Algorithms

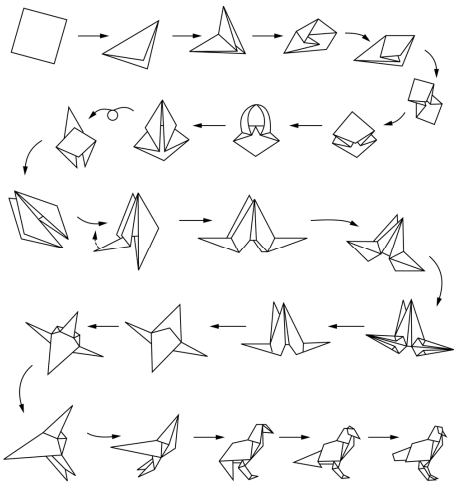
Definition of Algorithm

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

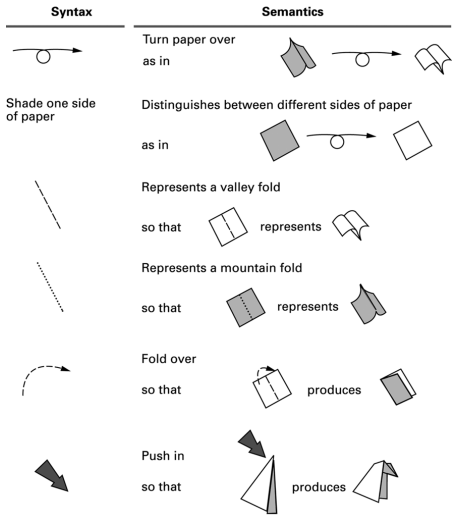
Algorithm Representation

- Requires well-defined primitives
- A collection of primitives that the computer can follow constitutes a programming language.

Folding a bird from a square piece of paper



Origami primitives



Pseudocode Primitives

- Pseudocode is "sort of" code that a computer can understand, but a higher level to be more easily human understandable
  - But becomes pretty straightforward to convert to an actual programming language
- Assignment

  name ← *expression*
- Conditional selection

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

Pseudocode Primitives (continued)

- Repeated execution

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

- Procedure (aka Method, Subroutine, Function)

  procedure *name*

  list of primitives associated with name

The procedure Greetings in pseudocode

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

Running Example

- You are running a marathon (26.2 miles) and would like to know what your finishing time will be if you run a particular pace. Most runners calculate pace in terms of minutes per mile. So for example, let's say you can run at 7 minutes and 30 seconds per mile. Write a program that calculates the finishing time and outputs the answer in hours, minutes, and seconds.

- Input:
  Distance : 26.2
  PaceMinutes: 7
  PaceSeconds: 30
- Output:
  3 hours, 16 minutes, 30 seconds

One possible solution

- Express pace in terms of seconds per mile by multiplying the minutes by 60 and then add the seconds; call this SecsPerMile
- Multiply SecsPerMile * 26.2 to get the total number of seconds to finish. Call this result TotalSeconds.
- There are 60 seconds per minute and 60 minutes per hour, for a total of 60*60 = 3600 seconds per hour. If we divide TotalSeconds by 3600 and throw away the remainder, this is how many hours it takes to finish.
- The remainder of TotalSeconds / 3600 gives us the number of seconds leftover after the hours have been accounted for. If we divide this value by 60, it gives us the number of minutes.
- The remainder of (the remainder of(TotalSeconds / 3600) / 60) gives us the number of seconds leftover after the hours and minutes are accounted for
- Output the values we calculated!

Pseudocode

SecsPerMile ← (PaceMinutes * 60) + PaceSeconds
TotalSeconds ← Distance * SecsPerMile
Hours ← Floor(TotalSeconds / 3600)
LeftoverSeconds ← Remainder of (TotalSeconds / 3600)
Minutes ← Floor(LeftoverSeconds / 60)
Seconds ← Remainder of (LeftoverSeconds /60)

Output Hours, Minutes, Seconds as finishing time

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.

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?

Solution

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

Iterative Structures

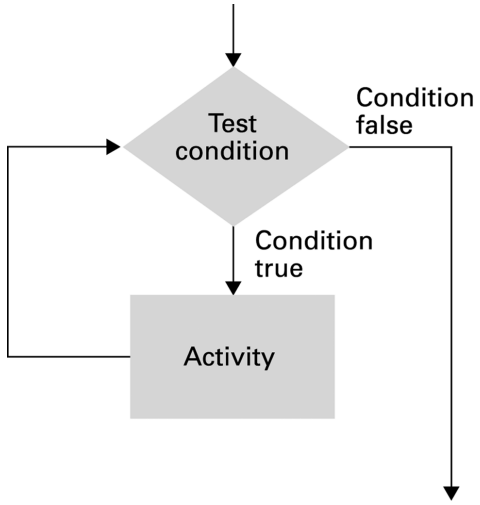
- Pretest loop:

```
while (condition) do
    (loop body)
```

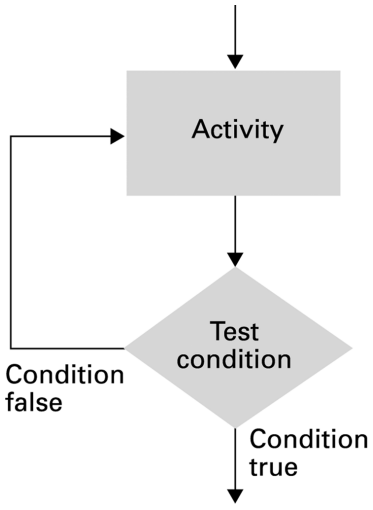
- Posttest loop:

```
repeat (loop body)
    until(condition)
```

The while loop structure



The repeat loop structure



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

Example: Sequential Search of a List

Fred

Alex

Diana

Byron

Carol

Want to see if Byron is in the list

The sequential search algorithm in pseudocode

```
procedure Search(List, TargetValue)
If (List is empty)
Then
        (Target is not found)
Else
        (
                name ← first entry in List
                while (no more names on the List)
                (
                        if (name = TargetValue)
                                (Stop, Target Found)
                        else
                                name ← next name in List
                )
                (Target is not found)
        )
```

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

Insertion Sort: Moving to the right, insert each name in the proper sorted location to its left

Fred Alex Diana Byron Carol

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
      )
```

1	2	3	4	5
Fred	Alex	Diana	Byron	Carol

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.
- Example: Binary Search

Applying our strategy to search a list for the entry John

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

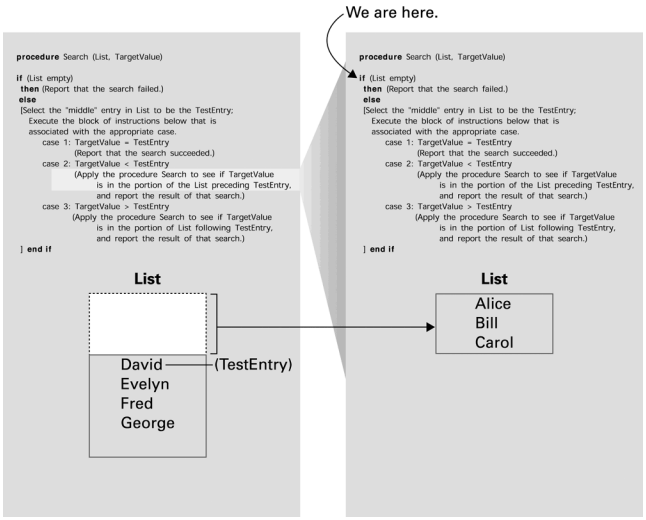
A first draft of the binary search technique

```
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
```

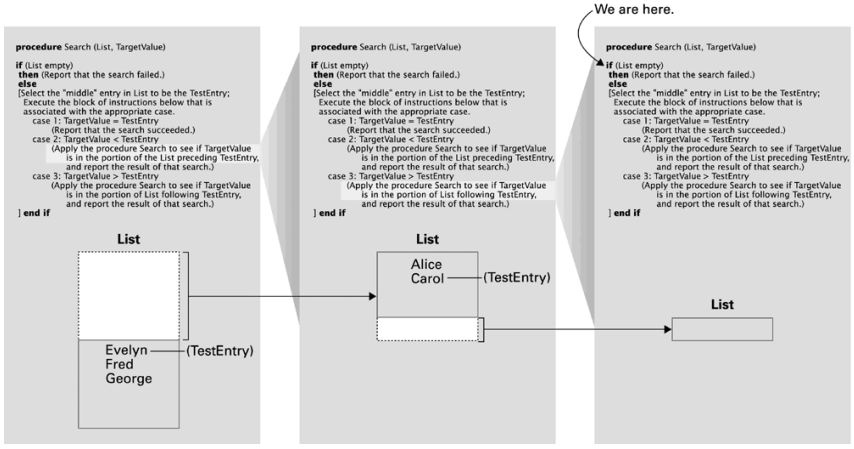
The binary search algorithm in pseudocode

```
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
```

Searching for Bill



Searching for David



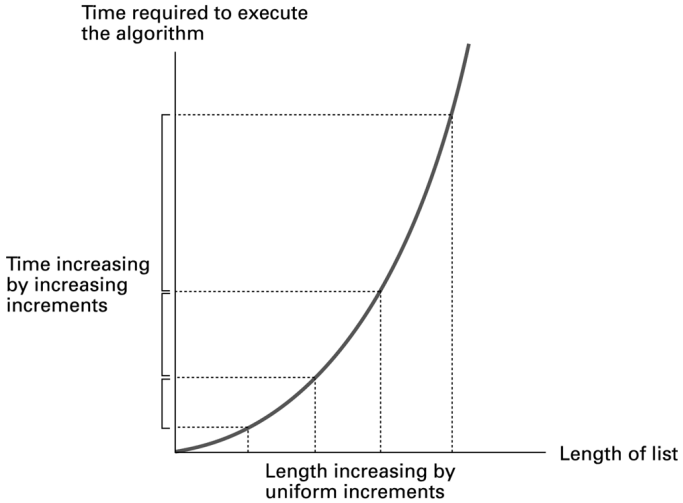
Algorithm Efficiency

- 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

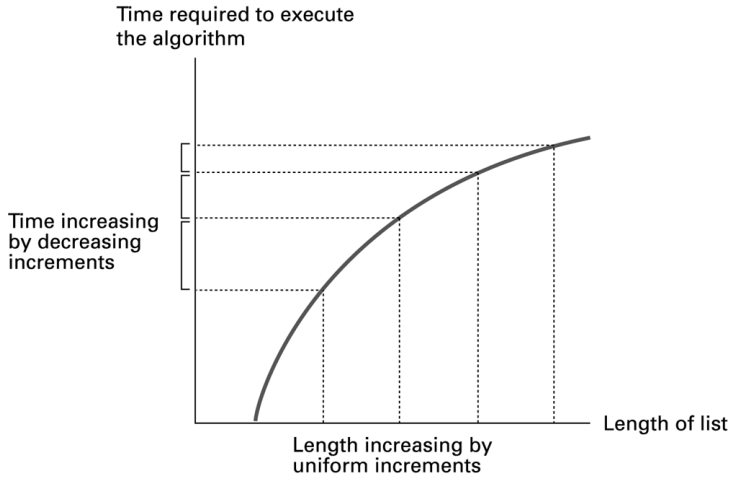
Applying the insertion sort in a worst-case situation

Initial list	Comparisons made for each pivot				Sorted list
	1st pivot	2nd pivot	3rd pivot	4th pivot	
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

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



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



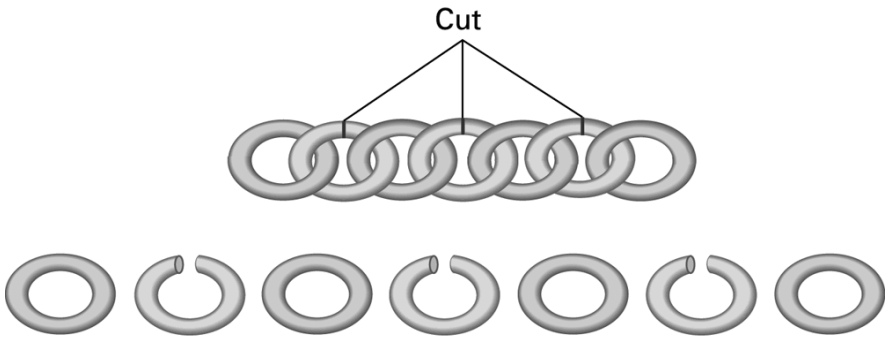
Software Verification

- 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?

Separating the chain using only three cuts



Solving the problem with only one cut