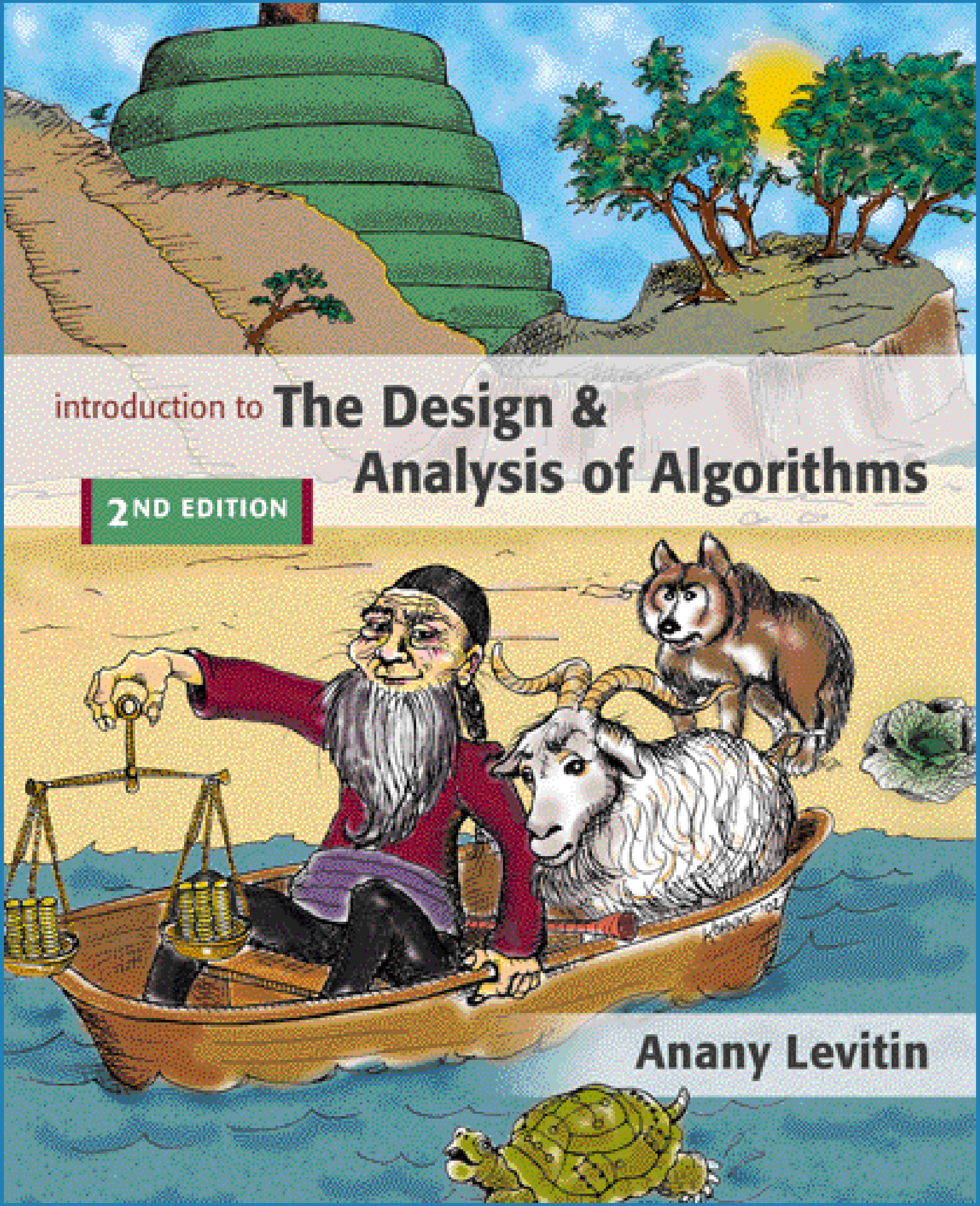
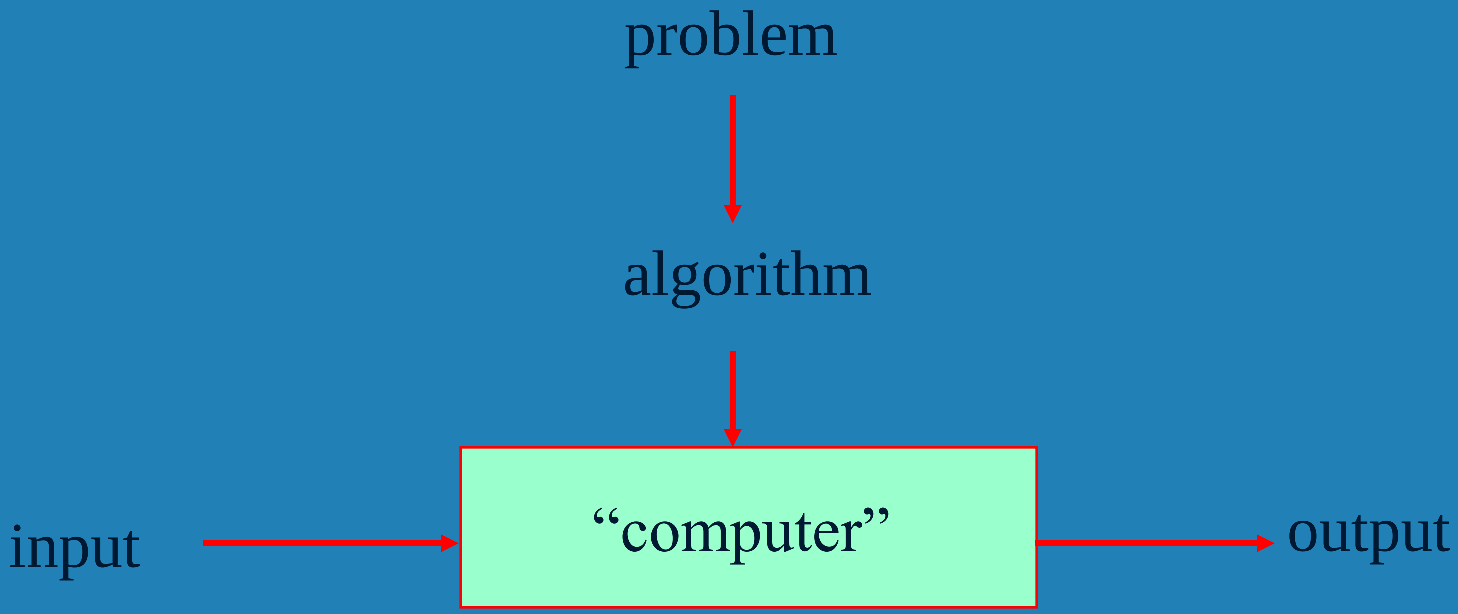
Chapter 1

Introduction



Algorithms

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



Example of Computational Problem: Sorting

- **Statement of problem:**
  - ***Input:*** **A sequence of *n* numbers** $<a_1, a_2, \ldots, a_n>$
  - ***Output:*** **A reordering of the input sequence** $<a'_1, a'_2, \ldots, a'_n>$ **so that** $a'_i \leq a'_j$ **whenever** $i < j$
- **Instance: The sequence <5, 3, 2, 8, 3>**
- **Algorithms:**
  - **Selection sort**
  - **Insertion sort**
  - **Merge sort**
  - **(many others)**

Properties of Algorithms

- What distinguish an algorithm from a recipe, process, method, technique, procedure, routine…?
  - Finiteness
    terminates after a finite number of steps
  - Definiteness
    Each step must be rigorously and unambiguously specified.
    -e.g., ”stir until lumpy”
  - Input
    Valid inputs must be clearly specified.
  - Output
    The data that result upon completion of the algorithm must be specified.
  - Effectiveness
    Steps must be sufficiently simple and basic.

Examples

- Is the following a legitimate algorithm?

```
i ← 1
While (i <= 10) do
    a ← i + 1
    Print the value of a
End of loop
Stop
```

Examples of Algorithms – Computing the Greatest Common Divisor of Two Integers

- gcd(m, n): <u>the largest integer that divides both m and n</u>
- First try -- Euclid's algorithm: gcd(m, n) = gcd(n, m mod n)
  - Step1: If n = 0, return the value of m as the answer and stop; otherwise, proceed to Step 2.
  - Step2: Divide m by n and assign the value of the remainder to r.
  - Step 3: Assign the value of n to m and the value of r to n. Go to Step 1.

Examples of Algorithms – Computing the Greatest Common Divisor of Two Integers

- gcd(m, n): the largest integer that divides both m and n
- First try -- Euclid's algorithm: gcd(m, n) = gcd(n, m mod n)
  - Step1: If n = 0, return the value of m as the answer and stop; otherwise, proceed to Step 2.
  - Step2: Divide m by n and assign the value of the remainder to r.
  - Step 3: Assign the value of n to m and the value of r to n. Go to Step 1.

gdc(60,24) = gdc(24,12) = gdc(12,0) = 12

Pseudocode of Euclid’s Algorithm

```
Algorithm Euclid(m, n)
//Computes gcd(m, n) by Euclid’s algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

- Questions:
  - Finiteness: how do we know that Euclid’s algorithm actually comes to a stop?
  - Definiteness: nonambiguity
  - Effectiveness: effectively computable

Second Try for gcd(m, n)

- Consecutive Integer Checking Algorithm
  - Step1: Assign the value of min{m, n} to t.
  - Step2: Divide m by t. If the remainder of this division is 0, go to Step3; otherwise, go to Step 4.
  - Step3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step4.
  - Step4: Decrease the value of t by 1. Go to Step2.
- Questions
  - Finiteness
  - Definiteness
  - Effectiveness
  - Which algorithm is faster, the Euclid’s or this one?

Second Try for gcd(m, n)

- Consecutive Integer Checking Algorithm
  - Step1: Assign the value of min{m, n} to t.
  - Step2: Divide m by t. If the remainder of this division is 0, go to Step3; otherwise, go to Step 4.
  - Step3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step4.
  - Step4: Decrease the value of t by 1. Go to Step2.
- Questions
  - Finiteness
  - Definiteness
  - Effectiveness - o que acontece quando m ou n for zero?
  - Which algorithm is faster, the Euclid’s or this one?

Third try for gcd(m, n)

- Middle-school procedure
  - Step1: Find the prime factors of m.
  - Step2: Find the prime factors of n.
  - Step3: Identify all the common factors in the two prime expansions found in Step1 and Step2. (If p is a common factor occurring Pm and Pn times in m and n, respectively, it should be repeated in min{Pm, Pn} times.)
  - Step4: Compute the product of all the common factors and return it as the gcd of the numbers given.
- gdc(60,24) =

Third try for gcd(m, n)

- Middle-school procedure
  - Step1: Find the prime factors of m.
  - Step2: Find the prime factors of n.
  - Step3: Identify all the common factors in the two prime expansions found in Step1 and Step2. (If p is a common factor occurring Pm and Pn times in m and n, respectively, it should be repeated in min{Pm, Pn} times.)
  - Step4: Compute the product of all the common factors and return it as the gcd of the numbers given.
- 60 = 2 . 2 . 3 . 5
- 24 = 2 . 2 . 2. 3
- gdc(60,24) = 2 . 2 . 3 = 12
- Question
  - Is this a legitimate algorithm?
  - Homework: algorithm for the Sieve (Crivo) of Erastosthenes

What can we learn from the previous 3 examples?

- **Each step of an algorithm must be unambiguous.**
- **The same algorithm can be represented in several different ways (different pseudocodes).**
- **There might exists more than one algorithm for a certain problem.**
- **Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.**

Fundamentals of Algorithmic Problem Solving

- **Understanding the problem**
  - **Asking questions, do a few examples by hand, think about special cases, etc.**
- **Deciding on**
  - **Exact vs. approximate problem solving**
  - **Appropriate data structure**
- **Design an algorithm** and specify it in some fashion
- **Proving correctness** (exact versus approximate approach)
- **Analyzing an algorithm**
  - **Time efficiency : how fast the algorithm runs**
  - **Space efficiency: how much extra memory the algorithm needs.**
  - **Simplicity and generality**
- **Coding an algorithm**

Algorithm design strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
    - Sorting, Matching, Hashing, B-Trees
- Dynamic programming
- Greedy approach
- Iterative Improvement
- Limitations of Algorithm Power
    - Decision Trees, P/NP, Numerical Algorithms
- Copy with Limitations
    - Backtracking, Branch and bound, Approximation

Important Problem Types

- **Sorting**
- **Searching**
- **String processing**
- **Graph problems**

Sorting (I)

- Rearrange the items of a given list in ascending order.
  - Input: A sequence of n numbers $<a_1, a_2, \ldots, a_n>$
  - Output: A reordering $<a'_1, a'_2, \ldots, a'_n>$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.
- Why sorting?
  - Help searching
  - Algorithms often use sorting as a key subroutine.
- Sorting key
  - A specially chosen piece of information used to guide sorting.

Sorting (II)

- Examples of sorting algorithms
  - Selection sort
  - Bubble sort
  - Insertion sort
  - Merge sort
  - Heap sort …
- Evaluate sorting algorithm complexity: the number of key comparisons.
- Two properties
  - Stability: A sorting algorithm is called stable if it preserves the relative order of any two equal elements in its input.
  - In place (in loco) : A sorting algorithm is in place if it does not require extra memory, except, possibly for a few memory units.

Selection Sort

```
Algorithm SelectionSort(A[0..n-1])
//The algorithm sorts a given array by selection sort
//Input: An array A[0..n-1] of orderable elements
//Output: Array A[0..n-1] sorted in ascending order
for i ← 0 to n – 2 do
    min ← i
    for j ← i + 1 to n – 1 do
        if A[j] < A[min]
            min ← j
    swap A[i] and A[min]
```

Searching

- **Find a given value, called a <u>search key</u>, in a given set.**
- **Examples of searching algorithms**
  - **Sequential searching**
  - **Binary searching…**

String Processing

- A string is a sequence of characters from an alphabet.
- Text strings, bit strings, gene sequences (A,C,G,T).
- String matching: searching for a given word/pattern in a text.

Graph Problems

- Informal (Visual) definition
  - A graph is a collection of points called vertices, some of which are connected by line segments called edges.
- Modeling real-life problems
  - Modeling WWW
  - Communication Networks
  - Project scheduling …
- Examples of graph algorithms
  - Graph traversal algorithms
  - Shortest-path algorithms
  - Topological sorting

Combinatorial Problems

- Finding a permutation, a combination or a subset that satisfies certain constraints and has some desired property
- Most difficult problems in computing
  - Number of candidates grows extremely fast with the problem´s size
  - There are no known exact or efficient algorithms for most such problems

Fundamental Data Structures

- Linear data structures
  - Stacks, queues, and heaps
- Graphs
- Trees

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: time consuming

Linked Lists

dynamic length

arbitrary memory locations

access by following links: time consuming

Insert/delete

Stacks, Queues, and Heaps (1)

- **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)

Stacks, Queues, and Heaps (2)

- Priority queues (implemented using heaps)
  - A data structure for maintaining a set of elements, each associated with a key/priority, with the following operations
    - Finding the element with the highest priority
    - Deleting the element with the highest priority
    - Inserting a new element
  - Scheduling jobs on a shared computer.

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 (digraph).
- What's the maximum number of edges in an undirected graph with |V| vertices?
  - (|V| * (|V|-1))/2
- Complete, dense, and sparse graph
  - A graph with every pair of its vertices connected by an edge is called complete $K_{|V|}$
- Weighted graphs
  - Graphs or digraphs with numbers assigned to the edges.

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.
  - Less space for sparse graphs

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 components
  - The maximum connected subgraphs of a given graph.

Graph Properties -- Acyclicity

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

Trees (I)

- Trees
  - A tree (or free tree) is a <u>connected acyclic graph</u>.
  - Forests: a graph that has no cycles but is not necessarily connected.
- Properties of trees
  - |E| = |V| - 1
  - For every two vertices in a tree there always exists exactly one simple path from one of these vertices to the other.
- <u>Rooted trees</u>: 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 of rooted tree.

Trees (II)

- **Ancestors**
  - **For any vertex *v* in a tree *T*, all the vertices on the simple path from the root to that vertex are called ancestors.**
- **Descendants**
  - **All the vertices for which a vertex *v* is an ancestor are said to be descendants of *v*.**
- **Parent, Child and Siblings**
  - **If *(u, v)* is the last edge of the simple path from the root to vertex *v* (and *u* ‡ *v*), *u* is said to be the parent of *v* and *v* is called a child of *u*.**
  - **Vertices that have the same parent are called siblings.**
- **Leaves**
  - **A vertex without children is called a leaf.**
- **Subtree**
  - **A vertex *v* with all its descendants is called the subtree of *T* rooted at *v*.**

Trees (III)

- **Depth of a vertex**
  - **The length of the simple path from the root to the vertex.**
- **Height of a tree**
  - **The length of the longest simple path from the root to a leaf.**

Ordered Trees

- Ordered trees
  - An ordered tree is a rooted tree in which all the children of each vertex are ordered.
- Binary trees
  - A binary tree is an ordered tree in which every vertex has no more than two children and each children is designated as either a *left child* or a *right child* of its parent.
- Binary search trees
  - Each vertex is assigned a number.
  - A number assigned to each parental vertex is larger than all the numbers in its left subtree and smaller than all the numbers in its right subtree.
- $\lfloor \log_2 n \rfloor \leq h \leq n - 1$, where h is the height of a binary tree and n is its number of nodes.