Chapter 5: Algorithms

Computer Science: An Overview
Tenth Edition

by
J. Glenn Brookshear

Presentation files modified by Farn Wang
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.
Folding a bird from a square piece of paper
## Origami primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shade one side</td>
<td>Distinguishes between different sides of paper</td>
</tr>
<tr>
<td>of paper</td>
<td>as in</td>
</tr>
<tr>
<td></td>
<td>Represented as a valley fold</td>
</tr>
<tr>
<td></td>
<td>so that</td>
</tr>
<tr>
<td></td>
<td>Represents a mountain fold</td>
</tr>
<tr>
<td></td>
<td>so that</td>
</tr>
<tr>
<td></td>
<td>Fold over</td>
</tr>
<tr>
<td></td>
<td>so that</td>
</tr>
<tr>
<td></td>
<td>Push in</td>
</tr>
<tr>
<td></td>
<td>so that</td>
</tr>
<tr>
<td></td>
<td>Turn paper over</td>
</tr>
<tr>
<td></td>
<td>as in</td>
</tr>
</tbody>
</table>

### Syntax
- Shade one side of paper
- Fold over
- Push in

### Semantics
- Distinguishes between different sides of paper
- Represents a valley fold
- Represents a mountain fold

### Diagrams
- Depicting the actions and their semantics.
Problem Solving Steps

*Designed the method in Hungary teaching the son of a baron.*

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.

George Pólya (1887-1985)
Stanford, ETH-Zurich
Getting a Foot in the Door

Basic approaches:

• 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)
A trial case study in solving problems

- 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?
Understanding the domain instances for a solution

- You can spend all the time fretting without a clue.
- Or you can start to play with the numbers.

The only pair that needs clue 3 to distinguish.

<table>
<thead>
<tr>
<th>a. Triples whose product is 36</th>
<th>b. Sums of triples from part (a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1,36)</td>
<td>1 + 1 + 36 = 38</td>
</tr>
<tr>
<td>(1,2,18)</td>
<td>1 + 2 + 18 = 21</td>
</tr>
<tr>
<td>(1,3,12)</td>
<td>1 + 3 + 12 = 16</td>
</tr>
<tr>
<td>(1,4,9)</td>
<td>1 + 4 + 9 = 14</td>
</tr>
<tr>
<td>(2,2,9)</td>
<td>1 + 6 + 6 = 13</td>
</tr>
<tr>
<td>(2,3,6)</td>
<td>2 + 2 + 9 = 13</td>
</tr>
<tr>
<td>(3,3,4)</td>
<td>2 + 3 + 6 = 11</td>
</tr>
<tr>
<td></td>
<td>3 + 3 + 4 = 10</td>
</tr>
</tbody>
</table>
Representation of algorithms
- Pseudocode Primitives

• Assignment

  \[ name \leftarrow expression \]

• Conditional selection

  \[ \text{if } condition \text{ then } action \]
Repeated execution

while condition do activity

Procedure

procedure name (generic names)
Representation of algorithms
- The procedure Greetings in pseudocode

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

Iterative Structures

• Pretest loop:
  
  while (condition) do
  (loop body)

• Posttest loop:
  
  repeat (loop body)
  until (condition)
The sequential search algorithm in pseudocode

Assumption: sorted list.

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
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
Representation of algorithms - flowchart

The while loop structure:

- **Test condition**
  - **Condition true**
  - **Activity**
  - **Condition false**
Representation of algorithms - flowchart

The repeat-until loop structure
How to design an algorithm?
- Intuition behind a sorting algorithm

Given a data item $b$ and a sorted list $\langle d_1, \ldots, d_k \rangle$, we can easily insert $b$ into $\langle d_1, \ldots, d_k \rangle$.

- sort $d_1$
  - Insert $d_1$ to the right position in $\langle \rangle$.
- sort $d_1, d_2$
  - Insert $d_2$ to the right position in $\langle d_1 \rangle$.
- sort $d_1, d_2, d_3$
  - Insert $d_3$ to the right position in $\langle d_1, d_2 \rangle$.
- ...
- sort $d_1, d_2, \ldots, d_n$
  - Insert $d_n$ to the right position in $\langle d_1, \ldots, d_{n-1} \rangle$. 
Test run of the algorithm in mind:

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

Initial list:

Fred
Alex
Diana
Byron
Carol

Sorted list:

Alex
Byron
Carol
Pseudo-code for insertion sort
- an example of loop algorithms

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
)

Why?
Too close to the real code.
Too many details.
Difficult to understand.
A better pseudo-code for insertion sort

Procedure sort(List) {
    Loop for $N = 2$ to $|List|$ {
        Insert List[N] to the right position in List[1..$N-1$]
        to make List[1..$N$] sorted;
    }
}

High-level mathematical notations.

High-level and intuitive operations in natural languages.
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

F(n) = F(n-1) + F(n-2)
F(0) = 0
F(1) = 1
F(n) {
    if (n <= 0) return(0);
    else if (n == 1) return 1;
    else return F(n-1)+F(n-2);
}

How many times F() is called ?
f(n) = 1+f(n-1)+f(n-2); f(0)=1, f(1)=1;
Recursion - implemented with iteration

\[ F(n) = F(n-1) + F(n-2) \]

\[ F(0) = 0 \]

\[ F(1) = 1 \]

\[ F(n) \{
    F(0) = 0; F(1) = 1; k = 2;
    while (k <= n) { F(k)=F(k-1)+F(k-2); k=k+1;}
    return (F(n));
\}

\[ \text{How many steps?} \quad f'(n) = 1 + n; \]
2013/04/17 stopped here.
Recursion
- with improved performance

\[ F(n) = F(n-1) + F(n-2) \; ; \; F(0) = 1; \; F(1) = 1 \]

F(n) {
    if (n < 0) exit(0);
    else if (n == 0 || n == 1) return n;
    else if (D[n] == -1)
        D[n] = F(n-1)+F(n-2);
    return(D[n]);
}

Main (n) {
    k=0; while (k<=n) {D[k]=-1; k=k+1;}
    return F(n);
}
Recursion for search?
- Trying out an example for the entry John

<table>
<thead>
<tr>
<th>Original list</th>
<th>First sublist</th>
<th>Second sublist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice, Bob, Carol, David, Elaine, Fred, George, Harry, Irene, John, Kelly, Larry, Mary, Nancy, Oliver</td>
<td>Irene, John, Kelly, Larry, Mary, Nancy, Oliver</td>
<td>Irene, John, Kelly</td>
</tr>
</tbody>
</table>
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
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
Binary search in recursion

bsearch(L[j,k], t) {
    if (j>k) return search failure.
    m = \lceil (j+k)/2 \rceil;
    if (t==L[m]) return m;
    else (t<L[m]) return bsearch(L[j,m-1], t);
    else return bsearch(L[m+1,k], t);
}
Binary search in recursion

bsearch(L[j,k], t) {
    while (j<=k) {
        m = ⌊(j+k)/2⌋;
        if (t==L[m]) return m;
        else (t<L[m]) k = m-1;
        else j=m+1;
    }
    return search failure.
}
Conceptualizing the recursion (I/III)

```
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

List

David
Evelyn
Fred
George

List

Alice
Bill
Carol
```
Conceptualizing the recursion (II/III)

**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
Conceptualizing the recursion (III/III)

```
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
```

We are here.

```
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
```
Algorithm Efficiency

• Measured as number of instructions executed

• Big O notation: Used to represent efficiency classes. Examples:
  – Insertion sort for n elements is in $O(n^2)$
  – binary search for n elements is in $O(\log n)$
  – $F(n)$ iterative: $O(n)$
  – $F(n)$ purely recursive: $O(f(n))$,
    • $f(n)=1+f(n-1)+f(n-2)$.

• Best, worst, and average case analysis
Applying the insertion sort in a worst-case situation

<table>
<thead>
<tr>
<th>Initial list</th>
<th>1st pivot</th>
<th>2nd pivot</th>
<th>3rd pivot</th>
<th>4th pivot</th>
<th>Sorted list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaine</td>
<td>Elaine</td>
<td>David</td>
<td>Carol</td>
<td>Barbara</td>
<td>Alfred</td>
</tr>
<tr>
<td>David</td>
<td>David</td>
<td>Elaine</td>
<td>David</td>
<td>Elaine</td>
<td>Barbara</td>
</tr>
<tr>
<td>Carol</td>
<td>Carol</td>
<td>Carol</td>
<td>Elaine</td>
<td>Barbara</td>
<td>Alfred</td>
</tr>
<tr>
<td>Barbara</td>
<td>Barbara</td>
<td>Barbara</td>
<td>Barbara</td>
<td>Alfred</td>
<td></td>
</tr>
<tr>
<td>Alfred</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Alfred</td>
</tr>
</tbody>
</table>

Comparisons made for each pivot:
- **1st pivot**: Elaine > David
- **2nd pivot**: Carol > Elaine
- **3rd pivot**: David > Elaine
- **4th pivot**: Barbara > David

Sorted list: Alfred, Barbara, Carol, David, Elaine
Graph of the worst-case analysis of the insertion sort algorithm
Graph of the worst-case analysis of the binary search algorithm
Algorithm Efficiency

Big O concepts for worst-case analysis

• Insertion sort for n elements is in $O(n^2)$
  – $0.00003n^2 + 0.00001 n + 0.00009$ sec
  – with efficient compiler & CPU

• Binary search for n elements is in $O(\log n)$
  – $123478999 \log n$ sec
  – with lousy compiler & CPU

• As n becomes bigger and bigger,
  $0.00003n^2 + 0.00001 n + 0.00009$

  $>> 123478999 \log n$ sec
The lower-bound for worst cases?
- An example

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
Lower-bound of worst-case efficiency

How do we know if an algorithm is good enough for a task?

• What is the assumptions of basic operations?
• Have we tied up ourselves?

Lemma: One cut is the minimum.

Proof:
• At least one chain is given in the first morning.
• So one cut is the least.
Lower-bound of worst-case efficiency

other techniques

• reduction:
  – translate any instance of a problem A with known lower-bound to the given problem B. Then we can show B’s lower-bound is no lower than A’s.

• information theory:
  – Is log(n) the best lower-bound for the worst case of searching algorithms?
  – Is n log(n) the best for sorting algorithms?
Software Verification

• Proof of correctness
  – Assertions
    • Preconditions
    • Loop invariants

• Testing
  – send testcases to SW
  – check the SW behavior or output

• Verification
  – analysis of models of SW
The assertions associated with a typical while structure

- **Precondition** (A)
- **Initialize**
- **Test** (B)
  - **False**
  - **Modify**
  - **Body**
  - **True**
  - **Loop invariant** (C)
    - **Loop invariant and termination condition**
Assertions
- an example on recursive code.

F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1

F(n) {
  F(0) = 0; F(1) = 1; k = 2;
  while (k <= n) {
    F(k) = F(k-1) + F(k-2); k = k + 1;
  }
  return (F(n));
}

precondition: true
precondition: k=2
Loop invariants:
k <= n,
\forall i < k, F(i) = F(i-1) + F(i-2)
can be proved with mathematical induction.
precondition: F(n) = F(n-1) + F(n-2)
Various algorithms (I)

• Numerical
  – factorization, prime number testing, …

• Constraint solving
  – equations, inequalities, differential systems

• Sequence:
  – Sorting, searching, merging, pattern matching, data mining
Various algorithms (II)

• Graph algorithms
  – searching, traversal,
  – critical paths, flow analysis, ....
  – structure identification,
  – spanning trees
  – .......
Various algorithms (II)

- Graph algorithms
  - searching, traversal,
  - critical paths, flow analysis, ....
  - structure identification,
  - spanning trees
Various algorithms (III)

• Logics
  – solutions, Boolean operations, greatest fixpoint, least fixpoint

• Distributed algorithms
  – leader elections,
  – snapshots,
  – deadlock detection,
  – consensus
  – ....
Algorithms for nonterminating systems

- Operating systems
  - deadlock
  - mutual exclusion
  - security
- Medical systems
When there is no algorithm

• genetic algorithm
• swamp intelligence
• ant intelligence
• machine learning
• heuristics
• solution as proofs
• …..
Complexities

Average case? Worst case?

Algorithms:
- time complexities
- space complexities

Problems:
- How efficient can we make the program run?
- Is there an algorithm?
Data structures

- linear structures
  - arrays, sequence, stacks, queues
- tree-structures
  - binary trees, 2-3 trees, splay trees
- general graphs
  - DAG (directed acyclic graphs)
  - directed, undirected, multigraphs