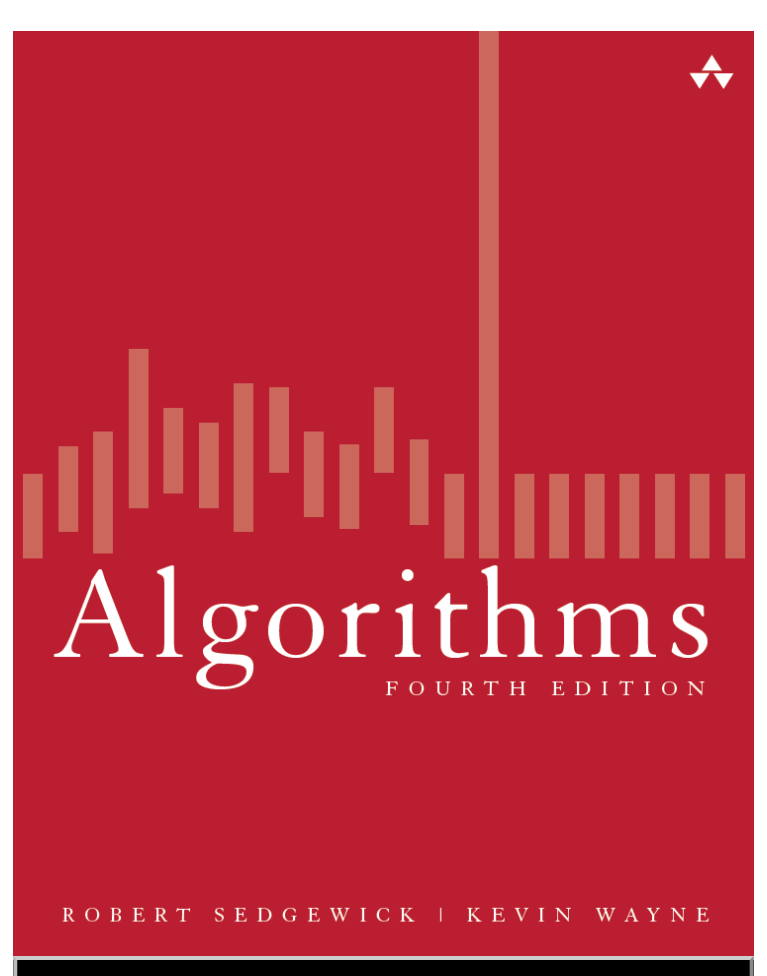

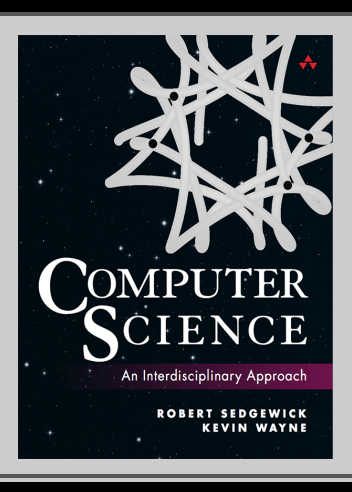

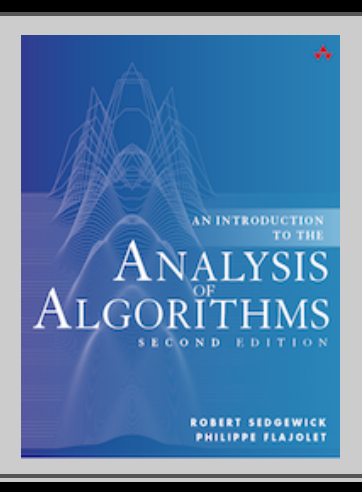

Lectures

This page provides information about online lectures and lecture slides for use in teaching and learning from the book *Algorithms, 4/e*. These lectures are appropriate for use by instructors as the basis for a "flipped" class on the subject, or for self-study by individuals.

Flipped classroom. If you are an an instructor teaching introductory computer science, an effective way for you to teach the material in a typical college class is to adhere to a weekly cadence, as follows:

- Each week, send an email to all students in the class that briefly describes activities for that week (lectures, reading, and programming assignments drawn from the book or from this booksite).
- Students watch the lecture videos at their own pace, do the readings, and work on the programming assignments.
- Schedule a weekly "class meeting" for discussion of the material, reviews for exams, informal interaction with students, and any enrichment material you may wish to cover.

This is just one suggestion—this material can support many different teaching styles and formats.

Important note: A common mistake in teaching a flipped class is to add too much enrichment material. Our experience is that time in class meetings is much better spent preparing students for success on programming assignments and exams. If an instructor makes it clear that the best way to prepare for exams is to watch the lecture videos and do the reading, most students will do so. Class meetings then can involve interacting with students and with the material in such a way as to reinforce understanding. For example, working with potential exam questions is an excellent activity.

Self-study. An effective way to learn the material on your own is to watch the lecture videos on some regular schedule, do the associated reading, and attempt to solve some of the exercises in the book or on the booksite on your own. If you get stuck on a particular exercise, find some others or try to solve some of the problems given in the lectures without looking at the solutions there.

Available lectures. The lecture videos are available by subscription from CUvids; the lecture slides are freely available in pdf format. When watching a lecture video, it is *very important* to choose an appropriate speed. If it is too slow, you are likely to be bored; if it is too fast, you are likely to get lost. Also be sure to make liberal use of pause and rewind.

- **Lecture 1: Union–Find.** We illustrate our basic approach to developing and analyzing algorithms by considering the dynamic connectivity problem. We introduce the union–find data type and consider several implementations (quick find, quick union, weighted quick union, and weighted quick union with path compression). Finally, we apply the union–find data type to the percolation problem from physical chemistry.
- **Lecture 2: Analysis of Algorithms.** The basis of our approach for analyzing the performance of algorithms is the scientific method. We begin by performing computational experiments to measure the running times of our programs. We use these measurements to develop hypotheses about performance. Next, we create mathematical models to explain their behavior. Finally, we consider analyzing the memory usage of our Java programs.
- **Lecture 3: Stacks and Queues.** We consider two fundamental data types for storing collections of objects: the stack and the queue. We implement each using either a singly linked list or a resizing array. We introduce two advanced Java features—generics and iterators—that simplify client code. Finally, we consider various applications of stacks and queues ranging from parsing arithmetic expressions to simulating queueing systems.
- **Lecture 4: Elementary Sorts.** We introduce the sorting problem and Java's Comparable interface. We study two elementary sorting methods (selection sort and insertion sort) and a variation of one of them (shellsort). We also consider two algorithms for uniformly shuffling an array. We conclude with an application of sorting to computing the convex hull via the Graham scan algorithm.
- **Lecture 5: Mergesort.** We study the mergesort algorithm and show that it guarantees to sort any array of n items with at most $n \log_2 n$ compares. We also consider a nonrecursive, bottom-up version. We prove that any compare-based sorting algorithm must make at least $\sim n \log_2 n$ compares in the worst case. We discuss using different orderings for the objects that we are sorting and the related concept of stability.
- **Lecture 6: Quicksort.** We introduce and implement the randomized quicksort algorithm and analyze its performance. We also consider randomized quickselect, a quicksort variant which finds the kth smallest item in linear time. Finally, we consider 3-way quicksort, a variant of quicksort that works especially well in the presence of duplicate keys.
- **Lecture 7: Priority Queues.** We introduce the priority queue data type and an efficient implementation using the binary heap data structure. This implementation also leads to an efficient sorting algorithm known as heapsort. We conclude with an applications of priority queues where we simulate the motion of n particles subject to the laws of elastic collision.
- **Lecture 8: Elementary Symbol Tables and BSTs.** We define an API for symbol tables (also known as associative arrays) and describe two elementary implementations using a sorted array (binary search) and an unordered list (sequential search). When the keys are Comparable, we define an extended API that includes the additional methods min, max, floor, ceiling, rank, and select. To develop an efficient implementation of this API, we study the binary search tree data structure and analyze its performance.
- **Lecture 9: Balanced Search Trees.** In this lecture, our goal is to develop a symbol table with guaranteed logarithmic performance for search and insert (and many other operations). We begin with 2–3 trees, which are easy to analyze but hard to implement. Next, we consider red–black binary search trees, which we view as a novel way to implement 2–3 trees as binary search trees. Finally, we introduce B-trees, a generalization of 2–3 trees that are widely used to implement file systems
- **Lecture 10: Geometric Applications of BSTs.** We start with 1d and 2d range searching, where the goal is to find all points in a given 1d or 2d interval. To accomplish this, we consider k-d trees, a natural generalization of BSTs when the keys are points in the plane (or higher dimensions). We also consider intersection problems, where the goal is to find all intersections among a set of line segments or rectangles.
- **Lecture 11: Hash Tables.** We begin by describing the desirable properties of hash function and how to implement them in Java, including a fundamental tenet known as the uniform hashing assumption that underlies the potential success of a hashing application. Then, we consider two strategies for implementing hash tables—separate chaining and linear probing. Both strategies yield constant-time performance for search and insert under the uniform hashing assumption. We conclude with applications of symbol tables including sets, dictionary clients, indexing clients, and sparse vectors.
- **Lecture 12: Undirected Graphs.** We define an undirected graph API and consider the adjacency-matrix and adjacency-lists representations. We introduce two classic algorithms for searching a graph—depth-first search and breadth-first search. We also consider the problem of computing connected components and conclude with related problems and applications.
- **Lecture 13: Directed Graphs.** In this lecture we study directed graphs. We begin with depth-first search and breadth-first search in digraphs and describe applications ranging from garbage collection to web crawling. Next, we introduce a depth-first search based algorithm for computing the topological order of an acyclic digraph. Finally, we implement the Kosaraju–Sharir algorithm for computing the strong components of a digraph.
- **Lecture 14: Minimum Spanning Trees.** In this lecture we study the minimum spanning tree problem. We begin by considering a generic greedy algorithm for the problem. Next, we consider and implement two classic algorithm for the problem—Kruskal's algorithm and Prim's algorithm. We conclude with some applications and open problems.
- **Lecture 15: Shortest Paths.** In this lecture we study shortest-paths problems. We begin by analyzing some basic properties of shortest paths and a generic algorithm for the problem. We introduce and analyze Dijkstra's algorithm for shortest-paths problems with nonnegative weights. Next, we consider an even faster algorithm for DAGs, which works even if the weights are negative. We conclude with the Bellman–Ford–Moore algorithm for edge-weighted digraphs with no negative cycles. We also consider applications ranging from content-aware fill to arbitrage.
- **Lecture 16: Maximum Flow.** In this lecture we introduce the maximum flow and minimum cut problems. We begin with the Ford–Fulkerson algorithm. To analyze its correctness, we establish the maxflow–mincut theorem. Next, we consider an efficient implementation of the Ford–Fulkerson algorithm, using the shortest augmenting path rule. Finally, we consider applications, including bipartite matching and baseball elimination.
- **Lecture 17: String Sorts.** In this lecture we consider specialized sorting algorithms for strings and related objects. We begin with a subroutine to sort integers in a small range. We then consider two classic radix sorting algorithms—LSD and MSD radix sorts. Next, we consider an especially efficient variant, which is a hybrid of MSD radix sort and quicksort known as 3-way radix quicksort. We conclude with suffix sorting and related applications.
- **Lecture 18: Tries.** In this lecture we consider specialized algorithms for symbol tables with string keys. Our goal is a data structure that is as fast as hashing and even more flexible than binary search trees. We begin with multiway tries; next we consider ternary search tries. Finally, we consider character-based operations, including prefix match and longest prefix, and related applications.
- **Lecture 19: Substring Search.** In this lecture we consider algorithms for searching for a substring in a piece of text. We begin with a brute-force algorithm, whose running time is quadratic in the worst case. Next, we consider the ingenious Knuth–Morris–Pratt algorithm whose running time is guaranteed to be linear in the worst case. Then, we introduce the Boyer–Moore algorithm, whose running time is sublinear on typical inputs. Finally, we consider the Rabin–Karp fingerprint algorithm, which uses hashing in a clever way to solve the substring search and related problems.
- **Lecture 20: Regular Expressions.** A regular expression is a method for specifying a set of strings. Our topic for this lecture is the famous grep algorithm that determines whether a given text contains any substring from the set. We examine an efficient implementation that makes use of digraph reachability.
- **Lecture 21: Data Compression.** We study and implement several classic data compression schemes, including run-length coding, Huffman compression, and LZW compression. We develop efficient implementations from first principles using a Java library for manipulating binary data that we developed for this purpose, based on priority queue and symbol table implementations from earlier lectures.
- **Lecture 22: Reductions.** In this lecture our goal is to develop ways to classify problems according to their computational requirements. We introduce the concept of reduction as a technique for studying the relationship among problems. People use reductions to design algorithms, establish lower bounds, and classify problems in terms of their computational requirements.
- **Lecture 23: Linear Programming.** The quintessential problem-solving model is known as linear programming, and the simplex method for solving it is one of the most widely used algorithms. In this lecture, we given an overview of this central topic in operations research and describe its relationship to algorithms that we have considered.

Table of lectures. Here are lecture slides and demos.

#	TOPIC	DEMOS
0	Introduction	–
1.1	Programming Model	–
1.2	Data Abstraction	–
1.3	Stacks and Queues	Dijkstra 2-stack
1.4	Analysis of Algorithms	Binary search
1.5	Case Study: Union Find	Quick-find Quick-union Weighted
2.1	Elementary Sorts	Selection Insertion h-sorting Knuth shuffle
2.2	Mergesort	Merging
2.3	Quicksort	Partitioning Quick-select
2.4	Priority Queues	Heap Heapsort
3.1	Elementary Symbol Tables	–
3.2	Binary Search Trees	BST
3.3	Balanced Search Trees	2–3 tree Red–black BST GrowingTree
–	Geometric Applications of BSTs	Kd tree Interval search tree
3.4	Hash Tables	Separate chaining Linear probing
3.5	Searching Applications	–
4.1	Undirected Graphs	DFS BFS Connected components
4.2	Directed Graphs	DFS BFS Topological sort Kosaraju–Sharir
4.3	Minimum Spanning Trees	Greedy Kruskal Prim
4.4	Shortest Paths	Dijkstra Acyclic Bellman–Ford
5.1	String Sorts	Key-indexed counting
5.2	Tries	Trie TST
5.3	Substring Search	KMP
5.4	Regular Expressions	NFA simulation NFA construction
5.5	Data Compression	Huffman LZW
6.4	Maximum Flow	Ford–Fulkerson
–	Linear Programming	Simplex

Last modified on November 03, 2020.