

Computer Science Guidance

Jianhui Zhang,

Ph.D., Associate Prof.

College of Computer Science and
Technology, Hangzhou Dianzi Univ.

Email: jh_zhang@hdu.edu.cn

Addison-Wesley
is an imprint of

PEARSON

Copyright © 2015 Pearson Education, Inc.

Chapter 5: Algorithms

**Computer Science: An Overview
Eleventh Edition**

**by
J. Glenn Brookshear
Dennis Brylow**

Addison-Wesley
is an imprint of

PEARSON

Copyright © 2015 Pearson Education, Inc.

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

Definition of Algorithm

An algorithm is an **ordered** set of **unambiguous, executable** steps that defines a **terminating** process.

Algorithm Representation

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

Figure 5.2 Folding a bird from a square piece of paper

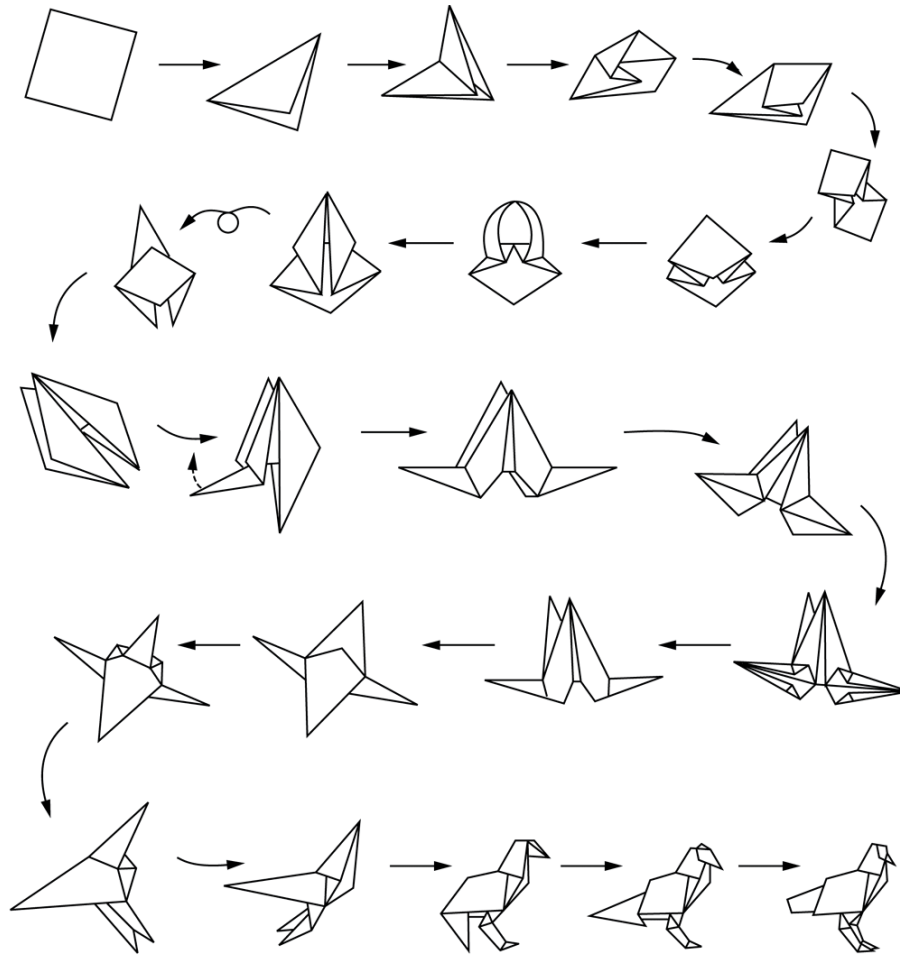
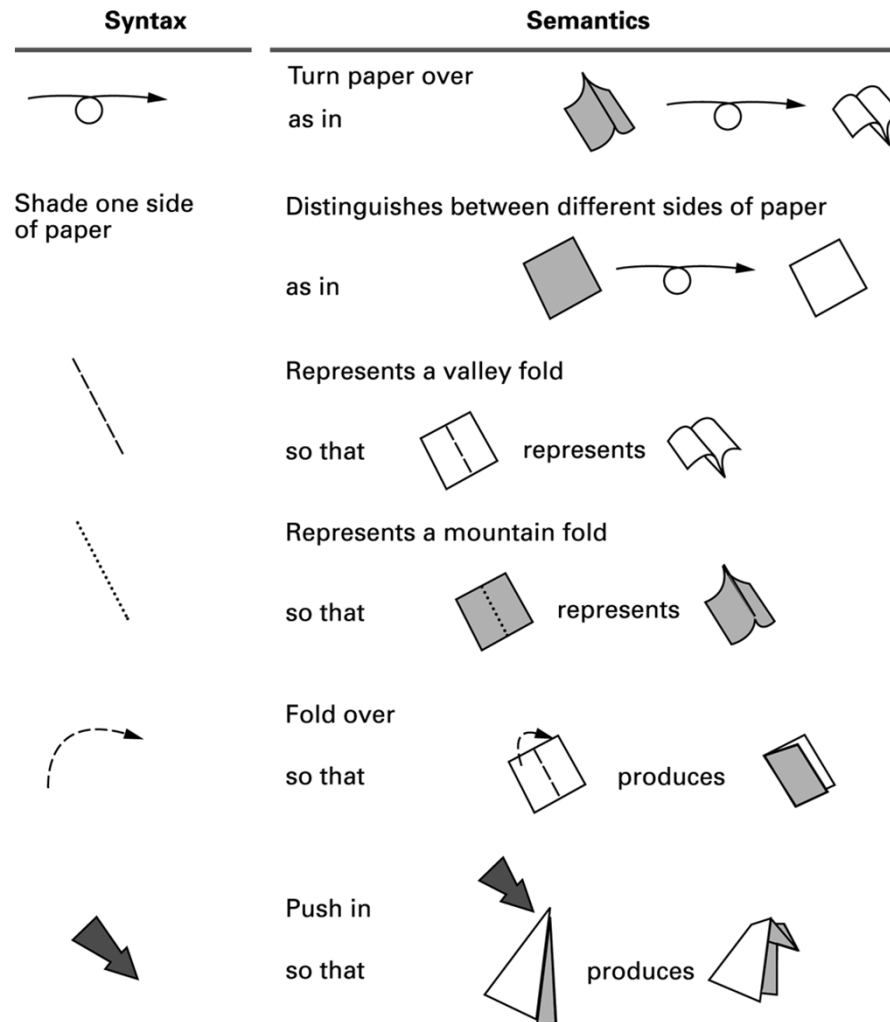


Figure 5.3 Origami primitives



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)
```

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.

Polya's Steps in the Context of Program Development

- 1. Understand the problem.
- 2. Get an idea of how an algorithmic function might solve the problem.
- 3. Formulate the algorithm and represent it as a program.
- 4. Evaluate the solution for accuracy and its potential as a tool for solving other problems.

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)

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

$$(1,1,36) \quad (1,6,6)$$

$$(1,2,18) \quad (2,2,9)$$

$$(1,3,12) \quad (2,3,6)$$

$$(1,4,9) \quad (3,3,4)$$

b. Sums of triples from part (a)

$$1 + 1 + 36 = 38$$

$$1 + 2 + 18 = 21$$

$$1 + 3 + 12 = 16$$

$$1 + 4 + 9 = 14$$

$$1 + 6 + 6 = 13$$

$$2 + 2 + 9 = 13$$

$$2 + 3 + 6 = 11$$

$$3 + 3 + 4 = 10$$

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

Iterative Structures

- Pretest loop:

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

- Posttest loop:

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

Figure 5.8 The while loop structure

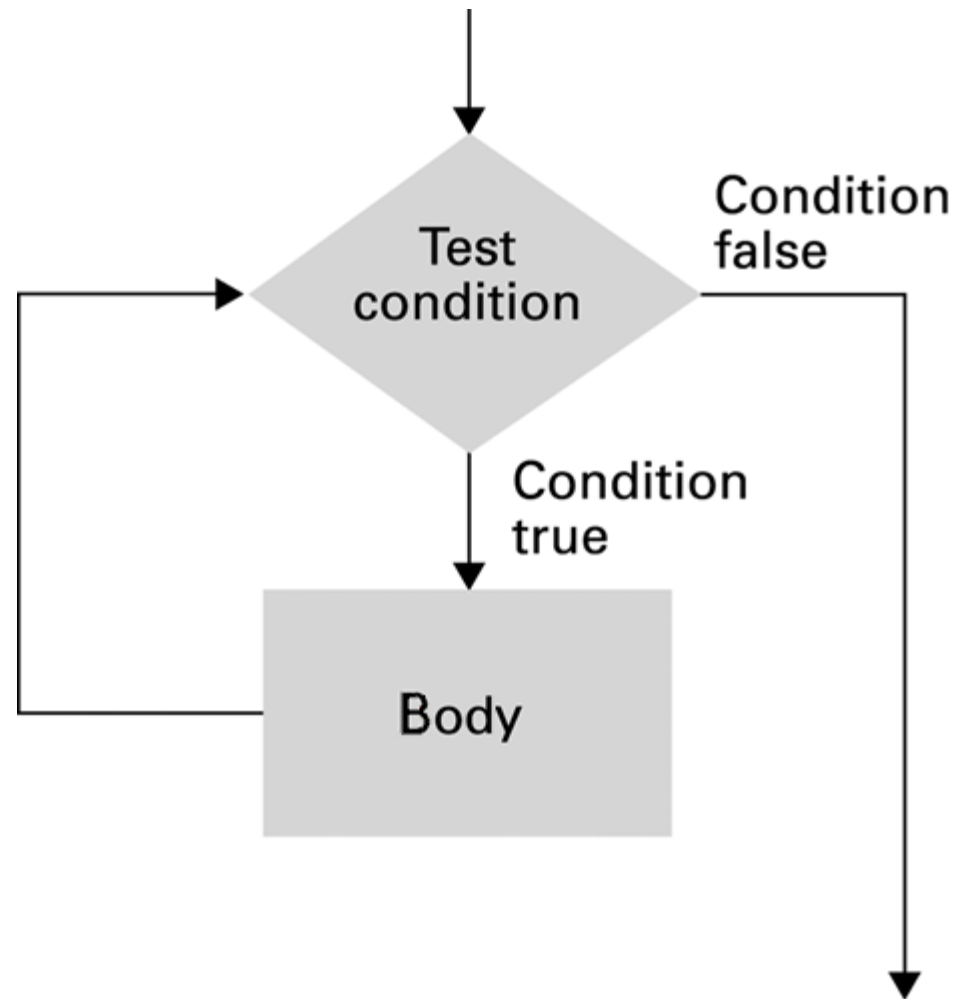


Figure 5.9 The repeat loop structure

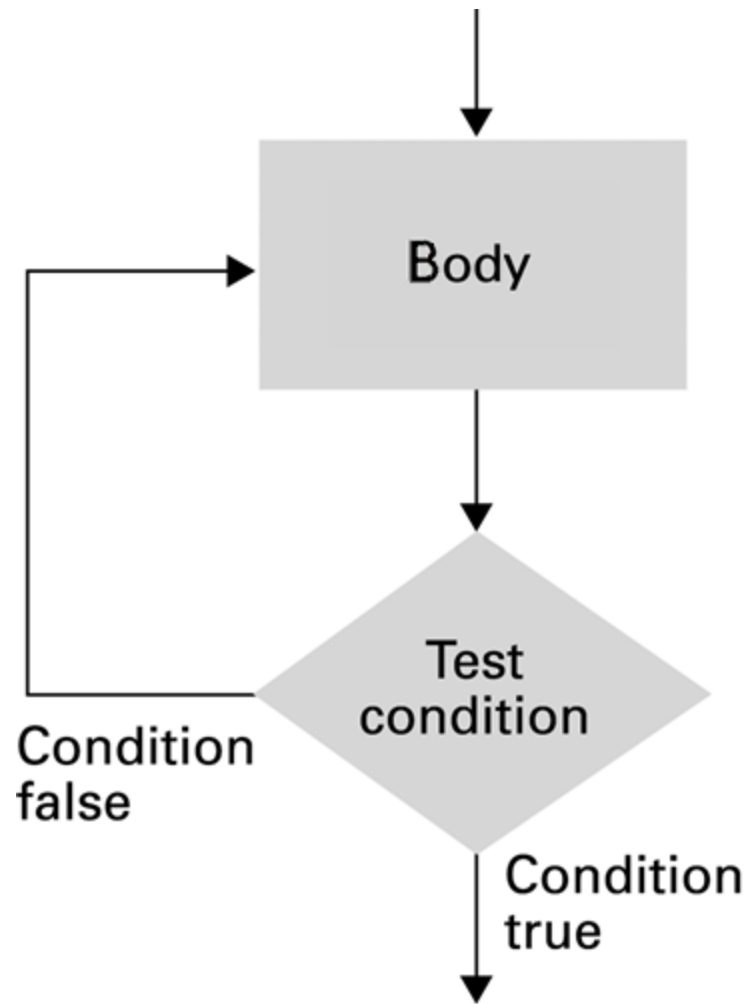


Figure 5.10 Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically

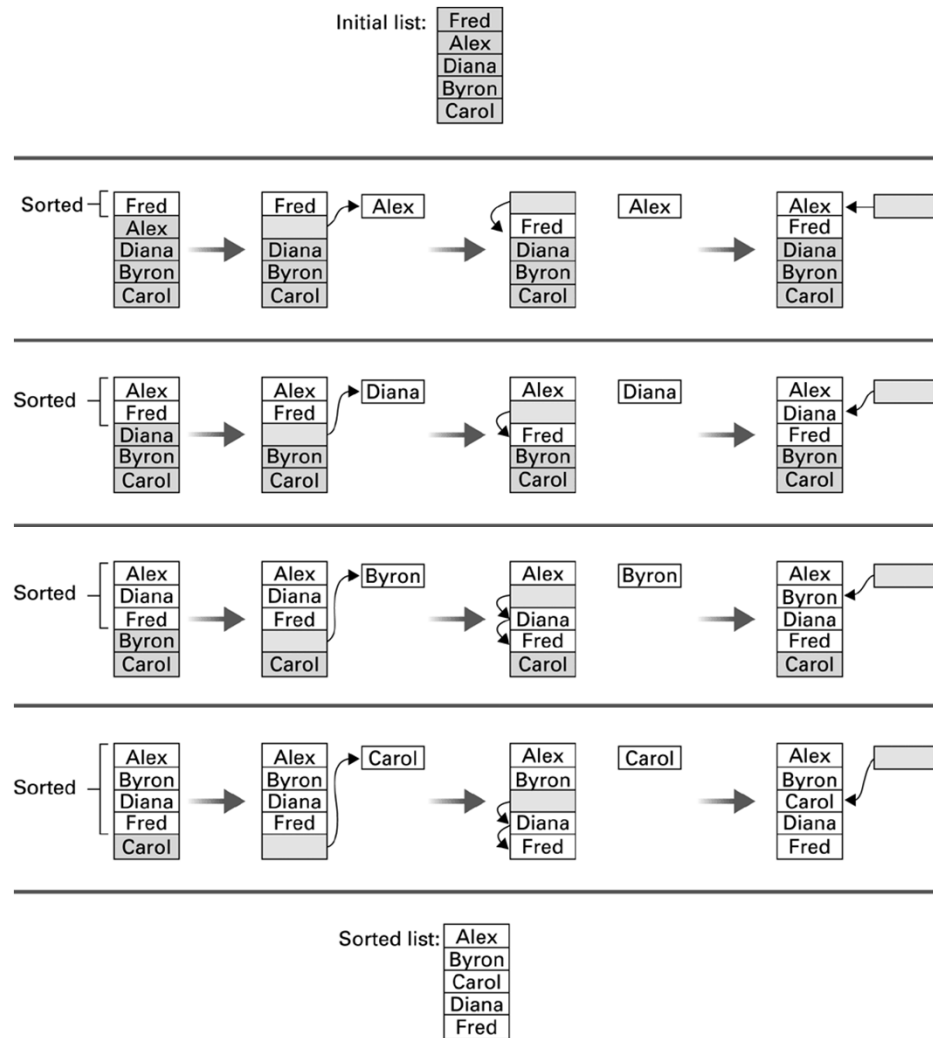


Figure 5.11 The insertion sort algorithm expressed in pseudocode

```
def Sort(List):  
    N = 2  
    while (N <= length of List):  
        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
```

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.

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

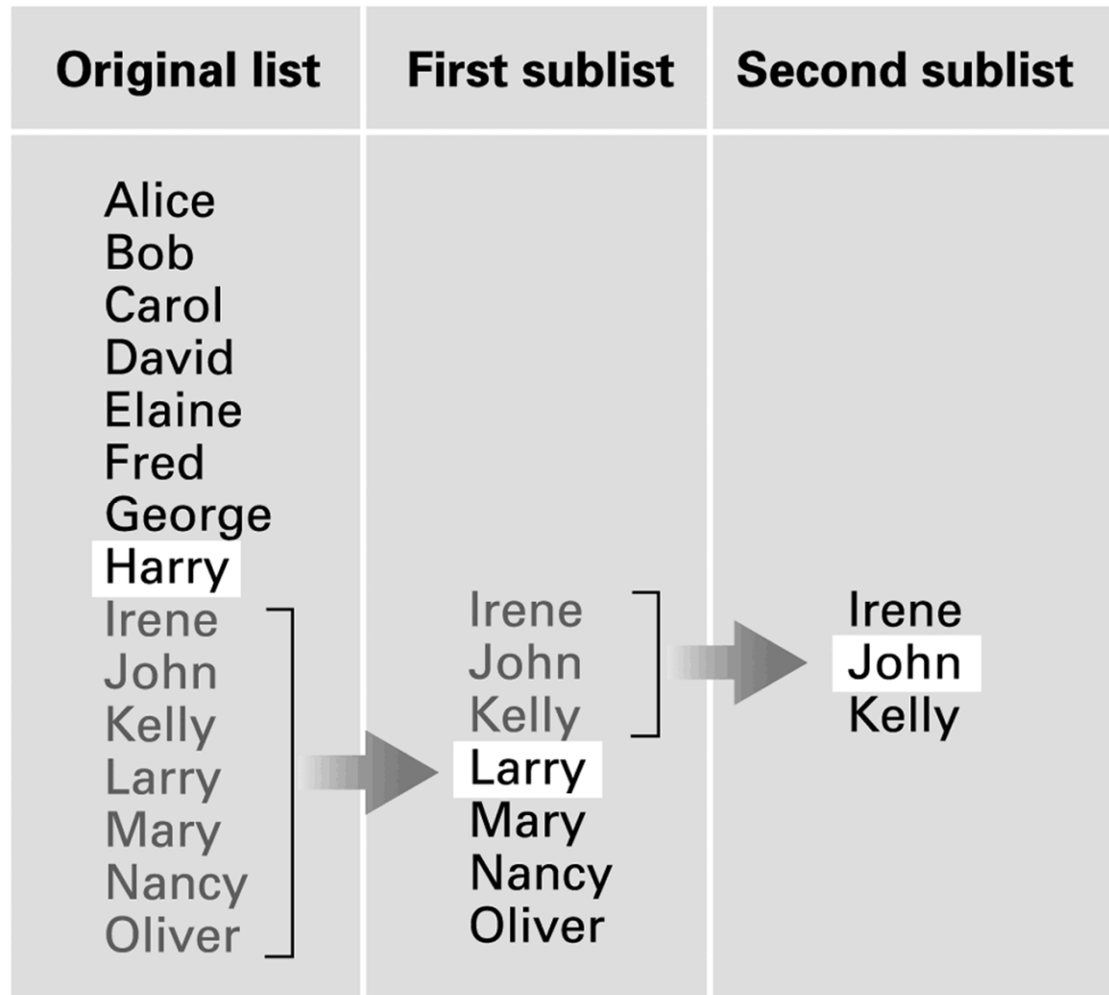


Figure 5.13 A first draft of the binary search technique

```
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

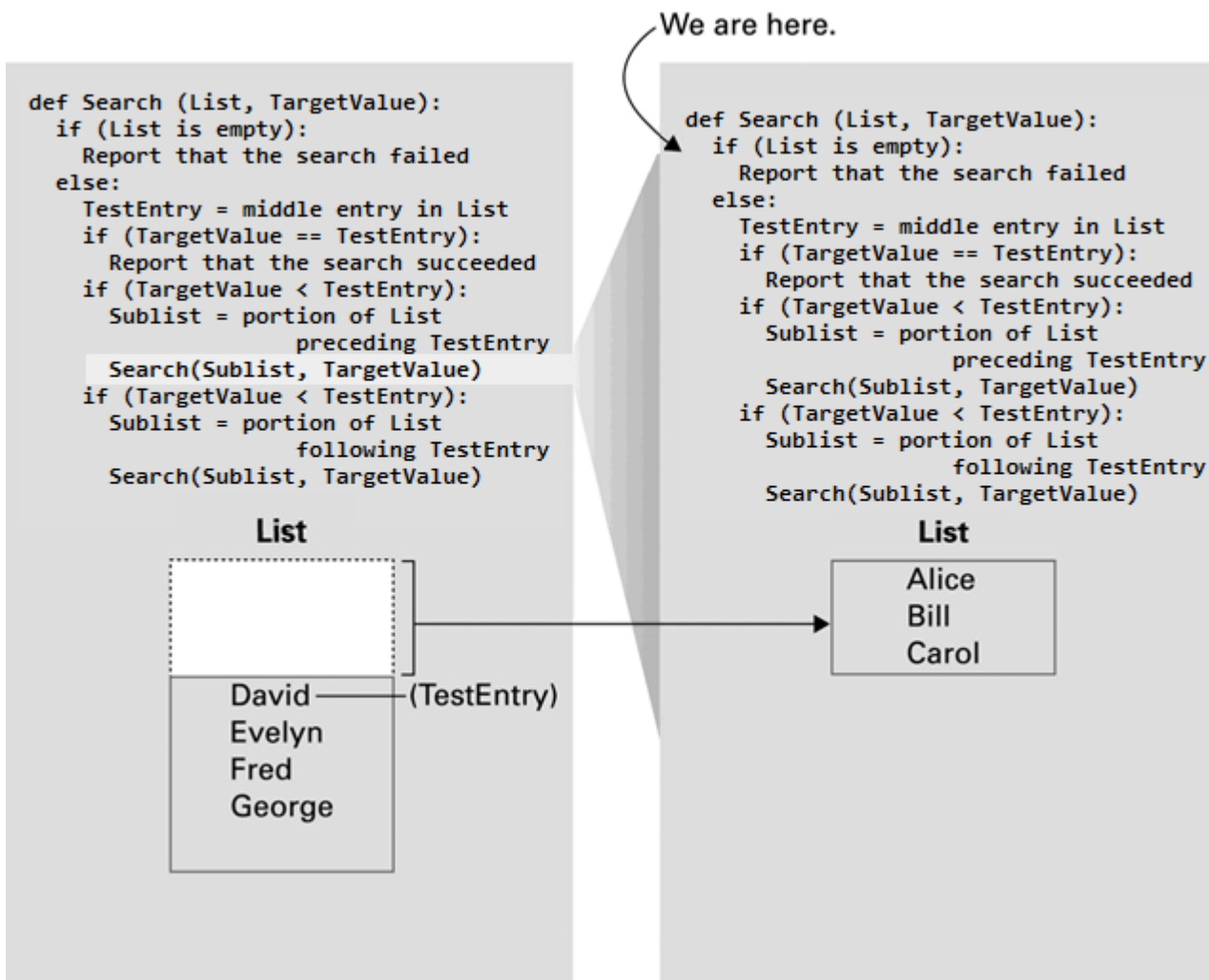


Figure 5.16

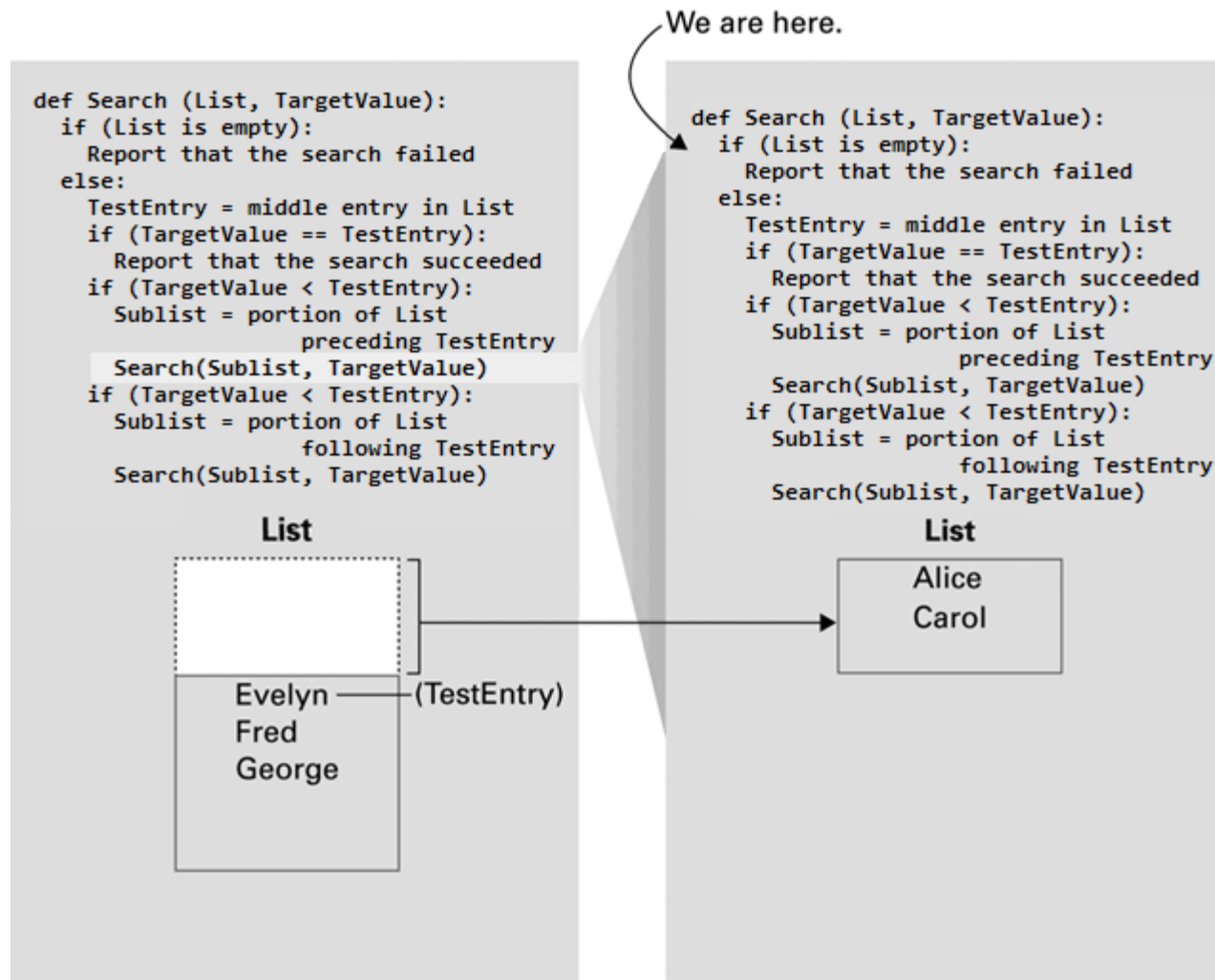
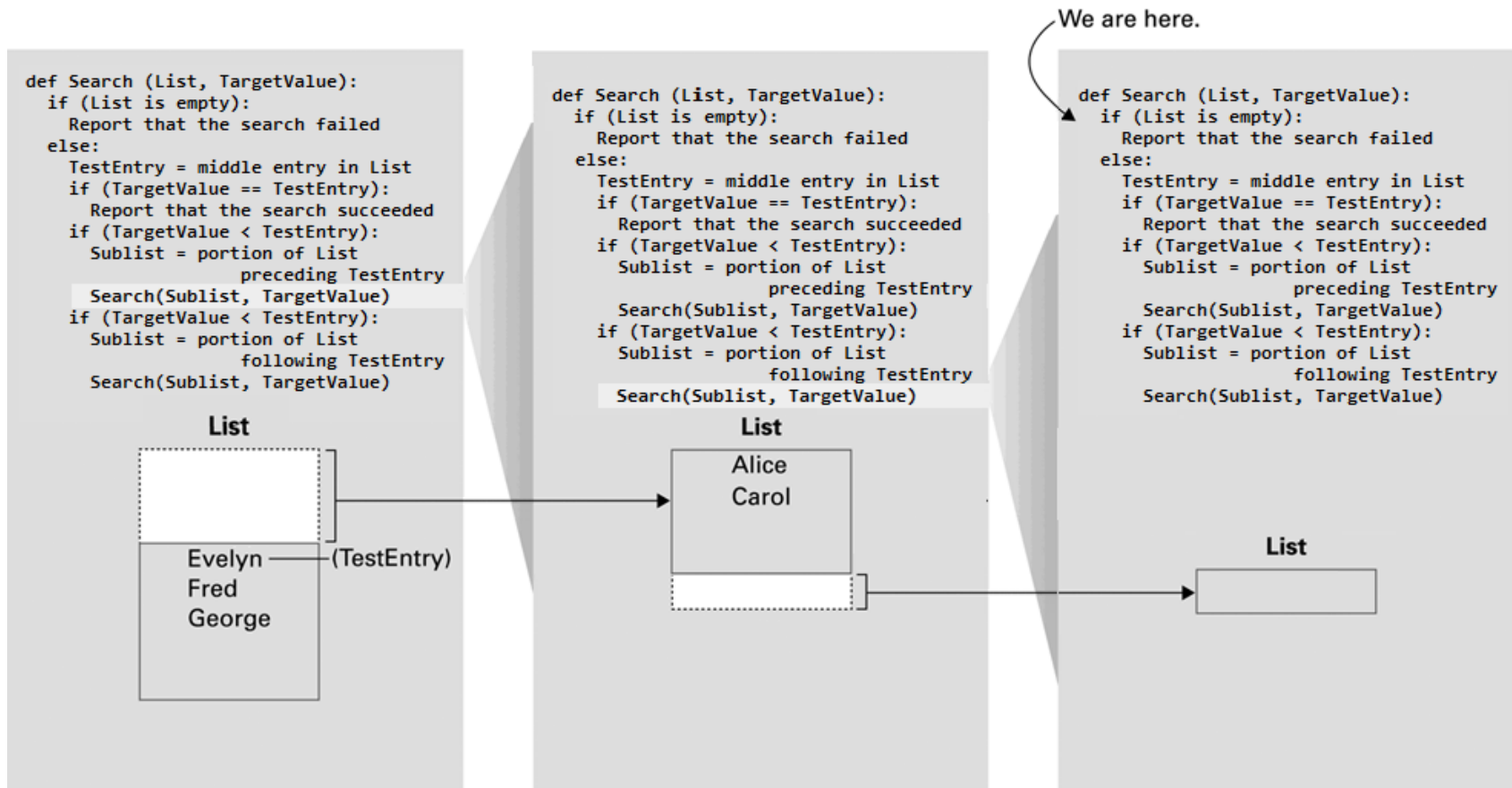


Figure 5.17



Algorithm Efficiency

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

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 David Carol Barbara Alfred	1 → Elaine David Carol Barbara Alfred	3 → David Elaine 2 → Carol Barbara Alfred	6 → Carol David 5 → Elaine 4 → Barbara Alfred	10 → Barbara 9 → Carol 8 → David 7 → Elaine Alfred	Alfred Barbara Carol David Elaine

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

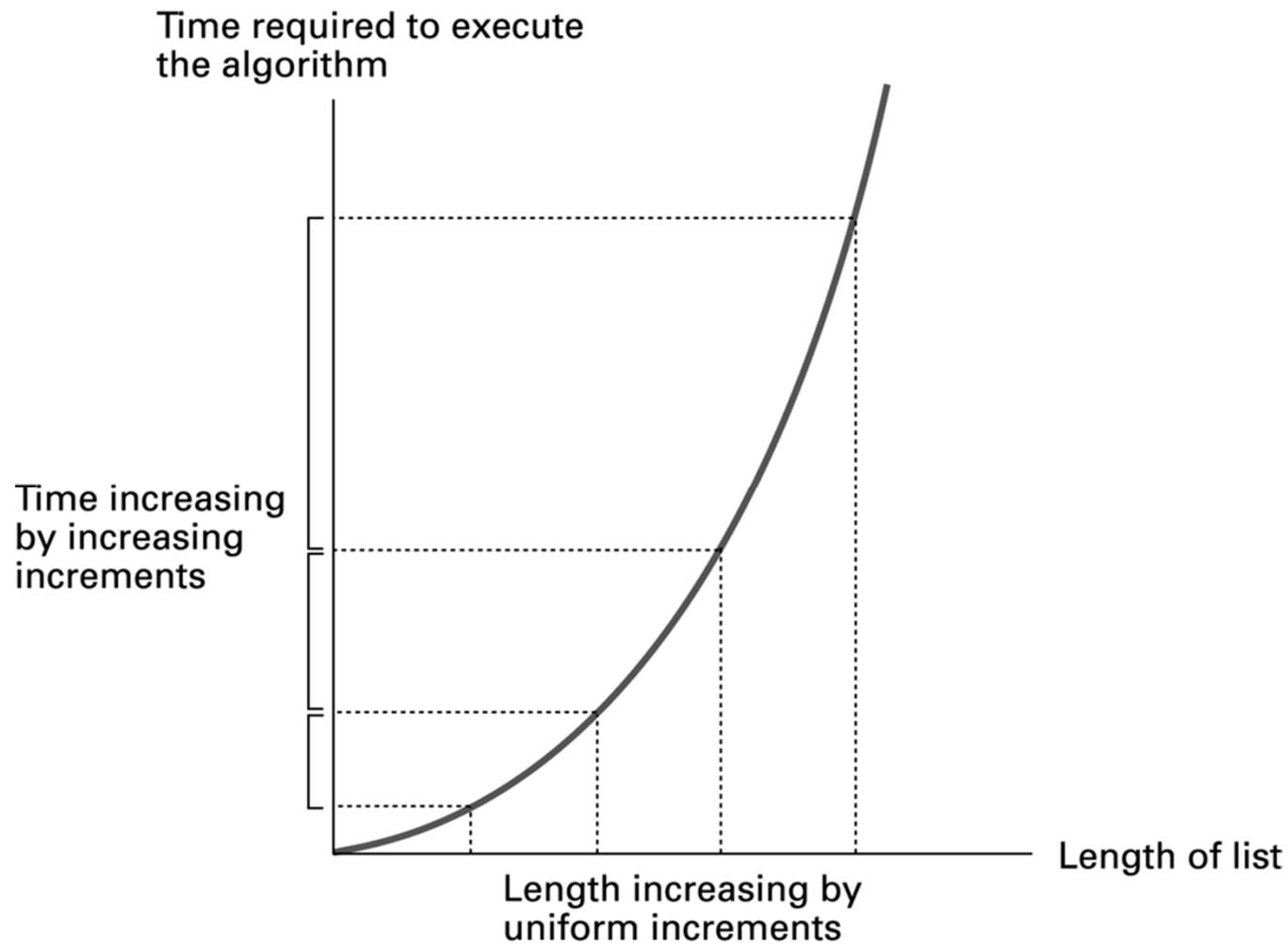
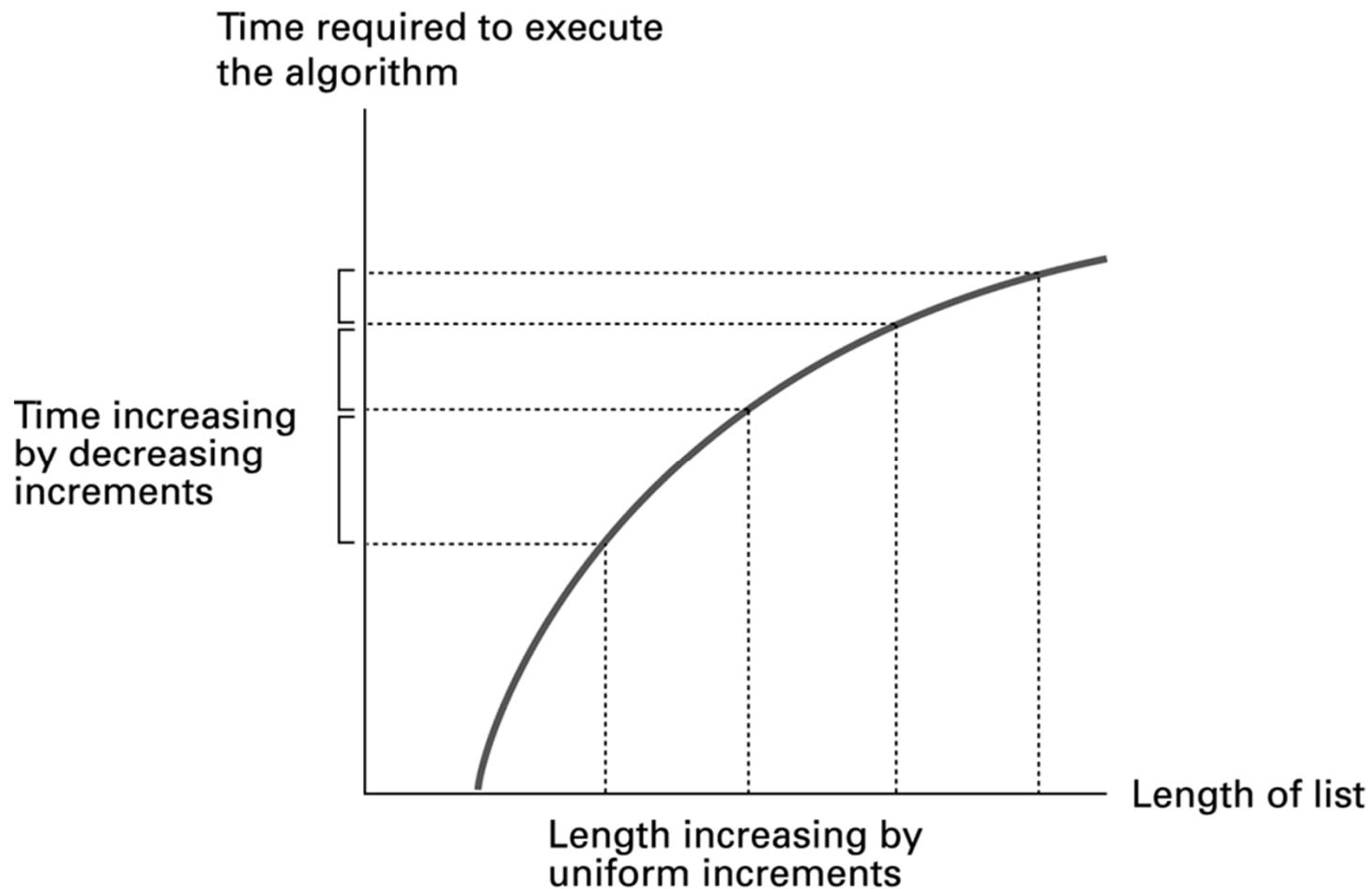


Figure 5.20 Graph of the worst-case analysis of the binary search algorithm



Software Verification

- Proof of correctness
 - Assertions
 - Preconditions
 - Loop invariants
- Testing

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

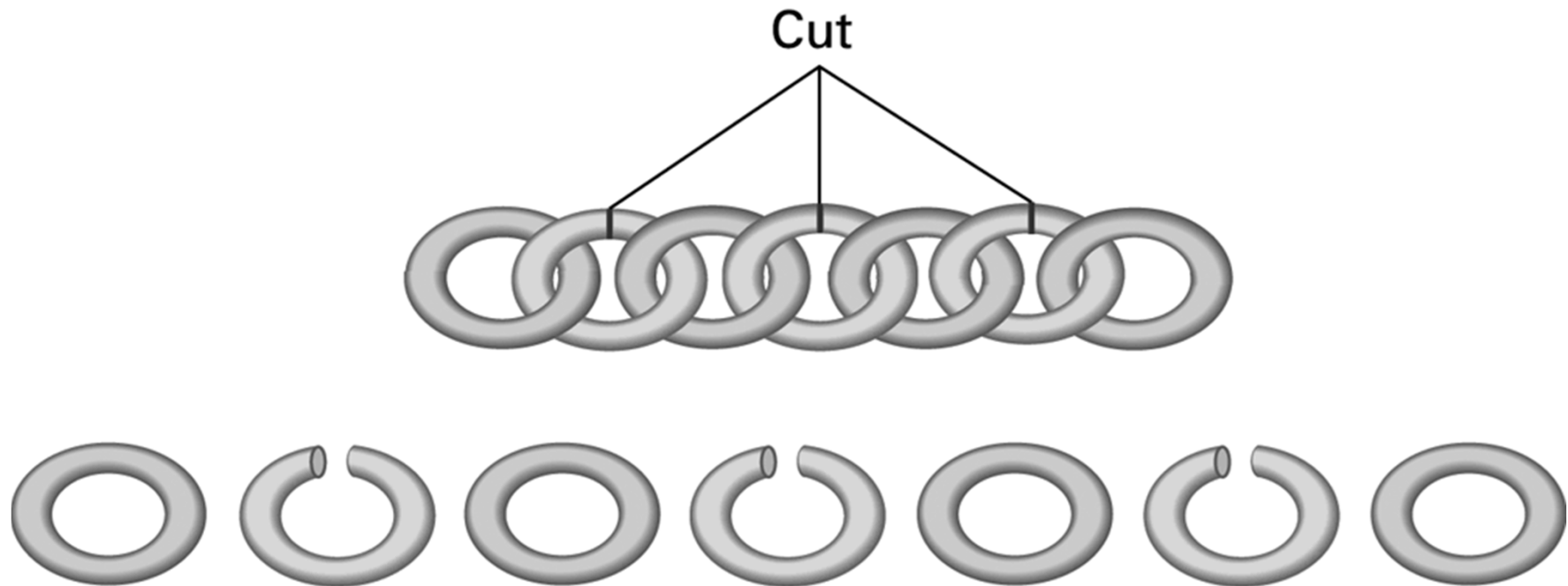


Figure 5.22 Solving the problem with only one cut

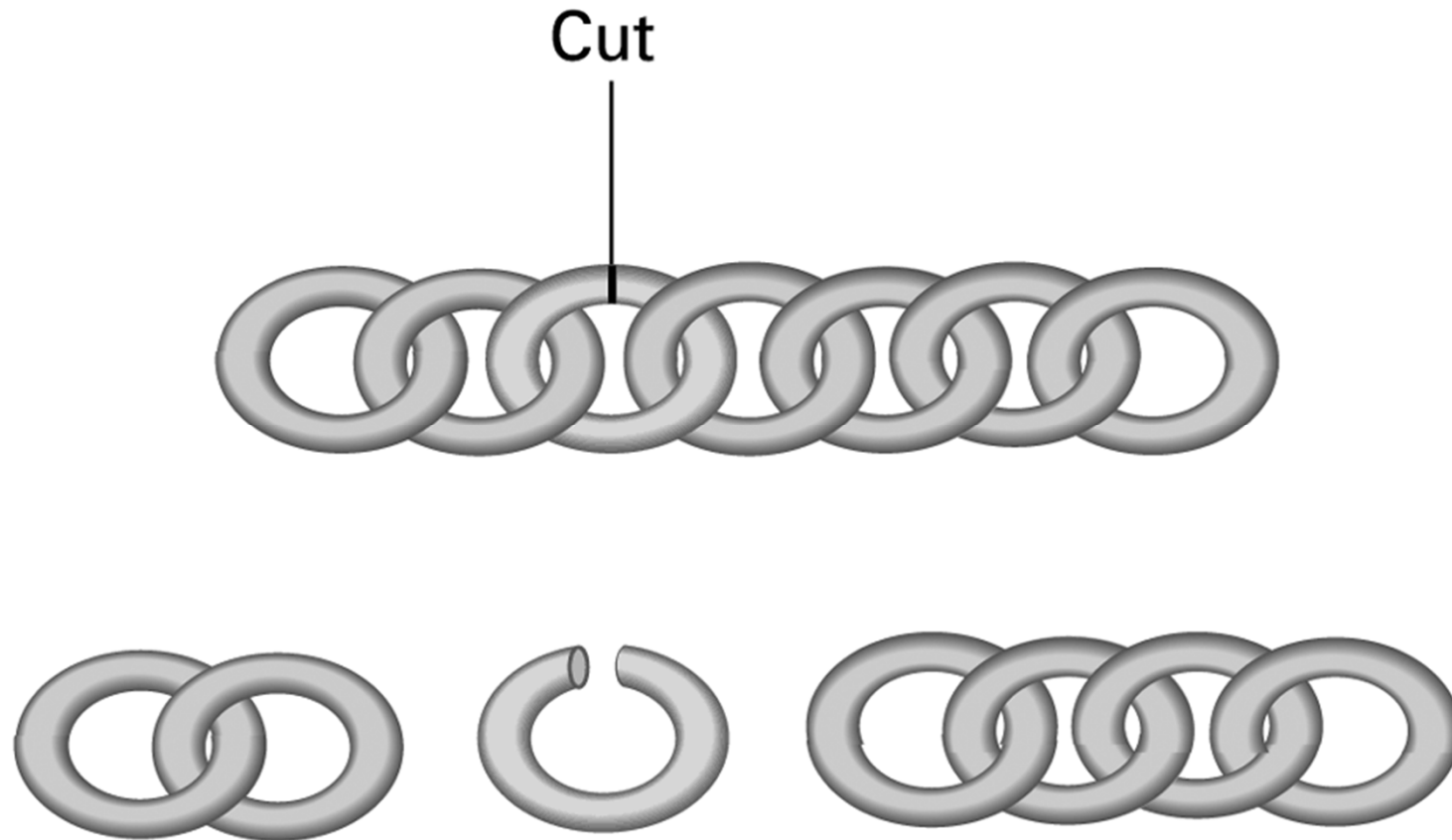
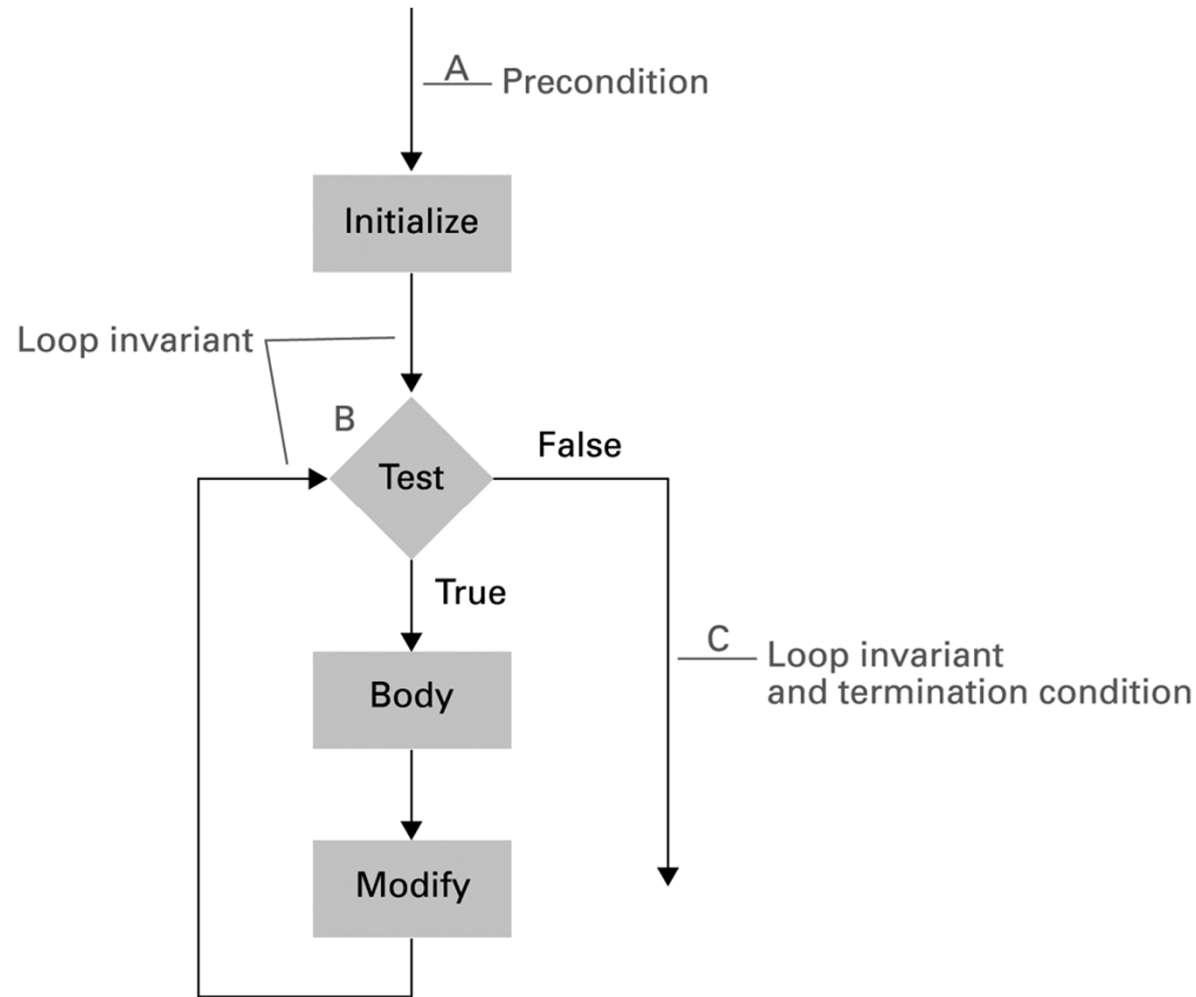


Figure 5.23 The assertions associated with a typical while structure



Q&A