# CSE 548: (*Design and*) Analysis of Algorithms

Fall 2017

R. Sekar

# Topics

# The topic of this course . . .

- Theoretical study of correctness and performance of (abstract) programs
  - Performance = run time + memory use
- Questions answered
  - How to design algorithms that perform better
  - How to prove their correctness
  - How to analyze their performance
- *Note: performance is often less important than:*
  - Functionality
  - Correctness, reliability
  - Programmer effort, maintainability, extensibility, modularity, simplicity

# Administrivia

- Course web page
  - http://seclab.cs.sunysb.edu/sekar/cse548/
    - Redirected from the department page for CSE 548
  - General information, lecture schedule and notes, etc.
- Blackboard/Piazza
  - All important announcements
  - Handouts, assignment submission
  - Discussion forum and emails

# Texts and References

- *Required:* "Algorithms,"
  *Dasgupta, Papadimitriou and Vazirani,* McGraw-Hill
  - This is the main text book
- *Online:* "Mathematics for Computer Science," *Lehman and Leighton*
  - We will cover selected background topics from this book.
- References and alternate texts
  - *Online:* "Algorithms," Jeff Erickson
  - "Algorithm Design,"
    *Kleinberg and Tardos,* Addison-Wesley
  - "Introduction to Algorithms,"
    *Cormen, Leisorson, Rivest and Stein,* MIT Press

# How the course is run

- A homework handed out every week, submissions due every other week
  - Assignments can be completed in groups of 2 to 4, but only one submission per group.
  - May include programming problems from time to time, but these are not for submisson.
- Two midterms and a final
- Grading (approximate)
  - 70% exams
  - 30% homework quizzes, homework assignments, etc.

# Academic Integrity

- Do not copy from any one, or any source (on the Internet or elsewhere)
- The penalty for cheating is an F-grade, plus referral to graduate school. *No exception,* regardless of the "amount" of copying involved.
- In addition, if you cheat, you will be unprepared for the exams, and will do poorly.
- To encourage you to work on your own, we scale up assignment scores by about 30% to a maximum 100%

# Representative Problem

## Stable Marriage

Find a "stable" pairing of men and women such that no one will leave their spouse for another's spouse.

- Participants rank members of opposite sex.
- Each man lists women in order of preference
- Each woman lists men in order of preference

Reference: [Kleinberg and Tardos]

Slides courtesy of Kevin Wayne (http://www.cs.princeton.edu/~wayne/kleinberg-tardos/)

# Understanding the solution

- Do solutions always exist?
  - Not intuitively obvious
  - but we just proved that Gale-Shapley algorithm always finds a solution!
- Are solutions unique?
  - If not, what are the characteristics of the solution found by [GS 1962]?

# Insertion Sort

**procedure** *InsSort*(**int** $A[n]$, **int** $n$)

```
for (j = 1; j < n; j++)  /*Invariant: A[0...j-1] is sorted (ascending)*/
    i = j - 1;
    key = A[j];
    while (i ≥ 0 && A[i] > key)
                /*A[i...j-1] < key, location A[i+1] is "free" */
        A[i+1] = A[i];
        i = i - 1;
    A[i+1] = key;
```

# Running time of Insertion Sort

- Given by a function $T(n)$ where $n$ is the array size
- Outer loop is executed $n$ times
- For the $j^{th}$ iteration of outer loop, the inner loop is executed at most $j$ times
- Runtime

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

- Often, we focus on *asymptotic complexity:* a function that matches the growth rate of $T(n)$ as $n \to \infty$

# Big-$O$ notation

- Expressing complexity in terms of "number of steps" is a simplification
  - Each such operation may in fact take a different amount of time
  - But it is too complex to worry about the details, esp. because they differ across architectures, languages, etc.
- Why not simplify further?
  - Capture just the growth rate of $T(n)$ as a function of $n$
  - Ignore constant factors (that tend to depend on machine specifics)
    - Don't have to count the number of operations in a loop, as long as they don't grow with input size
  - Ignore exceptions that occur for small values of $n$

# Big-$O$ notation (2)

## Definition

*Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. We say $f \in O(g)$ (which means that "f grows no faster than g") if there is a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$.*

*Notation abuse:* We often use "=" instead of $\in$.

## Examples

- $10n = O(n)$
- $0.0001n^3 + n = O(n^3)$
- $2^n + 10^n + n^2 + 2 = O(10^n)$
- $0.0001n \log n + 10000n = O(n \log n)$

# Types of complexity analyses

- **Worst case:** Bound that applies to any possible input.
  - *Guaranteed performance* — avoids assumptions about input.
- **Average case:** Average performance across expected inputs.
  - Useful sometimes, especially when typical performance much better than worst-case
  - *Use with caution:*
    - Requires details of input distributions that is rarely available
    - Often ends up making unrealistic assumptions or simplifications
- **Best case:** Not useful — unless you are trying to show than an algorithm is bad even in the best of circumstances!

# Big-$O$ versus $\Omega$ and $\Theta$

$10n = O(n^2)$ — Note that $O$ stands for upper bound

## $\Omega$ — Lower Bounds

$$f = O(g) \Leftrightarrow g = \Omega(f)$$

**Examples:**

$n^2 = \Omega(n)$ $\quad$ $n^2 = \Omega(n \log n)$ $\quad$ $2^n \neq \Omega(10^n)$

## $\Theta$ — Tight Bounds

$$f = \Theta(g) \Leftrightarrow f = O(g) \wedge f = \Omega(g)$$

**Example:** $0.2n^2 + 2n + 7 = \Theta(n^2)$

# Proofs

- Proofs arise in many contexts in this course
  - Correctness of algorithm
  - Optimality (or lack there of)
  - Time or space complexity
- What is a proof?
  - Establishes (or disproves) a predicate $P(x)$
    - May be universally ($\forall x\ Q(x)$) or existentially quantified ($\exists x\ Q(x)$)
  - *Explains* why $P(x)$ holds (or why not)
  - Identified using a search
- Often, proofs are difficult to construct (but fun!) because
  - search space is extremely large, and/or
  - lacks any structure

# Proving complexity of *InsSort*

Requires us to show

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

OK, how about

$$\sum_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}$$

or

$$\sum_{j=1}^{n} j^3$$

# Another Example

**Property**

$\forall n\ n^2 + n + 41$ *is a prime number*

# Yet Another Example

**Property**

$\forall n > 2\ a^n + b^n = c^n$ *has no integer solutions*

# Yet Another Example

**Property**

$\forall n > 2\ a^n + b^n = c^n$ *has no integer solutions*

- "Fermat's Last Theorem" — stated by Fermat in 1637 in the margin of a copy of Arithmetica
- Fermat said he had a proof, but it won't fit in the margin!
- Remained one of the most famous open mathematical problems for over 350 years! Finally proved in 1995 by Wiles.

# Proof Techniques and Strategies

- We have seen several techniques so far
  - identify and exploit some specific structure (pairing of $i$ and $n - i$)
  - geometric interpretation
  - factorization

  But this list can be endless
- Proof strategies are more limited in number
  - proof by contradiction
  - disproof by counter-example
    - but not proof by example!
  - induction
  - diagonalization

# Proof by Induction

Requires us to show

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

**Base:** For $j = 1$, easy to check that $1 = 1 * (1+1)/2$

**Induction hypothesis:** Assume that the equality holds for $k < n$

**Induction Step:**

# Proof by Induction

**Theorem**

*All horses have the same color*

**Base:** Trivial, as there is a single horse.

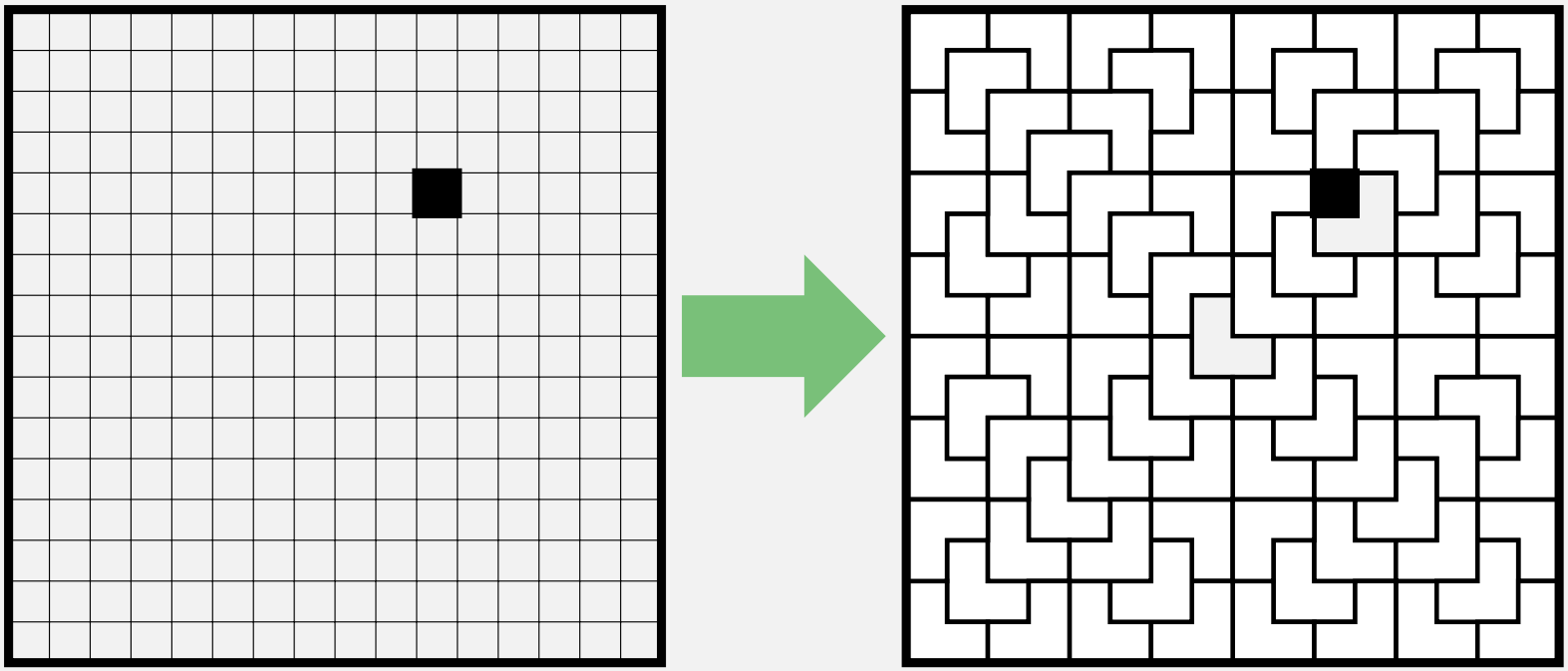
**Induction hypothesis:** All sets of horses with $n$ or fewer horses have the same color.

**Induction Step:** Consider a set of $h_1, h_2, \ldots, h_{n+1}$. By induction hypothesis:

$$\underbrace{h_1, h_2, \ldots, h_n}_{\text{same color}}, h_{n+1} \qquad h_1, \underbrace{h_2, \ldots, h_n, h_{n+1}}_{\text{same color}}$$

This obviously means that all $n+1$ horses have the same color!

# Tiling a $2^n \times 2^n$ board with Triominos



**Theorem**

*Any $2^n \times 2^n$ checkerboard with any single square removed can be tiled using L-shaped triominos.*

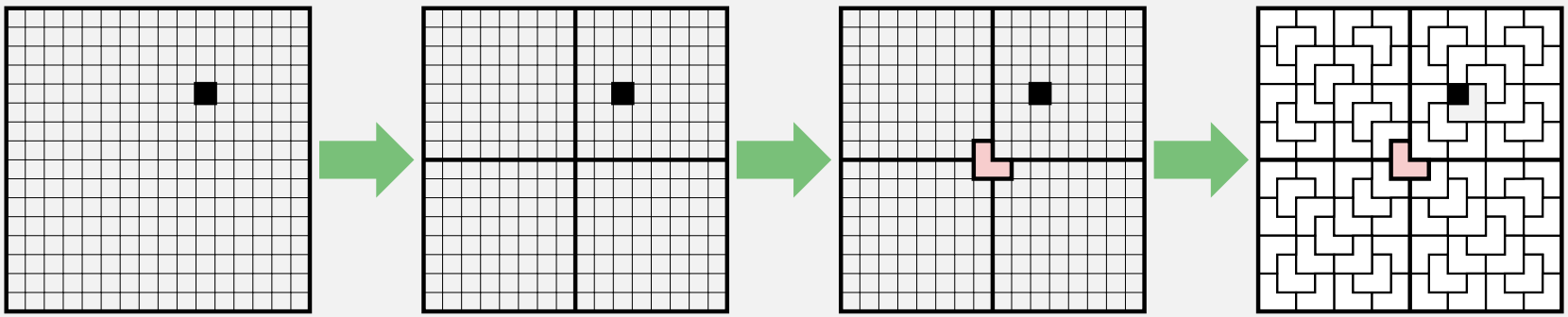
Figures/text from Jeff Erickson's "Algorithms"

# Tiling a $2^n \times 2^n$ board with Triominos

**Proof by top-down induction:** Let $n$ be an arbitrary non-negative integer. Assume that for any non-negative integer $k < n$, the $2^k \times 2^k$ grid with any square removed can be tiled using triominos. There are two cases to consider: Either $n = 0$ or $n \geq 1$.

- The $2^0 \times 2^0$ grid has a single square, so removing one square leaves nothing, which we can tile with zero triominos.
- Suppose $n \geq 1$. In this case, the $2^n \times 2^n$ grid can be divided into four smaller $2^{n-1} \times 2^{n-1}$ grids. Without loss of generality, suppose the deleted square is in the upper right quarter. With a single L-shaped triomino at the center of the board, we can cover one square in each of the other three quadrants. The induction hypothesis implies that we can tile each of the quadrants, minus one square.

In both cases, we conclude that the $2^n \times 2^n$ grid with any square removed can be tiled with triominos. □



# Recursion and Iteration Vs Induction

- Inductive proofs are often used in the context of recursive (and sometimes iterative) programs
- The recursive case closely resembles the inductive step in what we may call as top-down induction.
- Iterative programs are typically more closely related to bottom-up inductive proofs.

# Proof techniques for professors

- *by obviousness:* "It is too obvious to waste our time with the details ..."
- *by omission:* "Proof is easy and left as an after-class exercise..."

  "The other 100 cases are similar"
- *by lack of interest:* "Does anyone really want to see this?"
- *by lack of time:* "We don't have time to do this in class today ..."
- *by exhaustion:* Spend most of a lecture on background supposedly needed for a proof ...
- *by accumulated evidence:* "Long and diligent search has not revealed a counterexample."
- *by funding:* How could three different government agencies be wrong?
- *by asserting intellectual superiority:* "You don't have the background for the proof ..."
- *by authority:* "I saw Fermat in the elevator and he said he had a proof ..."
- *by intimidation:* "Don't be stupid; of course it's true!"
- *by terror:* When intimidation fails...

[1] Derived from a google search

# Proof techniques for students

- *by example:* Provide one example, and claim that the ideas hold for all cases. Frequently used for partial credit in exams.
- *by picture:* A more convincing form of proof-by-example.
- *by profusion of adjectives and adverbs:* "As is quite clear, the elementary aforementioned statement is obviously valid."
- *by vigorous handwaving:* For seminar settings, esp. if the presenter exudes supreme confidence ...
- *by cumbersome notation:* $\forall \varpi \in \mathfrak{P}\ \exists \nu \in \aleph\ \mathcal{P}(\varpi) \nrightarrow \overset{\curvearrowleft}{\mathcal{Q}}(\nu)$ — works well in journal papers
- *by throwing in the kitchen sink:* Write down all proofs vaguely related to the problem. A form of proof-by-exhaustion that is popular in exams.
- *by illegibility:* Combines well with many other techniques in exams
- *by mutual reference:* In reference A, Theorem 5 is said to follow from Theorem 3 in reference B, which is shown to follow from Corollary 6.2 in reference C, which is an easy consequence of Theorem 5 in reference A.