Algorithms

Chaper 1.1, 1.2, 1.3

Syllabus

FOLLOW THE WEB SITE FOR ANNOUNCEMENTS AND CHANGES

Web Page

http://pdc.ege.edu.tr/pdcworks/courses/344/344.html

Textbook

- Anany Levitin. Introduction to The Design and Analysis of Algorithms, Addison Wesley, 3rd edition.

Other Books

- Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press; 3rd edition, 2009 ISBN-10: 0262033844 ISBN-13: 978-0262033848

Syllabus-Catalog Description:

- Basic definitions and data structures.
- Introduction to analysis of algorithms.
- Standard algorithm design techniques;
  - divide-and-conquer,
  - greedy,
  - dynamic programming,
  - branch-and-bound,
  - backtracking,
  - etc.
- Basic algorithms;
  - sorting and searching,
  - graph algorithms,
  - etc.
- Introduction to complexity classes.

Syllabus - Educational Objectives:

This course introduces basic algorithms, algorithm design and analysis techniques which can be used in designing solutions to real life problems. After this course, you will

- able to design a new algorithms for a problem using the methods discussed in the class
- able to analyze an algorithm with respect to various performance criteria such as memory use and running time
- able to choose the most suitable algorithm for a problem to be solved,
- able to implement an algorithm efficiently.

Syllabus - Attendance Policy:

- Class attendance is advised but will not be a part of your final grade. However, a minimum of 70% attendance is required. DO NOT SIGN FOR OTHER STUDENTS.
- Please be considerate of your classmates during class. Students are expected to show courtesy and respect toward their classmates.
- Please do not carry on side discussions with other students during lecture time – when you have a question, please raise your hand and ask the question so that everyone may benefit from it.
- Also, please try to make sure that your cellular phone and/or pager does not interrupt during lecture time, and especially during exams.

Syllabus - Assessment:

No make up exam for the Quiz !
For assşignments: After 09:40 of the due date, the points received will be multiplied by %70. After the due date, no assignments will be accepted.
For projects: No late submission will be accepted.

Term Learning Activities	Count	Weight %	Contribution to Assesment %
Midterm	1	60	**36**
Quiz	1	10	**6**
Assignments	3	10	**6**
Projects	1	20	**12**
TOTAL		100	**60**
Contribution of Term Learning Activities to Success Grade		60	**60**
Contribution of Final Exam to Success Grade		40	**40**
	TOTAL	**100**	**100**

ROAD MAP

- **Introduction**
  - Definition and Properties of Algorithm
  - Fundamentals of Algorithmic Problem Solving
  - Important Problem Types
- Mathematical Background

Introduction

Algorithms are “methods for solving problems which are suited for computed implementation..” [Sedgewick]

An algorithm is “a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time.” [Aho, Hopcroft, & Ulman]

Introduction

"*Algorithmics* [defined as the study of algorithms -- A.L.] is more than a branch of computer science. <u>It is the core of computer science</u>, and, in all fairness, can be said to be relevant to most of science, business, and technology." [David Harel, "Algorithmics: The Spirit of Computing"]

What is an Algorithm ?

An *algorithm* is a finite, clearly specified sequence of instructions to be followed to solve a problem or compute a function

An *algorithm* generally

- takes some input
- carries out a number of effective instructions in a finite amount of time
- produces some output.

An effective instruction is an operation so basic that it is possible to carry it out using pen and paper.

Donald E. Knuth

Professor Emeritus of The Art of Computer Programming at Stanford University

Knuth has been called the "father" of the analysis of algorithms

A person well-trained in computer science knows how to deal with algorithms: how to construct them, manipulate them, understand them, analyze them. This knowledge is preparation for much more than writing good computer programs; it is a general-purpose mental tool that will be a definite aid to the understanding of other subjects, whether they be chemistry, linguistics, or music, etc. The reason for this may be understood in the following way: It has often been said that a person does not really understand something until after teaching it to someone else. Actually, a person does not *really* understand something until after teaching it to a *computer*, i.e., expressing it as an algorithm . . . An attempt to formalize things as algorithms leads to a much deeper understanding than if we simply try to comprehend things in the traditional way. [Knu96, p. 9]

Two main issues related to algorithms

- How to design algorithms
- How to analyze algorithm efficiency

Expressing Algorithms

Algorithms can be expressed in

- natural languages
  - verbose and ambiguous
  - rarely used for complex or technical algorithms
- pseudocode, flowcharts
  - structured ways to express algorithms
  - avoid ambiguities in natural language statements
  - independent of a particular implementation language
- programming languages
  - intended for expressing algorithms in a form that can be executed by a computer
  - can be used to document algorithms

Example:

Problem: Find the largest number in an (unsorted) list of numbers.

Idea: Look at every number in the list, one at a time.

Natural Language:

- Assume the first item is largest.
- Look at each of the remaining items in the list and if it is larger than the largest item so far, make a note of it.
- The last noted item is the largest in the list when the process is complete.

Example:

Pseudocode:

Algorithm LargestNumber

Input: A non-empty list of numbers L.

Output: The *largest* number in the list L.

largest $\leftarrow L_0$

for each *item* **in** the list $L_{i\geq 1}$, **do**

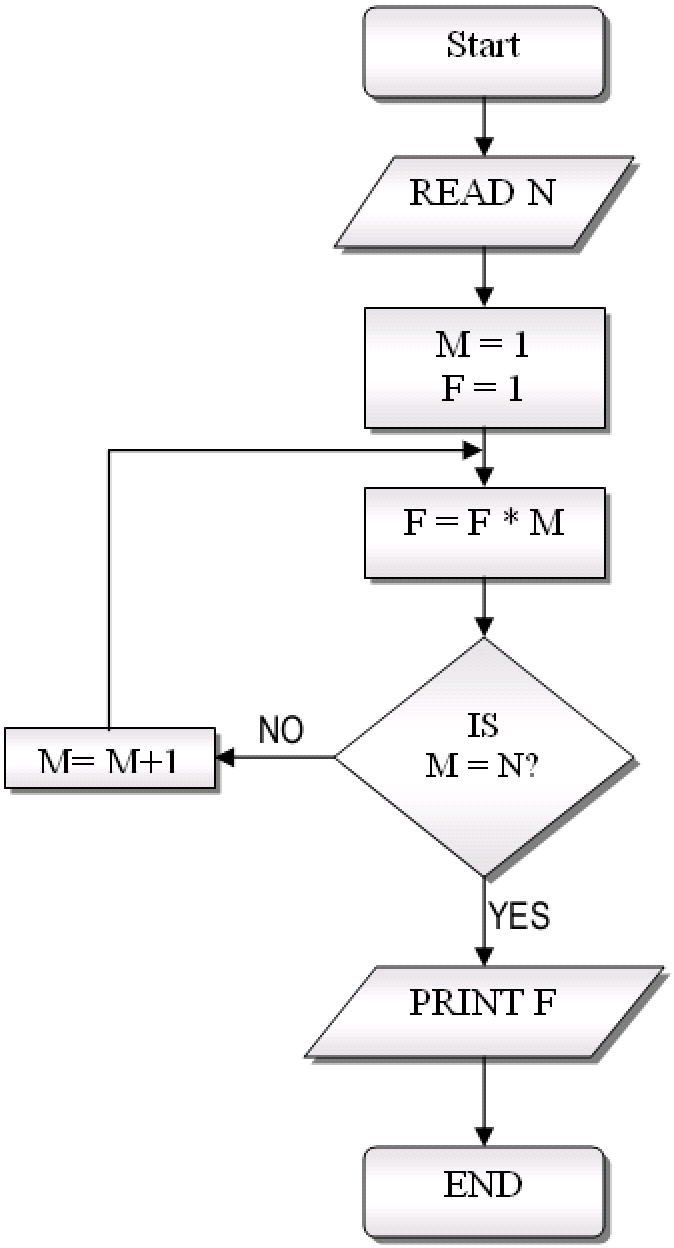
if the *item* > *largest*, **then**

largest $\leftarrow$ the *item*

return *largest*

Example:

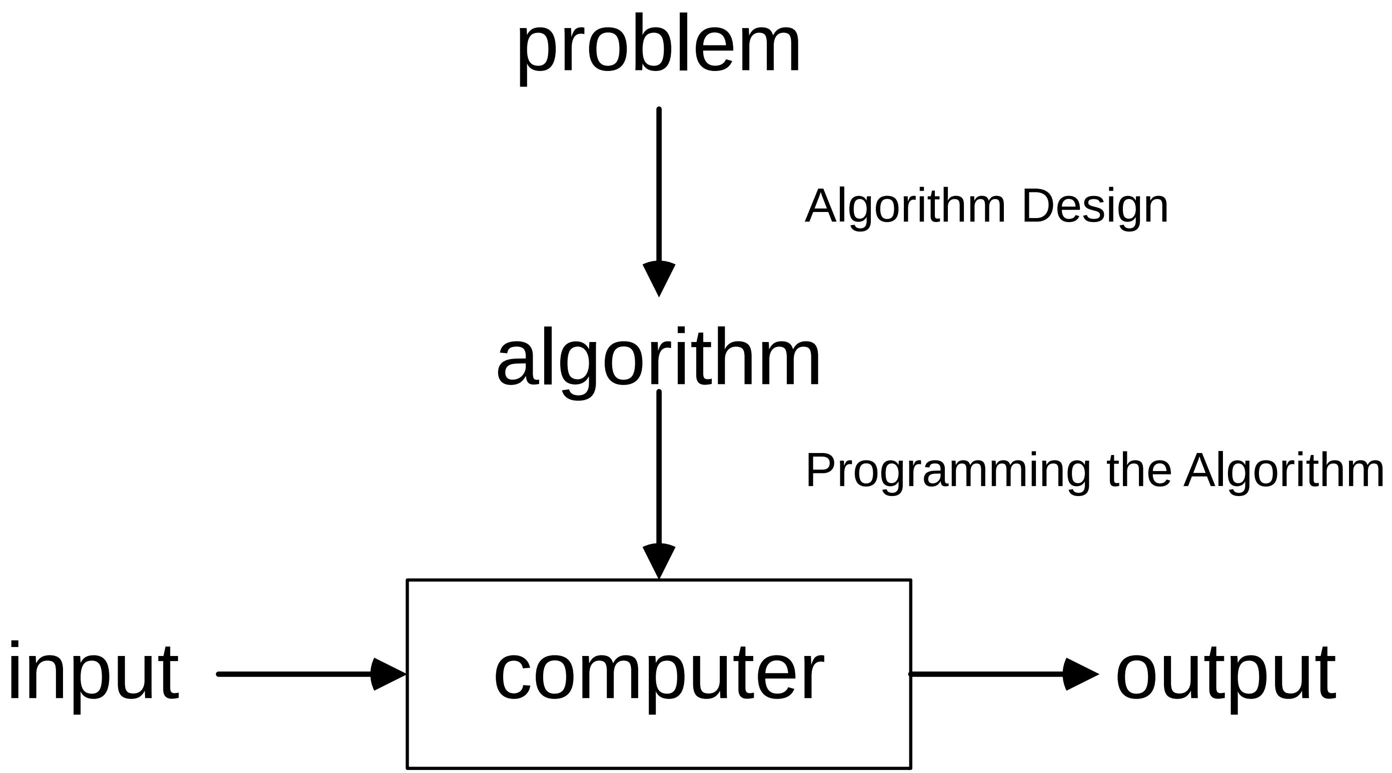
Flowchart:



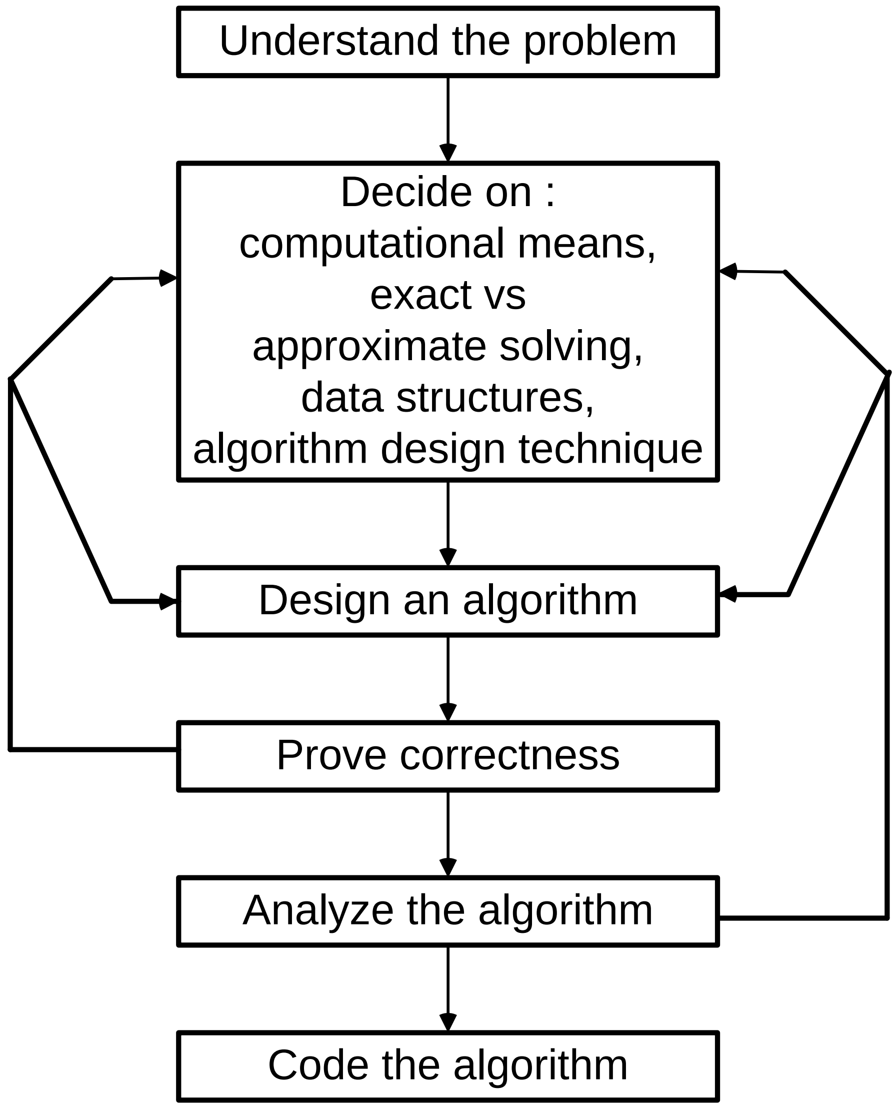
Properties of an Algorithm

- **Effectiveness**
  - Instructions are simple
    - can be carried out by pen and paper
- **Definiteness**
  - Instructions are clear
    - meaning is unique
- **Correctness**
  - Algorithm gives the right answer
    - for all possible cases
- **Finiteness**
  - Algorithm stops in reasonable time
    - produces an output

Notion of an Algorithm



Algorithm Design Process



Deciding on Appropriate Data Structures

Algorithms + Data Structures = Programs

What is an Algorithm ?

An *algorithm* is a finite, clearly specified sequence of instructions to be followed to solve a problem or compute a function

An *algorithm* generally

- takes some input
- carries out a number of effective instructions in a finite amount of time
- produces some output.

An effective instruction is an operation so basic that it is possible to carry it out using pen and paper.

Euclid's Algorithm

- **Problem:** Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n

- Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

- Euclid's algorithm is based on repeated application of equality

  gcd(m,n) = gcd(n, m mod n)
- until the second number becomes 0, which makes the problem trivial.

  Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Structured Description of Euclid's Algorithm

- **Step 1** If *n* = 0, return *m* and stop; otherwise go to Step 2
- **Step 2** Divide *m* by *n* and assign the value to the remainder to *r*
- **Step 3** Assign the value of *n* to *m* and the value of *r* to *n.* Go to Step 1.

Euclid's Algorithm (Pseudocode)

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 $n \neq 0$ **do**
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return m

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

Middle-school procedure for computing gcd(m, n)

Step 1 Find the prime factors of m.

Step 2 Find the prime factors 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*)

$$60 = 2 \times 2 \times 3 \times 5$$

$$24 = 2 \times 2 \times 2 \times 3$$

$$\gcd(60, 24) = 2 \times 2 \times 3 = 12$$

- *Is this an algorithm?*

Sieve of Eratosthenes

- **A simple Algorithm Generating Consecutive Primes Not Exceeding Any Given Integer n: Sieve of Eratosthenes**
- Example:

2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
2	3	x	5	x	7	x	9	x	11	x	13	x	15	x	17	x	19	x	21	x	23	x	**25**
2	**3**		5		7		x		11		13		x		17		19		x		23		**25**
2	**3**		**5**		**7**				**11**		**13**				**17**		**19**				**23**		**x**

Sieve of Eratosthenes

```
ALGORITHM Sieve(n)
    //Implements the sieve of Eratosthenes
    //Input: An integer n ≥ 2
    //Output: Array L of all prime numbers less than or equal to n
    for p ← 2 to n do A[p] ← p
    for p ← 2 to ⌊√n⌋ do    //see note before pseudocode
        if A[p] ≠ 0    //p hasn't been eliminated on previous passes
            j ← p * p
            while j ≤ n do
                A[j] ← 0    //mark element as eliminated
                j ← j + p
    //copy the remaining elements of A to array L of the primes
    i ← 0
    for p ← 2 to n do
        if A[p] ≠ 0
            L[i] ← A[p]
            i ← i + 1
    return L
```

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?
  - time efficiency
  - space efficiency

- 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
- graph
- tree
- set and dictionary

ROAD MAP

- Introduction
  - Definition and Properties of Algorithm
  - Fundamentals of Algorithmic Problem Solving
  - Important Problem Types
- **Mathematical Background**

Mathematical Background

- Functions
- Logarithm
- Summation
- Probability
- Asymptotic Notations
- Recursion
  - Recurrence equation

Properties of Logarithms

1. $\log_a 1 = 0$
2. $\log_a a = 1$
3. $\log_a x^y = y \log_a x$
4. $\log_a xy = \log_a x + \log_a y$
5. $\log_a \dfrac{x}{y} = \log_a x - \log_a y$
6. $a^{\log_b x} = x^{\log_b a}$
7. $\log_a x = \dfrac{\log_b x}{\log_b a} = \log_a b \log_b x$

Important Summation Formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1 + 1 + \cdots + 1}_{u-l+1 \text{ times}} = u - l + 1$ (l, u are integer limits, $l \le u$); $\sum_{i=1}^{n} 1 = n$

2. $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$

3. $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$

4. $\sum_{i=1}^{n} i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1}n^{k+1}$

Important Summation Formulas

5. $\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \ (a \neq 1); \quad \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

6. $\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$

7. $\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma$, where $\gamma \approx 0.5772 \ldots$ (Euler's constant)

8. $\sum_{i=1}^{n} \lg i \approx n \lg n$

Sum Manipulation Rules

1. $$\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$$

2. $$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

3. $$\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i, \text{ where } l \le m < u$$

4. $$\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$$