Algorithms and Complexity

Selected Topics in Computer Intelligence - 2015

Bioinformatics Programming

Computer Engineering, Chiang Mai University

What is an Algorithm?

- A sequence of instructions that we perform in order to solve a well-formulated problem
- A method of translating the inputs into outputs
- We can use pseudocode to describe algorithms
  - Operations can be grouped into subroutines that solve problems
  - Covers concepts of variables, arrays, and arguments

Examples

- Euclidean distance

$$
\begin{array}{l}
\textsc{Dist}(x1, y1, x2, y2) \\
1\ \ dx \leftarrow (x2 - x1)^2 \\
2\ \ dy \leftarrow (y2 - y1)^2 \\
3\ \ \textbf{return}\ \sqrt{(dx + dy)}
\end{array}
$$

  - Result: the Euclidean distance between points with coordinates *(x1,y1)* and *(x2,y2)*

- Maximum

$$
\begin{array}{l}
\textsc{Max}(a, b) \\
1\ \ \textbf{if}\ a < b \\
2\ \ \qquad \textbf{return}\ b \\
3\ \ \textbf{else} \\
4\ \ \qquad \textbf{return}\ a
\end{array}
$$

  - Result: the maximum of the numbers a and b

Examples

- Sum Integers

```
SUMINTEGERS(n)
sum ← 0
for i ← 1 to n
    sum ← sum + i
return sum
```

  - Result: sum of integers from *1* to n

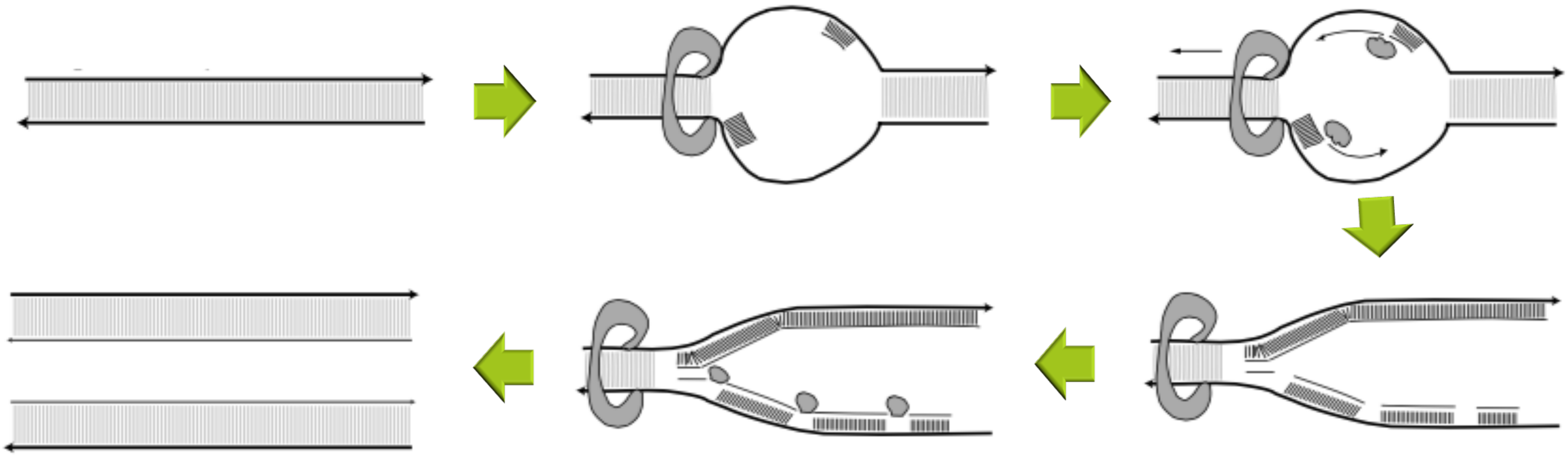
- Fibonacci

```
FIBONACCI(n)
F_1 ← 1
F_2 ← 1
for i ← 3 to n
    F_i ← F_{i-1} + F_{i-2}
return F_n
```

  - Result: the n^{th} Fibonacci number

Biological vs. Computer Algorithms

- DNA Replication Process
  - Uses algorithm-like procedures to solve this problem
  - Before a cell can divide, it must first make a copy of its genetic material
  - Requires an elaborate cooperation between different types of molecules and enzymes



Biological vs. Computer Algorithms

- In Computer Domain
  - "String Duplication Problem"
    - Given a string of letters, return a copy

```
StringCopy(s, n)
  for i ← 1 to n
      t_i ← s_i
  return t
```

  - A trivial algorithm but may require the number of operations
  - We have invented programming language to solve the problem quite easily
- Biologist have not yet invented a similar "language" to describe biological algorithms working in the cell…

Computer Algorithm

- To solve computational problem we need to …
  - Identify precisely what the problem is then we can devise an algorithms that solves it
  - Two important questions:
    - Does it work correctly?
    - How long will it take?
  - The Change Problem
    - How to make change in the least annoying way.
      - Minimize the number of coins returned
    - Is it specific to a certain country? (generalization?)

Computer Algorithm

- Returning coins in US currency

United States Change Problem:
Convert some amount of money into the fewest number of coins.

Input: An amount of money, M, in cents.

Output: The smallest number of quarters q, dimes d, nickels n, and pennies p whose values add to M (i.e., $25q + 10d + 5n + p = M$ and $q + d + n + p$ is as small as possible).

USCHANGE(M)

$r \leftarrow M$
$q \leftarrow r/25$
$r \leftarrow r - 25 \cdot q$
$d \leftarrow r/10$
$r \leftarrow r - 10 \cdot d$
$n \leftarrow r/5$
$r \leftarrow r - 5 \cdot n$
$p \leftarrow r$
return (q, d, n, p)

Problem?
- unlimited supply of each coin
- lacks generality

Computer Algorithm

- Returning coins in ANY currency

Change Problem:
Convert some amount of money M into given denominations, using the smallest possible number of coins.

Input: An amount of money M, and an array of d denominations $\mathbf{c} = (c_1, c_2, \ldots, c_d)$, in decreasing order of value ($c_1 > c_2 > \cdots > c_d$).

Output: A list of d integers $i_1, i_2, \ldots, i_d$ such that $c_1 i_1 + c_2 i_2 + \cdots + c_d i_d = M$, and $i_1 + i_2 + \cdots + i_d$ is as small as possible.

BETTERCHANGE($M, \mathbf{c}, d$)
1 $r \leftarrow M$
2 **for** $k \leftarrow 1$ **to** d
3 $\quad i_k \leftarrow r / c_k$
4 $\quad r \leftarrow r - c_k \cdot i_k$
5 **return** $(i_1, i_2, \ldots, i_d)$

- Correct algorithm
  - Can it translate EVERY input instance into the correct output
- Incorrect algorithm
  - At least one input instance does not produce the correct output

Problem?
- changing 40 cents
- $\mathbf{c}$ = (25, 20, 10, 5, 1)

Computer Algorithm

- Returning coins in ANY currency (cont.)
  - Brute force algorithm

BruteForceChange($M, \mathbf{c}, d$)

$smallestNumberOfCoins \leftarrow \infty$
for each $(i_1, \ldots, i_d)$ **from** $(0, \ldots, 0)$ **to** $(M/c_1, \ldots, M/c_d)$
    $valueOfCoins \leftarrow \sum_{k=1}^{d} i_k c_k$
    if $valueOfCoins = M$
        $numberOfCoins \leftarrow \sum_{k=1}^{d} i_k$
        if $numberOfCoins < smallestNumberOfCoins$
            $smallestNumberOfCoins \leftarrow numberOfCoins$
            bestChange $\leftarrow (i_1, i_2, \ldots, i_d)$
return (**bestChange**)

Type of Algorithms

- Recursive
  - An algorithm is recursive if it calls itself
  - Calculates Factorial and Fibonacci numbers
  - Towers of Hanoi puzzle – introduced in 1883

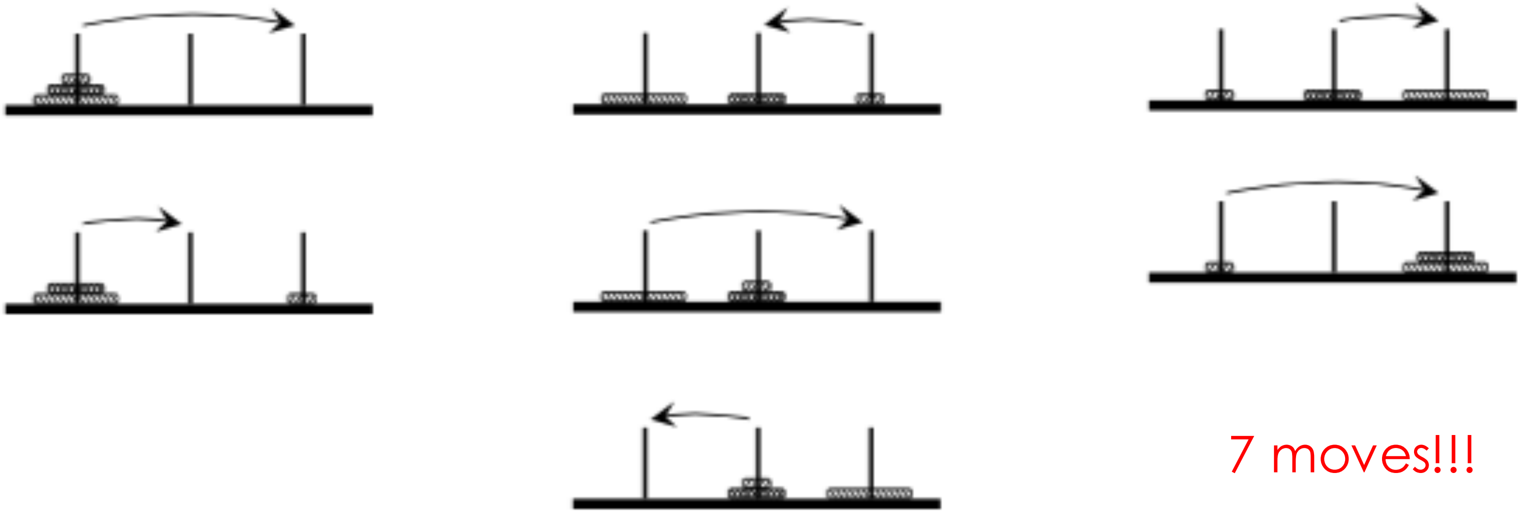
Type of Algorithms

- Recursion - Towers of Hanoi (cont.)

Towers of Hanoi Problem:
Output a list of moves that solves the Towers of Hanoi.

Input: An integer n.

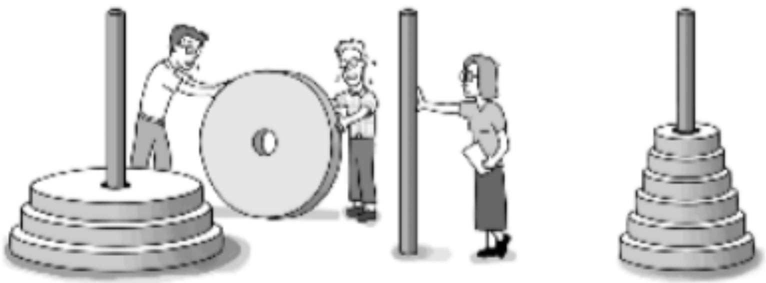
Output: A sequence of moves that will solve the n-disk Towers of Hanoi puzzle.



7 moves!!!

Type of Algorithms

- Recursion - Towers of Hanoi (cont.)
  - Cannot complete the game without moving the largest disk
  - How many steps for 4 disks (*n=4*)
    - move top 3 (*n-1*) disks to an empty peg – 7 moves
    - Move the largest disk, n^{th} disk, to the destination peg
    - Move again the 3 (*n-1*) disks from the temp peg to the destination peg – 7 moves

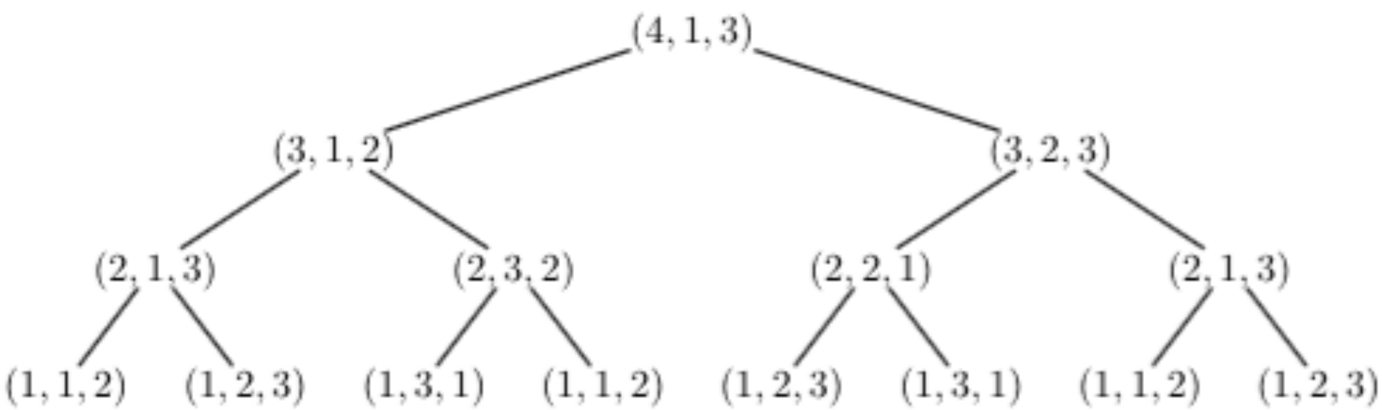


Type of Algorithms

Recursion - Towers of Hanoi (cont.)

HanoiTowers(n, $fromPeg$, $toPeg$)
if $n = 1$
  output "Move disk from peg $fromPeg$ to peg $toPeg$"
  return
$unusedPeg \leftarrow 6 - fromPeg - toPeg$
HanoiTowers($n-1$, $fromPeg$, $unusedPeg$)
output "Move disk from peg $fromPeg$ to peg $toPeg$"
HanoiTowers($n-1$, $unusedPeg$, $toPeg$)
return

$fromPeg$	$toPeg$	$unusedPeg$
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1



Type of Algorithms

- Iterative vs. Recursive Algorithm
  - Most recursive algorithm can be rewritten in iterative loops

RECURSIVEFIBONACCI(n)

if $n = 1$ **or** $n = 2$
    return 1
else
    $a \leftarrow$ RECURSIVEFIBONACCI($n - 1$)
    $b \leftarrow$ RECURSIVEFIBONACCI($n - 2$)
    return $a + b$

FIBONACCI(n)

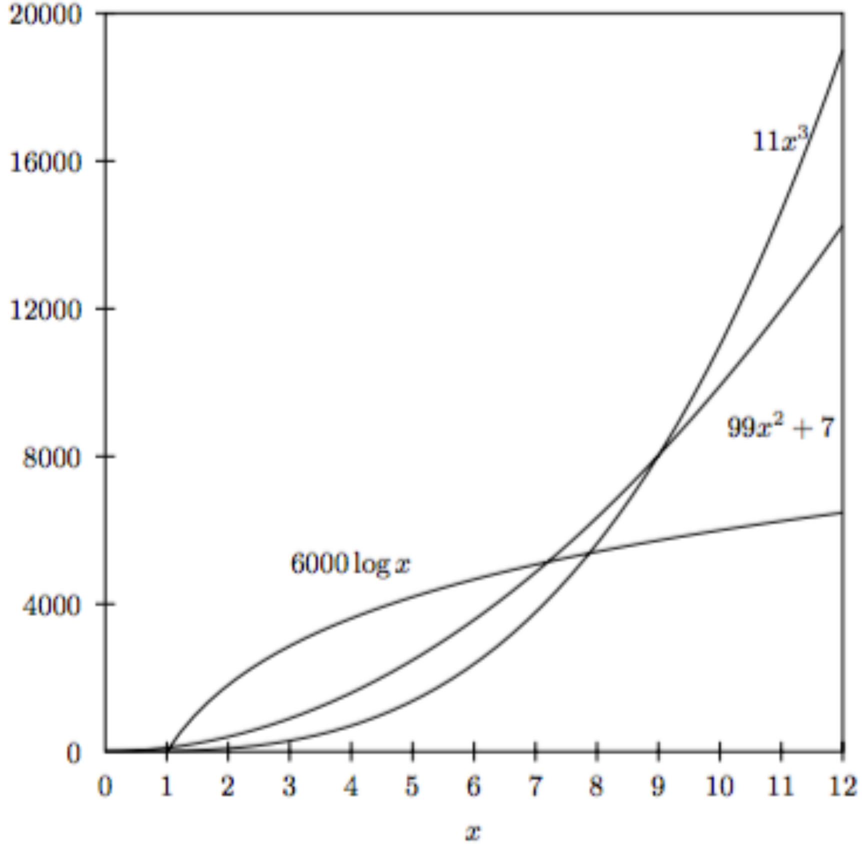
$F_1 \leftarrow 1$
$F_2 \leftarrow 1$
for $i \leftarrow 3$ **to** n
    $F_i \leftarrow F_{i-1} + F_{i-2}$
return F_n

Exponential-time

linear-time

Complexity

- Fast vs. Slow algorithm
  - It's about total number of operations that an algorithm perform to describe the running time
  - Algorithm complexity
    - Logarithmic
    - Quadratic
    - Cubic
    - What else?
  - The behavior of algorithms on small inputs does NOT matter



Complexity

- Big-O Notation
  - Describe the running time of an algorithm
  - Considering the term that grows the fastest with respect to ***n***

Definition 2.1 *A function $f(x)$ is $O(g(x))$ if there are positive real constants c and x_0 such that $f(x) \leq cg(x)$ for all values of $x \geq x_0$.* Upper bound

Definition 2.2 *A function $f(x)$ is $\Omega(g(x))$ if there are positive real constants c and x_0 such that $f(x) \geq cg(x)$ for all values of $x \geq x_0$.* Lower bound

Definition 2.3 *A function $f(x)$ is $\Theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.* Tight bound

Complexity

- Big-O Notation (cont.)
  - Worst case efficiency – the largest amount of time taking to solve the worst possible of input of a given ***size (n)***
  - E.g. measure sorting algorithm (min to max)
    - Given that the input of size ***n*** with max to min order

Algorithm Design Techniques

- Exhaustive Search (aka. Brute force)
- Branch-and-Bound (aka. Pruning)
  - Brute force with a bit of common sense
- Greedy – iterative procedure
  - Choosing the most attractive choice in each iteration
- Dynamic Programming
  - Organize computations to avoid computing known values
- Divide-and-Conquer
  - Splitting problem into smaller subproblems and combining the results
- Machine Learning
  - Based their strategies on the computation analysis of previous collected data