CSE 830: Design and Theory of Algorithms

Dr. Yanni Sun
Outline

• Administrative stuff ...

• Definitions
  – Data structures
  – Algorithms
  – Problems

• Course Objectives

• Analysis of algorithm complexity using sorting problem as an example
Definitions

• Data structures and algorithms
  – Data structure provides a way to organize data (e.g. organize all words in a dictionary)
  – Has associated operations
  – E.g. linear data structures, binary trees; insertion or deletion

• Algorithm: (intuitively) “a bag of tricks”
What is an Algorithm?

Algorithms are the ideas behind computer programs.

An algorithm is the thing that stays the same whether the program is in C++ running on a Cray in New York or is in BASIC running on a Macintosh in Alaska!

To be interesting, an algorithm has to solve a general, specified problem.
What is a problem?

• Definition
  – A mapping/relation between a set of input instances (domain) and an output set (range)

• Problem Specification
  – Specify what a typical input instance is
  – Specify what the output should be in terms of the input instance
Example problem: sorting

Input: A sequence of \( n \) numbers \( <a_1, a_2, \ldots, a_n> \)

Output: A permutation (reordering) \( <a'_1, a'_2, \ldots, a'_n> \) of the input sequence s.t. \( a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n \)

\[ 50 \ 100 \ 1 \ 0 \ \Rightarrow \ 0 \ 1 \ 50 \ 100 \]

\[ a_1 \ a_2 \ a_3 \ a_4 \ \Rightarrow \ a'_1 \ a'_2 \ a'_3 \ a'_4 \]
Types of Problems

Search: find X in the input satisfying property Y

Structuring: Transform input X to satisfy property Y

Construction: Build X satisfying Y

Optimization: Find the best X satisfying property Y

Decision: Does X satisfy Y?

Adaptive: Maintain property Y over time.
Two desired properties of algorithms

• Correctness
  – Always provides correct output when presented with legal input

• Efficiency
  – Computes correct output quickly given input
Correctness & Efficiency

- Example: Traveling Salesperson Problem (TSP)
- **Input:** A sequence of N cities with the distances $d_{ij}$ between each pair of cities
- **Output:** a permutation (ordering) of the cities $<c_1, \ldots, c_n>$ that minimizes the expression ($d_{i,j}$ is the distance between cities $c_i$ and $c_j$)
  \[ \sum_{j=1}^{n-1} d_{j,j+1} + d_{n,1} \]
- Which of the following algorithms is correct? How about efficiency?
  - Nearest neighbor: Initialize tour to city 1. Extend tour by visiting nearest unvisited city. Finally return to city 1.
  - All tours: Try all possible orderings of the points selecting the ordering that minimizes the total length:
Outline

• Administrative stuff …

• Definitions
  – Data structures
  – Algorithms
  – Problems

• Course Objectives

• Analysis of algorithm complexity using sorting problem as an example
Course Objectives

1. Details of classic algorithms
2. Methods for designing algorithms
3. Validate/verify algorithm correctness
4. Analyze algorithm efficiency
5. Prove (or at least indicate) no correct, efficient algorithm exists for solving a given problem
6. Writing clear algorithms and proofs
Classic Algorithms

- Lots of wonderful algorithms have already been developed
- Use these as examples to guide other objectives
Algorithm design methods

• Something of an art form
• Cannot be fully automated
• We will describe some general techniques and try to illustrate when each is appropriate
Algorithm correctness

• Proving an algorithm generates correct output for all inputs
• We will not emphasize algorithm correctness in this course
Analyzing algorithms

• The “process” of determining how much resources (time, space) are used by a given algorithm
• We want to be able to make quantitative assessments about the value (goodness) of one algorithm compared to another
• We want to do this WITHOUT implementing and running an executable version of an algorithm
• A way to think about computation. What is a “good” algorithm? Which of two algorithms is “better”? What does “fast”/“faster” mean?

analytical techniques - asymptotic notation lower bounds

• Evaluate which algorithm is “faster”
  • Implement and record the actual running time
  • Potential problems of this method?

We want to do this WITHOUT implementing and running an executable version of an algorithm
Proving hardness results

- We believe that no correct and efficient algorithm exists that solves many problems such as TSP
- We define a formal notion of a problem being hard (NP-hard)
- We develop techniques for proving hardness results
Clear Writing

• Methods for Expressing Algorithms
  – Implementations
  – Pseudo-code
  – English

• Writing clear and understandable proofs

• My main concern is not the specific language used but the clarity of your algorithm/proof
Outline

• Administrative stuff …

• Definitions
  – Data structures
  – Algorithms
  – Problems

• Course Objectives

• Analysis of algorithm complexity using sorting problem as an example
Input: A sequence of $n$ numbers $<a_1, a_2, \ldots, a_n>$

Output: A permutation (reordering) $<a'_1, a'_2, \ldots, a'_n>$ of the input sequence s.t. $a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n$

50 100 1 0 $\Rightarrow$ 0 1 50 100

$a_1$ $a_2$ $a_3$ $a_4$ $a'_1$ $a'_2$ $a'_3$ $a'_4$

Insertion Sort

1 50 55 $\Rightarrow$ 70 80 60

Insertion operation
Insertion Sort example

\( j = 1 \quad 5 \quad 2 \quad 4 \quad 6 \quad 1 \quad 3 \)

\( j = 2 \quad 2 \quad 5 \quad 4 \quad 6 \quad 1 \quad 3 \)

\( j = 3 \quad 2 \quad 4 \quad 5 \quad 6 \quad 1 \quad 3 \)

\( j = 4 \quad 2 \quad 4 \quad 5 \quad 6 \quad 1 \quad 3 \)

\( j = 5 \quad 1 \quad 2 \quad 4 \quad 5 \quad 6 \quad 3 \)

\( j = 6 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \)
**Insertion - SORT (A)**  // A is the input array of size n
\[ \text{length}[A] = n \]

1. \( \text{for } j = 2 \text{ to } \text{length}[A] \)

2. \( \text{do } \text{key} \leftarrow A[j] \)

3. \( \text{// insert } A[j] \text{ into a sorted list } A[i..j-1] \)

4. \( i \leftarrow j-1 \)

5. \( \text{while } i > 0 \text{ and } A[i] > \text{key} \)

6. \( \text{do } A[i+1] \leftarrow A[i] \)

7. \( i \leftarrow i-1 \)

8. \( A[i+1] \leftarrow \text{key} \)