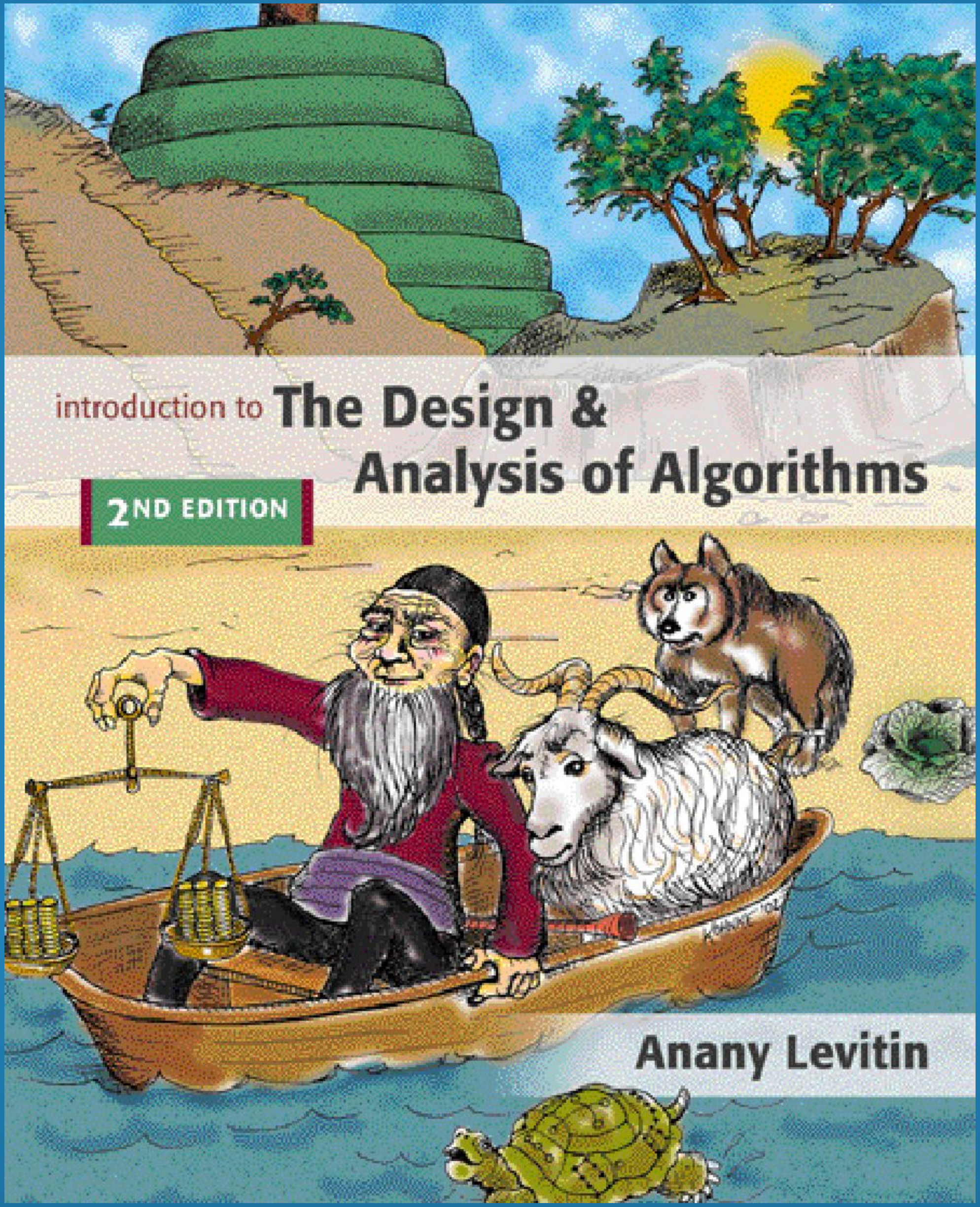
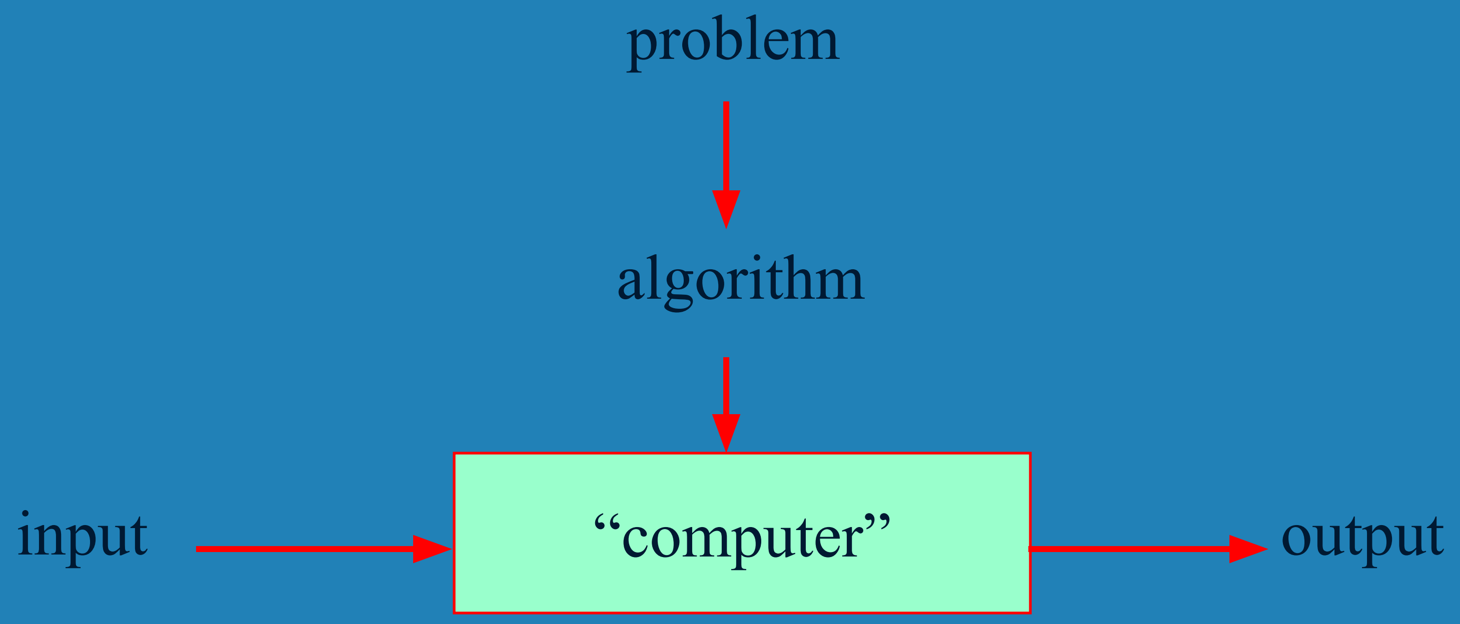
Chapter 1

Introduction



What is an algorithm?

An ***algorithm*** **is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.**



Algorithm

- **An _algorithm_ is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.**

  - Can be represented various forms
  - Unambiguity/clearness
  - Effectiveness
  - Finiteness/termination
  - Correctness

What is an algorithm?

- Recipe, process, method, technique, procedure, routine,… with the following requirements:

1. Finiteness
   - terminates after a finite number of steps
2. Definiteness
   - rigorously and unambiguously specified
3. Clearly specified input
   - valid inputs are clearly specified
4. Clearly specified/expected output
   - can be proved to produce the correct output given a valid input
5. Effectiveness
   - steps are sufficiently simple and basic

Historical Perspective

- **Muhammad ibn Musa al-Khwarizmi – 9th century mathematician** [**www.lib.virginia.edu/science/parshall/khwariz.html**](www.lib.virginia.edu/science/parshall/khwariz.html)

Some Well-known Computational Problems

- Sorting
- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Traveling salesman problem
- Knapsack problem
- Chess
- Towers of Hanoi
- Program termination

Some of these problems don't have efficient algorithms, or algorithms at all!

Basic Issues Related to Algorithms

- How to design algorithms
- How to express algorithms
- Proving correctness
- Efficiency (or complexity) analysis
  - Theoretical analysis
  - Empirical analysis
- Optimality

Why study algorithms?

- **Theoretical importance**
  - **the core of computer science**
- **Practical importance**
  - **A practitioner's toolkit of known algorithms**
  - **Framework for designing and analyzing algorithms for new problems**

Example: Google's PageRank Technology

Algorithm design techniques/strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound

Analysis of Algorithms

- How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency

- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

Other methods for computing gcd(*m*,*n*)

Consecutive integer checking algorithm

Step 1 Assign the value of min{*m*,*n*} to *t*

Step 2 Divide *m* by *t*. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide *n* by *t*. If the remainder is 0, return *t* and stop; otherwise, go to Step 4

Step 4 Decrease *t* by 1 and go to Step 2

Is this slower than Euclid's algorithm?
How much slower?

O(n), if n <= m , vs O(log n)

Other methods for gcd(*m*,*n*) [cont.]

Middle-school procedure

Step 1 Find the prime factorization of *m*

Step 2 Find the prime factorization of *n*

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd(*m*,*n*)

Is this an algorithm?

How efficient is it?

Time complexity: O(sqrt(n))

Two main issues related to algorithms

- How to design algorithms

- How to analyze algorithm efficiency

Analysis of algorithms

- How good is the algorithm?
  - time efficiency
  - space efficiency
  - correctness ignored in this course

- Does there exist a better algorithm?
  - lower bounds
  - optimality

Important problem types

- sorting
- searching
- string processing
- graph problems
- combinatorial problems
- geometric problems
- numerical problems

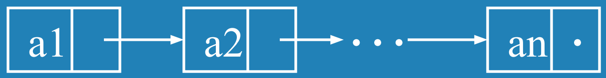
Fundamental data structures

- list
  - array
  - linked list
  - string
- stack
- queue
- priority queue/heap
- graph
- tree and binary tree
- set and dictionary

Linear Data Structures

- **Arrays**
  - **A sequence of n items of the same data type that are stored contiguously in computer memory and made accessible by specifying a value of the array's index.**
- **Linked List**
  - **A sequence of zero or more nodes each containing two kinds of information: some data and one or more links called pointers to other nodes of the linked list.**
  - **Singly linked list (next pointer)**
  - **Doubly linked list (next + previous pointers)**

- Arrays
  - fixed length (need preliminary reservation of memory)
  - contiguous memory locations
  - direct access
  - Insert/delete
- Linked Lists
  - dynamic length
  - arbitrary memory locations
  - access by following links
  - Insert/delete

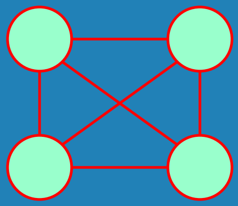


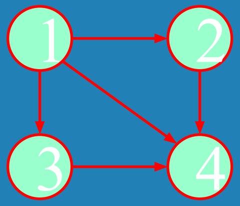
Stacks and Queues

- **Stacks**
  - **A stack of plates**
    - **insertion/deletion can be done only at the top.**
    - **LIFO**
  - **Two operations (push and pop)**
- **Queues**
  - **A queue of customers waiting for services**
    - **Insertion/enqueue from the rear and deletion/dequeue from the front.**
    - **FIFO**
  - **Two operations (enqueue and dequeue)**

Graphs

- Formal definition
  - A graph $G = <V, E>$ is defined by a pair of two sets: a finite set V of items called vertices and a set E of vertex pairs called edges.
- Undirected and directed graphs (digraphs).
- What's the maximum number of edges in an undirected graph with |V| vertices?
- Complete, dense, and sparse graphs
  - A graph with every pair of its vertices connected by an edge is called complete, $K_{|V|}$

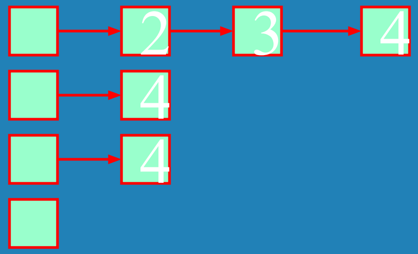




Graph Representation

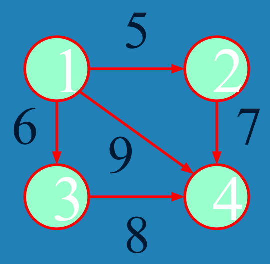
- **Adjacency matrix**
  - **n x n boolean matrix if |V| is n.**
  - **The element on the ith row and jth column is 1 if there's an edge from ith vertex to the jth vertex; otherwise 0.**
  - **The adjacency matrix of an undirected graph is symmetric.**
- **Adjacency linked lists**
  - **A collection of linked lists, one for each vertex, that contain all the vertices adjacent to the list's vertex.**
- **Which data structure would you use if the graph is a 100-node star shape?**

$$\begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$$



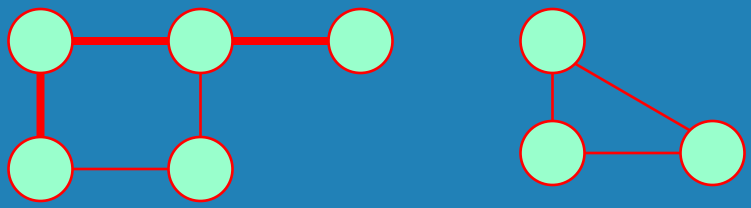
Weighted Graphs

- **Weighted graphs**
  - **Graphs or digraphs with numbers assigned to the edges.**



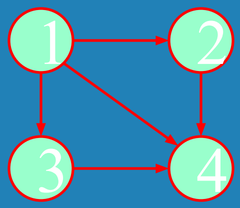
Graph Properties -- Paths and Connectivity

- **Paths**
  - A path from vertex u to v of a graph G is defined as a sequence of adjacent (connected by an edge) vertices that starts with u and ends with v.
  - **Simple paths**: All edges of a path are distinct.
  - Path lengths: the number of edges, or the number of vertices – 1.
- **Connected graphs**
  - A graph is said to be connected if for every pair of its vertices u and v there is a path from u to v.
- **Connected component**
  - The maximum connected subgraph of a given graph.



Graph Properties -- Acyclicity

- **Cycle**
  - **A simple path of a positive length that starts and ends a the same vertex.**
- **Acyclic graph**
  - **A graph without cycles**
  - **DAG (Directed Acyclic Graph)**

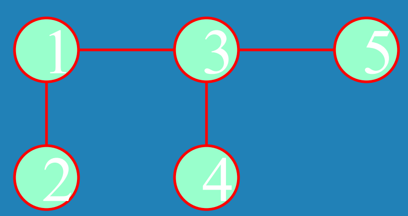


Trees

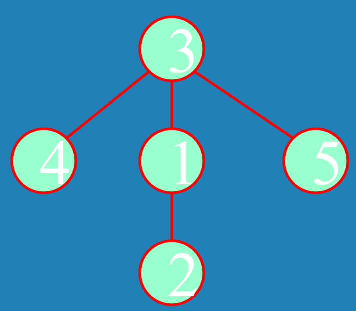
- Trees
  - A tree (or free tree) is a connected acyclic graph.
  - Forest: a graph that has no cycles but is not necessarily connected.
- Properties of trees

  - For every two vertices in a tree there always exists exactly one simple path from one of these vertices to the other. Why?
    - Rooted trees: The above property makes it possible to select an arbitrary vertex in a free tree and consider it as the root of the so called rooted tree.
    - Levels in a rooted tree.

- $|E| = |V| - 1$



rooted




Understand the problem
Decide on:
computational means,
exact vs. approximate solving,
algorithm design technique
Design an algorithm
Prove correctness
Analyze the algorithm
Code the algorithm

Algorithm design and analysis process

Basic Efficiency Classes

Class	Name	Comments
1	constant	May be in best cases
$\lg n$	logarithmic	Halving problem size at each iteration
n	linear	Scan a list of size n
$n \cdot \lg n$	linearithmic	Divide and conquer algorithms, e.g., mergesort
n^2	quadratic	Two embedded loops, e.g., selection sort
n^3	cubic	Three embedded loops, e.g., matrix multiplication
2^n	exponential	All subsets of n-elements set
$n!$	factorial	All permutations of an n-elements set