Introduction to Computer Science

Polly Huang
NTU EE
http://homepage.ntu.edu.tw/~pollyhuang
pollyhuang@ntu.edu.tw

Chapter 5

Algorithms

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

- An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.
- Program
  - Formal representation of an algorithm
- Process
  - Activity of executing a program

Ordered Set

- Steps in an algorithm must have a well-established structure in terms of the order in which its steps are executed
- Each step must be an executable instruction
  - Example: "Making a list of all the positive integers" is not an executable instruction
- May involve more than one thread (parallel algorithms)

Unambiguous Steps

- During execution of an algorithm, the information in the state of the process must be sufficient to determine uniquely and completely the actions required by each step
- The execution of each step in an algorithm does not require creative skills. Rather, it requires only the ability to follow directions.

Terminating Process

- All execution of an algorithm must lead to an end
- There are, however, many meaningful application for non-terminating processes
- Computer science seeks to distinguish both
  - Problems whose answers can be obtained algorithmically
  - Problems whose answers lie beyond the capabilities of algorithmic systems

An algorithm is

an ordered set of unambiguous, executable steps that defines a terminating process.

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

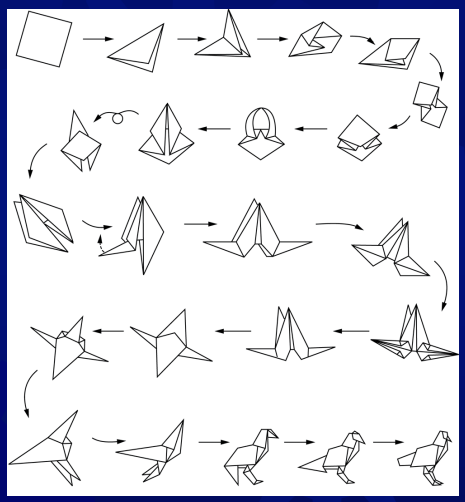
Algorithm and Its Representation

- Like a story and a story book
- Example: converting temperature readings from Celsius to Fahrenheit
  - 1. F = (9/5)C + 32
  - 2. Multiply the temperature reading in Celsius by 9/5 and then add 32 to the product
  - 3. Implemented by electronic circuit
- Underlying algorithm is the same, only the representations differ

Level of Details

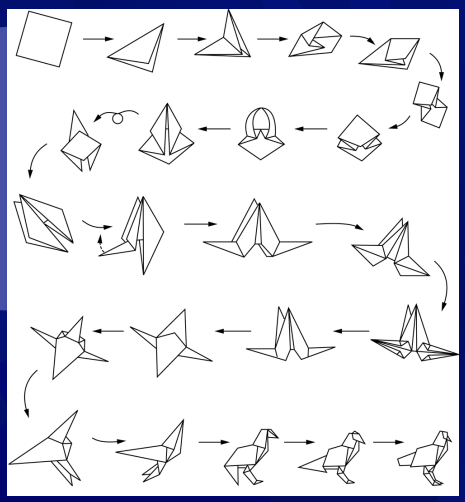
- May cause problems in communicating algorithms
- Example:
  - "Convert the Celsius reading to its Fahrenheit equivalent"
  - This might suffice for meteorologists
  - But a layperson would argue that this instruction is ambiguous
  - The problem is that the algorithm is not represented in enough detail for the layperson

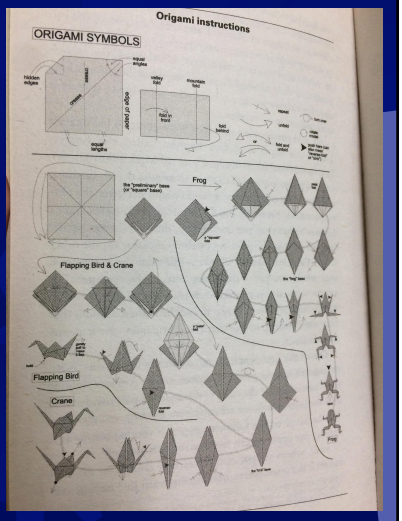
An Example: Origami



Quiz Time!

Another Example: Origami

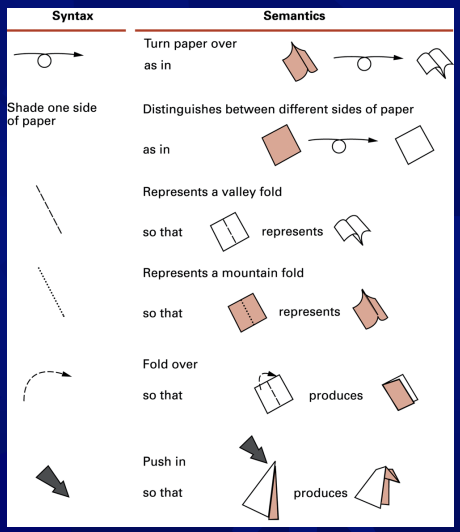




Algorithm Representation

- Primitive
  - Set of building blocks from which algorithm representations can be constructed
- Programming language
  - Collection of primitives
  - Collection of rules stating how the primitives can be combined to represent more complex ideas

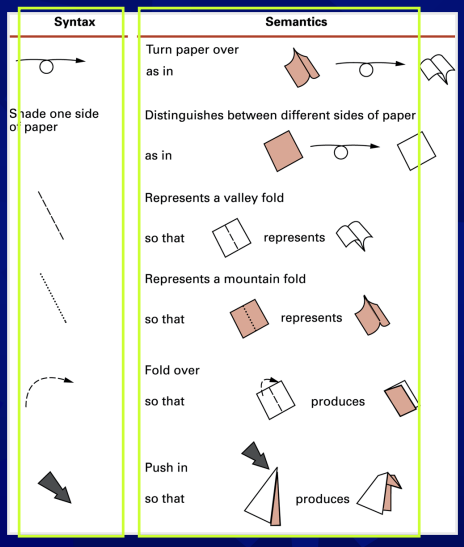
Origami Primitives



Primitives

- Syntax
  - Symbolic representation
- Semantics
  - Concept represented (meaning of the primitive)

Origami Primitives



Levels of Abstraction

- Algorithm
  - Procedure to solve the problem
  - Often one of many possibilities
- Representation
  - Description of algorithm sufficient to communicate it to the desired audience
  - Always one of many possibilities

Quiz Time!

Machine Instructions as Primitives

- Algorithm based on machine instructions is suitable for machine execution
- However, expressing algorithms at this level is tedious
- Normally uses a collection of higher level primitives, each being an abstract tool constructed from the low-level primitives provided in the machine's language

Pseudocode

- Less formal, more intuitive than the formal programming languages
- A notation system in which ideas can be expressed informally during the algorithm development process
- A consistent, concise notation for representing recurring semantic structure
- Comparison with flow chart

Pseudocode Primitives

- Assignment
  - *name* ← *expression*
- Conditional selection
  - **if** *condition* **then** *action*
- Repeated execution
  - **while** *condition* **do** *activity*
- Procedure
  - **procedure** *name* (*generic names*)

An Example: Greetings

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

Quiz Time!

Basic Primitives

- total ← price + tax
- if (sales have decreased)
  then (lower the price by 5%)
- if (year is leap year)
  then (divide total by 366)
  else (divide total by 365)
- while(tickets remain to be sold) do
  (sell a ticket)

Procedure Primitive

- total ← price + tax
- tax?
- A procedure to calculate tax

Tax as a Procedure

```
Procedure tax
  if (item is taxable)
  then (if (price > limit)
          then (return price*0.1000)
          else (return price*0.0825 )
        )
  else (return 0)
```

Nested Statements

- One statement within another

```
if (item is taxable)
then (if (price > limit)
       then (return price*0.1000)
       else (return price*0.0825 )
      )
else (return 0)
```


Indentations

- Easier to tell the levels of nested statements

```
if (item is taxable)
then (if (price > limit)
       then (return price*0.1000)
       else (return price*0.0825 )
      )
else (return 0)
```

Structured Program

- Divide the long algorithm into smaller tasks
- Write the smaller tasks as procedures
- Call the procedures when needed
- This helps the readers to understand the structure of the algorithm

```
if (customer credit is good)
then (ProcessLoan)
else (RejectApplication)
```


The Point of Pseudocode

- To communication the algorithm to the readers
- The algorithm will later turn into program
- Also help the program maintainer or developer to understand the program

Quiz Time!

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

Algorithm Discovery

- Development of a program consists of
  - Discovering the underlying algorithm
  - Representing the algorithm as a program
- Algorithm discovery is usually the more challenging step in the software development process
- Requires finding a method of solving the problem

Problem Solving Steps

1. Understand the problem
2. Get an idea
3. Formulate the algorithm and represent it as a program
4. Evaluate the program
   1. For accuracy
   2. For its potential as a tool for solving other problems

Not Yet Sure What to Do

1. Understand the problem
2. Get an idea
3. Formulate the algorithm and represent it as a program
4. Evaluate the program
   1. For accuracy
   2. For its potential as a tool for solving other problems

Difficulties

- Understanding the problem
  - There are complicated problems and easy problems
  - A complete understanding of the problem before proposing any solutions is somewhat idealistic
- Get an idea
  - Take the 'Algorithm' course
  - Mysterious inspiration

Getting a Foot in the Door

- Work the problem backwards
  - Solve for an example and then generalize
  - Solve an easier related problem
    - Relax some of the problem constraints
- Divide and conquer
  - Stepwise refinement
    - top-down methodology
    - Popular technique because it produces modular programs
  - Solve easy pieces of the problem first
    - bottom up methodology

Work the Problem Backwards

- Simplify the problem
- Build up a scenario for the simplified problem
- Try to solve this scenario
- Generalize the special solution to general scenarios
- Consider a more general problem
- Repeat the process

Example Problem

- Person A is assigned the task of determining the ages of B's three children.
- B tells A that the product of the children's ages is **X**.
- A replies that another clue is required.
- B tells A the sum of the children's ages **Y**.
- 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.

- Come out with an algorithm for person A

Try This First!

a.k.a. Quiz Time!

Solving the Problem

a. Triples whose product is 36		**b.** Sums of triples from part (a)	
(1,1,36)	(1,6,6)	1 + 1 + 36 = 38	1 + 6 + 6 = 13
(1,2,18)	(2,2,9)	1 + 2 + 18 = 21	2 + 2 + 9 = 13
(1,3,12)	(2,3,6)	1 + 3 + 12 = 16	2 + 3 + 6 = 11
(1,4,9)	(3,3,4)	1 + 4 + 9 = 14	3 + 3 + 4 = 10

Now Try This Problem Again!

- Person A is assigned the task of determining the ages of B's three children.
- B tells A that the product of the children's ages is **X**.
- A replies that another clue is required.
- B tells A the sum of the children's ages **Y**.
- 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.

- Come out with a solution for A

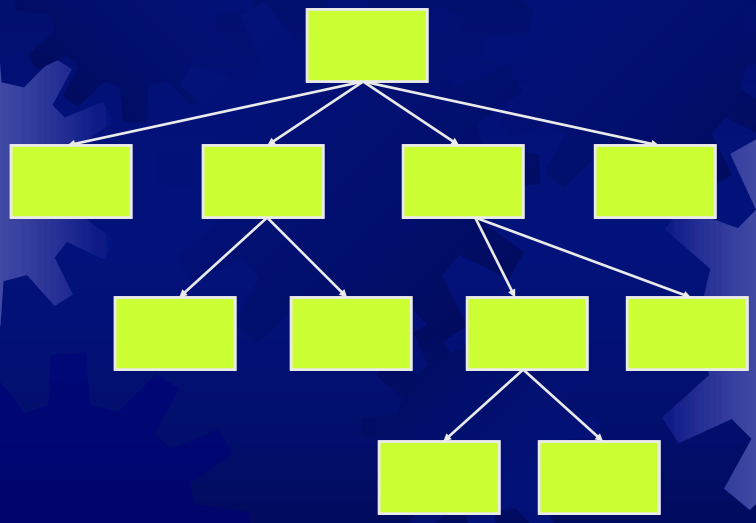
Divide and Conquer Concept

- Not trying to conquer an entire task at once
- First view the problem at hand in terms of several subproblems
- Approach the overall solution in terms of steps, each of which is easier to solve than the entire original problem

Divide and Conquer

- Steps be decomposed into smaller steps
- These smaller steps be broken into still smaller ones
- Until the entire problem has been reduced to a collection of easily solved subproblems
- Solve from the small subproblems and gradually have it all.

Divide and Conquer Illustrated



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

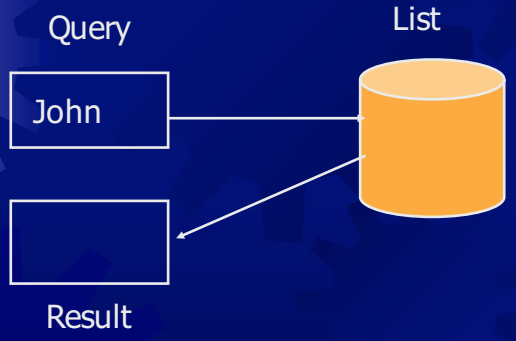
Iterative Structures

- Used in describing algorithmic process
- A collection of instructions is repeated in a looping manner

Search Problem

- Search a list for the occurrence of a particular target value
- If the value is in the list, we consider the search a success; otherwise we consider it a failure
- Assume that the list is sorted according to some rule for ordering its entries

Search Scenario

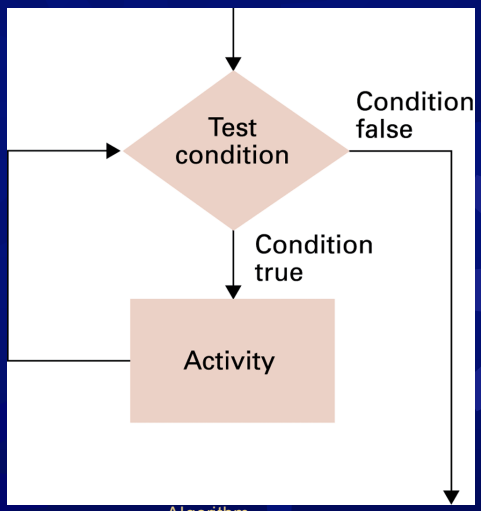


Alice
Bob
Carol
David
Elaine
Fred
George
Harry
Irene
John
Kelly
Larry
Mary
Nancy
Oliver

Sequential Search Algorithm

```
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
```

The while Loop



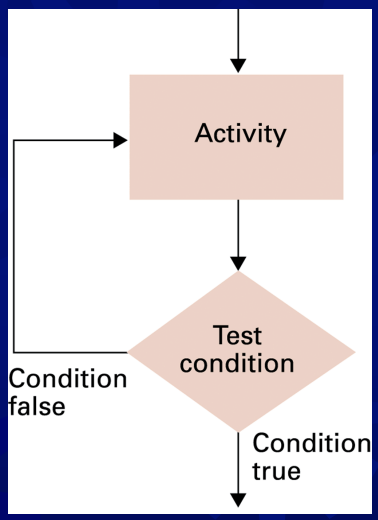
Components of Repetitive Control

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

while vs. *repeat* Structure

- In repeat structure the loop's body is always performed at least once (posttest loop)
- While in while structure, the body is never executed if the termination is satisfied the first time it is tested (pretest loop)

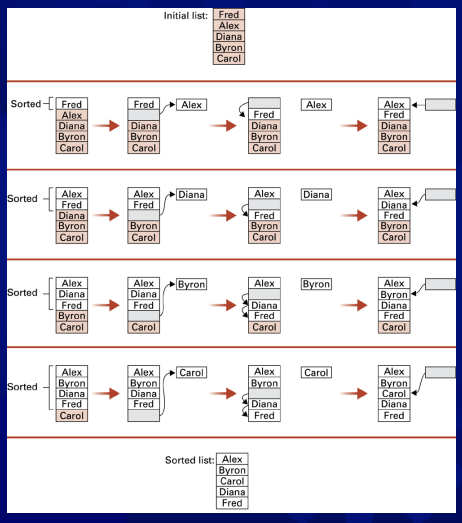
The repeat Loop



A Sort Problem

- Sort a list of names into alphabetical order
- The constraint is to sort the list "within itself"
- In other words, sort by shuffling its entries as opposed to moving the list to another location
- Typical in computer applications to use the storage space efficiently

Insertion Sort



Insertion Sort Algorithm

```
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
      )
```

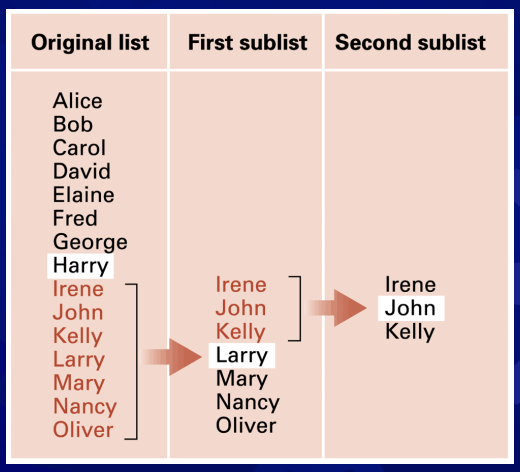

Chapter 5: Algorithms

- 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

Recursive Structures

- Involves repeating the set of instructions as a subtask of itself
- An example is in processing incoming telephone calls using the call-waiting feature
  - An incomplete telephone conversation is set aside while another incoming call is processed
  - Two conversations are performed
  - But not in a one-after-the-other manner as in the loop structure
  - Instead one is performed within the other

Binary Search (for John)



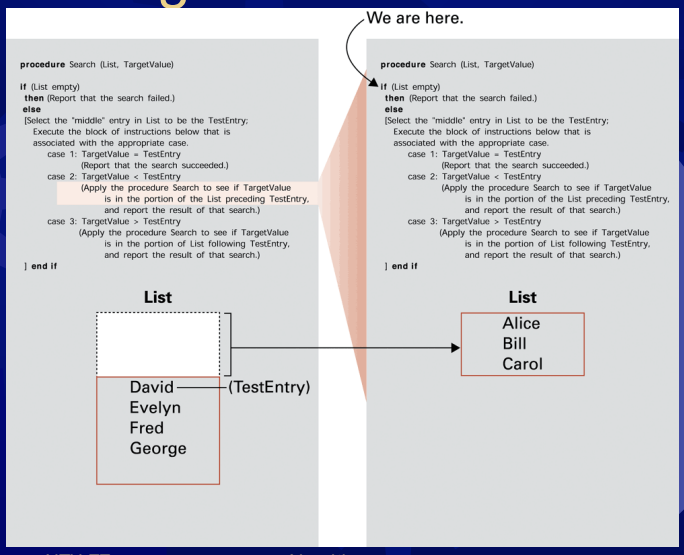
Binary Search Algorithm

```
if (List empty)
 then
  (Report that the search failed.)
else
   [Select the "middle" entry in the 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
                  (Search the portion of List preceding TestEntry for
                        TargetValue, and report the result of that search.)
        case 3: TargetValue > TestEntry
                  (Search the portion of List following TestEntry for
                        TargetValue, and report the result of that search.)
   ] end if
```

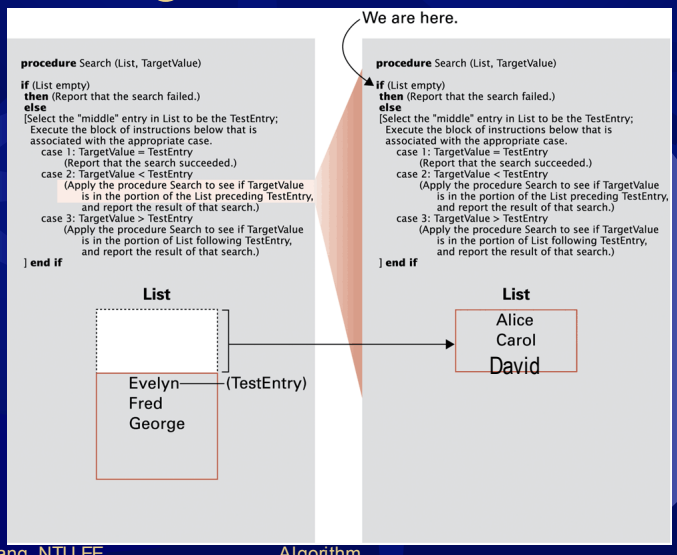

Binary Search Algorithm in Pseudocode

```
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
```

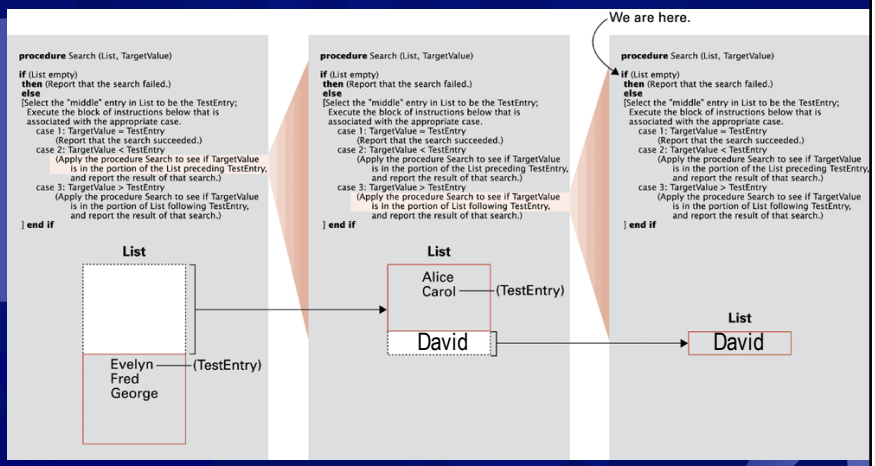
Searching for Bill



Searching for David



Searching for David



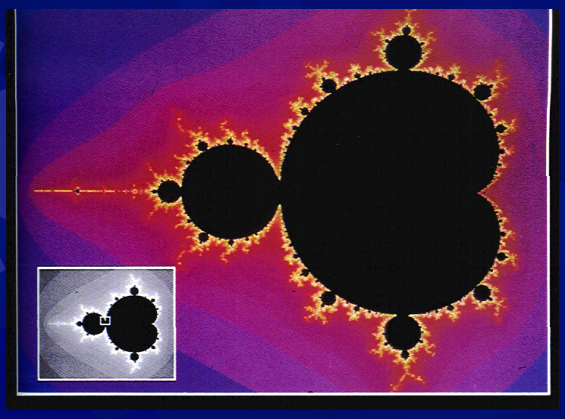
Recursion

- Execution is performed in which each stage of repetition is as a subtask of the previous stage
- Example: divide-and-conquer in binary search

Characteristics of Recursion

- Existence of multiple copies of itself (or multiple activations of the program)
- At any given time only one is actively progressing
- Each of the others waits for another activation to terminate before it can continue

Mandelbrot Set



•Credit: J. Gleick 原著, 林和譯, 混沌: 不測風雲的背後(Chaos), 天下文化, 1991

Recursive Control

- Also involves
  - Initialization
  - Modification
  - Test for termination (degenerative case)
- Test for degenerative case
  - Before requesting further activations
  - If not met, assigns another activation to solve a revised problem that is closer to the termination condition
- Similar to a loop control

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

Software Efficiency

- Measured as number of instructions executed
- Θ notation for efficiency classes
- Best, worst, and average case

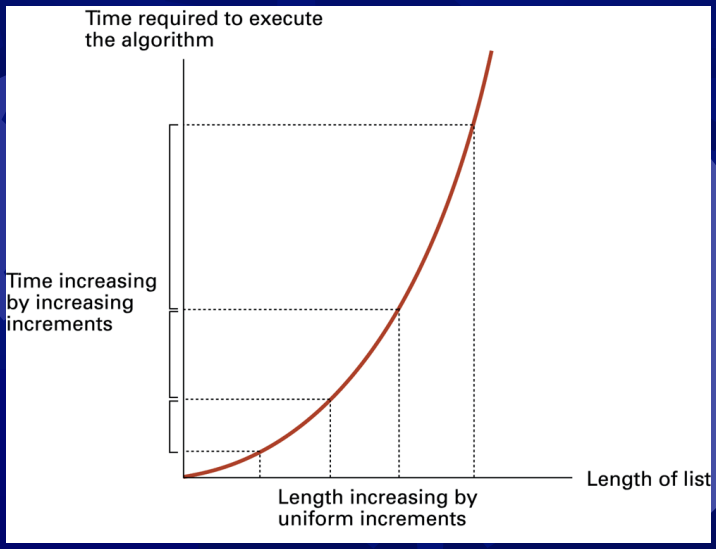
Insertion Sort in Worst Case

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

Number of Executions

- # of shifts for a list of N entries in the worst case?
- 1+2+3+…+N-1
- (1+N-1)*(N-1)/2 → $(N^2-N)/2$

Worst-Case Analysis Insertion Sort

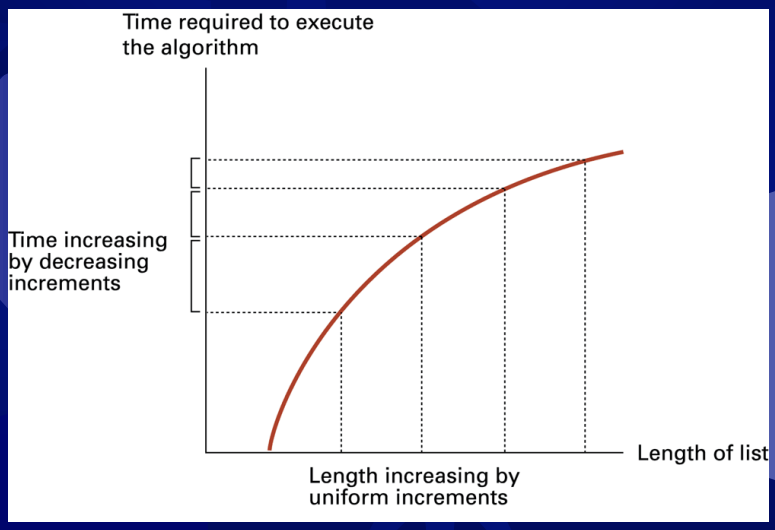


Binary Search in Worst Case

Number of Executions

- # of comparing to the middle entry for a list of N entries in the worst case?
- 1, 3, 7, 15, 31, 63 → N
- 2, 4, 8, 16, 32, 64 → N+1
- $\log_2(N+1)$ → # of comparsions

Worst-Case Analysis Binary Search



Big-theta Notation

- Identification of the shape of the graph representing the resources required with respect to the size of the input data
  - Normally based on the worst-case analysis
  - Insertion sort: $\Theta(n^2)$ $(n^2-n)/2$
  - Binary search: $\Theta(\lg n)$ $\lg_2(n+1)$

Formal Definition

- $\Theta(n^2)$: complexity is $kn^2+o(n^2)$
  - $f(n)/n^2 \rightarrow k$, $n \rightarrow \infty$
- $o(n^2)$: functions grow slower than n^2
  - $f(n)/n^2 \rightarrow 0$, $n \rightarrow \infty$

$(n^2-n)/2 = \frac{1}{2} n^2 - \frac{1}{2} n \rightarrow o(n^2)$

Problem Solving Steps

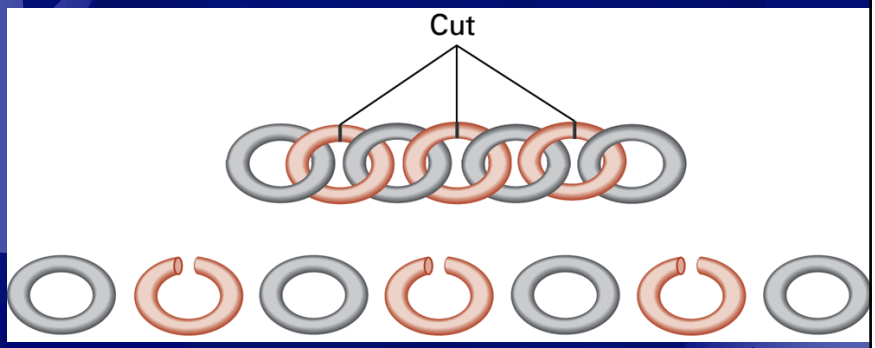
1. Understand the problem
2. Get an idea
3. Formulate the algorithm and represent it as a program
4. Evaluate the program
   1. For its potential as a tool for solving other problems (the faster, the better)
   2. For accuracy

Software Verification

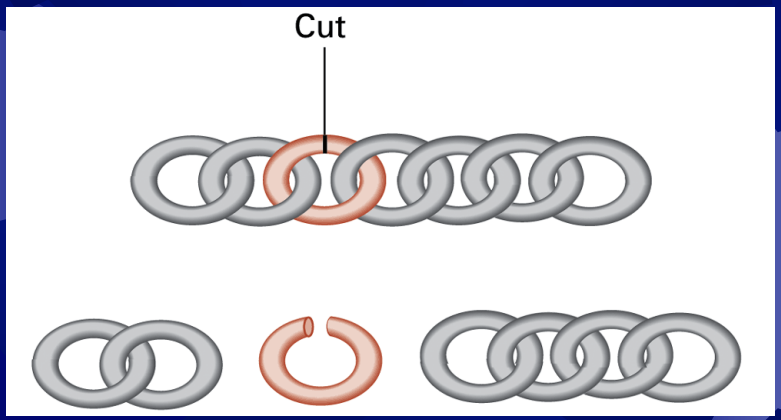
- Evaluate the accuracy of the solution
- This is not easy
- The programmer often does not know whether the solution is accurate (enough)
- Example: Traveler's gold chain

Quiz Time!

Separating the chain using only three cuts



Solving the problem with only one cut



Moral of the Story

- You thought there is no better way
- You thought it is accurate enough
- But really, who knows?

Ways to Level the Confidence

- For perfect confidence
  - Prove the correctness of a algorithm
  - Application of formal logic to prove the correctness of a program
- For high confidence
  - Exhaustive tests
  - Application specific test generation
- For some confidence
  - Program verification
  - Assertions

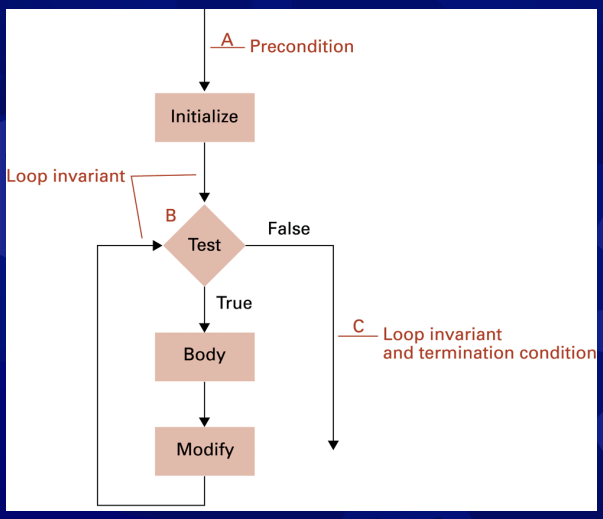
Program Conditions

- Preconditions
  - Conditions satisfied at the beginning of the program execution
- Propagations
  - The next step is to consider how the consequences of the preconditions propagate through the program

Program Verification

- Assertions
  - Statements that can be established at various points in the program
  - To check for rightful precondition & propagation
- Proof of correctness to some degree
  - Establish a collection of assertions
  - If all assertions pass
  - The program is correct to some degree

In a *while* Structure



Insertion Sort Algorithm

```
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
      )
```

Asserting of Insertion Sort

- Precondition
  - N == 2, trivial
- Loop invariant of the outer loop
  - The names preceding the Nth entry form a sorted list
- Termination condition
  - The value of N is greater than the length of the list
  - The list is sorted

Questions?