Welcome to CS-250: Algorithms!

Alessandro Chiesa, Ola Svensson

EPFL

School of Computer and Communication Sciences

Lecture 1, 18.02.2025

Slides inspired by

http://www.cs.princeton.edu/courses/archive/fall13/cos226/lectures/00Intro+15UnionFind.pdf

CS-250 course overview

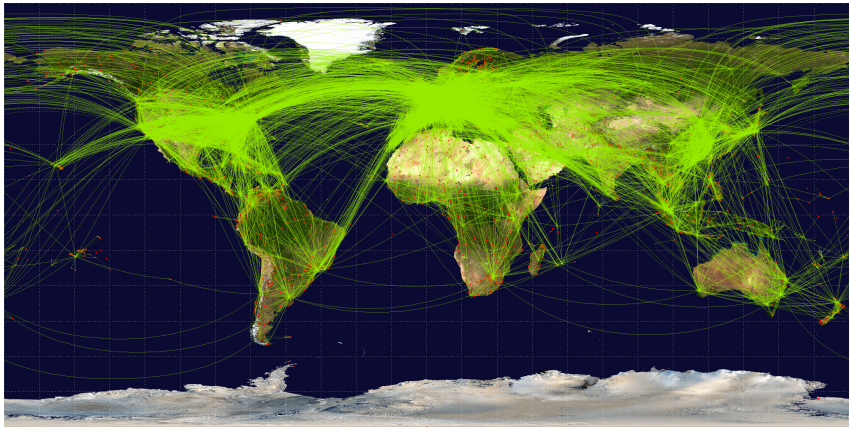
What is CS-250?

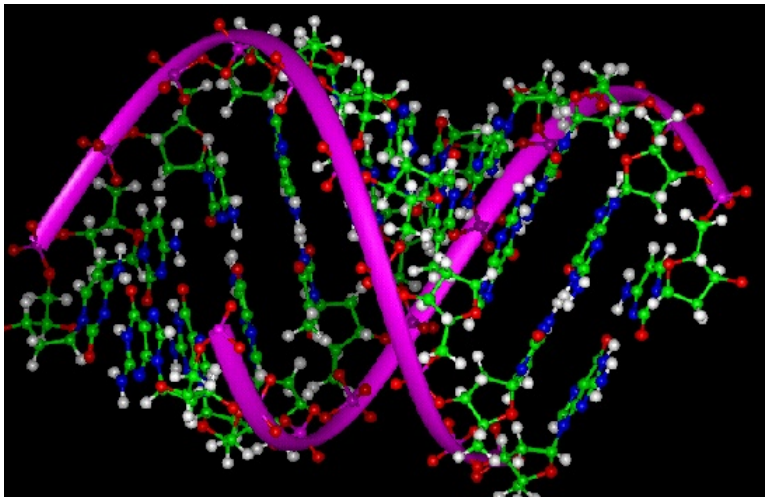
- Bachelor-intermediate level course
- Survey of tools and their application to problem solving
- Algorithm: method for solving a problem
- Data structure: method to store information

data types	lists, stack, queue, union-find, priority queue
sorting	insertion sort, mergesort, heapsort, quicksort
searching	binary search, BST, hash table
graphs	BFS, DFS, Prim, Kruskal, Bellman-Ford
and so on	max-flow min-cut, probabilistic analysis

Why study algorithms?

Their impact is broad and far-reaching





Internet. Web search, packet routing, distributed file sharing, . . .

Biology. Human genome project, protein folding

Why study algorithms?

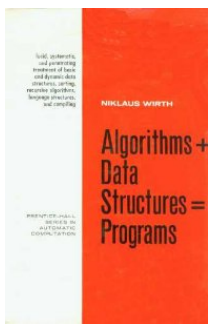
To become a proficient programmer

"The difference between a bad programmer and a good one is whether [the programmer] considers code or data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

– Linus Torvalds (creator of Linux)

"Algorithms + Data Structures = Programs."

– Niklaus Wirth (Turing award winner from Switzerland)



Why study algorithms?

They may unlock the secrets of life and of the universe

Scientists are replacing mathematical models with computational models

"Algorithms: a common language for nature, human, and computer."

– Avi Wigderson

Why study algorithms?

For **fun**, intellectual stimulation and profit!

WHO WILL TEACH YOU ALL THE COOL MATERIAL?

The Dream Team of Teaching Assistants

Antoni

Arthur

Aruzhan

Davide

Giacomo

Guy

Lina

Lukas

Marina

Matheo

Miltos

Radu

Rémy

Siba

Sofia

Victor

Wiktoria

Yassine

Youssef

Zihan

Instructor

- Alessandro Chiesa
  - *alessandro.chiesa@epfl.ch*
  - `https://ic-people.epfl.ch/~achiesa/`
- Faculty of computer science
  - research on cryptography, complexity
- Office: BC 245

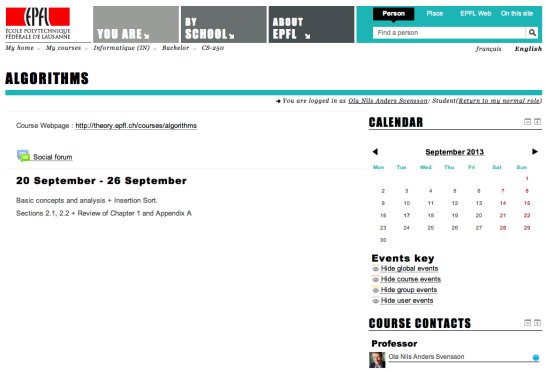
Instructor

- Ola Svensson
  - *ola.svensson@epfl.ch*
  - `http://theory.epfl.ch/osven/`
- Faculty of computer science
  - research on algorithms
- Office: INJ 114

THE COURSE ESSENTIALS

Online Resources

Moodle: *http://moodle.epfl.ch/*

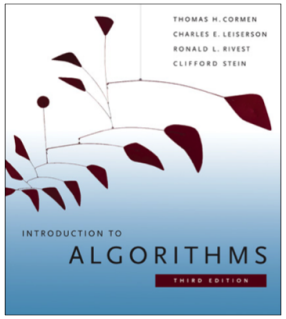


Dynamic: course material (these slides will be there), exercises+solutions, quizzes, etc.

Forum where you can ask questions!

Textbook

Introduction to Algorithms, Third edition (2009)



by T. Cormen, C. Leiserson, R. Rivest, C. Stein

- To learn best, solve as many of the exercises at the end of sections and problems at the end of chapters as you can.
- For mathematical background, look at the appendices at the end of the book.
- Many other good books available (see e.g. chapter notes at the end of Chapter 1)
  A classic: *The Art of Computer Programming* Volumes 1, 2, 3, 4A by D. E. Knuth

Time and Location

Lectures:

- Tuesday 13:15 - 15:00 (RLC E1 240)
- Wednesday 11:15 - 13:00 (RLC E1 240)

Exercise Session:

- Friday 10:15 - 12:00 in CM11, CM1100, CM1104, CM1105, CM1106

Online forum: Around the clock :)

Grading

FINAL: exam to be scheduled during exam period in June

MID-TERM: in-class exam **during the lecture April 1**

- Both the final and mid-term will test basic knowledge as well as more advanced knowledge required for the higher grades

IMPLEMENTATION EXERCISE: on codeforces, in May

QUIZZES: ≈weekly

GRADE: The **maximum** of

(i) 10% implementation exercise + 10% quizzes + 30% mid-term + 50% final

(ii) 100% final

A note on cheating

Don't!
We will report any cases

Do not copy, look at, share your solutions to problems during quizzes, mid-terms or finals.

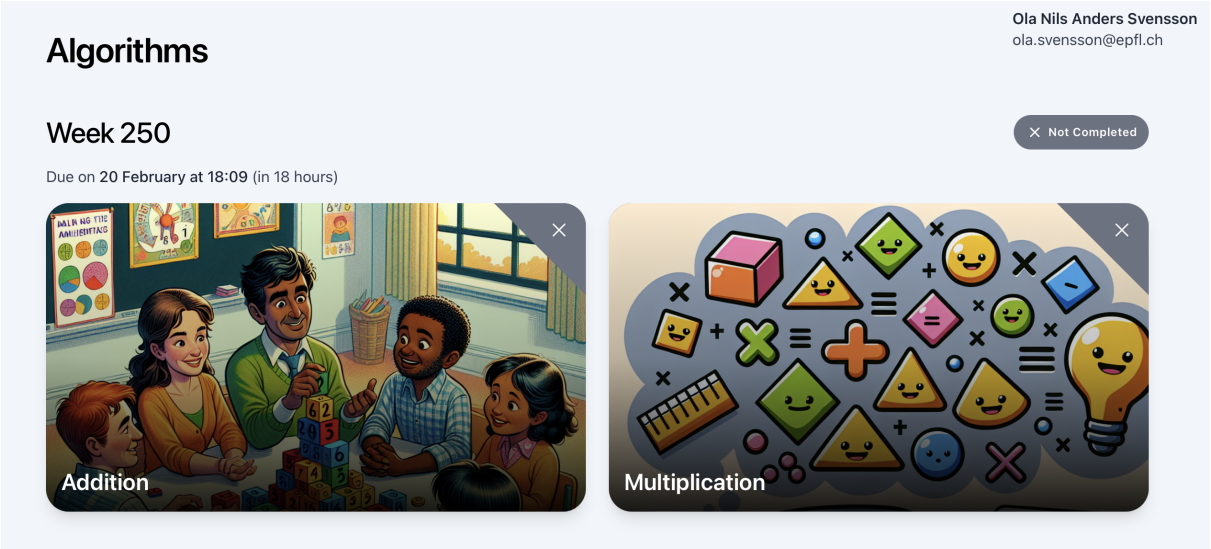
Always contact us (Ale or Ola) if in doubt whether something is allowed or not.

An exciting new addition

Instead of only listening, you will do the explaining

Explaining concepts

- Every week (starting ≈next week), we will have two concepts, for example addition and multiplication



- Explain one of the concepts and read about the other
- Following explaining/reading, do a quiz until you succeed

Let's start our journey!

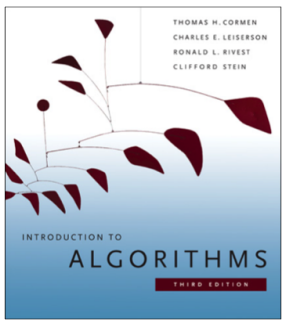
Small example: calculating an arithmetic series

Let's start our journey!

Small example: calculating an arithmetic series

Let's start our journey!

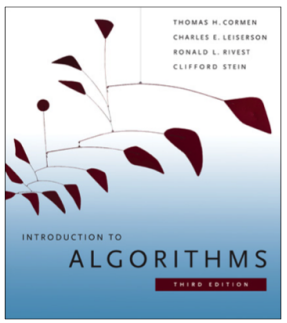
Small example: calculating an arithmetic series



Let's start our journey!

Small example: calculating an arithmetic series

"Alkhwarizmi $\rightarrow$ Algorism $\rightarrow$ Algorithm"



What is an Algorithm?

"Informally, an **algorithm** is any well-defined computational procedure that takes some value, or set of values as **input** and produces some value, or set of values, as **output**. An algorithm is thus a sequence of computational steps that transform the input into the output."

- We can view an algorithm as a tool for solving a well-specified computational problem
- The statement of the problem specifies in general terms the desired input/output relationship
- The algorithm describes a specific computational procedure for achieving that input/output relationship

Problem: Calculating an Arithmetic Series

Problem Definition (Input/Output relationship)

Input: A number n

Output: The value of $\sum_{i=1}^{n} i$

- For example, given the input 4, a correct algorithm outputs **???**

Problem: Calculating an Arithmetic Series

Problem Definition (Input/Output relationship)

Input: A number n

Output: The value of $\sum_{i=1}^{n} i$

- For example, given the input 4, a correct algorithm outputs 10. Such an input is called an instance of the problem.
- Another instance is 5, another 10142141 and so on (there are infinitely many instances).

Which one is an instance of the computational problem defined above?

A. a positive integer B. 232 C. (4,10)

First Algorithm

- Well simply calculate the sum :)

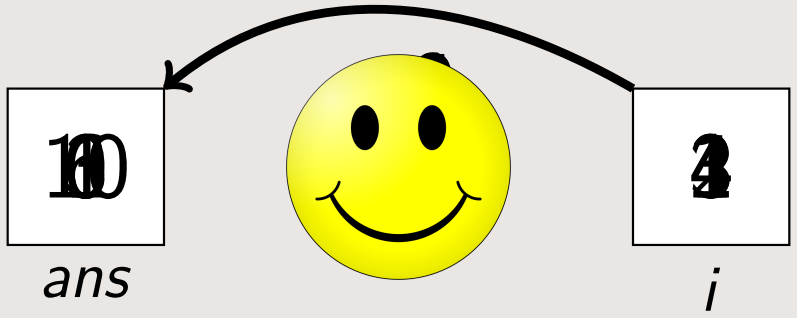
CalculateSum(n):

1. $ans = 0$
2. **for** $i = 1, 2, \ldots, n$
3. $\quad ans = ans + i$
4. **return** ans

CalculateSum(n):

1. $ans = 0$
2. **for** $i = 1, 2, \ldots, n$
3. $\quad ans = ans + i$
4. **return** ans

Example (CalculateSum(4))



- Is the algorithm efficient?

Clever Algorithm

- Well simply calculate the sum in a smart way :)

CalculateSumClever(n):

1. **return** $n(n+1)/2$

- Is the algorithm efficient?

  Space? Yes Time? Yes, constant # elementary operations

- Is the algorithm correct?

Carl-Friedrich Gauss

Correctness

Theorem

For any integer $n \geq 0$, $\frac{n(n+1)}{2} = 1 + 2 + \cdots + n$.

Most likely "Gauss' argument":

$$1 + 2 + 3 + \cdots + n - 2 + n - 1 + n$$
$$= (1 + n) + (2 + n - 1) + (3 + n - 2)$$

$n/2$ terms each equal to $n + 1$

$$= \mathbf{\frac{n}{2}(n + 1)}$$

- Rigorous **proof by induction** ← refresh this technique!

The message

- Straightforward algorithm not always the best
- Clever insight into structure of the problem
- Provably better/faster algorithm

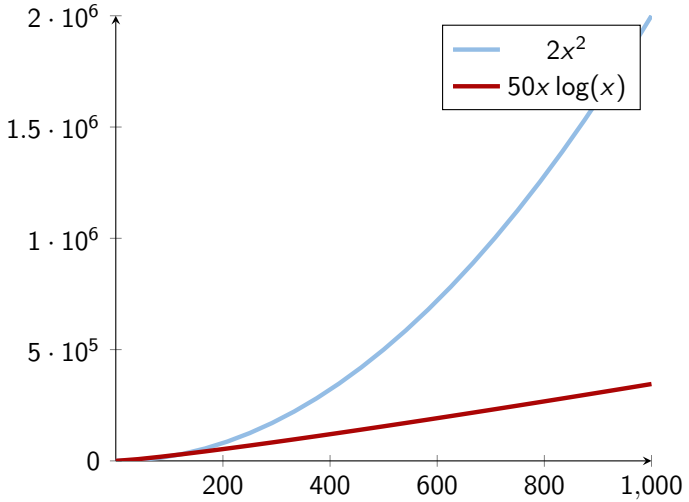
Some of you are still not convinced...

I buy a 10 times faster computer next year

Asymptotics

If insertion sort takes $2n^2$ steps and merge sort takes $50n \lg n$ steps, and if computer FAST runs 1000 faster than computer SLOW which one takes more time if we have to sort $10,000,000$ numbers?

A. Insertion sort on computer FAST B. Merge sort on computer SLOW

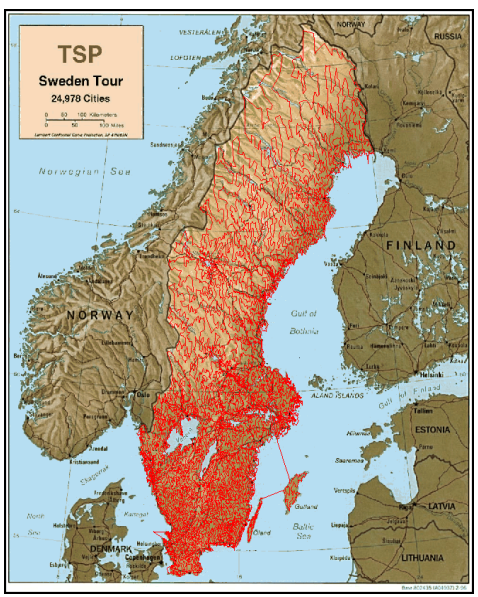


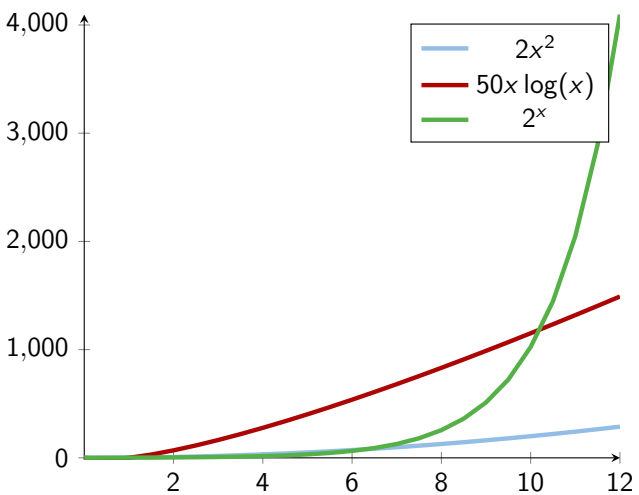
- Constant coefficients do not matter asymptotically!
  Refresh the notions $O(\cdot), \Omega(\cdot), \Theta(\cdot), o(\cdot), \omega(\cdot)$

Be Aware, There are Hard Problems

- Most problems in this course can be solved by an efficient algorithm that runs in time $O(n)$, $O(n^2)$ or $O(n^3)$.
- Many more complex problems do not have efficient algorithms (they are NP-hard)

Example: Traveling Salesman Problem (best algorithm runs in time $\approx 2^n$)





The 280-Year-Old Algorithm Inside Google Trips

Tuesday, September 20, 2016

Posted by Bogdan Arsintescu, Software Engineer & Sreenivas Gollapudi, Kostas Kollias, Tamas Sarlos and Andrew Tomkins, Research Scientists

Algorithms Engineering is a lot of fun because algorithms do not go out of fashion: one never knows when an oldie-but-goodie might come in handy. Case in point: Yesterday, Google announced Google Trips, a new app to assist you in your travels by helping you create your own "perfect day" in a city. Surprisingly, deep inside Google Trips, there is an algorithm that was invented 280 years ago.

In 1736, Leonhard Euler authored a brief but beautiful mathematical paper regarding the town of Königsberg and its 7 bridges, shown here:

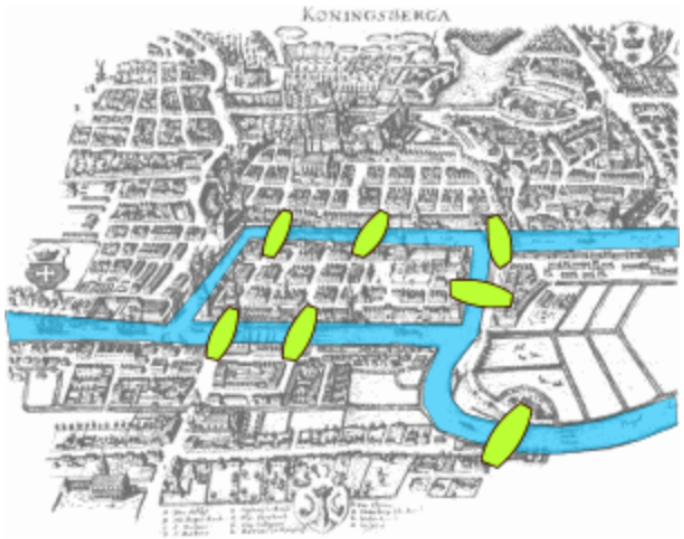
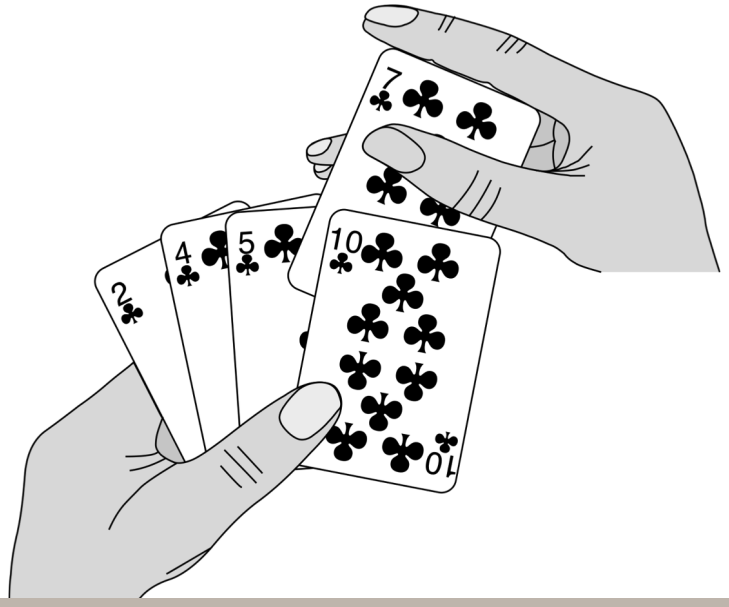


Image from Wikipedia



SORTING

Insertion Sort

The sorting problem

Definition

INPUT: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

OUTPUT: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

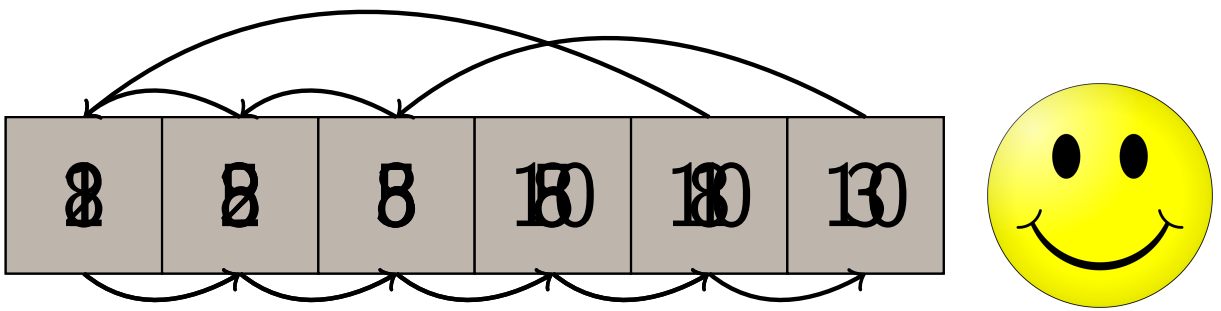
For example

- Given the input $\langle 5, 2, 4, 6, 1, 3 \rangle$
- a correct output is $\langle 1, 2, 3, 4, 5, 6 \rangle$

Insertion Sort - The Idea

Like sorting a hand of playing cards

- Start with an empty left hand of playing cards and the cards face down on the table
- Then remove one card at a time from the table, and insert it into the correct position in the left hand
- To find the correct position for a card, compare it with each of the cards already in the hand, from right to left.
- At all times, the cards, held in the left hand are sorted, and these cards were originally the top cards of the pile on the table



Insertion Sort

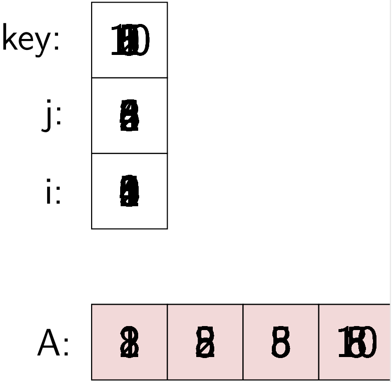
The Algorithm

- Takes as parameters an array $A[1 \ldots n]$ and the length n of the array

```
INSERTION-SORT(A, n)
 for j = 2 to n
     key = A[j]
     // Insert A[j] into the sorted sequence A[1 .. j − 1].
     i = j − 1
     while i > 0 and A[i] > key
         A[i + 1] = A[i]
         i = i − 1
     A[i + 1] = key
```

Insertion Sort

Example on $\langle 8, 2, 5, 10, 1, 3 \rangle$




```
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1 .. j − 1].
      i = j − 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = key
```


```
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1 .. j − 1].
      i = j − 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = key
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1 .. j − 1].
      i = j − 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = key
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1 .. j − 1].
      i = j − 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i − 1
```

Correctness

```
INSERTION-SORT(A, n)
for j = 2 to n
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Loop invariant:

At the start of each iteration of the "outer" **for** loop – the loop indexed by j– the subarrary $A[1 \ldots, j-1]$ consists of the elements originally in $A[1, \ldots, j-1]$ but in sorted order.

Similar to induction

Need to verify:

- **Initialization:** It is true prior to the first iteration of the loop.
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
- **Termination:** When the loop terminates, the invariant — usually along with the reason that the loop terminated — gives us a useful property that helps show that the algorithm is correct.

Analyzing Algorithms

Computational Model

We want to predict the resources that the algorithm requires. Usually, running time.

For that we need a computational model

Random-access machine (RAM) model

- Instructions are executed one after another
- Simplification basic instructions take constant (O(1)) time
  - Arithmetic: add, subtract, multiply, divide, remainder, floor, ceiling
  - Data movement: load, store, copy.
  - Control: conditional/unconditional branch, subroutine call and return
- We don't worry about precision, although it is crucial in certain numerical applications

Analyzing an algorithm's running time (1/2)

Time it takes depend on the input

- Sorting 1000 numbers take longer than sorting 3 numbers
- A given sorting algorithm may even take different amounts of time on two inputs of the same size

Input size: depends on the problem being studied

- Usually, the number of items in the input. Like the size n of the array being sorted
- If multiplying two integers, could be the total number of bits in the two integers
- Could be described by more than one number: e.g. graph algorithm running times are usually expressed in terms of the number of vertices and the number of edges in the input graph.

Analyzing an algorithm's running time (2/2)

Running time: on a particular input, it is the number of primitive operations (steps) executed

- Figure that each line of pseudocode requires a constant amount of time
- One line may take a different amount of time than another, but each execution of line i takes the same amount of time c_i
- This is assuming that the line consists only of primitive operations
  - If the line is a subroutine call, then the actual call takes constant time, but the execution of the subroutine might not
  - If the line specifies operations other than primitive ones, then it might take more than constant time. Example: "sort the points by x-coordinate"

Analysis of insertion sort

```
INSERTION-SORT(A, n)                                   cost   times
  for j = 2 to n                                       c1     n
      key = A[j]                                       c2     n − 1
      // Insert A[j] into the sorted sequence A[1 .. j − 1].   0      n − 1
      i = j − 1                                        c4     n − 1
      while i > 0 and A[i] > key                       c5     Σ_{j=2}^{n} t_j
          A[i + 1] = A[i]                              c6     Σ_{j=2}^{n} (t_j − 1)
          i = i − 1                                    c7     Σ_{j=2}^{n} (t_j − 1)
      A[i + 1] = key                                   c8     n − 1
```

number of times line executed based on the value of j

Best case: The array is already sorted

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = \Theta(n)$$

A note on worst-case analysis

We usually concentrate on finding the **worst-case running time:** the longest running time for *any* input of size n

Reasons:

- Gives a guaranteed upper bound on the running time for any input
- For some algorithms, the worst case occurs often.
  For example, when searching, the worst case often occurs when the item being searched for is not present
- Average case often as bad as worst-case: Suppose that we randomly choose n numbers as the input to insertion sort

Order of growth: Focus on the important features

- Drop lower-order terms
- Ignore the constant coefficient in the leading term

Summary

- Welcome to “Algorithms”!
- You will learn a lot of cool and useful material
- We will do our best to help you, but ask questions
- Today: what is an algorithm and simple example + analysis.
- Refresh from previous courses:
  - Discrete math such as mathematical induction, graphs and trees, and probability (covered in Appendices B, C)
  - Asymptototics and summation formulas (covered in Chapter 3 and Appendix A)
  - Programming basics such as arrays, recursive calls, and pointers.