Data Structures and Algorithm Analysis (CSC317)

"I think I finally figure out how Google's indexing algorithms work."

Introduction

Poll: Why study algorithms?

CSC317 Introductions

- Introductions…

  Your major and what do you hope to get out of this course?

CSC317 House Keeping

- **Course homepage:**
  http://www.cs.miami.edu/home/odelia/teaching/csc317_fall20/index.html

- Online class
- My typed slides will be posted on a regular basis on the course website and on Blackboard
- Recordings will be posted on Blackboard
- Assignments will be posted and submitted through Blackboard

CSC317 House Keeping

- **Course homepage:**
  http://www.cs.miami.edu/home/odelia/teaching/csc317_fall20/index.html

- **Instructor:** Odelia Schwartz (odelia at cs miami dot edu). Encouraged to email to make zoom appointment
  office hour: TBA online

- **TA:** Xu Pan (and also Alison Cohen, Emily Silvershein, Dylan Aron Noah)
  Xu Pan will run problem-solving sessions online. Times TBA.

  Office hours: TBA online

CSC317 House Keeping

- Practicum optional CSC401: self-study programming using dynamic HTML and Javascript

https://www.cs.miami.edu/home/odelia/teaching/csc317_fall20/practicum/index.html

TA: Alexandros Khan
Office hours online: TBA

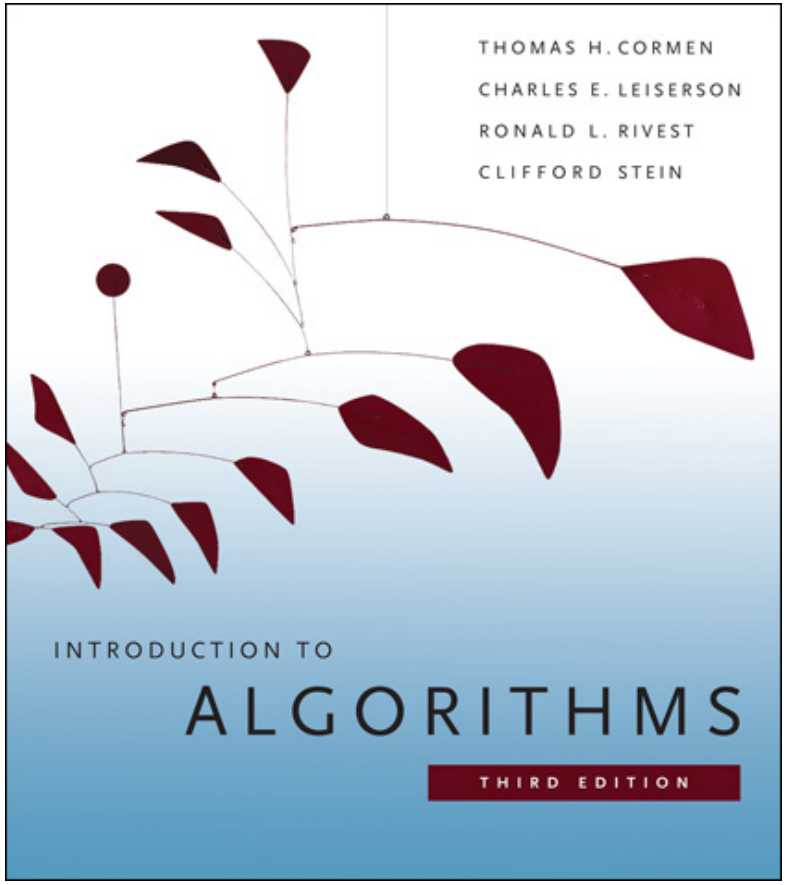
CSC317 House Keeping

- Weekly problem sets on Tuesday, due following Tuesday by midnight on BB
- Feel free to stop me and ask questions during online lectures; interactive
- Please turn on your camera during lecture
- Homework Assignments (around 11): 60%
  Quizzes (5): 50%
  No final exam

CSC317 House Keeping

- Online class structure
- Show course syllabus

Data Structures and Algorithm Analysis (CSC317)



Introduction to Algorithms (3rd edition), Cormen, Leiserson, Rivest and Stein (yes, it's thick; not the whole book…)

Algorithms – Why?

- You answer

Algorithms – Why?

- You answer
- How it has been important for me in my field…

Algorithms

- At the heart of computer science

Algorithms

- At the heart of computer science
- How to solve problems. We'll encounter prototypical examples – toolkit of approaches. Introduce many prototypical cases you will most likely later come back to…

Algorithms

- At the heart of computer science
- How to solve problems. We'll encounter prototypical examples – toolkit of approaches. Introduce many prototypical cases you will most likely later come back to…
- Not only solve the problems, but figure out how quickly algorithms run; figure our correctness of solution

Algorithms

- At the heart of computer science
- How to solve problems. We'll encounter prototypical examples – toolkit of approaches. Introduce many prototypical cases you will most likely later come back to…
- Not only solve the problems, but figure out how quickly algorithms run; figure our correctness of solution
- Many possible languages that can then implement an algorithm

Algorithms

- At the heart of computer science
- How to solve problems. We'll encounter prototypical examples – toolkit of approaches. Introduce many prototypical cases you will most likely later come back to…
- Not only solve the problems, but figure out how quickly algorithms run; figure our correctness of solution
- Many possible languages that can then implement an algorithm
- Algorithms are important in job interviews…

When have you last used an algorithm??

In the news …

New York Times, 2016:

"Instagram May Change Your Feed, Personalizing It With An Algorithm … That could mean that if your best friend posted a photo of her new Bernese mountain dog's puppies five hours ago while you were on a flight without Internet connectivity, Instagram might place that image at the top of your feed the next time you open the app. Based on your history of interaction with that friend, Instagram knows you probably would not want to miss that picture."

In the news …

New York Times, J
an 2018:

Facebook Is Changing. What Does That Mean for Your News Feed?

“Facebook is making the changes to the News Feed by tinkering under the hood and reconfiguring the algorithms.”

In the news …

New York Times, 2015:

“Can an Algorithm Hire Better Than a Human?...
A new wave of start-ups — including Gild, Entelo, Textio, Doxa and GapJumpers — is trying various ways to automate hiring. They say that software can do the job more effectively and efficiently than people can…”

In the news …

New York Times, 2020:

https://www.nytimes.com/2020/03/10/us/algorithms-learn-our-workplace-biases-can-they-help-us-unlearn-them.html

"Algorithms Learn Our Workplace Biases. Can They Help Us Unlearn Them?"

In the news …

Washington Post, 2015:

¨ The algorithm was given this photo of buildings, left, and a copy of Vincent Van Gogh's "The Starry Night." In about an hour it taught itself to mimic Van Gogh's style, and apply it to the photo of the buildings. (University of Tuebingen)" How? Deep learning.

In the news …

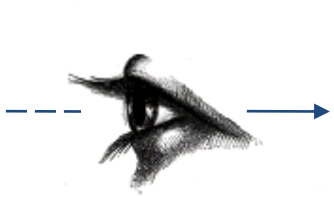
Jan 2018: Google arts & culture app

What is an algorithm?

- Description of a problem and expected input and output
- Solution to the problem; method of providing required output for every valid input

In my field… Computational neuroscience

Brain receives input, processes information, and computes outputs. What algorithms does the brain use??



What is an algorithm?

Marr's (1982) three levels of analysis

The three levels at which any machine carrying out an Information processing task must be understood

1. Computational theory: goal of the computation; why is it important…?

2. Representation and algorithm: How can this computational theory be solved? In particular, what is the representation for the input and output, and what is the algorithm for the transformation?

3. Hardware implementation: How can the representation and algorithm be realized physically?

What is an algorithm?

- Description of a problem and expected input and output
- Solution to the problem; method of providing required output for every valid input

Correct algorithm solves the problem properly for every Possible input

Incorrect algorithm doesn't stop for every input or provides wrong output

What is an algorithm?

- Description of a problem and expected input and output
- Solution to the problem; method of providing required output for every valid input

Is there a unique algorithm for solving each problem?

What is an algorithm?

Correct algorithm solves the problem properly for every possible input

Incorrect algorithm doesn’t stop for every input or provides wrong output

Do we always require correct answer?

What is an algorithm?

Tradeoffs are…

- Correctness
- Efficiency (Why do we care?)

What is an algorithm?

Tradeoffs are…

- Correctness
- Efficiency (Why do we care?)

  Computers are getting faster but…

  Big data (genome; image web search; social networks; etc)

What is an algorithm?

Tradeoffs are…

- Correctness
- Efficiency (Do we need to be efficient for every possible input?)
- Deterministic versus probabilistic. Modern algorithms often probabilistic…

What is an algorithm?

Tradeoffs are…

- Correctness
- Efficiency (Do we need to be efficient for every possible input?)
- Deterministic versus probabilistic. Modern algorithms often probabilistic…
- “Hard” problems

Steps in solving a problem?

- Determine input and output
- Define abstract mathematical model to represent the problem
- Find an algorithm to solve the problem

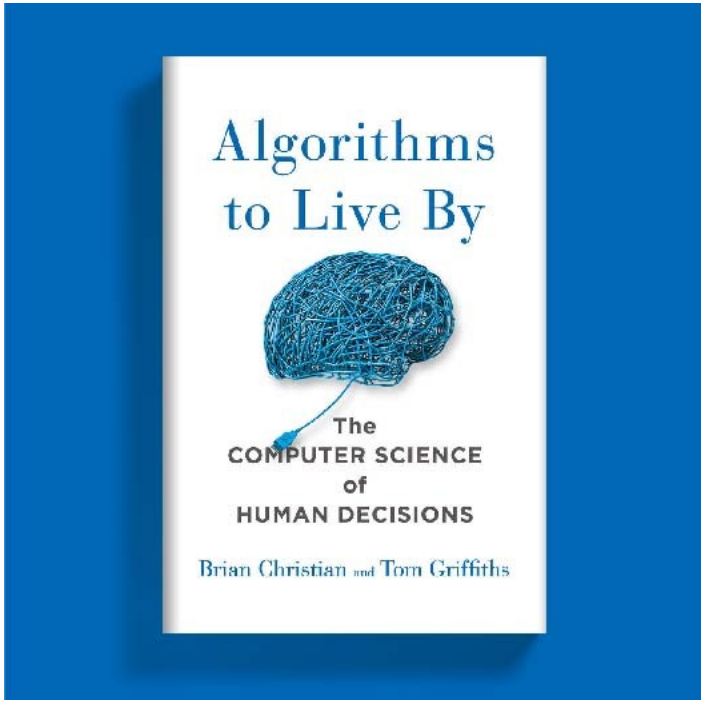
  High level description

  More detailed description (Pseudo code; data structures)
- Tests and proof of correctness
- Analysis of efficiency, run time, complexity…

CSC317 Main topics

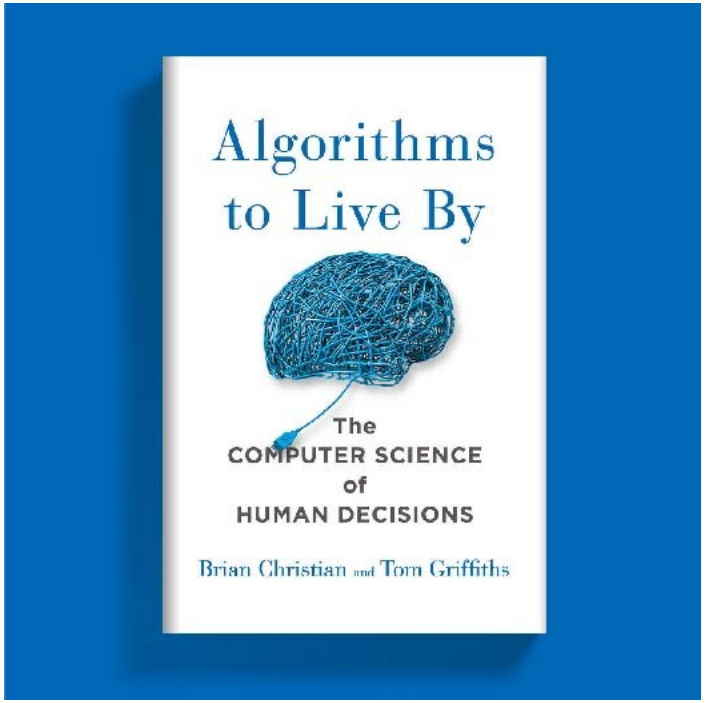
- Introduction to algorithms
- Sorting as example
- Correctness
- Growth of functions and Big-Oh notation
- Divide and conquer and solving recursion equations
- Randomized algorithms
- Introduction/review of data structures
- Hashing
- Trees and Red Black Trees
- Algorithmic paradigms: Dynamic programming, Greedy algorithms
- Graph Algorithms
- Some recent/modern applications

Sorting



“The roomate pulled a sock out of the clean laundry hamper. Next he pulled another sock out at random. If it didn’t match the first one, he tossed it back in. Then he continued this process, pulling out socks one by one and tossing them back in until he found a match for the first… It was enough to make any budding computer scientist request a room transfer…”

Sorting



“With just 10 different pairs of socks … on average 19 pulls merely to complete first pair, and 17 more pulls to complete the second … in total … fishing in the hamper 110 times just to pair 20 socks”

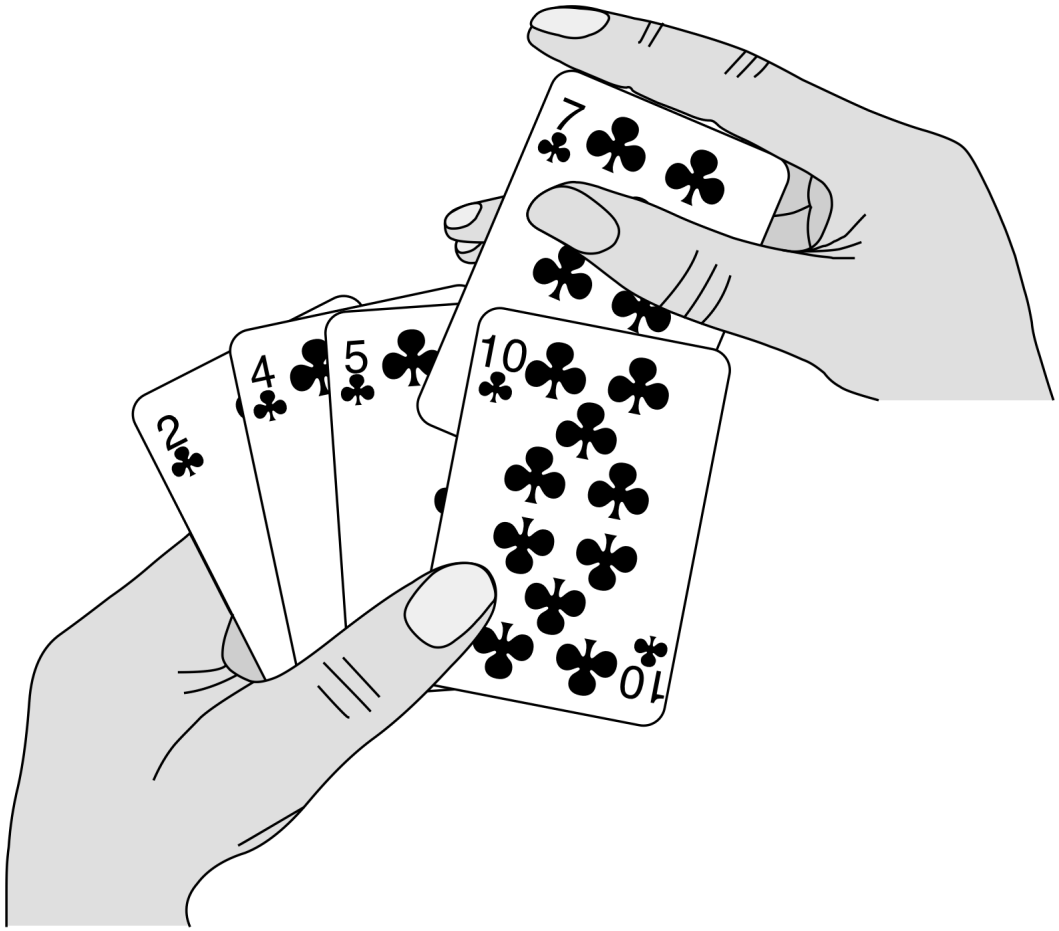
Sorting

To introduce some of the concepts

Why sort?

Sorting as example: Insertion sort

- Input: n numbers
- Output: sorted numbers, e.g., in increasing order



Sorting as example: Insertion sort

Animation example:

http://cs.armstrong.edu/liang/animation/web/InsertionSort.html

Sorting as example: Insertion sort

- Input: n numbers
- Output: sorted numbers, e.g., in increasing order

Pseudo code: (book; more formal version)

```
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
```

Sorting as example: Insertion sort

- Input: n numbers
- Output: sorted numbers, e.g., in increasing order

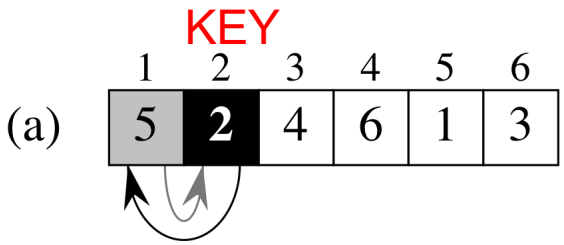
Pseudo code: (high level)

1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Sorting as example: Insertion sort

Pseudo code:

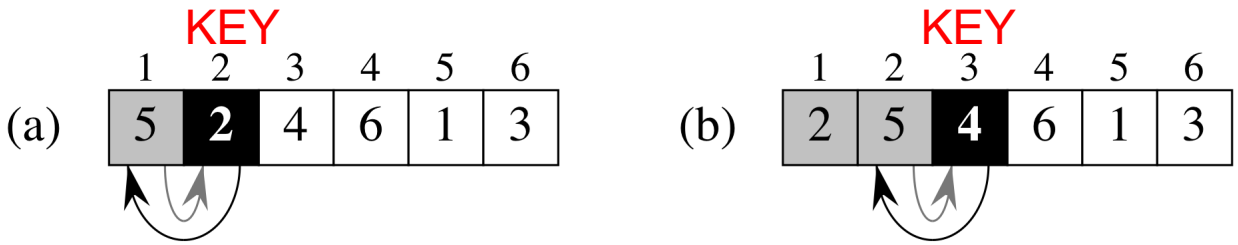
1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position



Sorting as example: Insertion sort

Pseudo code:

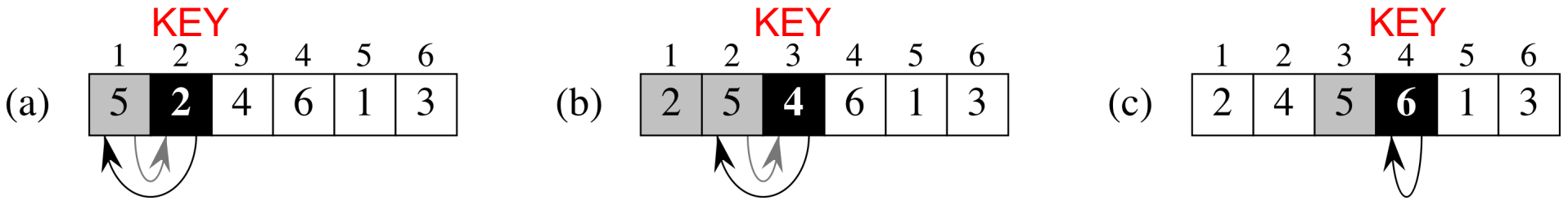
1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position



Sorting as example: Insertion sort

Pseudo code:

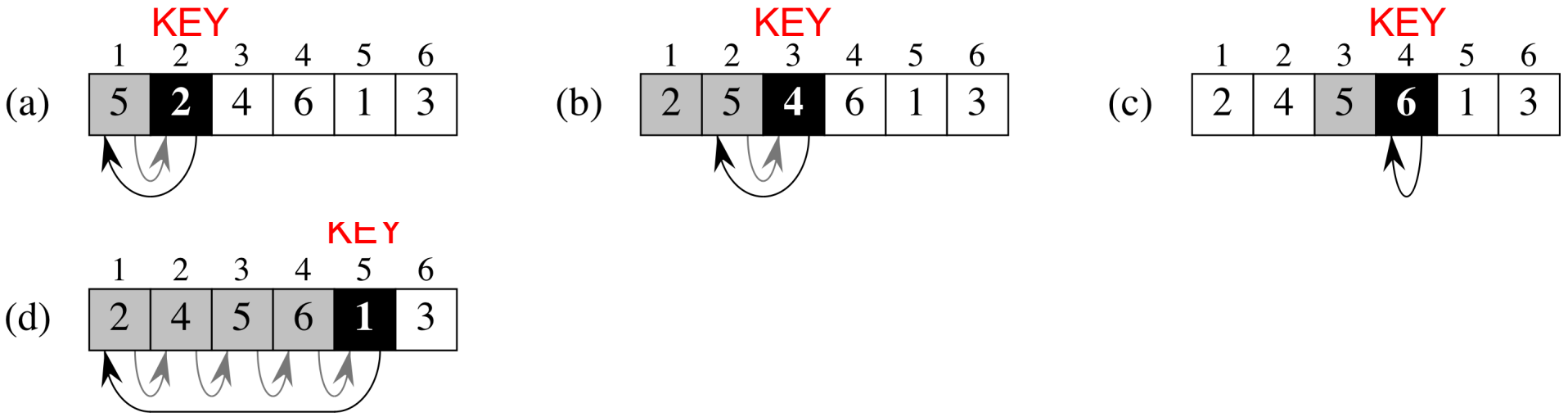
1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position



Sorting as example: Insertion sort

Pseudo code:

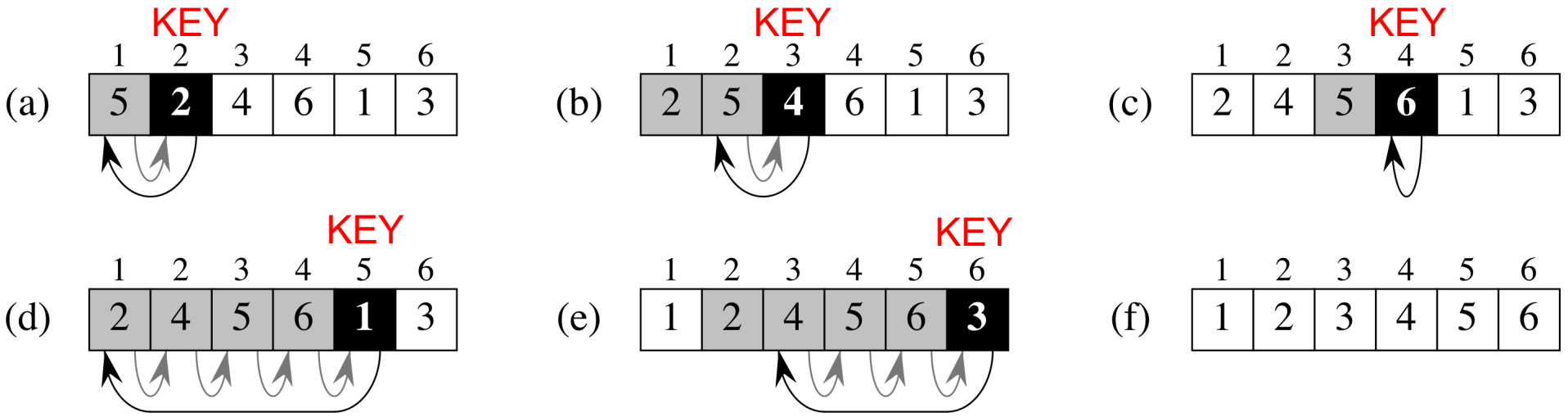
1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position



Sorting as example: Insertion sort

Pseudo code:

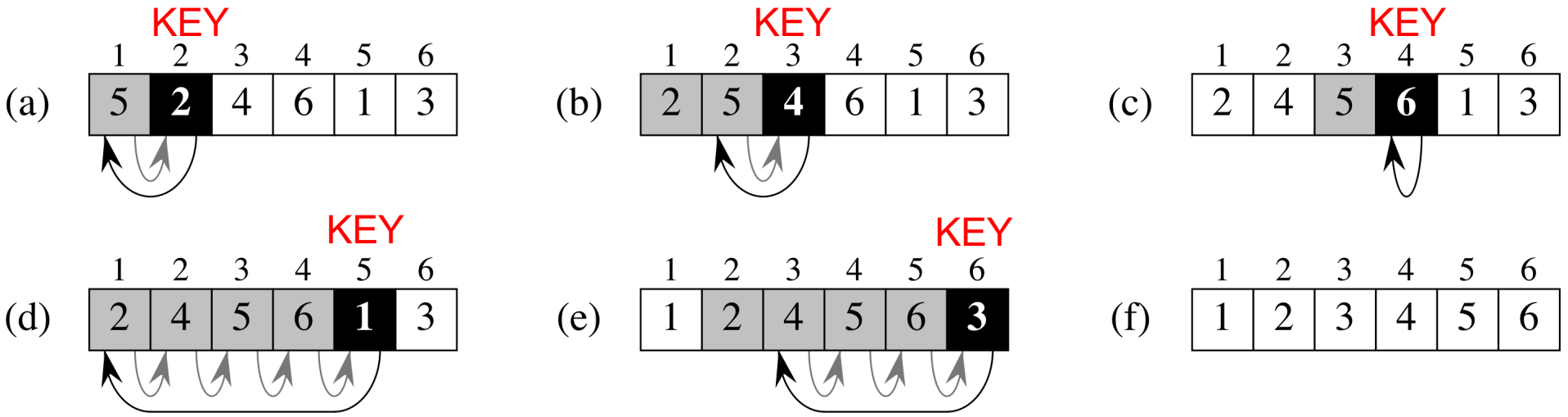
1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing pairs and swapping into correct position



Sorting as example: Insertion sort

Pseudo code:

1. For j = 2 to n
2. Key = A[j];
3. Insert Key into sorted array A[1 .. j-1]
   by comparing pairs and swapping into correct position

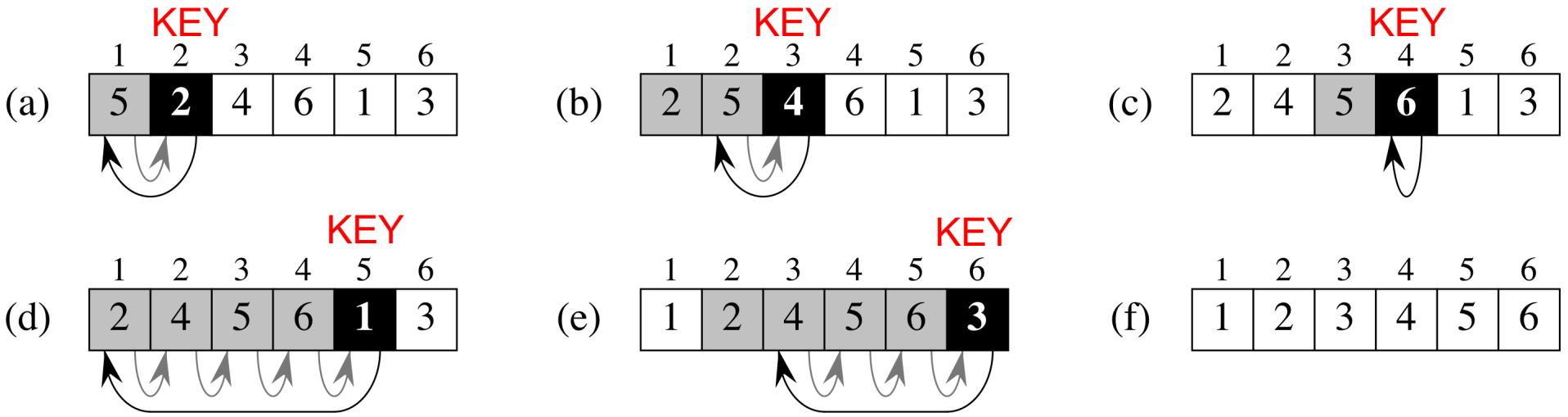


What is striking about the gray boxes??

Sorting as example: Insertion sort

Pseudo code:

1. For j = 2 to n
2. Key = A[j];
3. **Insert Key into sorted array A[1 .. j-1]**
   by comparing pairs and swapping into correct position

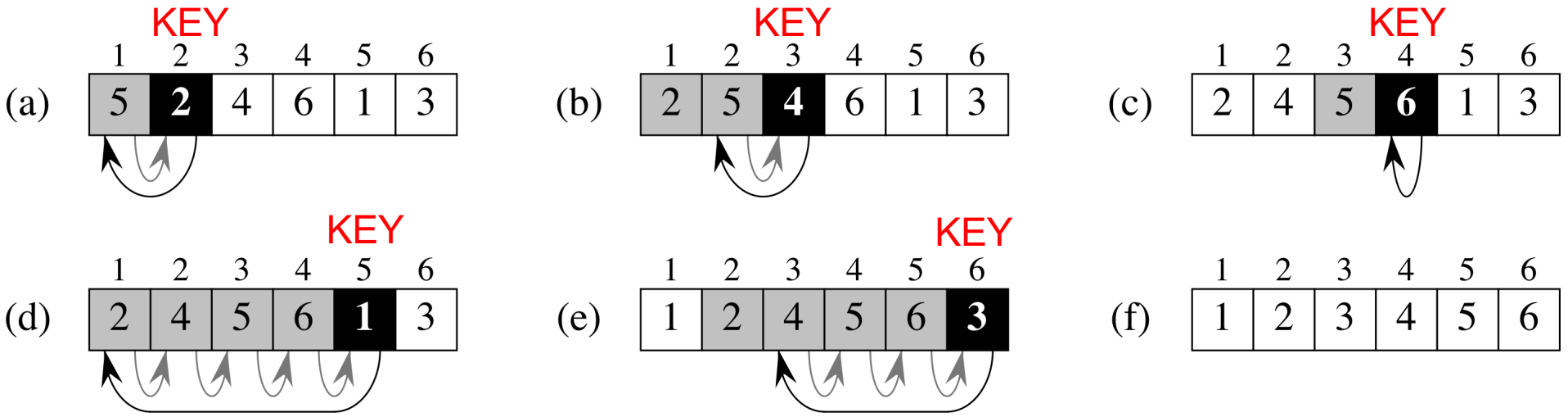


Sorting as example: Insertion sort

Pseudo code:

1. For j = 2 to n
2. Key = A[j];
3. **Insert Key into sorted array A[1 .. j-1]**
   by comparing pairs and swapping into correct position

We will come back to this later when we talk about loop invariants and proving correctness



Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

If there are n inputs, what is the cost to perform this algorithm?

Poll: Cost of insertion sort

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

We'll go through each statement and calculate costs

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Cost?

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

$c_1 n$

We'll usually ignore constants… "grows like n"

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

$c_2 n$

We'll usually ignore constants… "grows like n"

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Cost?

Best case?

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Best case: Already sorted (How many comparisons and swaps?)
Input: [1 2 3 4 5 6]
Output: [1 2 3 4 5 6]

Cost (why?)

$c_3 n$

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Best case total cost:

$$T(n) = c_1 n + c_2 n + c_3 n$$

We'll usually ignore constants… "grows like n"

Insertion sort: analysis of run time

For j = 2 to n
   Insert A[j] into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

But…

We are almost never handed a best case (eg, deck of cards)!

Insertion sort: analysis of run time

For j = 2 to n
   Insert A[j] into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case?

Insertion sort: analysis of run time

For j = 2 to n
   Insert A[j] into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case? In reverse order.

Input: [6 5 4 3 2 1]
Output: [1 2 3 4 5 6]

How many comparisons/swaps?

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost:

$$T(n) = \sum_{j=2}^{n}$$ number of comparisons/swaps for j

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost (why?):

$$T(n) = \sum_{j=2}^{n} (j-1) =$$

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost (why?):

$$T(n) = \sum_{j=2}^{n} (j-1) =$$

What kind of series is this??

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost (why?):

$$T(n) = \sum_{j=2}^{n} (j-1) =$$

What kind of series is this??

Arithmetic series!

Arithmetic series

$$\sum_{j=1}^{n} j = 1+2+3+ \ldots n = ?$$

Arithmetic series

$$\sum_{j=1}^{n} j = 1+2+3+ \ldots n = ?$$

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

Wolfram: " … trick Gauss used as a school boy to solve sum of 1 to 100 … classmates toiled away … when the answers were examined, Gauss's proved to be correct…"

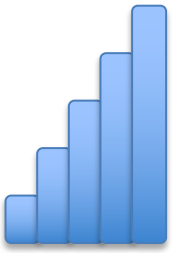
Arithmetic series

$$\sum_{j=1}^{n} j = 1+2+3+ \ldots n = ?$$

Multiplied by number of elements

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

Sum of first times last divided by 2



1 2 n

Back to our question about Insertion Sort…

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost:

$$T(n) = \sum_{j=2}^{n}$$ number of comparisons/swaps for j

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost (why?):

$$T(n) = \sum_{j=2}^{n} (j-1) =$$

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost. Note that (why?):

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2};$$

Arithmetic series

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1;$$

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost:

$$T(n) = \sum_{j=2}^{n} (j-1) = \frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

Worst case cost:

$$T(n) = \sum_{j=2}^{n} (j-1) = \frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Whenever we see two loops, one for j going through all or nearly all elements (here 2 to n), and the other over 1 to j-1 (or j+1 to n!) — we are in the n squared regime

Insertion sort: analysis of run time

1. For j = 2 to n
2. Key = A[j]
3. Insert Key into sorted array A[1 .. j-1]
   by comparing and swapping into correct position

WORST case total:

$$T(n) = c_1 n + c_2 n + c_3 (\frac{n^2}{2} - \frac{n}{2})$$

We'll usually ignore constants and lower order terms for the form $an^2 + bn + c$ (why?)

grows like n^2

Insertion sort: analysis of run time

For j = 2 to n
   Insert A[j] into sorted array A[1 .. j-1]
   by swapping into correct position

- Best case grows like n
- Worst case grows like n squared

Is average case more like best or worst case? Why?

Insertion sort: analysis of run time

- We've slightly simplified notation from book – don't care if repeated 2 or 3 constant times…
- We only care about limiting step relative to input size n – ignore constant and lower order terms
- We usually care about worst case scenario
- Average case often roughly as bad as worst case
- More on "grows like" later…