CSC 225

Algorithms and Data Structures I
Spring 2018
Venkatesh Srinivasan

Lectures and Labs

Venkatesh Srinivasan
- E-mail: srinivas@uvic.ca
- Voice: 250-472-5731
- Office: ECS 626
- Office hours:
  R 12:30 pm – 2:30 pm

- Lectures
  - A01 MR 10:00 am – 11:20 pm MAC A144
- Labs
  - Labs start week of January 8, 2017
  - Lab Instructors: Michael, Nishat
  - B01 M 11:30 – 12:20 pm ECS 258
  - B02 M 12:30 – 1:20 pm ECS 258
  - B03 R 11:30 – 12:20 pm ECS 258
  - B04 R 12:30 - 13:20 pm ECS 258
  - B05 F 2:30 – 3:20 pm ECS 258
  - B06 F 3:30 – 4:20 pm ECS 258

- Course Web pages
  - Official Webpage on the Department Website
  - Detailed Course Website on ConneX

Administrative Officer Announcements

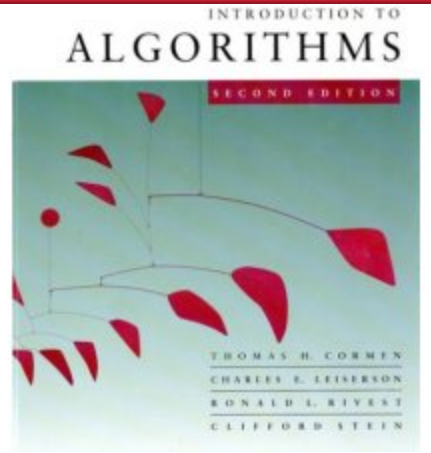
- CSC Administrative Officer is Sue Butler
  E-mail: cscadvisor@uvic.ca Office: ECS 512
- Any student who has registered in CSC 225 and **does not** have the required pre-requisites and no waiver **must drop the class**. Otherwise: student will be dropped and a pre-requisite drop is recorded on the student's record.
- Taking the course more than twice:

  you must request, in writing, permission from the Chair of the Department and the Dean of the Faculty to be allowed to stay registered in the class (University Rule). The letter should be given to Sue Butler, Undergraduate Advisor. Otherwise: student will be dropped from class.
- Always use and check your UVic e-mail account and use CSC 225 as part of the subject line.
- Do not send messages from other accounts (such messages are filtered and discarded).
- Register for labs!

Books

- **Required Textbook**
  R. Sedgewick and K. Wayne
  Algorithms, Fourth Edition
  Addison-Wesley, Toronto, 2011
  ISBN: 0-321-57351-X
- http://algs4.cs.princeton.edu/home/

- **Optional Textbook**
  T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. *Introduction to Algorithms.*
  MIT Press (2001), 2nd edition.



Lectures and Labs

- **Attendance of Lectures**
  - Essential for doing well on assignments and exams
- **Labs**
  - Extra details and hints on assignments

Evaluation

Assignments & Labs	25%
Midterm	25%
Final	50%

- Marks will be posted on conneX
- Midterm exam will be in-class, one hour, closed books, closed notes, no calculators, no gadgets **February 8, 2018**
- The final exam will be three hours, closed books, closed notes, no calculators, no gadgets scheduled by the registrar

Assignment Schedule

A1	February 1, 2018 (due date)
A2	February 19, 2018
A3	March 15, 2018
A4	March 29, 2018

Reading Assignment

- **Chapter 1 – 1.1, 1.2 in Sedgewick and Wayne**
- Algorithm wiki
  - http://en.wikipedia.org/wiki/Algorithm
  - History: Development of the notion of "algorithm"
- Data structures wiki
  - http://en.wikipedia.org/wiki/Data_structure
  - http://en.wikipedia.org/wiki/List_of_data_structures
  - http://en.wikibooks.org/wiki/Data_Structures

Assignments

- **4 assignments during the course**
- **Late submissions are <u>not</u> accepted**
  - ➢ if valid excuse (e.g., doctor's statement), raise weight of other assignments to compensate
- **Programming: work in the labs or at home**
  - ➢ use your favorite Java environment
- **Cheating: zero-tolerance policy**
  - ➢ first time fail assignment, second time fail course

Prerequisites

- **CSC 115**
  - Basic Java knowledge and programming skills
  - Object-oriented programming
  - Basics in fundamental algorithms and data structures as discussed in CSC 115

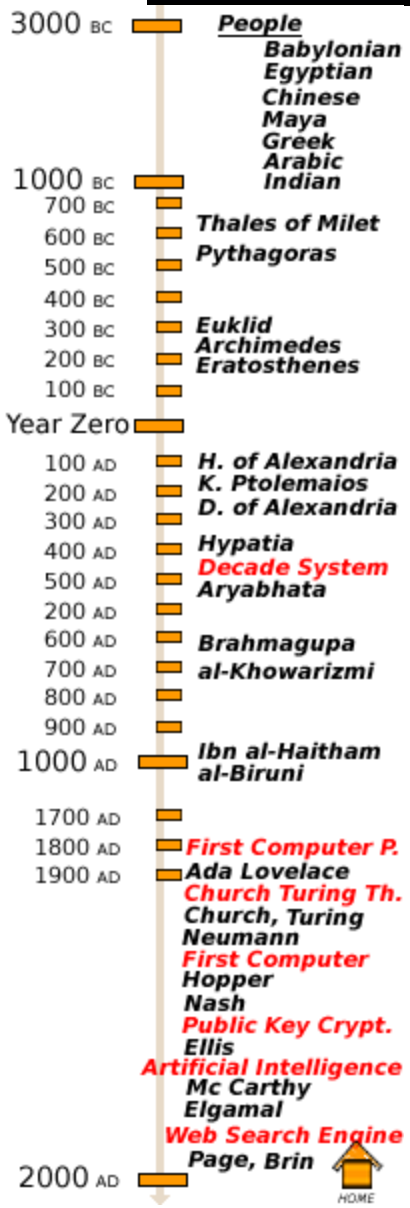
Lecture Notes

- **Acknowledgments**
  - Most of the slides for this course were prepared by Dr. Ulrike Stege. Thank you!!
- Consider posted lecture slides as *additional* information
- **Note**
  - Not all materials required for the midterm and final exams are on the lecture slides

Questions?

- Regarding questions on lectures, assignments, algorithms, data structures, programming, Java, etc. consult in the following order:
  - Study group
  - ConneX web page
  - Lab instructor
  - Instructor

History of Algorithms



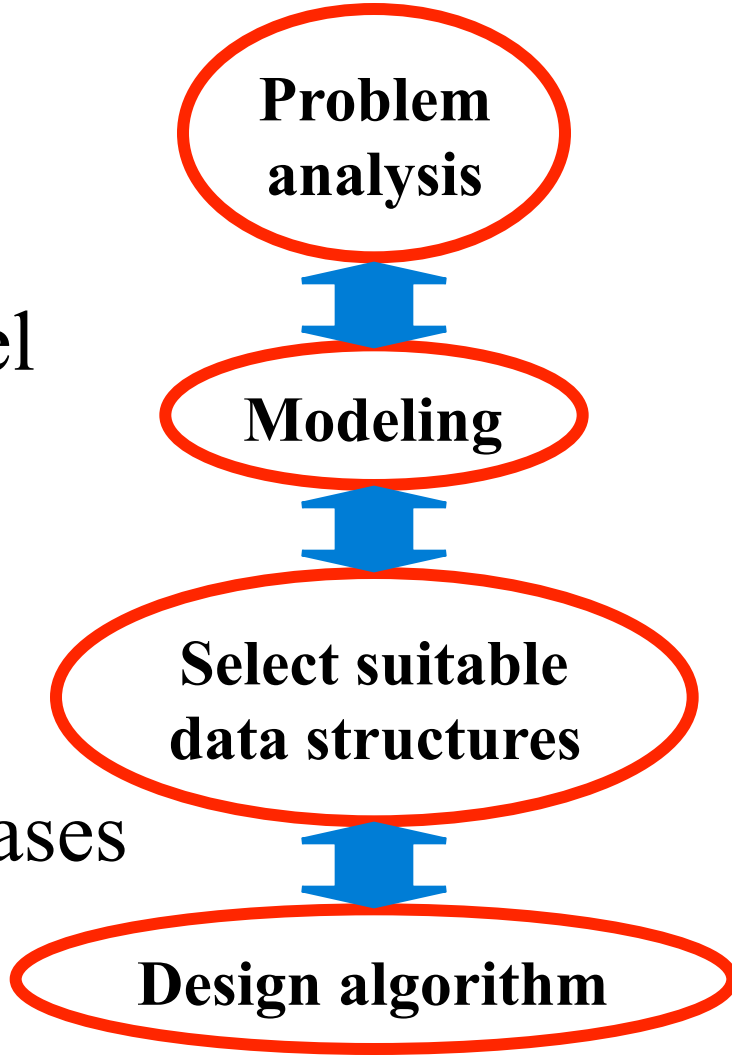
- The word algorithm can be traced back to the 9th century to the Persian scientist, astronomer and mathematician Abdullah Muhammad bin Musa al-Khwarizmi, often cited as "The father of Algebra"
- In the 12th century one of his books was translated into Latin, where his name was rendered as "Algorithmi"
- Algorithms are everywhere. They have been developed to ease our daily life from calculating algorithms, to artificial intelligence and molecular biology. The searching and sorting algorithms embodied in Google are a good example of our daily use of algorithms.
- In the age of information, people are inundated with data. With the aid of powerful algorithms and data structures, we can make sense of volumes of data that come in many forms: text, numbers, images, video, audio.

- **History of Algorithms**
  http://cs-exhibitions.uni-klu.ac.at/index.php?id=193

Algorithmics

- Problem analysis
- Design an appropriate model
- Select data structures
- Select algorithms

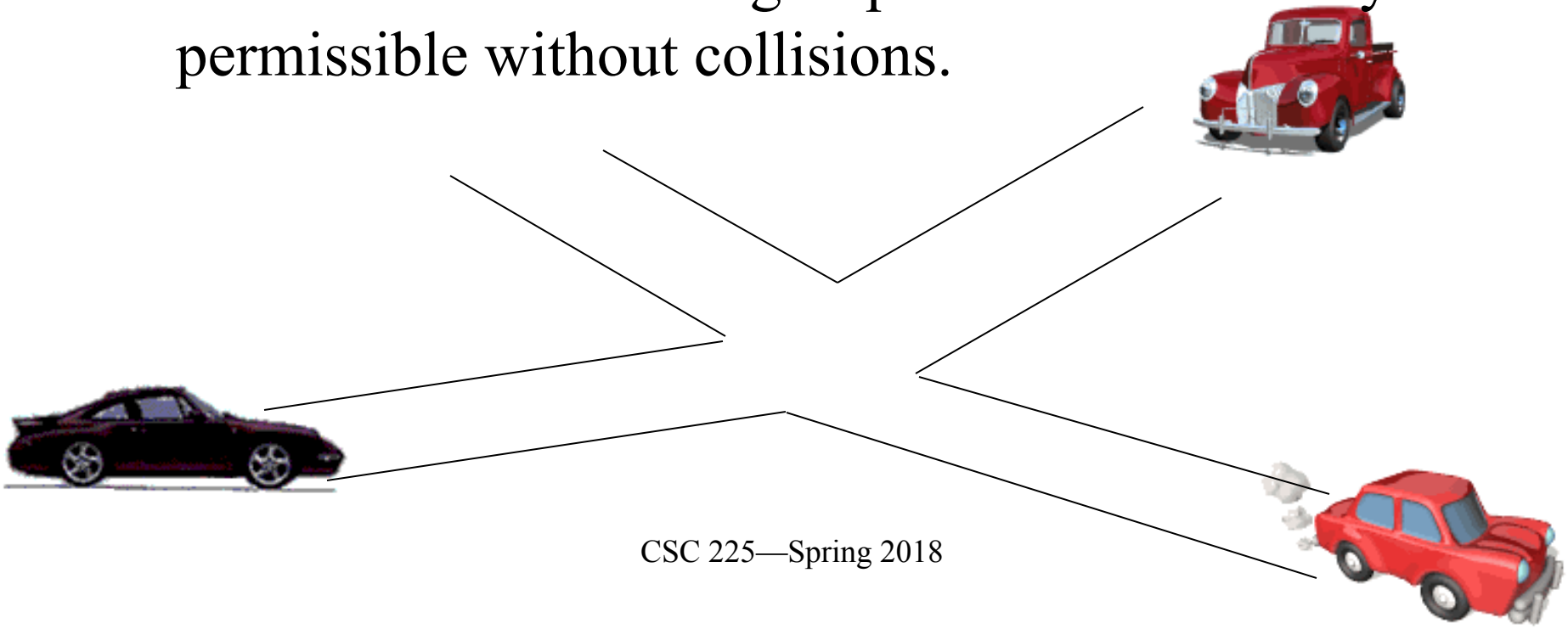
- Iterate over these design phases



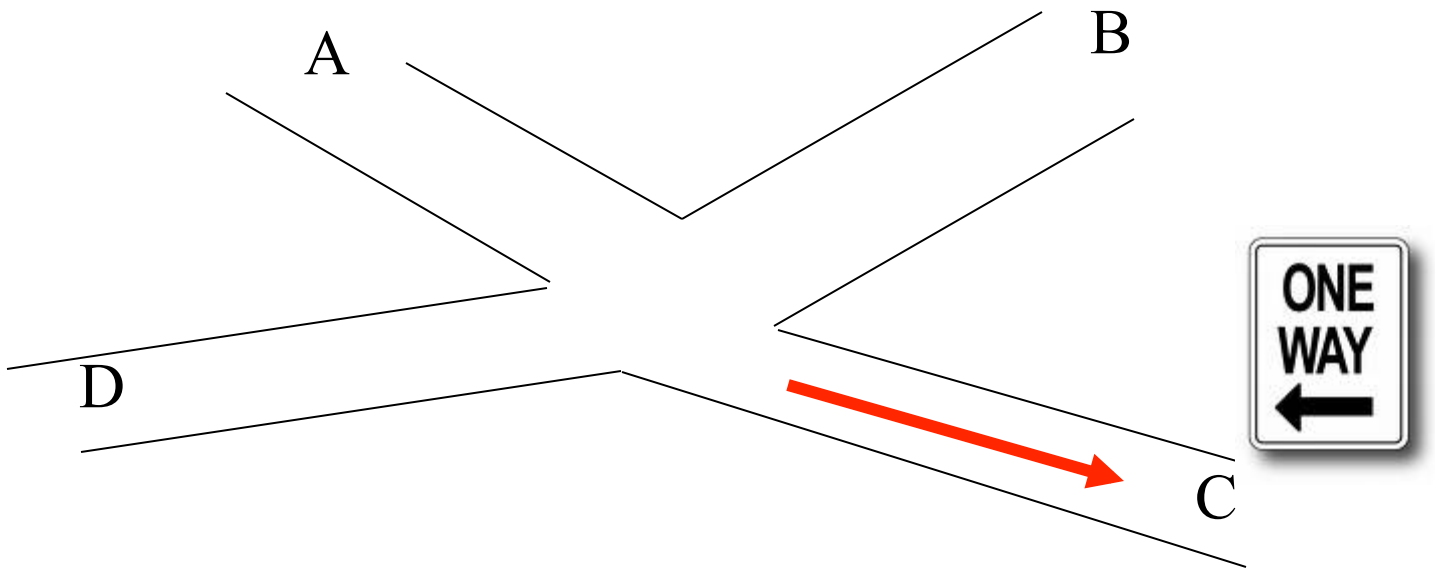
Design of traffic light phases for an intersection of roads

Given: A set of permitted turns of intersections

Goal: Partition the set into as few groups as possible such that all turns in a group are simultaneously permissible without collisions.

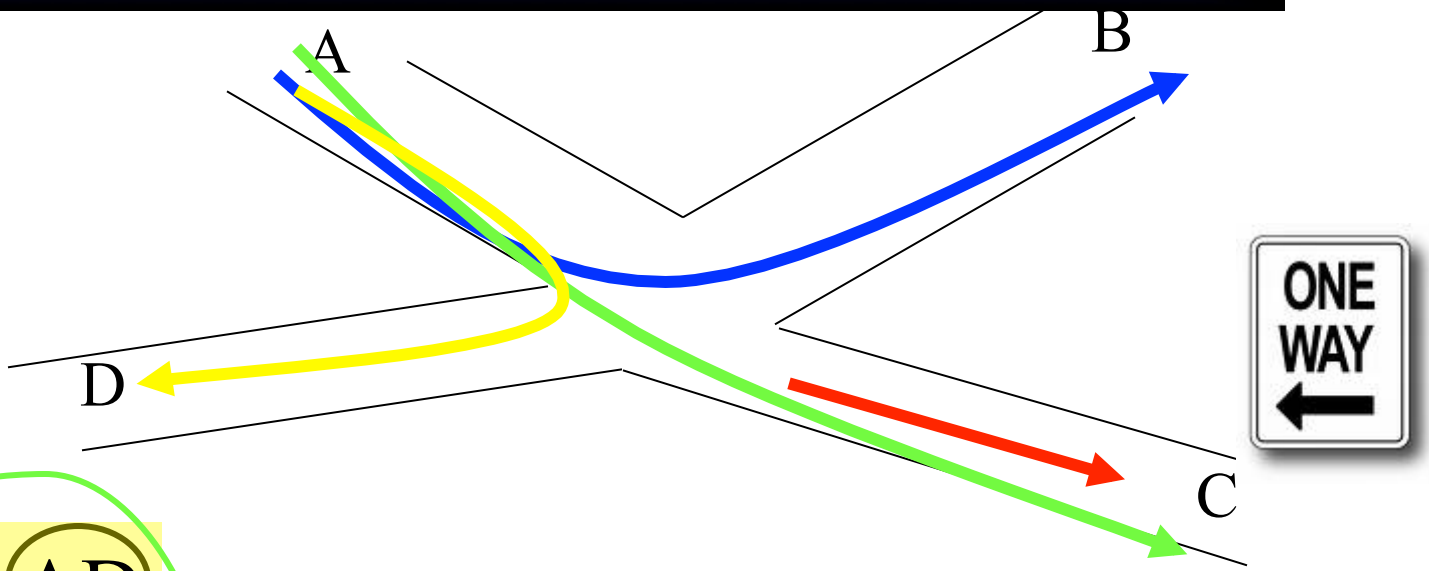


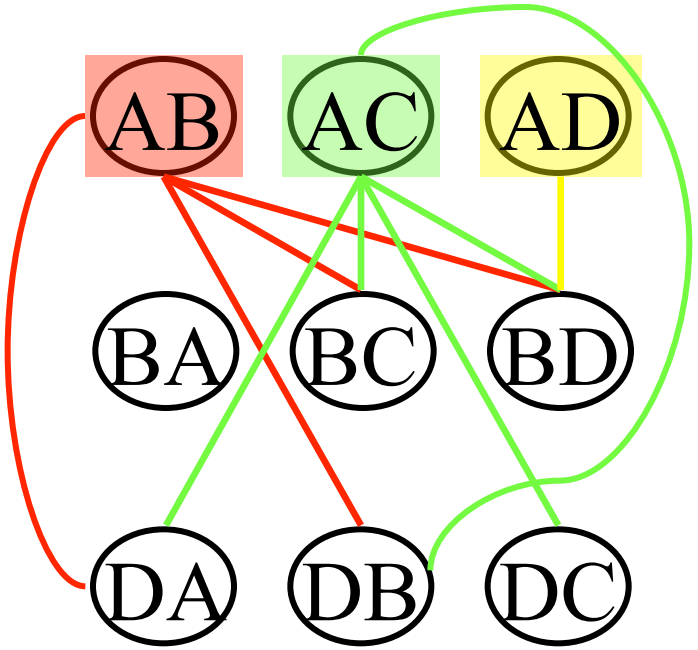
Possible Turns



- AB, AC, AD
- BA, BC, BD
- DA, DB, DC

The Model: A Conflict Graph





Possible turns ≡ *vertices*
Incompatible turns ≡ *edges*

The Model: A Conflict Graph

- Complete the conflict graph for the other two rows of the graph

- Find a traffic phase assignment such that no two turns that have a possible collision are allowed at the same time
- Minimize the number of phases
- Implement domain-specific optimization

Problem Solving

- **Model:** Conflict Graph
- **Data Structure:** graph
- **Algorithm:** graph colouring

Graph Colouring Problem

Input: A *graph* $G = (V, E)$ with *vertices* and *edges* E.

- Possible turns ≡ *vertices*
- Incompatible turns ≡ *edges*

Output: A minimum set of colours and an assignment from the colours to the vertices such that no two vertices that have an edge in common have the same colour.

- Colour ≡ *traffic light phase*

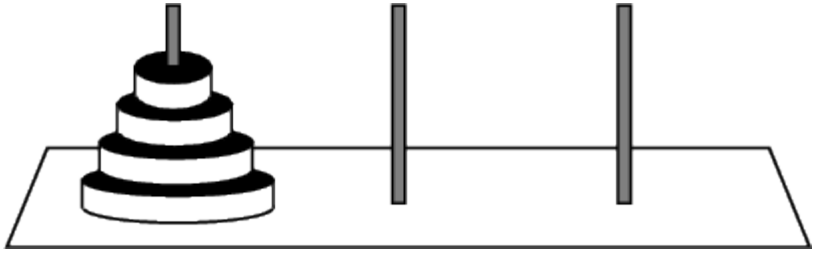
Algorithm Design Techniques

- **Algorithm Design Techniques**
  - Greedy algorithms
  - Divide and conquer
  - Backtracking
  - Dynamic programming

Design of traffic light phases for an intersection of roads

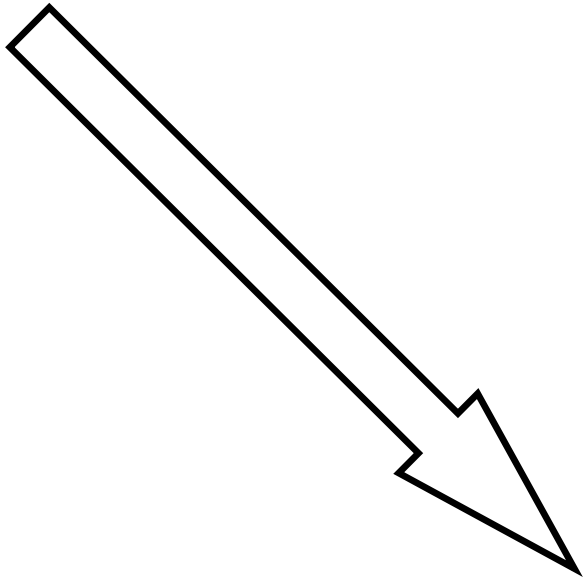
- Find optimal solution this problem
  - Hard
  - Graph colouring is NP-complete
- Find "a" solution using greedy approach
  - Heuristic: choose an arbitrary node as the first node to be coloured
  - Colour all the nodes that are not connected to this node with the same colour
  - Iterate
- Enumerate all solutions using greedy approach and select best or most appropriate solution
  - Works for small graphs

Which Type of Algorithm Solves the Towers of Hanoi Problem?

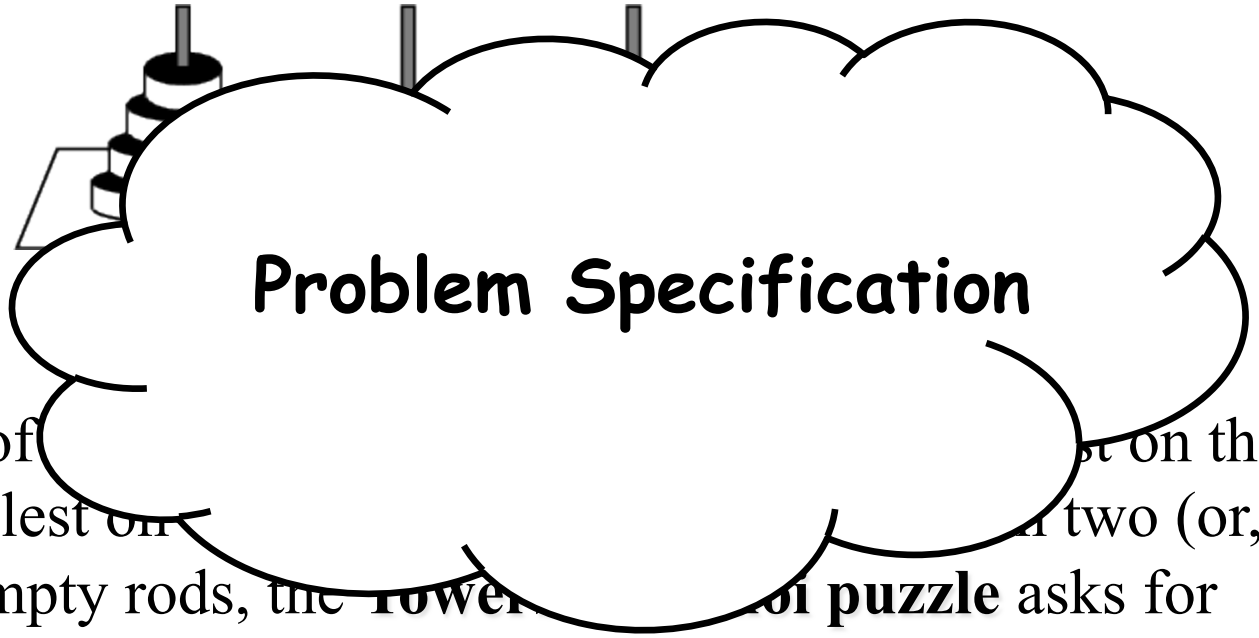


Recursion and Backtracking

Towers of Hanoi



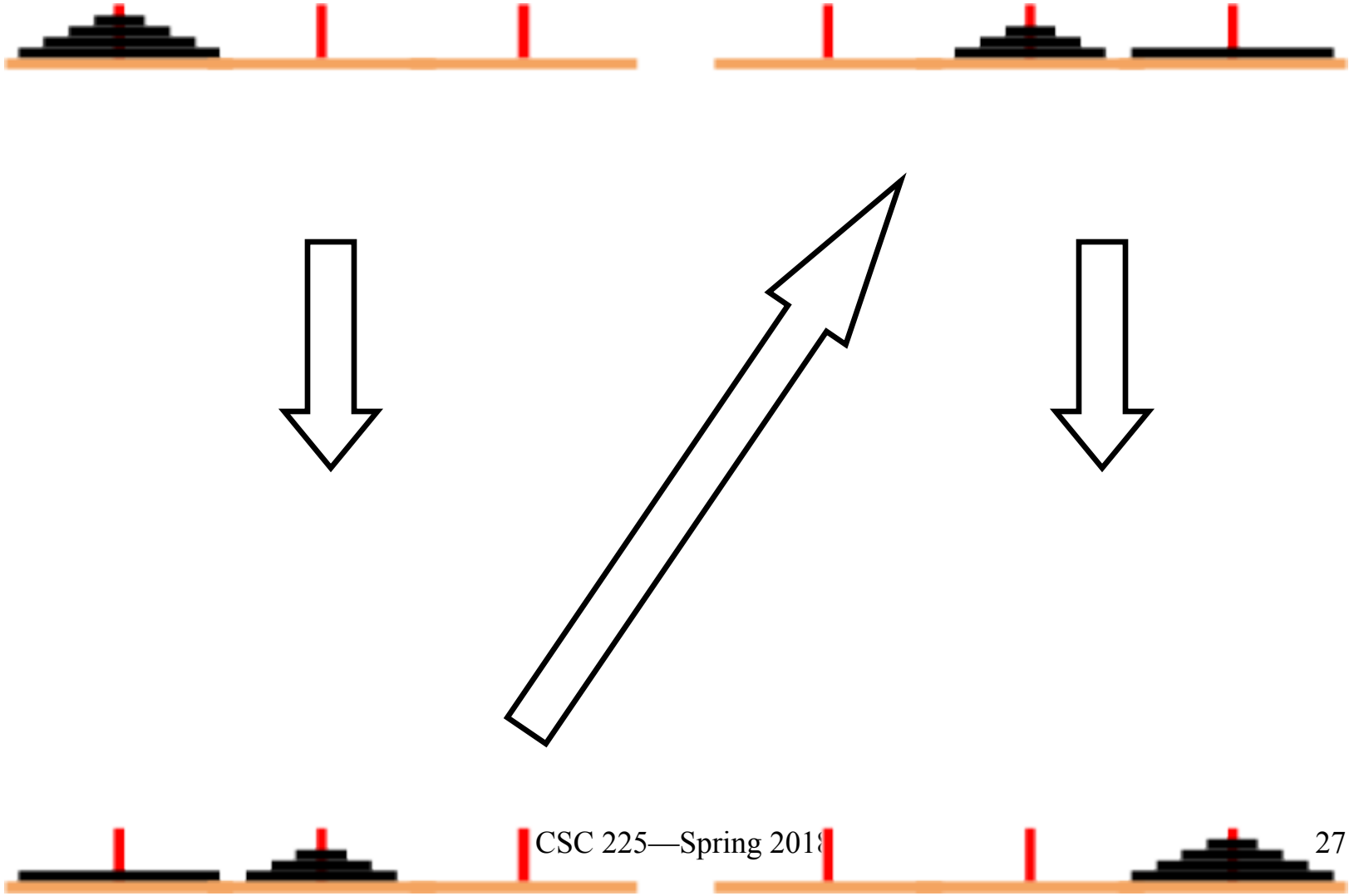
Towers of Hanoi



Given a stack of ... on the bottom to smallest on ... two (or, in general *r*) empty rods, the **Tower... puzzle** asks for the *minimum number* of disk-moves required to move the stack from one rod to another. **...e moves are not allowed only if they place larger disks on top of smaller disks.**

$n = 4$

Solving for $n = 3$ helps!



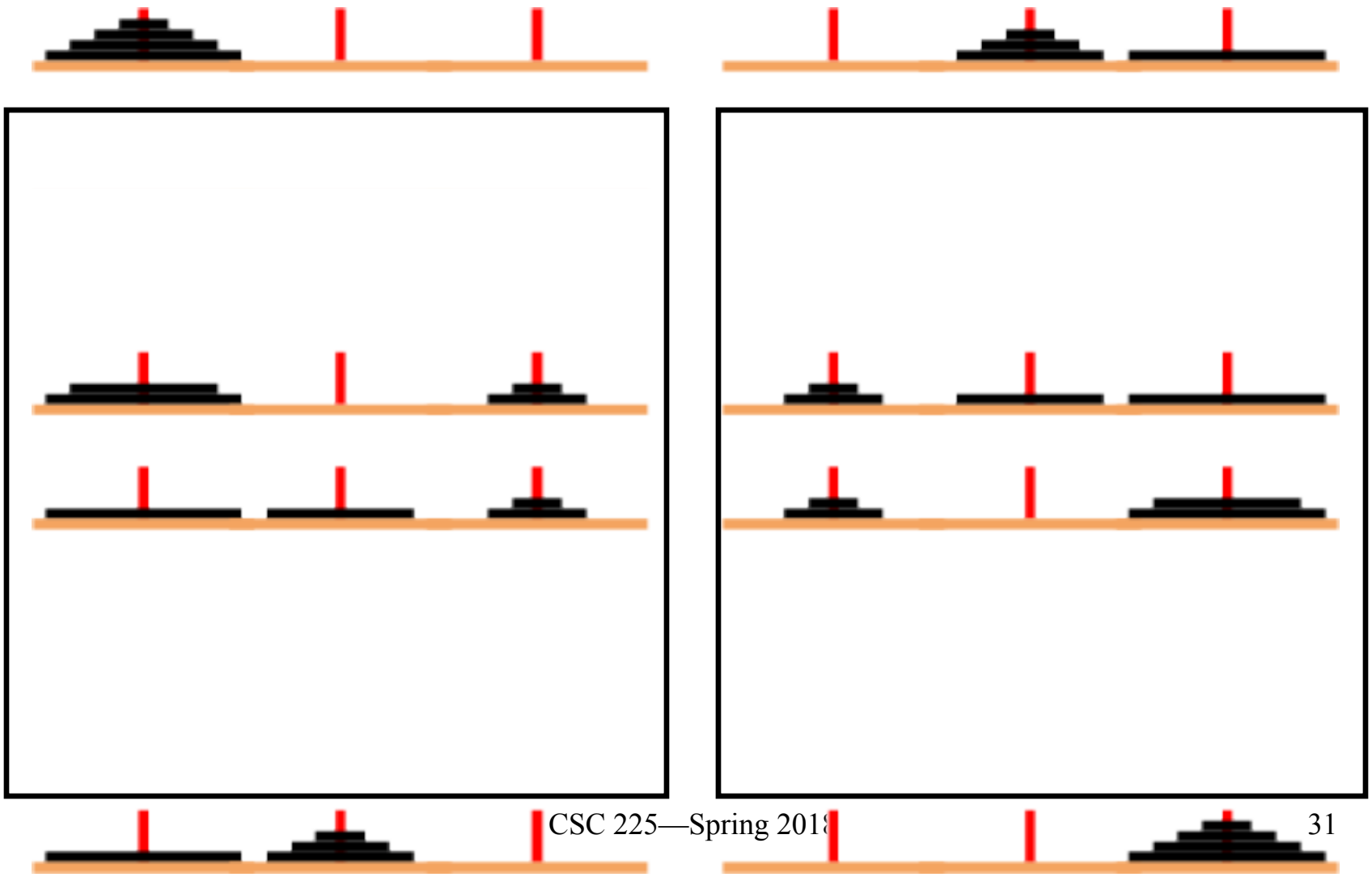
$n = 3$

- Solving for $n = 2$ helps!

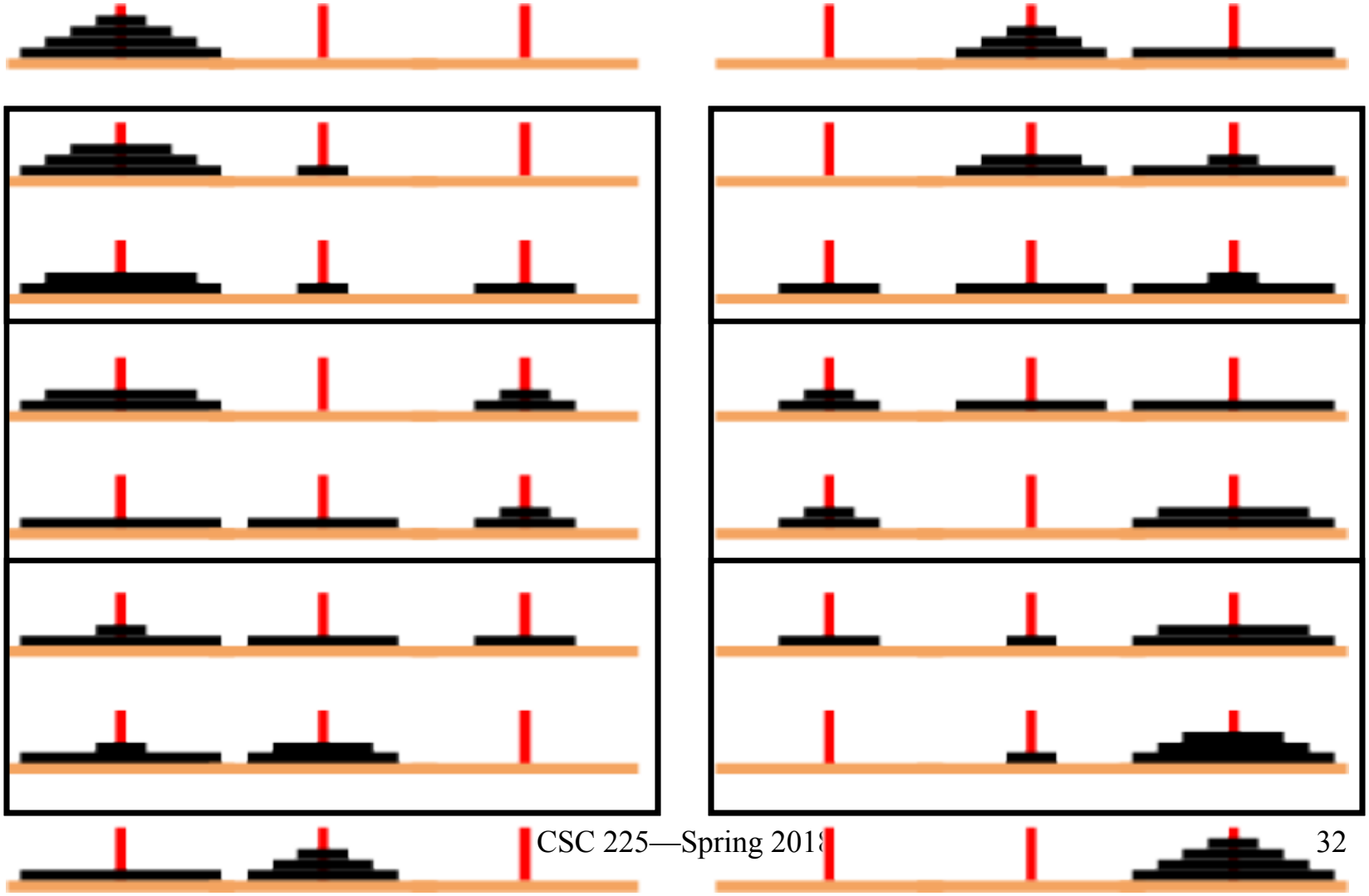
$n = 2$

$n = 4$

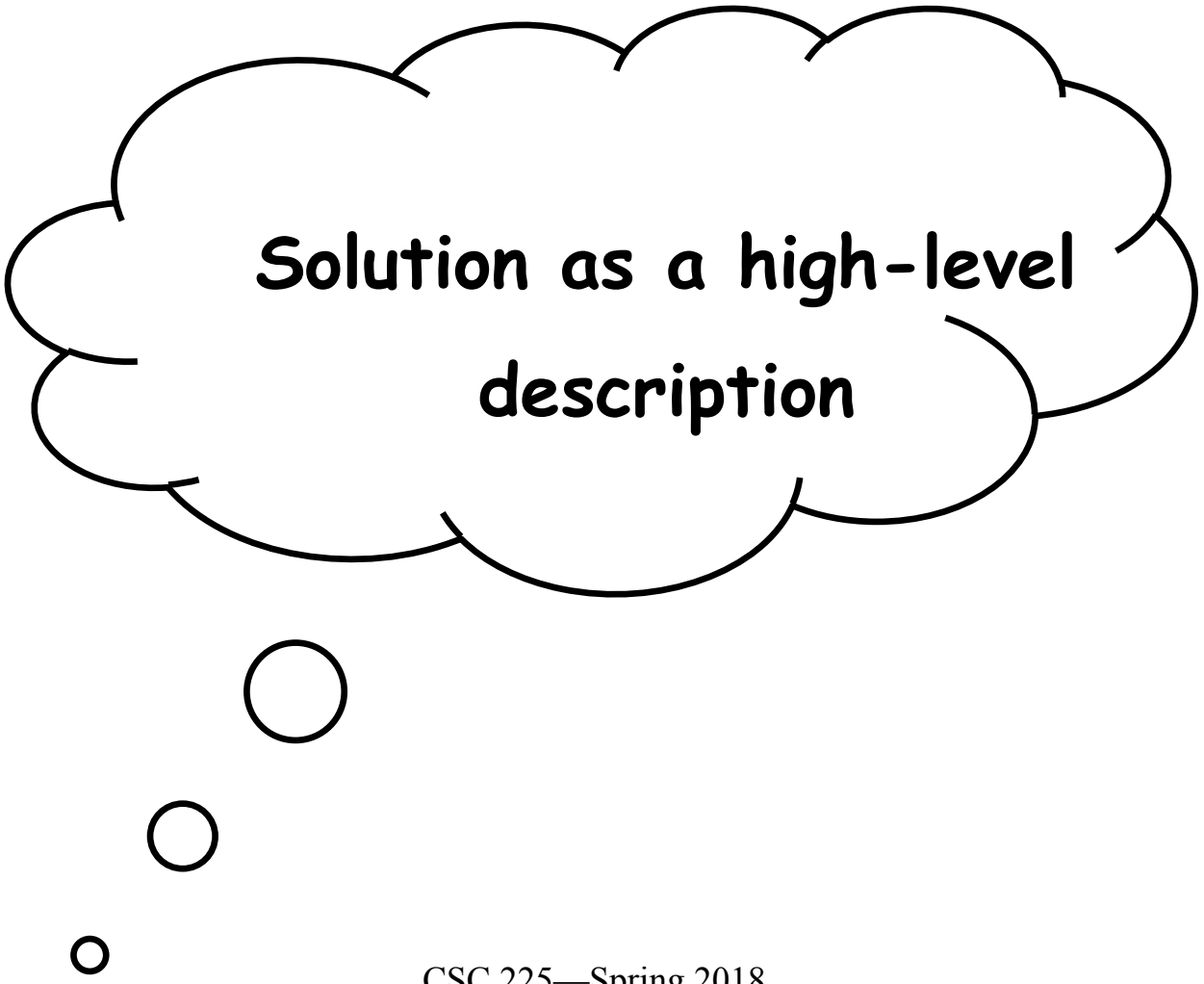
Solution for $n = 4$ (15 moves required)



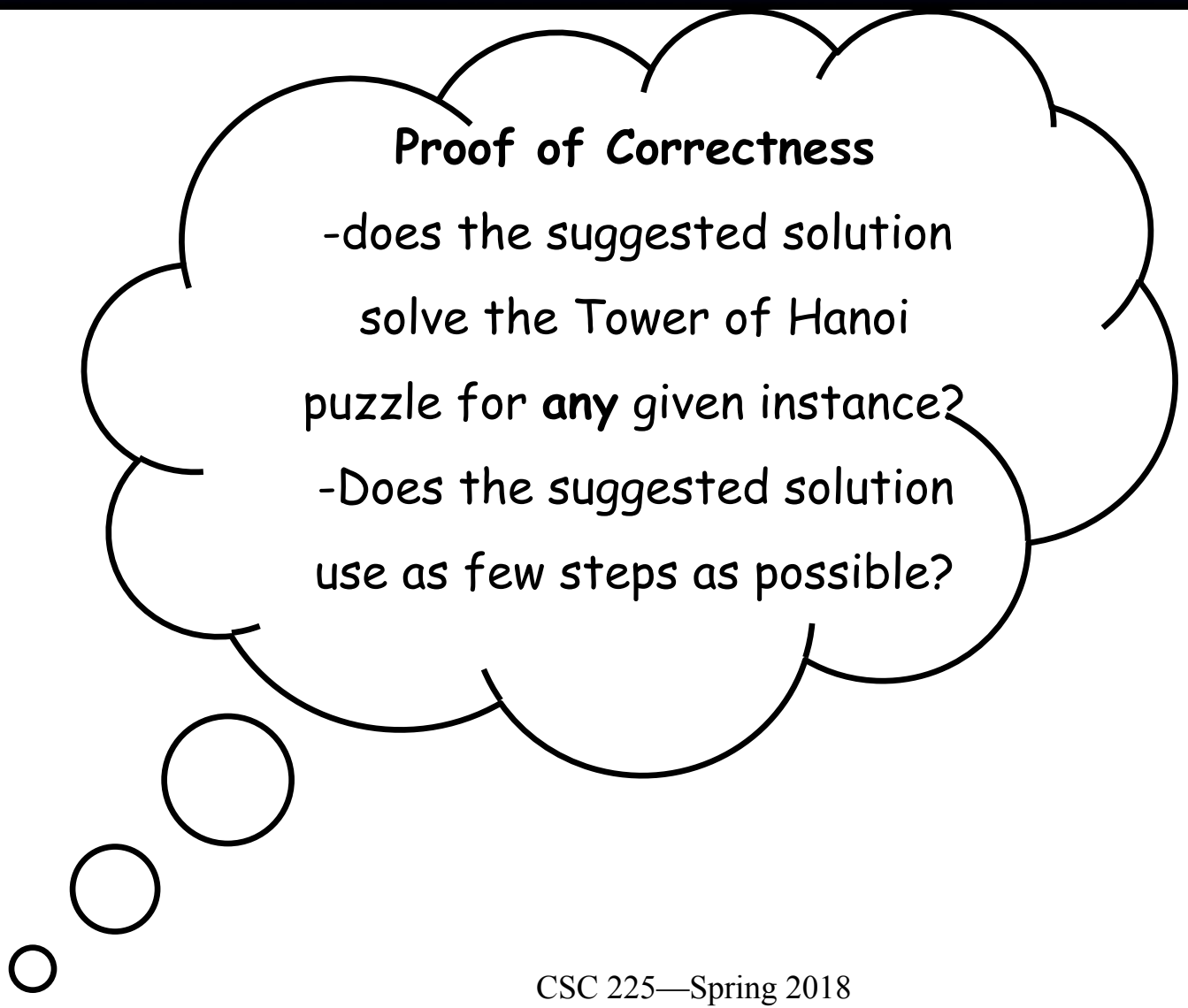
Solution for $n = 4$ (15 moves required)



Still missing



Still missing



Quiz

- How many moves are required for $n = 5$?
- How many moves are required for n discs?
- Does the number of moves change if 4 rods (instead of 3 rods) are allowed?
  - $n = 1$
  - $n = 2$
  - $n = 3$
  - $n = 4$
  - $n = 5$

What is an Algorithm?

- An ***algorithm*** is a sequence of unambiguous instructions for solving a problem for obtaining the desired output for any legitimate input in a finite amount of time.

- An ***algorithm*** is a finite procedure, written in a fixed symbolic vocabulary, governed by precise instructions, moving in discrete steps, 1, 2, 3, …, whose execution requires no insight, cleverness, intuition, intelligence, or perspicuity, and that sooner or later comes to an end.

- The nonambiguity requirement is critical
- The range of inputs has to be carefully specified
- An algorithm can be implemented in several different ways
- Algorithms for the same problem can be based on very different ideas and can solve the problem with very different speeds

Fundamental Algorithms

- Selection (Linear Median)
- Quicksort, Heapsort
- Linear search, Hash search
- Tree traversals
- Graph traversals
- Depth first search, breadth first search

Two Problems with several Algorithms

Sorting

- Bubblesort
- Insertionsort
- Selectionsort
- Quicksort
- Heapsort
- Smoothsort

Searching

- Linear search
- Binary search
- Hash search

What is a Data Structure?

- A data structure is a particular scheme for organizing and accessing related data items
- The nature of the data elements is dictated by the problem at hand

Fundamental Data Structures

- Primitive data types
  - int, char, string, Boolean, double
- Compound data types
  - Arrays, records
  - Combination thereof
- Records or structs
  - Nested records or structs
- Arrays
  - 1-D arrays or vectors
  - 2-D arrays or tables
  - n-D arrays
- Lists
  - Linked lists, singly and doubly linked lists, skip lists
  - Stack, queues
- Trees
  - Rooted trees and forests
  - Ordered trees
  - Binary trees and m-way trees
  - AVL-trees
  - Heaps
  - Tries
- Graphs
  - Directed and undirected graphs
  - Weighted graphs
  - Paths and cycles
  - Representations: adjacency list, adjacency matrix, doubly connected edge list
- Sets
- Dictionaries
- Priority queues

Pseudo Code

- An algorithm is the procedural or step-by-step solution of a problem
- The procedural solution can be at different levels of abstraction

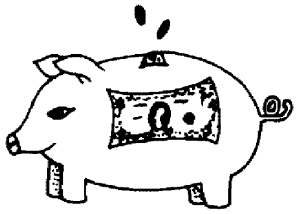
- Machine code
- Assembly language
- C

- Pascal
- C++
- Java
- C#

- Pseudo code

Banking Example

To honor their "most economical" young client, a Canadian bank wants to find the child or student having the largest amount of money in his or her savings account.



- *Problem 1*: For each child/student: sum up all the amounts in his or her saving accounts
- *Problem 2*: Find the largest number over all children

Problem 2: Find the maximum value in an array of *n* numbers

```
                                …
public int arrayMax(int[] A, int n) {
  int currentMax = A[0];
  for (int k = 1; k<n; i++) {
   if (currentMax < A[k]) {
     currentMax = A[k];
   }
  }
  return currentMax;
}
                                …
```

Pseudo-code vs. Java code

```
currentMax ← A[0]
for k ← 1 to n-1 do
 if currentMax < A[k]
then
    currentMax ← A[k]
return currentMax
```

```
                    …
public int arrayMax(int[] A,
   int n) {
   int currentMax = A[0];
   for (int i=1; i<n; i++) {
    if (currentMax < A[i]) {
     currentMax = A[i];
    }
   }
 return currentMax;
}
                    …
```

Problem 2: Find the maximum value in an array of *n* numbers

```
Algorithm arrayMax(A,n):
  Input: An array A storing n ≥ 1
                integers
  Output: The maximum element in A

  currentMax ← A[0]
  for k ← 1 to n-1 do
     if currentMax < A[k] then
          currentMax ← A[k]
  return currentMax
```

Pseudo Code: Expressions

- Assignment operator (←)
  - `currentMax ← A[i]`
- Equality relation (=)
  - `currentMax = A[i]`

  is `true` if *`currentMax`* and `A[i]` have the same value, otherwise `false`.
- Smaller than (<), greater than (>), smaller or equal to (≤), greater or equal to (≥)

Pseudo Code: Method declarations

- **Algorithm** name(param1, param2,…)

```
Algorithm arrayMax(A,n)
```

Pseudo Code: Decision structures

- **if** `condition` **then**
  `true-actions`
  [**else**
  `false-actions`]
  end

- **if** `currentMax < A[k]` **then**
  `currentMax ← A[k]`
  end

Pseudo Code: While loops

- **while** condition **do**
  actions
  end

- **while** currentMax < A[k] **do**
  count ← 2*count
  i ← k+1
  end

Pseudo Code: Repeat loops

- **repeat**
  actions
  until condition

- **repeat**
  i ← k+1
  count ← 2*count
  until currentMax < A[i]

Pseudo Code: For loops

- **for** step-definition **do**
  actions
  end

- **for** k ← 1 **to** n-1 **do**
  if currentMax < A[k] **then**
  currentMax ← A[k]
  end
  end

Pseudo Code: Array indexing and record field selection

- A[k] represents the k^{th} cell in array A, indexed from $A[0]$ to $A[n-1]$.
- $r.key$ represents the field *key* in record or struct r

Pseudo Code: Method calls and return statements

- Method call
  - object.method(args)
  - Example: `arrayMax(X,13)`

- Return statement
  - **return** value

Algorithmics

- Design and analysis process of algorithms and their underlying data structures
- Given a problem
  - Understand and specify
  - Model the problem
  - Design of data structures and algorithms
    - Investigate algorithm design techniques
    - Investigate data structures
  - Prove correctness of the algorithm
  - Analyze the efficiency of the algorithm
  - Implement solution
  - Test implementation

Important Problem Types

- **Sorting:** arrange the items in a list in ascending order according to a key
- **Searching:** finding a search key in a given set of items
- **String processing:** searching for a key word or phrase in a large information space
- **Graph problems:** given a set of vertices and connecting edges, traverse the graph and assess properties

- **Combinatorial problems:** find a combinatorial object—permutation, combination, subset, spanning tree
- **Geometric problems:** find a geometric object—convex hull, Voronoi diagram; computational geometry, computer graphics, robotics, tomography
- **Numerical problems:** find mathematical objects—solving systems of equations, evaluating functions
- **Computational biology problems:** find a biology object

- **CSC 225, 326, 425** concentrate on sorting, searching, string processing, graph problems
- **CSC 340, 349A, 349B, 445, 446** deal with numerical problems
- **CSC 426** deals with computational geometry problems
- **CSC 428** deals with computational biology problems

Analysis of Algorithm Efficiency

No everything that can be counted counts,
and not everything that counts can be counted.
—Albert Einstein (1879-1955)

- Time efficiency
  - indicates how fast an algorithm runs
- Space efficiency
  - indicates how much extra space an algorithm requires

Measuring an Algorithm's Running Time

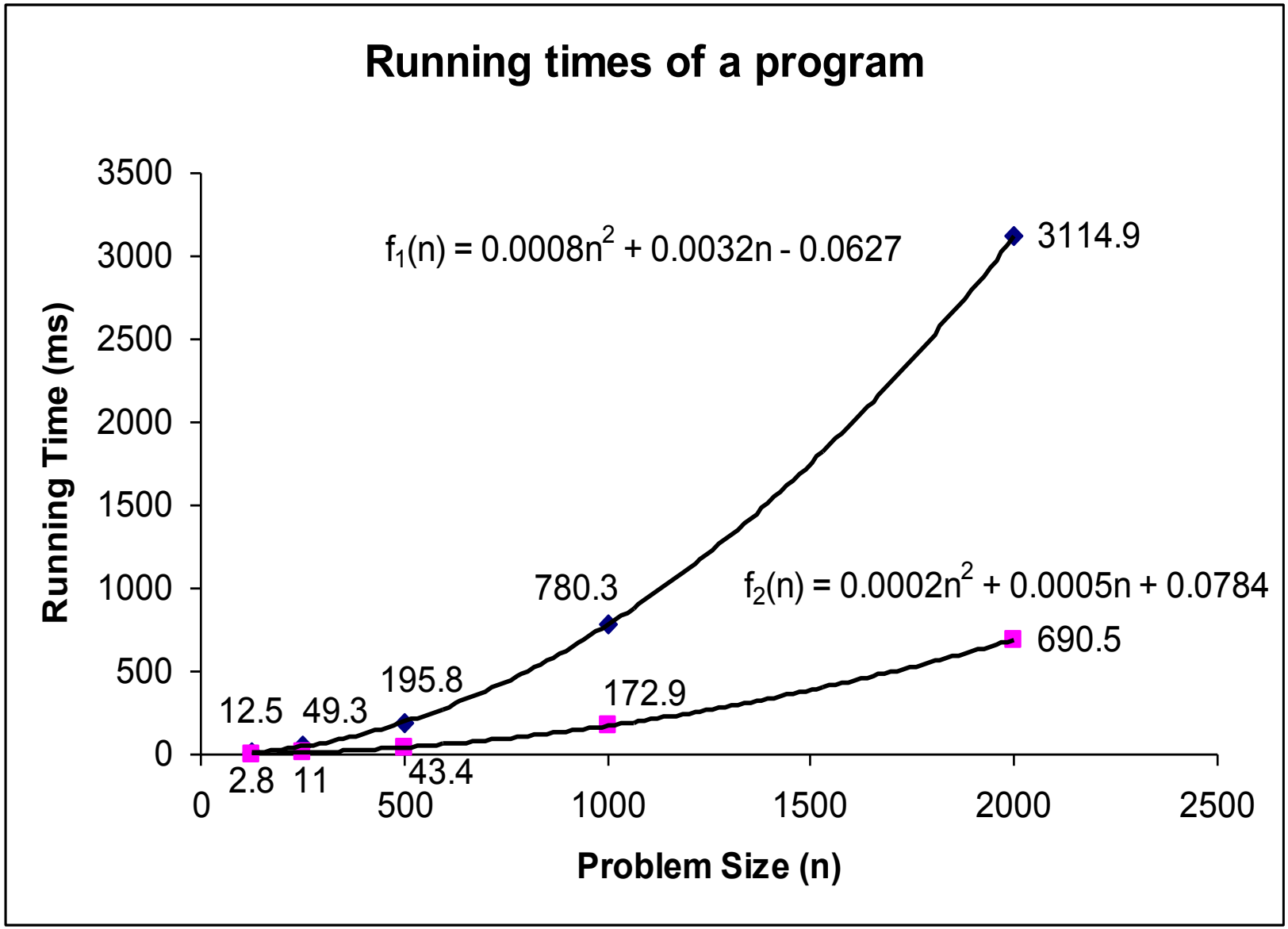
- Two approaches
  - Counting primitive operations and computing upper and lower bounds—topic in CSC 115/160, CSC 225, CSC 326, CSC 425, CSC 426, CSC 428 and many other courses
  - Instrument the code to measure computer clock cycles—topic in SENG 265

What is the running time of this algorithm?

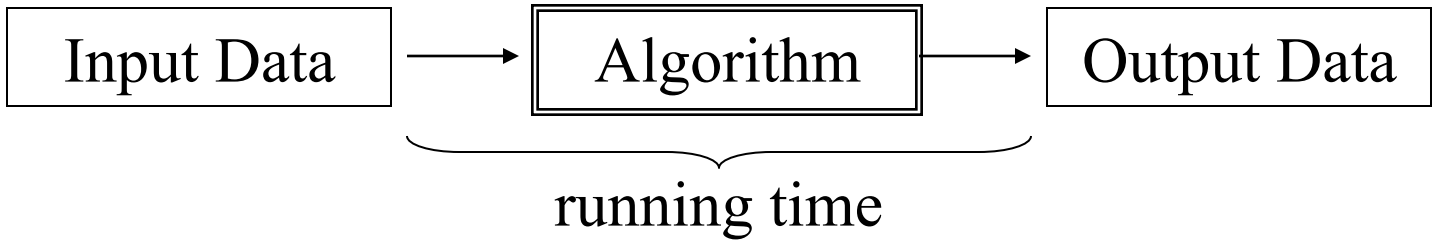
```
Algorithm arrayMax(A,n):
  Input: An array A storing n ≥ 1
     integers.
  Output: The maximum element in A.

  currentMax ← A[0]
  for k ← 1 to n-1 do
     if currentMax < A[k] then
           currentMax ← A[k]
     end
  end
  return currentMax
```

Instrumenting the Code to Measure Clock Cycles

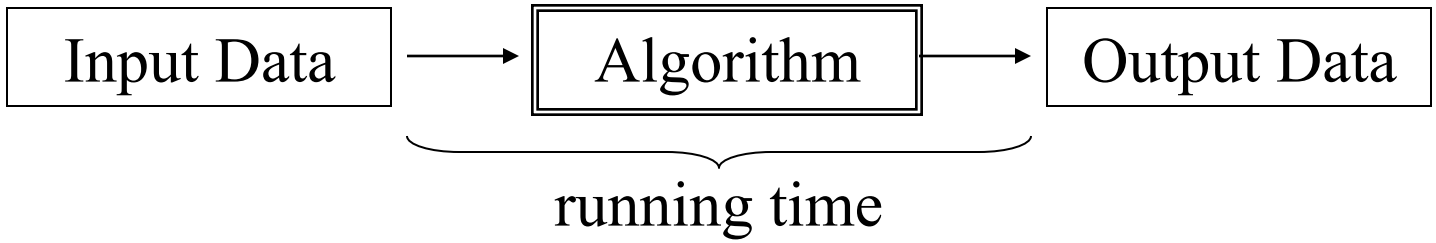


Limitations of Experiments Results



- The algorithm must be implemented
- Experiments can typically be done only on a limited set of test inputs
- When is a set of test inputs representative?
- To compare two algorithms the experiments must run on the same machine

Algorithm Analysis Wish List



- Take all possible inputs into account
- Compare the efficiency of two different algorithms independent from computer and implementation
- Analyze algorithm *before* starting implementation

Time Complexity Analysis

- Measuring the size of the algorithm's input
- Defining units for measuring running time
- Computing function for running time
- Performing worst-case, best-case, average-case analysis
- Insert function into the order of growth
- Asymptotic notations and efficiency classes
- Analysis of recursive and non-recursive algorithms

Primitive Operations

- Assignments (A)
- Comparisons (C)
- Boolean expressions (B)
- Array indexing (I)
- Record selector or object reference (R)
- Arithmetic operations
  - Add, subtract (S)
  - Multiplication, division (D)
- Trigonometric operations (e.g., sin, cos, tan) (T)
- Calling a method, function, procedure, routine (M)

Worst Case Running Time T(n)
Counting Assignments, Comparisons & Indexing

```
currentMax ← A[0]
for k ← 1 to n-1 do
    if currentMax < A[k] then
        currentMax ← A[k]
    end
end
return currentMax
```

- How has the input to be arranged to produce the best case and the worst case?

Worst case

- 1 A + 1 I +
- 1 A + (n-1)*{
  - 1 A + 1 S +
  - 1 C +
  - 1 C + 1 I +
  - 1 A + 1 I +
- }
- 1 C (to terminate loop) +
- 1 A

$T(n) = 5+(n-1)*7$

$T(n) = 7n - 2$

Worst Case Running Time T(n)
Counting Assignments & Comparisons

```
currentMax ← A[0]
for k ← 1 to n-1 do
    if currentMax < A[k] then
        currentMax ← A[k]
    end
end
return currentMax
```

- How has the input to be arranged to produce the best case and the worst case?

Worst case

- 1 A +
- 1 A + (n-1)*{
  - 1 A +
  - 1 C +
  - 1 C +
  - 1 A +
- }
- 1 C (to terminate loop) +
- 1 A

$T(n) = 3+(n-1)*4$

$T(n) = 4n - 1$

Design & Analysis of Algorithms Go Hand in Hand

For almost all computational problems *many* candidate solutions are possible, but most of them we don't want! Sometimes we even want only one.

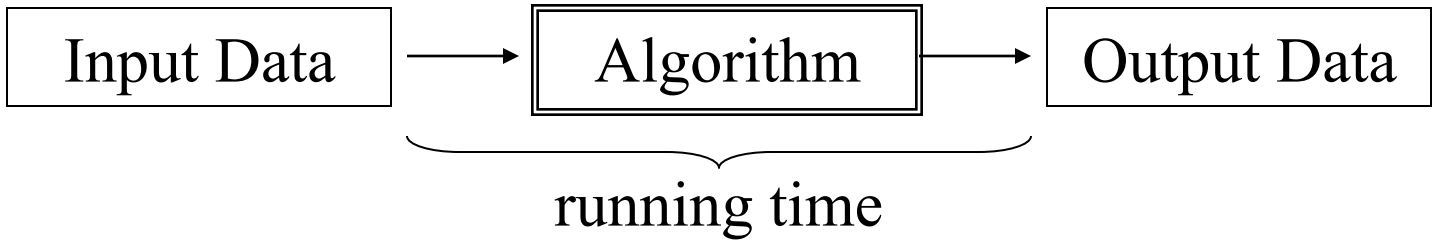
Examples

1. **Sorting** Any sequence of *n* given numbers is a possible candidate solution for a sorted sequence of these numbers.
2. **15-puzzle** Any sequence of moves is a possible candidate solution to solve the puzzle.

Tools for Algorithm Analysis

- Language for describing algorithms
  - *pseudo-code*
- Metric for measuring algorithm running time
  - *primitive operations*
- Most common approach for characterizing running times
  - *Worst-case analysis*

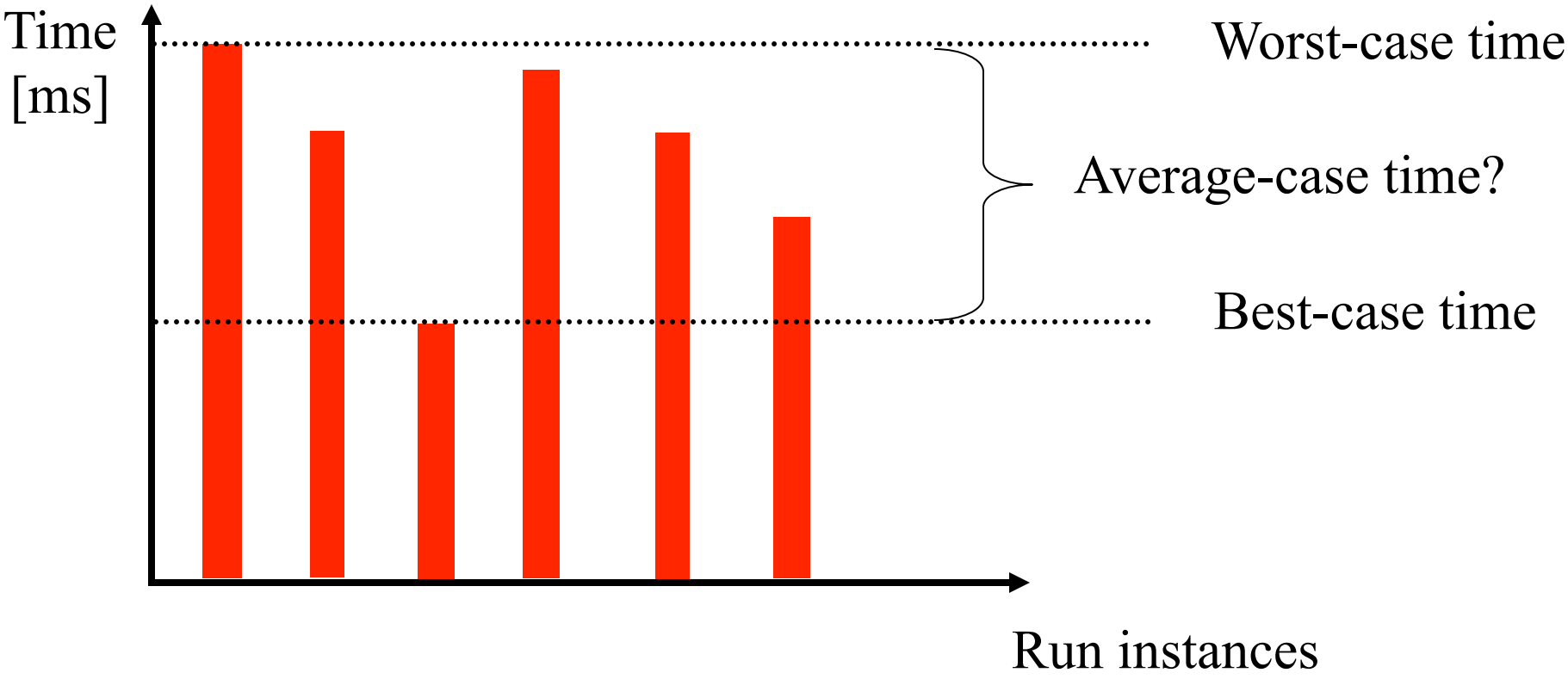
Algorithm analysis



Categories of algorithm running times

- Best case analysis $T_b(n)$
- Average-case analysis $T_a(n)$
- Worst-case analysis $T(n)$

Best-case, Average-case and Worst-case Analysis



Determining the Average-case Time

- Calculate expected running times based on a given input distribution
- Typically the analysis requires heavy math and probability theory

Determining the Best-case and Worst-case Running Time

- Worst-case T(n)
  - What is the maximum number of primitive operations (depending on *n*) executed by the algorithm, taken over all inputs of size *n*?
  - Worst-case analysis is most common and may aid in the design of the algorithm (e.g., Linear Median algorithm)
- Best-case $T_b(n)$
  - What is the minimum number of primitive operations (depending on *n*) executed by the algorithm, given the most advantageous or best input configuration of size *n*?

Examples: Worst-case, Best-case and Average-case Analysis

Basic units: A & C

```
a = 3 * n
cnt = 1
while a > n do
   a = a - 1
   cnt = cnt + 1
end
```

$T(n) = 2 + 2n(3)+1$

$T(n) = 3 + 6n$

$T_b(n) = T(n)$

Basic units: A & C

```
A: array[0..n-1]
k = 0
while k < n do
   if A[k] = key then
      return k
   end
   k = k + 1
end
return "not found"
```

$T(n) = 1 + n(3) + 2$

$T(n) = 3 + 3n$

$T_b(n) = 4$

$T_a(n) = 3 + 3n/2$

Examples: Worst-case, Best-case and Average-case Analysis

Basic units: A & C

```
A: array[0..n-1]
Swap: 3 A per swap
For: 2 A per iteration;
   ignore initial A in loops
for k=0 to n-1 do
   for j=0 to n-1 do
      if A[k]<=A[j] then
         swap(A[k],A[j])
      end
   end
end
```

$T(n) = (6n+2)n$

$T(n) = 6n^2 + 2n$

Basic units: A & C

```
For: 2 A per iteration;
   ignore initial A;
   ignore return A
s = 0
for k=1 to n do
   s = s + k*k
end
return s
```

$T(n) = 1 + 3n$