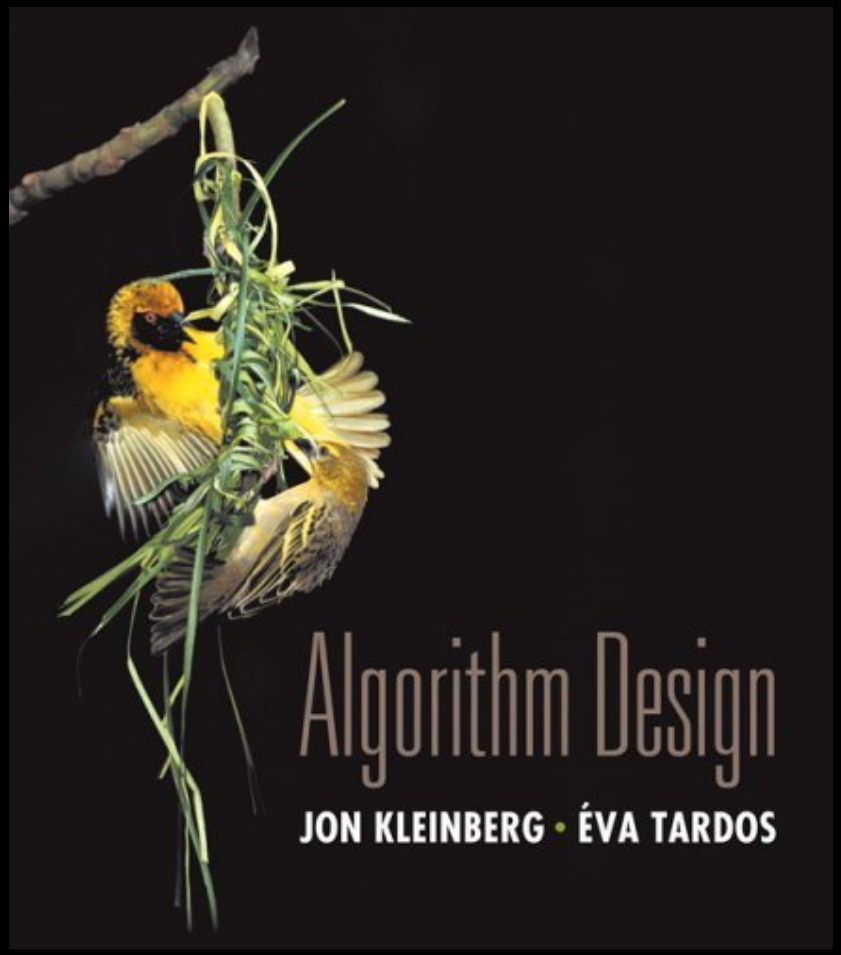



# Chapter 1

## Introduction to the Course

Modified versions of Slides by Kevin Wayne (Princeton)

# Administravia

- ***BLG 336E - Analysis of Algorithms II***

- **Grading, course plan, VF conditions, please see:**
- http://web.itu.edu.tr/~cataltepe/aoa2/index.html

**Reference Book:** - J. Kleinberg and E. Tardos, *Algorithm Design*, Addison Wesley, 2006.

**Other References:**

-*Fundamentals of Algorithmics*,Brassards and Bratley, Prentice Hall (Available at the **Central Library**, QA9.58.B73 1996).

-*Introduction to Algorithms*, Cormen, Leiserson and Rivest, The MIT Pres/McGraw-Hill.

- *Algorithms and Complexity* , Wilf. You may download free from www.upenn.edu.

# Please Note that

- **Programming Projects:** Will be in C++. There will be explicit instructions on which files to submit, how it should be compiled, example cases etc. Some examples will be provided so that you can test your program yourselves. However, your program will be graded based on how good it is in solving the given problem.
- **A project that does not compile or does not provide the necessary files will get zero, no exceptions! Projects not submitted through the online submission system will not get accepted, even if they are a couple of minutes late. So, please submit to ninova ahead of the due second.**
- Plagiarism (copying (parts) of projects **from each other or internet resources**) is strictly discouraged. You may discuss the problem with your classmates, but you must write your own code. You must not take somebody else's code and modify it and submit it. If we realize that you did copy, you will get zero from that project, you may get an FF from the course and the Student Affairs Office will be informed.

## Chapter1 Contents (Week 1 and 2)

- What is an Algorithm
- Algorithmic Paradigms:
  - Greedy.
  - Divide-and-conquer.
  - Dynamic programming.
  - Network flow.
  - Intractability.
- Stable Matching Problem
  - Gale-Shapley (Propose and Reject) Algorithm

# Algorithms

## Algorithm.

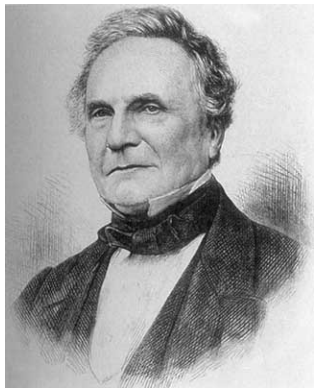
- [webster.com] A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.

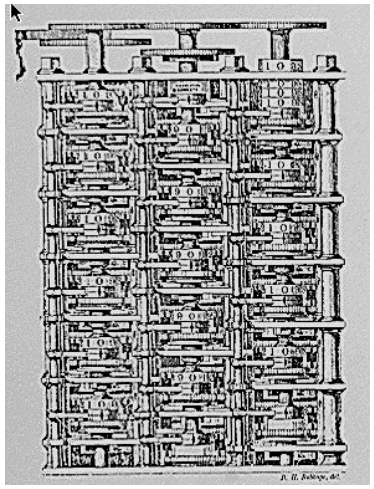
- [Knuth, TAOCP] An algorithm is a finite, definite, effective procedure, with some input and some output.

> Great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing. - *Francis Sullivan*

# Theory of Algorithms

"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the *shortest time*? - *Charles Babbage*





# Algorithmic Paradigms

Design and analysis of computer algorithms.

- Greedy
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Intractability.
- Coping with intractability.

Critical thinking and problem-solving.

# Applications

Wide range of applications.

- Caching.
- Compilers.
- Databases.
- Scheduling.
- Networking.
- Data analysis.
- Signal processing.
- Computer graphics.
- Scientific computing.
- Operations research.
- Artificial intelligence.
- Computational biology.
- . . .

We focus on algorithms and techniques that are useful in practice.

# Contents

# Introduction, Some representative problems

Stable matching problem

- How to prove the number of iterations of the algorithm
- How to prove that the solution returned is correct (stable)

Five representative problems:

- Interval Scheduling (Greedy Algorithm)
- Weighted Interval Scheduling (Dynamic Programming)
- Bipartite Matching (network flow problems, augmentation)
- Independent Set (NP Complete problem, can check a given solution in linear time, finding soln takes a lot of time)
- Competitive Facility Location (PSPACE-complete poblem)

# Basics of Algorithm Analysis

Computational Tractability = running efficiently=in polynomial time

Asymptotic Order of Growth ($O()$, $\Omega()$, $\Theta()$)

Implementation of Stable Matching Problem Using Arrays and Lists

Survey of Common Running Times (Linear, nlogn, Quadratic, Cubic, $O(n^k)$, Beyond Polynomial, Sublinear)

Implementation of Stable Matching Problem Using Priority Queues (Heap)

# Graphs

Definition and Applications (Transportation networks, communication networks, information networks, social networks, dependency networks)

Paths and connectivity, trees

Graph Connectivity and Graph Traversal (Breadth-First (BFS), Depth First Search (DFS))

Implementation using Queues and Stacks

Testing bipartideness: An Application of BFS

Directed Acyclic Graphs, Topological Ordering

# Greedy Algorithms

An algorithm that builds up a solution in small steps, choosing a decision at each step myopically to optimize some underlying criterion.

Interval Scheduling: Design, Analysis (How to prove that a greedy algorithm produces an optimal solution)

Scheduling all intervals

Scheduling to minimize lateness

//Optimal Caching

Shortest Paths in a Graph (Dijksta)

Minimum(-cost) spanning trees (Kruskal, Prim)

Implementation of Kruskal's Algo: Union-Find Data Structure

//?Clustering

//?Huffman Codes and Data Compression

# Divide and Conquer

An algorithm that breaks up the problem into smaller problems, solves each part separately and then combines them. Recurrence relation.

Mergesort

Recurrences

Divide and Conquer Applications

- Counting inversions (collaborative filtering)
- Finding the closest pair of points
- Integer Multiplication
- //?Convolutions and the FFT

# Dynamic Programming

Drawn from the intuition behind divide and conquer and is opposite of greedy strategy. Explore space of all possible solutions by carefully decomposing things into a series of subproblems. Then build up solutions to larger and larger subproblems. Dynamic programming is better than brute force search. Does not explore all possibilites but only the ones it needs to explore.

Weighted Interval Scheduling

Principles of Dynamic Programming

Segmented Least Squares

Subset sums and knapsacks

//RNA secondary structure

//?Sequence Alignment

Shortest Paths in a Graph

# Network Flow

Bipartite matching is a special case, but there are many diverse applications.

Max flow problem and Ford-Fulkerson algorithm

Maximum flows and minimum cuts

Choosing good augmenting paths

A first application: bipartite matching

Disjoint paths in directed and undirected graphs

Extensions to max flow problem

//Survey Design

//Image segmentation

//Project selection

//Baseball elimination

# NP and Computational Intractability

NP complete problems: A large set of problems for which we do not know an efficient (polynomial time) solution. Once a solution is given though, we can check if it is correct in polynomial time.

Polynomial time reductions (of one problem to another)

Satisfiability problem reductions

Definition of NP.

NP Complete problems

//?Sequencing problems (TSP, Hamiltonian Cycle, Partitioning problems, Graph coloring, numerical problems)

//Co-NP and Asymmetry of NP

//Partial Taxonomy of Hard Problems