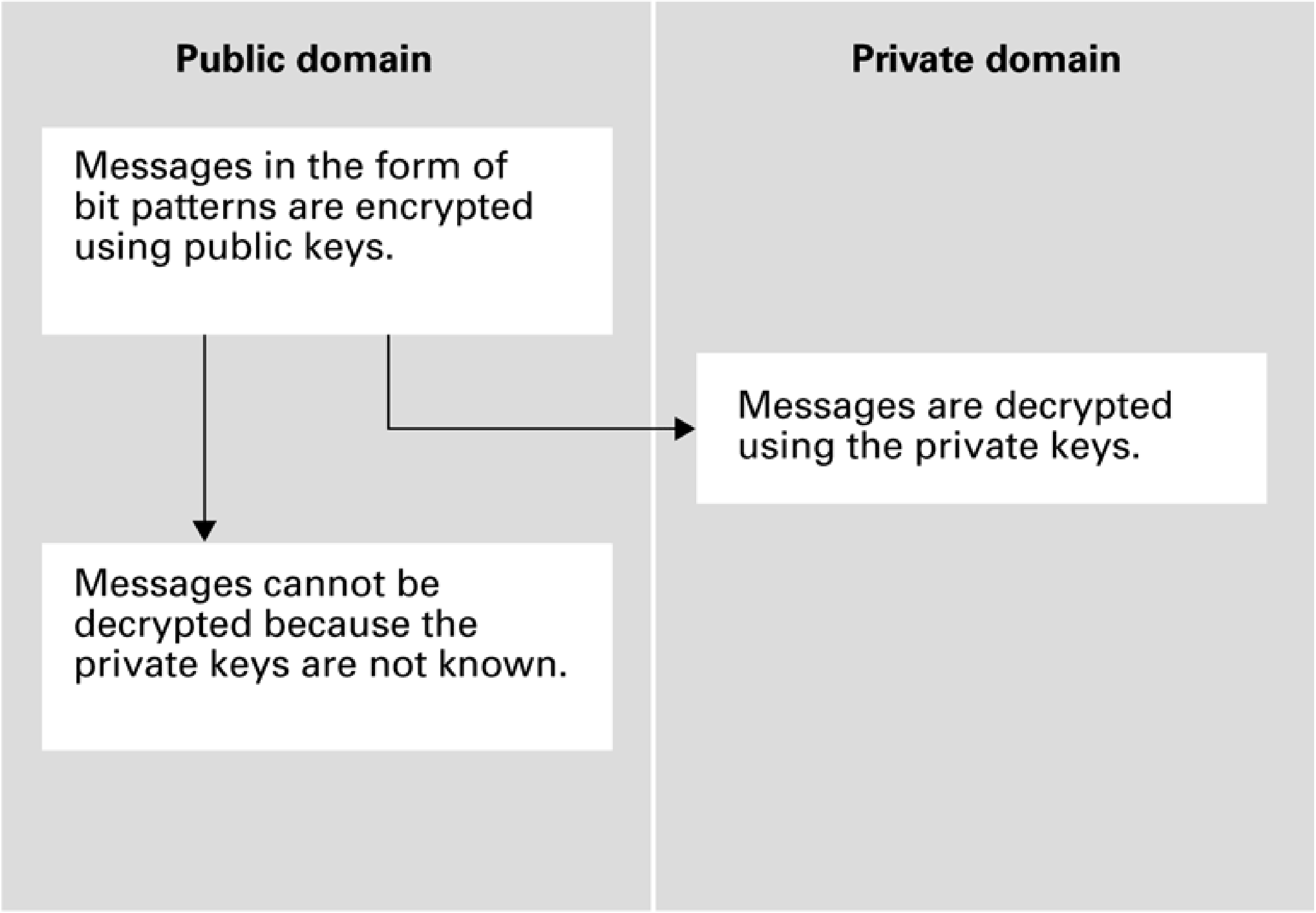
Figure 12.12 **A graphic summation of the problem classification**



Public-Key Cryptography

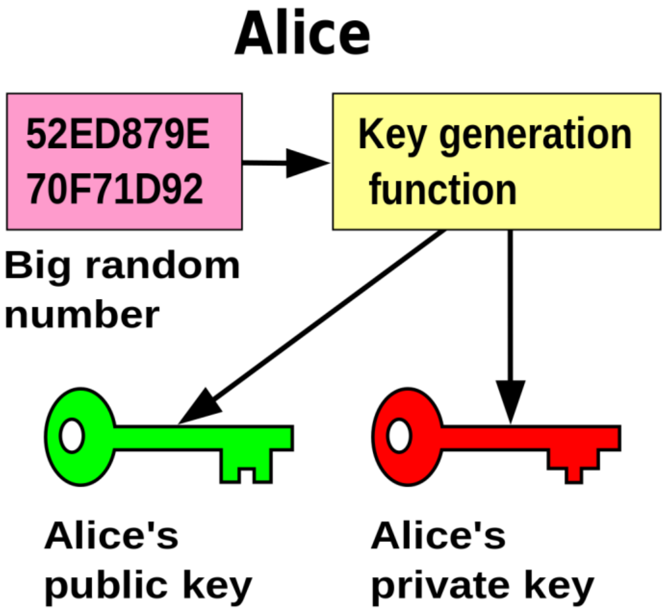
- **Key:** A value used to encrypt or decrypt a message
  - **Public key**: Used to encrypt messages
  - **Private key**: Used to decrypt messages

Figure 12.13 **Public key cryptography**



RSA by Ron Rivest, Adi Shamir, and Leonard Adleman

- **RSA:** A popular public key cryptographic algorithm (1977)
  - Relies on the (presumed) intractability of the problem of factoring large numbers (integer factorization)



Cryptography By Factoring Large Numbers

- p, q为两个质数，m是0到p*q的一个整数，对于任意的正整数k存在：$1 = m^{k(p-1)(q-1)} (\mathrm{mod}\ p*q)$
- RSA公钥系统：选出两个不同的质数，p, q。令 n=p*q。
- 选出两个正整数：e, d。使得对某个正整数k，$e*d=k(p-1)(q-1)+1$
- 那么，e,n为加密密钥，d,n为解密密钥，**p,q用来构建加密系统。**
- 加密m：$c=m^e (\mathrm{mod}\ n)$
- 解密c：$c^d (\mathrm{mod}\ n)$
  - 原理：$c^d=m^{e*d} (\mathrm{mod}\ n) = m^{k(p-1)(q-1)+1} (\mathrm{mod}\ n)=m^{k(p-1)(q-1)} (\mathrm{mod}\ n) * m^1 (\mathrm{mod}\ n)=m\ (\mathrm{mod}\ n)= m.$

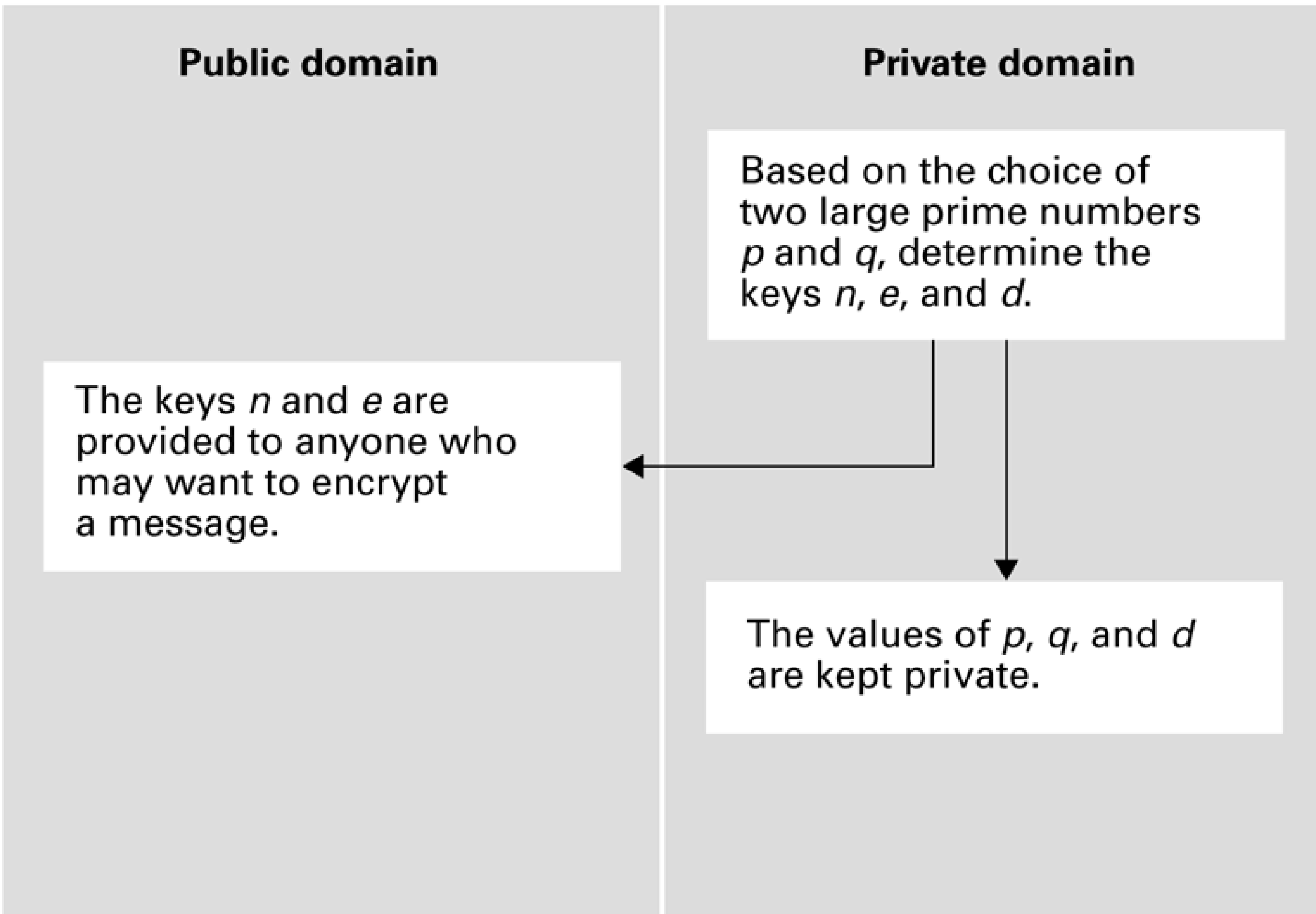
Encrypting the Message 10111

- Encrypting keys: n = 91 and e = 5
- $10111_{two} = 23_{ten}$
- $23^e = 23^5 = 6{,}436{,}343$
- $6{,}436{,}343 \div 91$ has a remainder of 4
- $4_{ten} = 100_{two}$
- Therefore, encrypted version of 10111 is 100.

Decrypting the Message 100

- Decrypting keys: d = 29, n = 91
- $100_{two} = 4_{ten}$
- $4^d = 4^{29} = 288{,}230{,}376{,}151{,}711{,}744$
- $288{,}230{,}376{,}151{,}711{,}744 \div 91$ has a remainder of 23
- $23_{ten} = 10111_{two}$
- Therefore, decrypted version of 100 is 10111.

Figure 12.14 **Establishing an RSA public key encryption system**



Reference :http://www.ruanyifeng.com/blog/2013/07/rsa_algorithm_part_two.html

Crack RSA?

- 有无可能在已知n和e的情况下，推导出d
  - $e*d=k(p-1)(q-1)+1$ 只有知道p和q，才能算出d
  - n=pq只有将n因数分解，才能算出p和q。

- Questions?