

# Algorithms: An Introduction

*'Algorithm' is a distortion of Al-Khwarizmi, a Persian mathematician*



## Section 3.1 of Rosen

Spring 2017

CSCE 235 Introduction to Discrete Structures (Honors)

Course web-page: [cse.unl.edu/~cse235h](http://cse.unl.edu/~cse235h)

Questions: Piazza

# Outline

---

- Introduction & definition
- Algorithms categories & types
- Pseudo-code
- Designing an algorithm
  - Example: MAX
- Greedy Algorithms
  - CHANGE

# Computer Science is About Problem Solving

---

- A Problem is specified by
  1. **The givens** (a formulation)
    - A set of objects
    - Relations between them
  2. **The query**
    - The information one wants to extract from the formulation, the question to answer

Real World	↔	Computing World
Objects	represented by...	data Structures, ADTs, Classes
Relations	implemented with...	relations & functions (e.g., predicates)
Actions	Implemented with...	algorithms: a sequence of instructions

- An algorithm is a method or procedure that solves instances of a problem

# Algorithms: Formal Definition

---

- **Definition:** An algorithm is a sequence of unambiguous instructions for solving a problem.
- Properties of an algorithm
  - **Finite:** the algorithm must eventually terminate
  - **Complete:** Always give a solution when one exists
  - **Correct (sound):** Always give 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
- Efficient= runs in polynomial time. Thus, **effective ≠ efficient**
- There can be many algorithms to solve the same problem

# Outline

---

- Introduction & definition
- **Algorithms categories & types**
- Pseudo-code
- Designing an algorithm
  - Example: MAX
- Greedy Algorithms
  - CHANGE

# Algorithms: General Techniques

---

- There are many broad categories of algorithms
  - Deterministic versus Randomized (e.g., Monte Carlo)
  - Exact versus Approximation
  - Sequential/serial versus Parallel, etc.
- Some general styles of algorithms include
  - Brute force (enumerative techniques, exhaustive search)
  - Divide & Conquer
  - Transform & Conquer (reformulation)
  - Greedy Techniques

# Outline

---

- Introduction & definition
- Algorithms categories & types
- **Pseudo-code**
- Designing an algorithm
  - Example: MAX
- Greedy Algorithms
  - CHANGE

# Good Pseudo-Code: Example

---

## INTERSECTION

*Input:* Two finite sets  $A, B$

*Output:* A finite set  $C$  such that  $C = A \cap B$

1.  $C \leftarrow \emptyset$
2. **If**  $|A| > |B|$  **Then**  $\text{SWAP}(A, B)$
3. **For every**  $x \in A$  **Do**
4.     **If**  $x \in B$  **Then**  $C \leftarrow C \cup \{x\}$                        $\text{UNION}(C, \{x\})$
5. **End**
6. **Return**  $C$



# Algorithms: Pseudo-Code

---

- Algorithms are usually presented using pseudo-code
- 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 such as set union)
- Good pseudo-code
  - Is a balance between clarity and detail
  - Abstracts the algorithm
  - Makes good use of mathematical notation
  - Is easy to read and
  - Facilitates implementation (reproducible, does not hide away important information)

# Writing Pseudo-Code: Advice

---

- Input/output must properly defined
- All your variables must be properly initialized, introduced
- Variables are instantiated, assigned using  $\leftarrow$
- All 'commands' (while, if, repeat, begin, end) boldface `\bf`  
**For  $i \leftarrow 1$  to  $n$  Do**
- All functions in small caps `UNION(s,t)` `\sc`
- All constants in courier: `pi  $\leftarrow$  3.14` `\tt`
- All variables in italic: `temperature  $\leftarrow$  78` `\mathit{}`
- LaTeX: Several algorithm formatting packages exist on WWW

# Outline

---

- Introduction & definition
- Algorithms categories & types
- Pseudo-code
- **Designing an algorithm**
  - **Example: MAX**
- Greedy Algorithms
  - CHANGE

# Designing an Algorithm

---

- A general approach to designing algorithms is as follows
  - Understanding the problem, **assess its difficulty**
  - Choose an approach (e.g., exact/approximate, deterministic/probabilistic)
  - (Choose appropriate data structures)
  - Choose a strategy
  - Prove
    1. Termination
    2. Completeness
    3. Correctness/soundness
  - Evaluate complexity
  - Implement and test it
  - Compare to other known approach and algorithms

# Algorithm Example: MAX

---

- When designing an algorithm, we usually give a formal statement about the problem to solve
- **Problem**
  - **Given:** a set  $A=\{a_1, a_2, \dots, a_n\}$  of integers
  - **Question:** find the index  $i$  of the maximum integer  $a_i$
- A straightforward idea is
  - Simply store an initial maximum, say  $a_1$
  - Compare the stored maximum to every other integer in  $A$
  - Update the stored maximum if a new maximum is ever encountered

# Pseudo-code of Max

---

MAX

*Input:* A finite set  $A = \{a_1, a_2, \dots, a_n\}$  of integers

*Output:* The largest element in the set

1.  $temp \leftarrow a_1$
2. **For**  $i = 2$  **to**  $n$  **Do**
3.     **If**  $a_i > temp$
4.         **Then**  $temp \leftarrow a_i$
5.     **End**
6. **End**
7. **Return**  $temp$

# Algorithms: Other Examples

---

- Check Bubble Sort and Insertion Sort in your textbooks
- ... which you should have seen ad nauseum in CSE 155 and CSE 156
- And which you will see again in CSE 310
- Let us know if you have any questions

# Outline

---

- Introduction & definition
- Algorithms categories & types
- Pseudo-code
- Designing an algorithm
  - Example: MAX
- **Greedy Algorithms**
  - **CHANGE**



# Greedy Algorithms

---

- 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
  - Choose what appears now to be the best choice
  - Choose the most immediate best solution (i.e., think locally)
- Greedy algorithms
  - Work well on some (simple) problems
  - Usually they are not guaranteed to produce the best globally optimal solution

# Change-Making 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  $(c_1, c_2, \dots, c_r)$  with  $c_1 > c_2 > \dots > c_r$
  - **Query:** Find a set of coins  $d_1, d_2, \dots, d_k$  such that  $\sum_{i=1}^k d_i = n$  and  $k$  is minimized

# Greedy Algorithm: CHANGE

---

CHANGE

*Input:* An integer  $n$  and a set of coin denominations  $\{c_1, c_2, \dots, c_r\}$   
with  $c_1 > c_2 > \dots > c_r$

*Output:* A set of coins  $d_1, d_2, \dots, d_r$  such that  $\sum_{i=1}^r d_i \cdot c_i = n$  and  $\sum_{i=1}^r d_i$  is minimized

1. **For**  $i=1$  **to**  $r$  **Do**

2.      $d_i \leftarrow 0$

3.     **While**  $n \geq c_i$  **Do**

4.          $d_i \leftarrow d_i + 1$

5.          $n \leftarrow n - c_i$

6.     **End**

7. **Return**  $\{d_i\}$

# CHANGE: Analysis (1)

---

- Will the algorithm always produce an optimal answer?
- Example
  - Consider a coinage system where  $c_1=20$ ,  $c_2=15$ ,  $c_3=7$ ,  $c_4=1$
  - We want to give 22 ‘cents’ in change
- What is the output of the algorithm?
- Is it optimal?
- It is not optimal because it would give us two  $c_4$  and one  $c_1$  (3 coins). The optimal change is one  $c_2$  and one  $c_3$  (2 coins)

# CHANGE: Analysis (2)

---

- What about the US currency system: is the algorithm correct in this case?
- Yes, in fact it is. We can prove it by contradiction.
- For simplicity, let us consider

$$c_1=25, c_2=10, c_3=5, c_4=1$$

# Optimality of CHANGE (1)

---

- Let  $C = \{d_1, d_2, \dots, d_k\}$  be the solution given by the greedy algorithm for some integer  $n$ .
- By way of contradiction, assume there is a better solution  $C' = \{d_1', d_2', \dots, d_l'\}$  with  $l < k$
- Consider the case of quarters. Say there are  $q$  quarters in  $C$  and  $q'$  in  $C'$ .
  1. If  $q' > q$ , the greedy algorithm would have used  $q'$  by construction. Thus, it is impossible that the greedy uses  $q < q'$ .
  2. Since the greedy algorithm uses as many quarters as possible,  $n = q(25) + r$ , where  $r < 25$ . If  $q' < q$ , then,  $n = q'(25) + r'$  where  $r' \geq 25$ .  $C'$  will have to use more smaller coins to make up for the large  $r'$ . Thus  $C'$  is not the optimal solution.
  3. If  $q = q'$ , then we continue the argument on the smaller denomination (e.g., dimes). Eventually, we reach a contradiction.
- Thus,  $C = C'$  is our optimal solution

# Optimality of CHANGE (2)

---

- But, how about the previous counterexample? Why (and where) does this proof?
- We need the following lemma:

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

- Has at most two dimes
- Has at most one nickel
- Has at most four pennies, and
- Cannot have two dimes and a nickel

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

# Greedy Algorithm: Another Example

---

- Check the problem of Scenario I, page 25 in the slides [IntroductiontoCSE235.ppt](#)
- We discussed then (remember?) a greedy algorithm for accommodating the maximum number of customers. The algorithm
  - terminates, is complete, sound, and satisfies the maximum number of customers (finds an optimal solution)
  - runs in time linear in the number of customers



# Summary

---

- Introduction & definition
- Algorithms categories & types
- Pseudo-code
- Designing an algorithm
  - Example: MAX
- Greedy Algorithms
  - Example: CHANGE