#### Algorithms: A Brief Introduction

Notes

Notes

Notes

Slides by Christopher M. Bourke Instructor: Berthe Y. Choueiry

Fall 2007

Computer Science & Engineering 235 Introduction to Discrete Mathematics Section 3.1 of Rosen cse235@cse.unl.edu

#### Algorithms Brief Introduction

Real World Objects Relations Actions **Computing World** Data Structures, ADTs, Classes Relations and functions Operations

Problems are specified by (1) a formulation and (2) a query.

Formulation is a set of objects and a set of relations between them

 $\ensuremath{\textbf{Query}}$  is the information one is trying to extract from the formulation, the question to answer.

 $\boldsymbol{\mathsf{Algorithms}}^1$  are methods or procedures that solve instances of a problem

<sup>1</sup>"Algorithm" is a distortion of *al-Khwarizmi*, a Persian mathematician

# Algorithms

Formal Definition

Definition

An  ${\bf algorithm}$  is a sequences of unambiguous instructions for solving a problem. Algorithms must be

- ► Finite must eventually *terminate*.
- ▶ Complete *always* gives a solution when there is one.
- ► Correct (sound) *always* gives a "correct" solution.

For an algorithm to be an acceptable solution to a problem, it must also be *effective*. That is, it must give a solution in a "reasonable" amount of time.

There can be many algorithms for the same problem.

# Algorithms

General Techniques

There are many broad categories of Algorithms: Randomized algorithms, Monte-Carlo algorithms, Approximation algorithms, Parallel algorithms, et al.

Usually, algorithms are studied corresponding to relevant data structures. Some general  $\mathit{styles}$  of algorithms include

- 1. Brute Force (enumerative techniques, exhaustive search)
- 2. Divide & Conquer
- 3. Transform & Conquer (reformulation)
- 4. Greedy Techniques

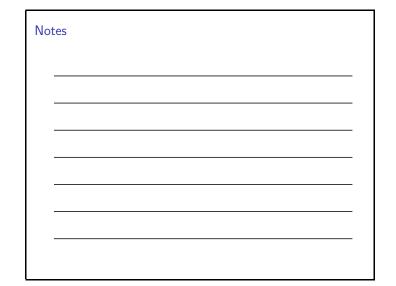
# Pseudo-code

Algorithms are usually presented using some form of *pseudo-code*. Good pseudo-code is a balance between clarity and detail.

*Bad* pseudo-code gives too many details or is too implementation specific (i.e. actual C++ or Java code or giving every step of a sub-process).

 ${\it Good}$  pseudo-code abstracts the algorithm, makes good use of mathematical notation and is easy to read.

IN	TERSECTION
	INPUT : Two sets of integers, A and B
	OUTPUT : A set of integers $C$ such that $C = A \cap B$
1	$C \leftarrow \emptyset$
2 3	IF $ A  >  B $ THEN swap $(A, B)$
4	END
5 6 7	FOR every $x \in A$ DO IF $x \in B$ THEN $C \leftarrow C \cup \{x\}$
8	END
9	END
10	output C



Notes

# Designing An Algorithm

A general approach to designing algorithms is as follows.

- 1. Understand the problem, assess its difficulty
- 2. Choose an approach (e.g., exact/approximate, deterministic/probabilistic)
- 3. (Choose appropriate data structures)
- 4. Choose a strategy
- 5. Prove termination
- 6. Prove correctness
- 7. Prove completeness
- 8. Evaluate complexity
- 9. Implement and test it.
- 10. Compare to other known approaches and algorithms.

# MAX

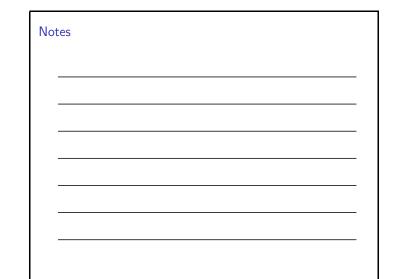
When designing an algorithm, we usually give a formal statement about the problem we wish to solve.  $% \left( {{{\rm{s}}_{\rm{s}}}} \right)$ 

Problem

Given a set  $A = \{a_1, a_2, \dots, a_n\}$  integers. Output the index i of the maximum integer  $a_i$ .

A straightforward idea is to simply store an initial maximum, say  $a_1$  then compare it to every other integer, and update the stored maximum if a new maximum is ever found.

# MAX Pseudo-code MAX INPUT : A set $A = \{a_1, a_2, \dots, a_n\}$ of integers. OUTPUT : An index i such that $a_i = \max\{a_1, a_2, \dots, a_n\}$ 1 index $\leftarrow 1$ 2 FOR $i = 2, \dots, n$ DO 3 IF $a_i > a_{index}$ THEN 4 index $\leftarrow i$ 5 END 6 END 7 output i



Notes

# MAX

# Analysis

This is a simple enough algorithm that you should be able to:

- Prove it correct
- Verify that it has the properties of an algorithm.
- ► Have some intuition as to its *cost*.

That is, how many "steps" would it take for this algorithm to complete its run? What constitutes a step? How do we measure the complexity of the step?

These questions will be answered in the next few lectures, for now let us just take a look at a couple more examples.

# Other examples

Check Bubble Sort and Insertion Sort in your textbooks, which you have seen ad nauseum, in CSE155, CSE156, and will see again in CSE310.

I will be glad to discuss them with any of you if you have not seen them yet.

# Greedy algorithm

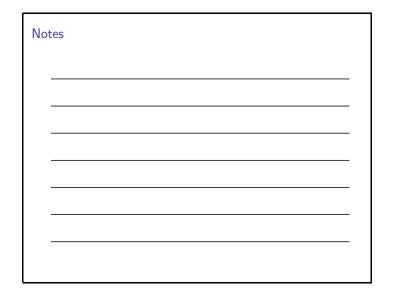
Optimization

In many problems, we wish to not only find *a* solution, but to find the best or *optimal* solution.

A simple technique that works for *some* optimization problems is called the *greedy technique*.

As the name suggests, we solve a problem by being greedy—that is, choosing the best, most immediate solution (i.e. a *local* solution).

However, for some problems, this technique is not guaranteed to produce the best *globally optimal* solution.



Notes

# Example

. Change-Making Problem

For anyone who's had to work a service job, this is a familiar problem: we want to give change to a customer, but we want to minimize the number of total coins we give them.

Problem

Given An integer n and a set of coin denominations  $(c_1, c_2, \ldots, c_r)$ with  $c_1 > c_2 > \cdots > c_r$ 

 $\textbf{Output} \ \mathsf{A} \ \mathsf{set} \ \mathsf{of} \ \mathsf{coins} \ d_1, d_2, \cdots, d_k \ \mathsf{such} \ \mathsf{that} \ \sum_{i=1}^k d_i = n \ \mathsf{and} \ k$ is minimized.

#### Example Change-Making Algorithm

# Change

_		
	Input	: An integer $n$ and a set of coin denominations $(c_1,c_2,\ldots,c_r)$ with $c_1>c_2>\cdots>c_r.$
	Output	: A set of coins $d_1, d_2, \cdots, d_k$ such that $\sum_{i=1}^k d_i = n$ and $k$ is minimized.
1	$C \gets \emptyset$	
2	For $i = 1,$	, <i>r</i> do
3	WHILE	$n \ge c_i$ do
4	(	$C \leftarrow C \cup \{c_i\}$
5	1	$n \leftarrow n - c_i$
6	END	
7	END	
8	output $C$	
-		

# Change-Making Algorithm Analysis

Will this algorithm *always* produce an optimal answer?

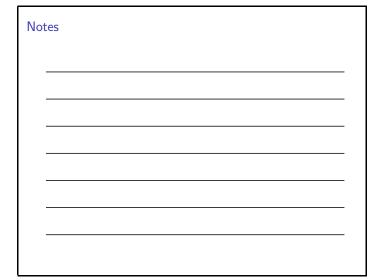
Consider a coinage system:

- where  $c_1 = 20, c_2 = 15, c_3 = 7, c_4 = 1$
- $\blacktriangleright$  and we want to give 22 "cents" in change.

What will this algorithm produce?

#### Is it optimal?

It is *not* optimal since it would give us one  $c_4$  and two  $c_1$ , for three coins, while the optimal is one  $c_2$  and one  $c_3$  for two coins.





# Change-Making Algorithm Optimal?

What about the US currency system—is the algorithm correct in this case?

Yes, in fact, we can prove it by contradiction.

For simplicity, let  $c_1 = 25, c_2 = 10, c_3 = 5, c_4 = 1$ .

# Change-Making Algorithm

Proving optin Proof.

- Let  $C = \{d_1, d_2, \ldots, d_k\}$  be the solution given by the greedy algorithm for some integer n. By way of contradiction, assume there is *another* solution  $C' = \{d'_1, d'_2, \ldots, d'_l\}$  with l < k.
- Consider the case of quarters. Say in C there are q quarters and in C' there are q'. If q' > q, the greedy algorith would have used q'.
- Since the greedy algorithm uses as many quarters as possible, n = q(25) + r. where r < 25, thus if q' < q, then in n = q'(25) + r',  $r' \ge 25$  and so C' does not provide an optimal solution.
- ▶ Finally, if q = q', then we continue this argument on dimes and nickels. Eventually we reach a contradiction.
- ▶ Thus, C = C' is our optimal solution.

#### Change-Making Algorithm Proving optimality

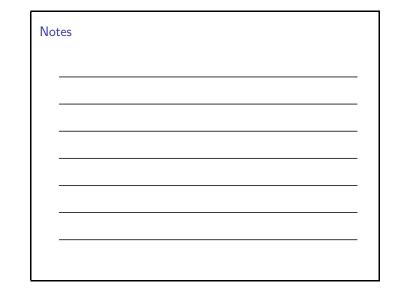
Why (and where) does this proof fail in our previous counter example to the general case?

We need the following lemma:

If n is a positive integer then n cents in change using quarters, dimes, nickels, and pennies using the fewet coins possible

- 1. has at most two dimes, at most one nickel, at most most four pennies, and
- 2. cannot have two dimes and a nickel.

The amount of change in dimes, nickels, and pennies cannot exceed 24 cents.



Notes