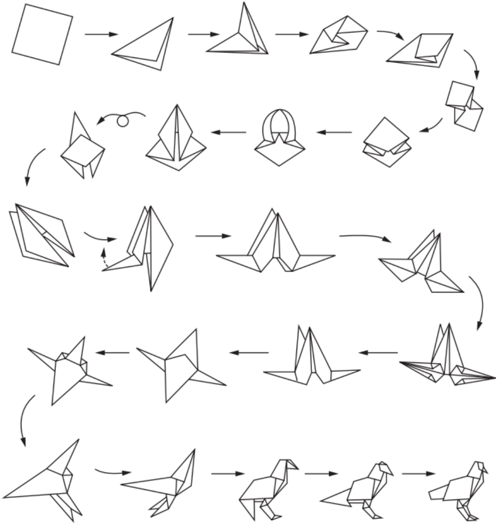
Introduction to Computer Science
Lecture 5: ALGORITHMS

Department of Electrical Engineering
National Taiwan University

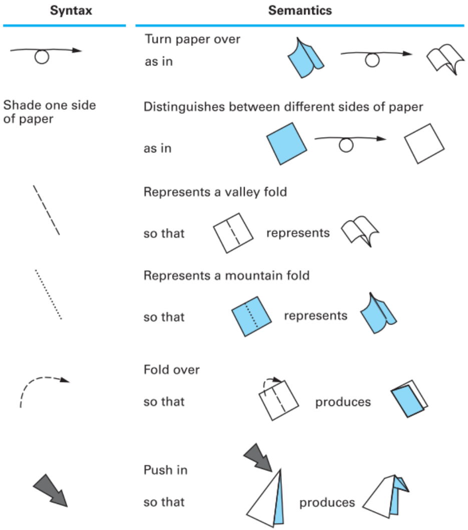
Definitions

- Algorithm: **ordered** set of **unambiguous**, **executable** steps that defines a **terminating** process.
- Program: formal representation of an algorithm.
- Process: activity of executing a program.
- Primitives, programming languages.
- Abstraction

Folding a Bird

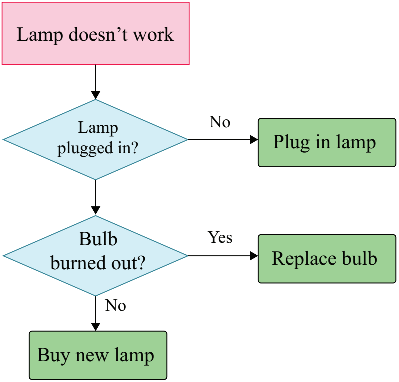


Origami Primitives



Algorithm Representation

- Flowchart
    - Popular in 50s and 60s
    - Overwhelming for complex algorithms
- Pseudocode
    - A loosen version of formal programming languages



Pseudocode Primitives

- Assignment

  name $\leftarrow$ expression

- Conditional selection

  if (condition) **then** (activity)

- Repeated execution

  while (condition) **do** (activity)

- Procedure

  procedure name

procedure GREETINGS
$Count \leftarrow 3$
while $(Count > 0)$ **do**
(print the message "Hello" and
$Count \leftarrow Count - 1)$

Pólya's Problem Solving Steps

How to Solve It by George Pólya, 1945.

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.

Problem Solving

- Top-down
  - Stepwise refinement
  - Problem decomposition
- Bottom-up
- Both methods often complement each other
- Usually,
  - planning $\rightarrow$ top-down
  - implementation $\rightarrow$ bottom-up

Iterations

- Loop control

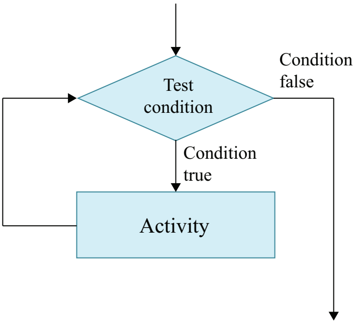
Initialize: Establish an initial state that will be modified toward the termination condition

Test: Compare the current state to the termination condition and terminate the repetition if equal

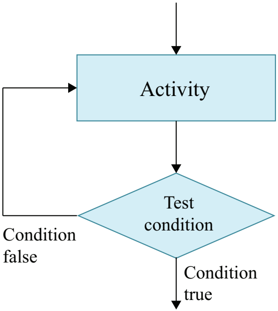
Modify: Change the state in such a way that it moves toward the termination condition

Loops

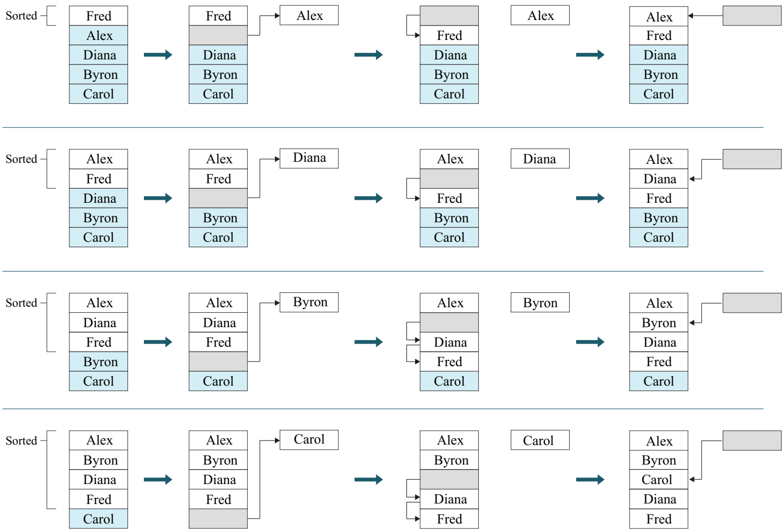
- Pre-test (while. . .)



- Post-test (do. . . while, repeat. . . until)



Insertion Sort



Pseudocode for Insertion Sort

procedure InsertionSort (*List*)

```
N ← 2
while (the value of N does not exceed the length of List) do
    (Select the N-th entry in List as the pivot entry
    Move the pivot 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
    )
```

Binary Search

Original list	First sublist	Second sublist
Alice		
Bob		
Carol		
David		
Elaine		
Fred		
George		
Harry		
Irene	Irene	Irene
John	John	John
Kelly	Kelly	Kelly
Larry	Larry	
Mary	Mary	
Nancy	Nancy	
Oliver	Oliver	

Pseudocode for Binary Search

procedure BinarySearch (*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: TagetValue = TestEntry
                        (Report that the search succeeded.)
                case 2: TagetValue < TestEntry
                        (Search the portion of List preceding TestEntry f
                        TargetValue, and report the result of that search.)
                case 3: TagetValue > TestEntry
                        (Search the portion of List succeeding TestEntry f
                        TargetValue, and report the result of that search.)
) end if
```

Recursive Problem Solving (contd.)

- Factorial

```
int factorial (int x) {
    if (x==0) return 1;
        return x * factorial(x-1);
}
```

- Do not abuse
  - Calling functions takes a long time
  - Avoid **tail recursions**

```
int factorial (int x) {
    int product = 1;
    for (int i=1; i<=x; ++i)
        product *= i;
    return product;
}
```

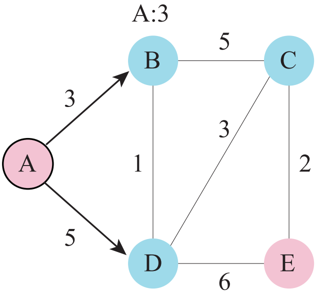
```
int Fibonacci (int x) {
    if (x==0) return 0;
    if (x==1) return 1;
    return Fibonacci(x-2) + Fibonacci(x-1);
}
```

Divide and Conquer vs. Dynamic Programming

- Divide and conquer (D&C):
  - Subproblems
  - Top-down
  - Binary search, merge sort, ...
- Dynamic programming (DP):
  - Subprograms share subsubproblems
  - Bottom-up
  - Shortest path, matrix-chain multiplication, ...

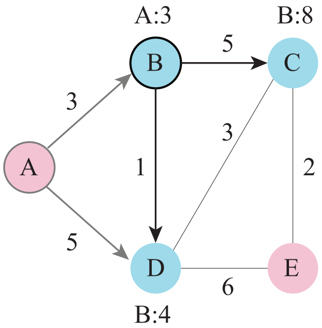
Shortest Path

$$Shortest_{AE} = \min_{i \in \{A,B,C,D,E\}}(Shortest_{Ai} + Shortest_{iE})$$



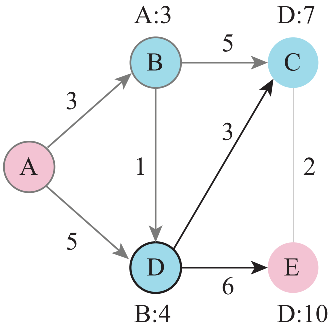
Shortest Path (contd.)

$$Shortest_{AE} = \min_{i \in \{A,B,C,D,E\}}(Shortest_{Ai} + Shortest_{iE})$$



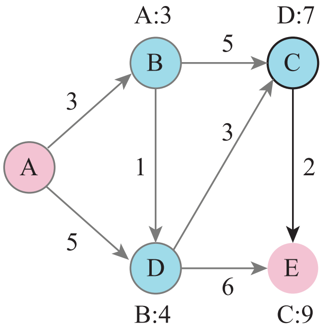
Shortest Path (contd.)

$$Shortest_{AE} = \min_{i \in \{A,B,C,D,E\}}(Shortest_{Ai} + Shortest_{iE})$$



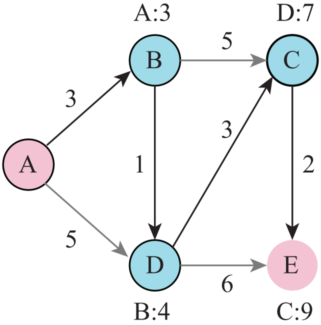
Shortest Path (contd.)

$$Shortest_{AE} = \min_{i \in \{A,B,C,D,E\}}(Shortest_{Ai} + Shortest_{iE})$$



Shortest Path (contd.)

$$Shortest_{AE} = \min_{i \in \{A,B,C,D,E\}} (Shortest_{Ai} + Shortest_{iE})$$



Matrix-Chain Multiplication

- Matrices: $A : p \times q$; $B : q \times r$
  - Then $C = A \cdot B$ is a $p \times r$ matrix.

$$C_{i,j} = \sum_{k=1}^{q} A_{i,k} \cdot B_{k,j}$$

  - Time complexity: pqr scalar multiplications
- The matrix-chain multiplication problem
  - Given a chain $< A_1, A_2, ..., A_n >$ of n matrices, which A_i is of dimension $p_{i-1} \times p_i$, parenthesize properly to minimize # of scalar multiplications.

Matrix-Chain Multiplication

- $(p \times q) \cdot (q \times r) \;\rightarrow\; (p \times r)$
  - (pqr) scalar multiplications

- $A_1,\; A_2,\; A_3 :\; (10 \times 100),\; (100 \times 5),\; (5 \times 50)$
- $(A_1A_2)A_3 \;\rightarrow (10 \times 100 \times 5) + (10 \times 5 \times 50) = 7500$
- $A_1(A_2A_3) \;\rightarrow (100 \times 5 \times 50) + (10 \times 100 \times 50) = 75000$

- 4 matrices:
  - $((A_1A_2)A_3)A_4$
  - $A_1(A_2A_3)A_4$
  - $(A_1A_2)(A_3A_4)$
  - $A_1(A_2(A_3A_4))$

The Minimal # of Multiplications

- $m[i,j]$: minimal # of multiplications to compute matrix $A_{i,j} = A_i A_{i+1} \ldots A_j$, where $1 \leq i \leq j \leq n$.

$$m[i,j] = \begin{cases} 0 & , i = j \\ \min_k \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & , i \neq j \end{cases}$$

Bottom-Up DP

- $A_1 : 7 \times 3$
- $A_2 : 3 \times 1$
- $A_3 : 1 \times 2$
- $A_4 : 2 \times 4$

- $p_0 = 7$
- $p_1 = 3$
- $p_2 = 1$
- $p_3 = 2$
- $p_4 = 4$

- $m[i,i] = 0$

- $m[1,2] = 0 + 0 + 7 \times 3 \times 1 = 21$
- $m[2,3] = 6$
- $m[3,4] = 8$

- $m[1,3] = 35$
  $\min\{21 + 0 + 7 \times 1 \times 2, 0 + 6 + 7 \times 3 \times 2\}$

- $m[2,4] = 20$
  $\min\{6 + 0 + 3 \times 2 \times 4, 0 + 8 + 3 \times 1 \times 4\}$

Bottom-Up DP (contd.)

- $A_1 : 7 \times 3$
- $A_2 : 3 \times 1$
- $A_3 : 1 \times 2$
- $A_4 : 2 \times 4$

- $p_0 = 7$
- $p_1 = 3$
- $p_2 = 1$
- $p_3 = 2$
- $p_4 = 4$

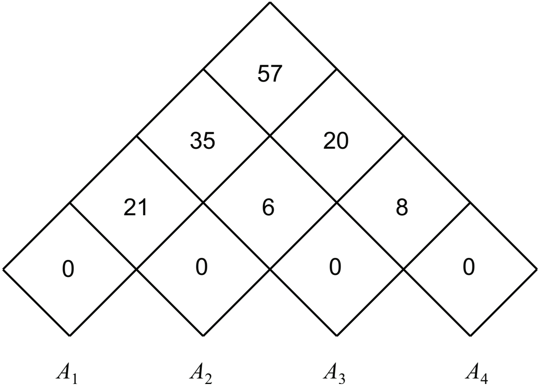
- $$\begin{aligned} m[1,4] = \min\{ & \\ & m[1,1] + m[2,4] + 7 \times 3 \times 4, \\ & m[1,2] + m[3,4] + 7 \times 1 \times 4, \\ & m[1,3] + m[4,4] + 7 \times 2 \times 4\} \\ = 57 & \end{aligned}$$

- Ans: $(A_1 A_2)(A_3 A_4)$

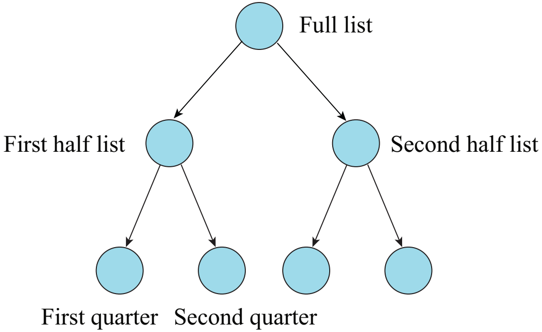
Table Filling

- $A_1 : 7 \times 3$
- $A_2 : 3 \times 1$
- $A_3 : 1 \times 2$
- $A_4 : 2 \times 4$

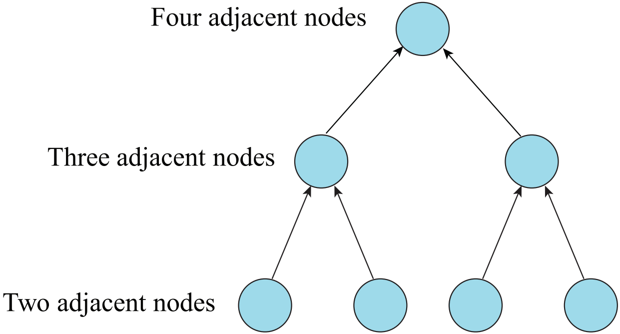
- $p_0 = 7$
- $p_1 = 3$
- $p_2 = 1$
- $p_3 = 2$
- $p_4 = 4$



Top-Down Manner (Binary Search)



Bottom-up Manner (Shortest Path)



Algorithm Efficiency

- Number of instructions executed
- Execution time

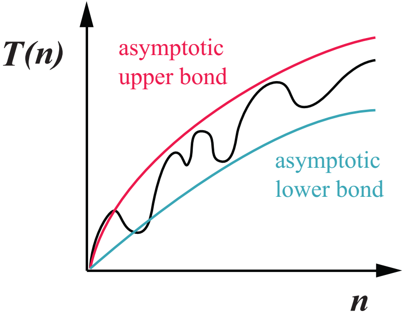
- What about on different machines?

- O, Ω, Θ notations
- Pronunciations: big-o, big-omega, big-theta

Asymptotic Analysis

- Exact analysis is often difficult and tedious.
- Asymptotic analysis emphasizes the behavior of the algorithm when n tends to infinity.

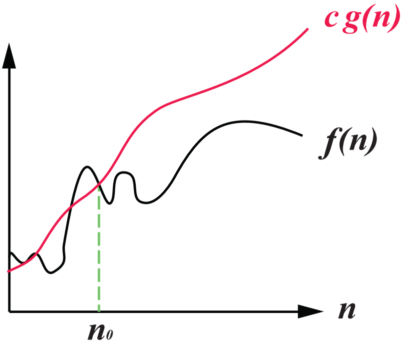
- Asymptotic
  - Upper bound
  - Lower bound
  - Tight bound



Big-O

$$O(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \text{ s.t. } \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n)\}$$

- Asymptotic upper bound
- If $f(n)$ is a member of the set of $O(g(n))$, we write $f(n) = O(g(n))$.
- Examples
  $100n = O(n^2)$
  $n^{100} = O(2^n)$
  $2n + 100 = O(n)$

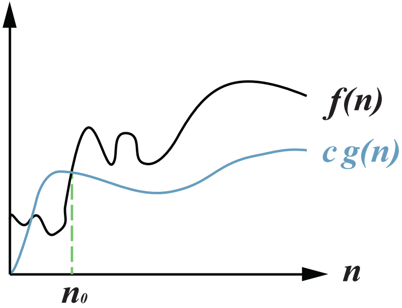


$f(n) = O(g(n))$

Big-Omega

$$\Omega(g(n)) = \{f(n) | \; \exists c > 0, n_0 > 0 \text{ s.t.} \forall n \geq n_0, \; 0 \leq cg(n) \; \leq f(n)\}$$

- Asymptotic lower bound
- If $f(n)$ is a member of the set of $\Omega(g(n))$, we write $f(n) = \Omega(g(n))$.
- Examples
  $0.01n^2 = \Omega(n)$
  $2^n = \Omega(n^{100})$
  $2n + 100 = \Omega(n)$

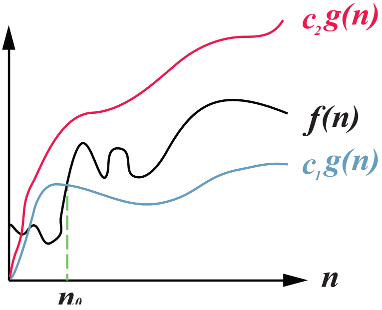


$f(n) = \Omega(g(n))$

Big-Theta

$$\Theta(g(n)) = \{f(n) | \exists c_1, c_2, n_0 > 0 \text{ s.t. } \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

- Asymptotic tight bound
- If $f(n)$ is a member of the set of $\Theta(g(n))$, we write $f(n) = \Theta(g(n))$.
- Examples
  $0.01n^2 = \Theta(n^2)$
  $2n + 100 = \Theta(n)$
  $n + \log n = \Theta(n)$



$f(n) = \Theta(g(n))$

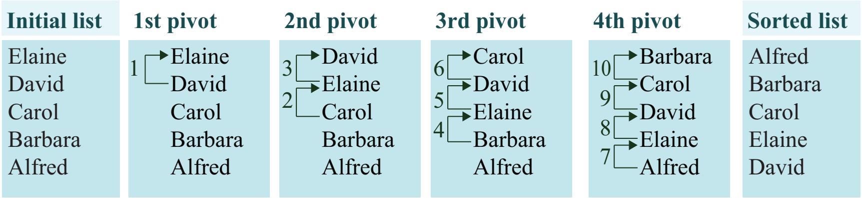
Theorem

$f(n) = \Theta(g(n))$ **iff** $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Efficiency Analysis

- Best, worst, average cases

Comparisons made for each pivot

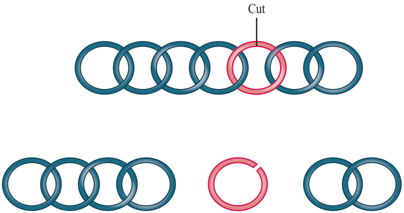


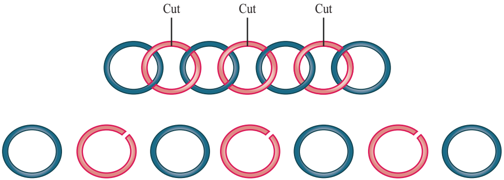
Worst case for insertion sort

Worst: $(n^2 - n)/2$, best: $(n - 1)$, average: $\Theta(n^2)$

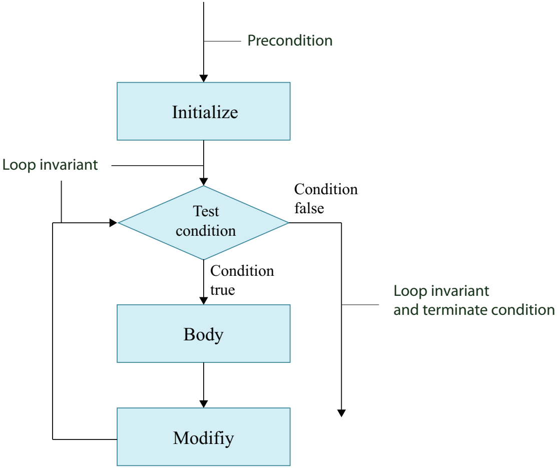
Software Verification

Traveler's gold chain





Assertion for "While"



- Precondition
- Loop invariant
- Termination condition

Correct or Not?

$Count \leftarrow 0$
$Remainder \leftarrow Dividend$
repeat $(Remainder \leftarrow Remainder - Divisor$
$\quad Count \leftarrow Count + 1)$
until $(Remainder < Divisor)$
$Quotient \leftarrow Count$

Problematic

$$Remainder > 0?$$

- **Preconditions:**
  - $Dividend > 0$
  - $Divisor > 0$
  - $Count = 0$
  - $Remainder = Dividend$
  - $Remainder > 0$
- **Loop invariants:**
  - $Dividend > 0$
  - $Divisor > 0$
  - $Dividend = Count \cdot Divisor + Remainder$
- **Termination condition:**
  - $Remainder < Divisor$

Verification of Insertion Sort

- Loop invariant of the outer loop
  - Each time the test for termination is performed, the names preceding the N-th entry form a sorted list
- Termination condition
  - The value of N is greater than the length of the list.
- If the loop terminates, the list is sorted

Final Words for Software Verification

- In general, not easy.
- Need a formal PL with better properties.