CSE 100: Algorithm Design and Analysis
Course Overview (Ch01) and Administrivia

Sungjin Im

University of California, Merced

Last Update: 1-17-2024

The greatest pleasure in life is doing what people say you cannot do.

Walter Bagehot

Outline

1. Personnel (instructor and TAs)
2. Course Overview
3. Administravia

Personnel

See the Syllabus on Catcourses.

What is Algorithm?

An algorithm is

What is Algorithm?

An algorithm is a *well-defined computational procedure* that takes an *input* and produces a desired *output* to solve a *well-specified* problem.

What is Algorithm?

An algorithm is a *well-defined computational procedure* that takes an *input* and produces a desired *output* to solve a *well-specified* problem.
An algorithm is not necessarily a program... Not necessarily tied to a specific programming language... It is an 'abstract' computational procedure...

Why do you care about algorithms rather than actual codes?

What is Algorithm?

An algorithm is a *well-defined computational procedure* that takes an *input* and produces a desired *output* to solve a *well-specified* problem.
An algorithm is not necessarily a program... Not necessarily tied to a specific programming language... It is an 'abstract' computational procedure...

Why do you care about algorithms rather than actual codes?

- Algorithms help abstract away details related to each language or system and focus on key procedural ideas. For example, in the algorithmic world, we can create an arbitrarily large array without worrying about RAM size.
- Algorithms can be readily turned into actual codes.

An Example of Well-specified Problem

The Sorting Problem:

Input: A sequence of n numbers $\langle a_1, a_2, \cdots, a_n \rangle$.
Output: A permutation (reordering) $\langle a'_1, a'_2, \cdots, a'_n \rangle$ of the input sequence s.t. $a'_1 \leq a'_2 \leq a'_3 \leq \cdots \leq a'_n$.

Input is often referred to as instance.

Say the instance is 5, 2, 4, 6, 1, 3.
Then, the desired output is 1, 2, 3, 4, 5, 6

Algorithms for the Sorting Problem

Insertion Sort

The Sorting Problem:

Instance: 5, 2, 4, 6, 1, 3.

Algorithms for the Sorting Problem

Insertion Sort

The Sorting Problem:

Instance: 5, 2, 4, 6, 1, 3.

An idea (Insertion Sort): Let's first sort the first number.

Add the 2nd number to get the sorted order of the first two numbers.

Add the 3rd number to get the sorted order of the first three numbers.

...

Algorithms for the Sorting Problem

Insertion Sort

The Sorting Problem:

Instance: 5, 2, 4, 6, 1, 3.
An idea (Insertion Sort): Let's first sort the first number.
Add the 2nd number to get the sorted order of the first two numbers.
Add the 3rd number to get the sorted order of the first three numbers.

...

Execution:
(5)
(2) 5
2 (4) 5
2 4 5 (6)
(1) 2 4 5 6

1 2 (3) 4 5 6

Algorithms for the Sorting Problem

Insertion Sort

The Sorting Problem:

Instance: 5, 2, 4, 6, 1, 3.
An idea (Insertion Sort): Let's first sort the first number.
Add the 2nd number to get the sorted order of the first two numbers.
Add the 3rd number to get the sorted order of the first three numbers.
...
This is an algorithmic idea (80-90% complete solution), but not sufficiently formal to be said to be an algorithm.

Algorithms for the Sorting Problem

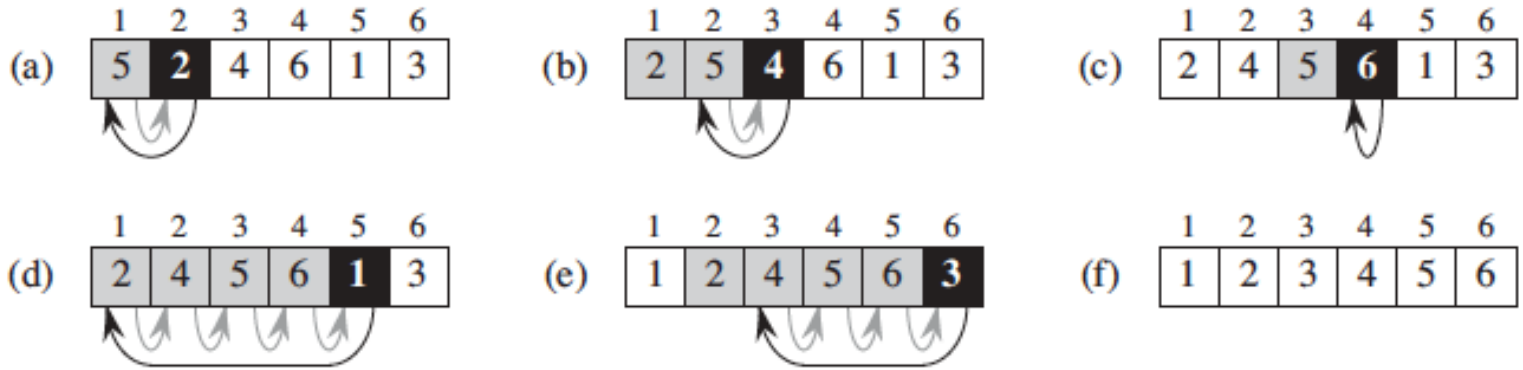
Insertion Sort

The Sorting Problem:

Instance: 5, 2, 4, 6, 1, 3.
An idea (Insertion Sort): Let's first sort the first number.
Add the 2nd number to get the sorted order of the first two numbers.
Add the 3rd number to get the sorted order of the first three numbers.
...
This is an algorithmic idea (80-90% complete solution), but not sufficiently formal to be said to be an algorithm. Data structure?
How do we "insert" the considered number?

Insertion sort

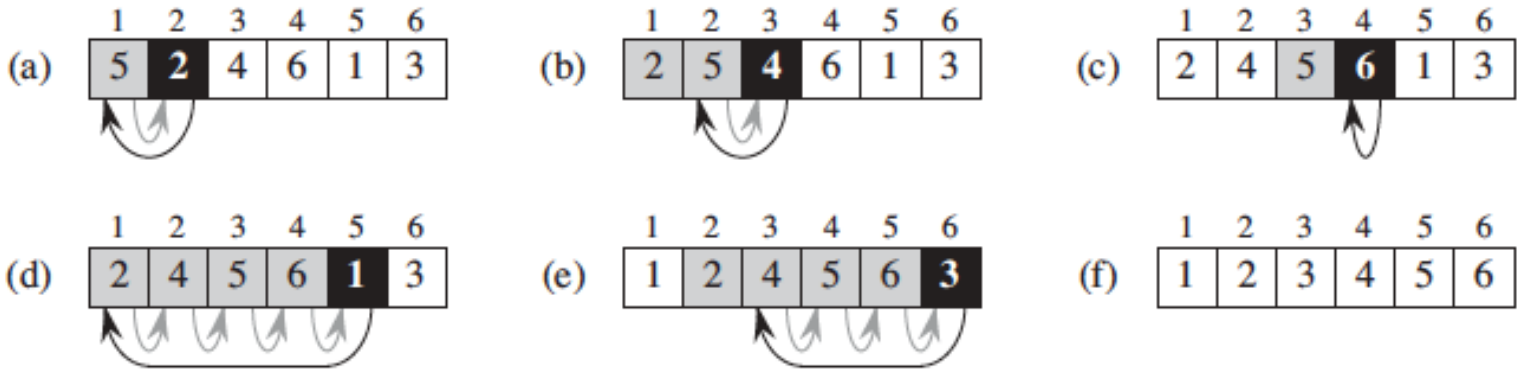
Assume that the input is given as an array $A[1 \cdots n]$.



How to describe an algorithm?

By plain English

Insertion Sort:
In the kth iteration ($1 \leq k \leq n$), we ensure that $A[1 \cdots k]$ is sorted by inserting the kth number in the the 'right' position of $A[1 \cdots k-1]$. Here we find the position by scanning the array $A[1 \cdots k-1]$ *sequentially* backward and pushing back elements greater than the kth element, one by one.
(Remark: You can abstract something if it doesn't affect the asymptotic running time of the algorithm.)



How to describe an algorithm?

By plain English

Insertion Sort:
In the kth iteration ($1 \leq k \leq n$), we ensure that $A[1 \cdots k]$ is sorted by inserting the kth number in the the 'right' position of $A[1 \cdots k-1]$. Here we find the position by scanning the array $A[1 \cdots k-1]$ *sequentially* backward and pushing back elements greater than the kth element, one by one.
(Remark: You can abstract something if it doesn't affect the asymptotic running time of the algorithm.)
This is also acceptable: In the kth iteration ($1 \leq k \leq n$), we ensure that $A[1 \cdots k]$ is sorted by inserting the kth number in the the 'right' position of $A[1 \cdots k-1]$ that is already sorted. Here we find the position by linear scanning and pushing back some elements one by one.

How to describe an algorithm?

By Pseudocode

Insertion Sort:

Input: $A[1 \cdots n]$

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted
           sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

How to describe an algorithm?

By plain English or pseudocode

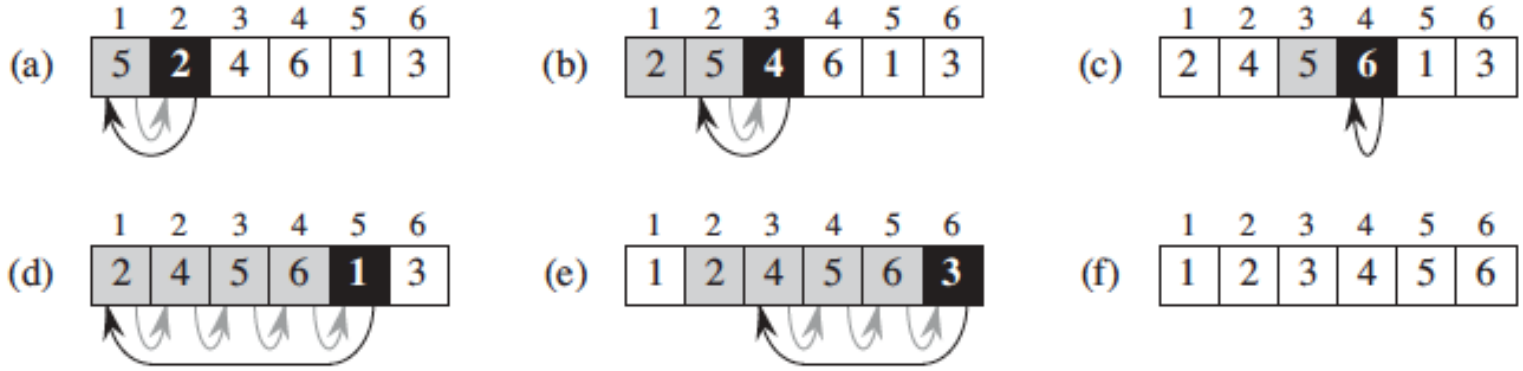
You can describe your algorithm by either plain English or pseudocode — or their combination. Your code should be sufficiently formal. Conciseness will be a plus though.

Common mistakes:

1. In this example, the algorithm works as follows... The algorithm works like this for all inputs. Giving an example helps sometimes but it cannot replace your algorithm description. It should be complete.
2. Missing data structures.

Insertion sort: Running Time

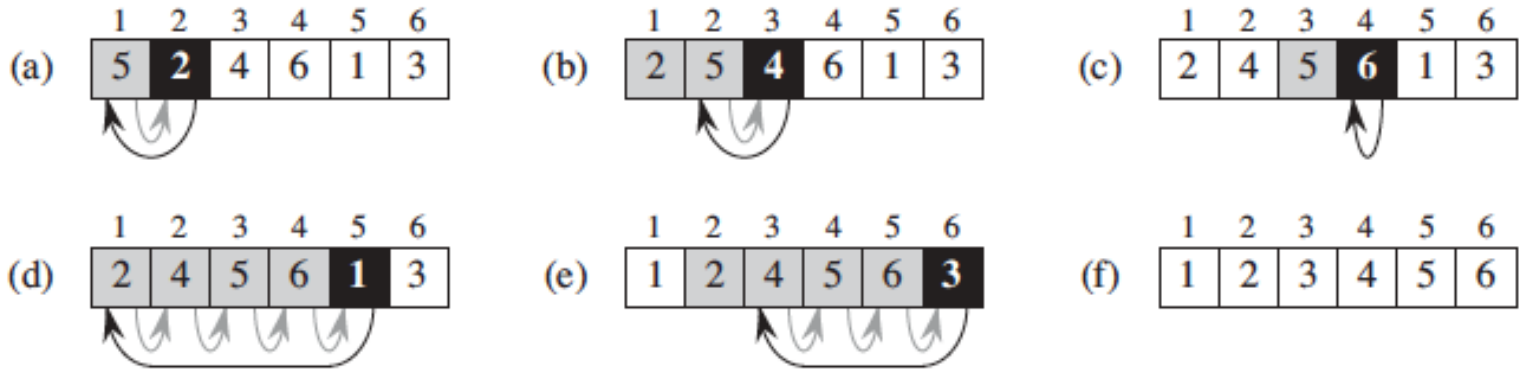
Assume that the Input is given as an array $A[1 \cdots n]$.



Running time:

Insertion sort: Running Time

Assume that the Input is given as an array $A[1 \cdots n]$.



Running time: Roughly speaking, at most n^2 within a constant factor. ($O(n^2)$).
(No worries. We will revisit the asymptotic running time later).
(Later, we will learn $O(n \log n)$-time sorting algorithms).

Compare $O(n^2)$ vs. $O(n \log n)$

Who is the winner?

Criteria for Good Algorithms

Criteria for Good Algorithms

- Correctness

Criteria for Good Algorithms

- Correctness
- (Asymptotic) Running Time

Criteria for Good Algorithms

- Correctness
- (Asymptotic) Running Time
- (Asymptotic) Memory Usage

Criteria for Good Algorithms

- Correctness
- (Asymptotic) Running Time
- (Asymptotic) Memory Usage
- Easy-to-implement

Criteria for Good Algorithms

- Correctness
- (Asymptotic) Running Time
- (Asymptotic) Memory Usage
- Easy-to-implement

Applications of Algorithms

Various Applications

- Finding the Shortest Path.
- Electronic Commerce. Public-key cryptography.
- Human Genome Sequencing

Why Should We Care?

Useful and Fundamental. Found in all areas.

- CSE 150 (operating systems) scheduling algorithms and efficient data structures.
- CSE 176 (machine learning) fast geometric algorithms and similarity search.
- CSE 178 (cryptography) fast number theoretic and algebraic algorithms.
- CSE 111 (databases) query optimization, join selection, recovery and isolation algorithms.
- CSE 160 (networking) uses shortest-path algorithms, spanning tree, diverse routing algorithms.
- CSE 173 (computational neuroscience) algorithms that operate on strings often employs the dynamic programming paradigm.

And helps you get a job.

Why Should We Care?

Algorithmic Ideas Never Die and They are Beautiful.

Why Should We Care?

Algorithmic Ideas Never Die and They are Beautiful. Today's

irrelevant problem might be tomorrow's multibillion-dollar industry.

Bruce Maggs
Professor at Duke University
Vice President, Research, Akamai Technologies

(Akamai was co-founded by Dan Lewin and Tom Leighton.)

Course goals

- The design and analysis of algorithms
  - These go hand-in-hand

In this course you will:

- Learn how to think analytically about algorithms
- Learn how to flesh out an "algorithmic toolkit"
- Learn how to communicate clearly about algorithms

Topics to be Covered

subject to changes

1. Getting started: How to describe algorithms. How to prove the correctness (loop invariants/inductions). Asymptotic notation. Divide-and-conquer. Recurrent equations and the master theorem.
2. (More) Sorting, Selection, Data Structures: Quicksort, Heapsort, etc. Sorting lower bounds. Heaps. Binary search trees. Hashing.
3. Dynamic programming and greedy algorithms.
4. Graph algorithms: BFS/DFS, Topological sort, Shortest path, Minimum spanning trees, Max flow and min cut.

Administravia

Course Webpage

https://catcourses.ucmerced.edu/courses/30623
In particular, Syllabus will serve as the Portal. So, check it frequently.

CSE 100 in the CSE curriculum

A core course. You must get a grade C- or higher to graduate. A prerequisite of many upper division CSE courses.

Prerequisites:
CSE 030: Data Structures
MATH 032: Probability and Statistics

Prerequisites with a Concurrent Option:
CSE 031: Computer Organization
MATH 024: Linear Algebra and Differential Equations

Schedule

Lectures: 4:30-5:45pm, Thu and Fri

Labs:
In each lab,

- One half (1.15 hrs): Discussion session. Discussing problems. Not a lecture. You're expected to go over the problem before you come.
- The other half (1.15 hrs): Programming assignments. Kind of OH to help you with the assignments.

Asking questions online

If you have questions, please first consider to ask TAs unless you have good reasons that you believe you should contact me directly (you can always Cc me). We will try to reply within 24 business hours.

Lecture Slides

Lecture slides will be uploaded to CatCourse within 24 hours after class. If you don't find lecture slides after 24 hours, please let the instructor know.

Textbook

T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein: Introduction to Algorithms, 3rd ed. MIT Press, 2009.

Other recommended books:

- J. Kleinberg and E. Tardos: Algorithm Design, Addison-Wesley, 2005. Companion site for the book.
- S. S. Skiena: The Algorithm Design Manual, 2nd ed. Springer, 2008. This book is accessible online from within UC Merced.
- S. Dasgupta, C. H. Papadimitriou and U. V. Vazirani: Algorithms, McGraw-Hill, 2006.
- Donald Knuth: The Art of Computer Programming.
- A. V. Aho, J. E. Hopcroft and J. D. Ullman: Data Structures and Algorithms, Prentice-Hall, 1983.

Grading

- Programming Assignments (Lab): 20%
- Midterms 1, 2, 3: 18% each. During class time. Not cumulative.
- Final: 26% Cumulative.
  For the date/time of exams, see the course syllabus.

* All exams in classroom.
* Note. No written assignments. Attendance has no direct effect on grades — active participants will be given some bonus points, though.

Exams and Course Objectives

Three types of knowledge

- Type 1: Basic algorithmic questions. eg. run a certain algorithm on a given input.
- Type 2: Implementation level. eg. provide or correct the code.
- Type 3: Application level. eg. applying algorithmic principles to solve new problems.

Exams and Course Objectives

Three types of knowledge

- Type 1: Basic algorithmic questions. eg. run a certain algorithm on a given input.
- Type 2: Implementation level. eg. provide or correct the code.
- Type 3: Application level. eg. applying algorithmic principles to solve new problems.

Related CSE Course Objectives:

- A: An ability to apply knowledge of computing and mathematics appropriate to the discipline;
- H: An ability to apply mathematical foundations, algorithmic principles, and computer science theory to the modeling and design of computer based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.

Exams

Expectation

- Type 1: Basic algorithmic questions. eg. run a certain algorithm on a given input. A: 90% or higher; B: 80% or higher; C: 70% or higher.
- Type 2: Implementation level. eg. provide or correct the code. A: 70% or higher; B: 60% or higher; C: 50% or higher.
- Type 3: Application level. eg. applying algorithmic principles to solve new problems. A: 50% or higher; B: 30% or higher; C: 10% or higher.

Type 1: 60-70%, Type 2: 20-30%, Type 3: 10-15%

Letter Grades

Based on the percentage of problems of each type and your scorese. Some bonus points will be given to students who actively participate in discussion during class or labs.