Welcome to
CSEP 521 - Applied Algorithms

Lecturer: Tami Tamir (tami@cs)
Office Hours: Mon. 9:20 - 10:00 pm
and by appointment

TA: David Richardson (daverich@cs)
Office Hours: Mon. 5:20 - 6:30 pm

Administrivia

- Books
  - The Algorithm Design Manual. Steven Skiena
  - Introduction to Algorithms. Cormen, Leiserson, and Rivest (CLR)
- Most important resource: course web page
  - http://www.cs.washington.edu/education/courses/csep521
- Papers and sections from other Books.
- Grading:
  - Weekly problem sets - 50% (average of n-1 best).
  - Final exam - 50%

Course Goals

- An appreciation for applications of algorithmic techniques in the real world.
- A larger toolbox.
- A better sense of how to model problems you encounter as well-known algorithmic problems.
- A deeper understanding of the issues and tradeoffs involved in algorithm design.
- Fun!! Beauty!!!

Some Topics

- Graph algorithms.
- Analysis of algorithms
- Data structures
- NP-completeness
- Dynamic programming
- Linear Programming
- Greedy algorithms
- On-line algorithms
- Approximation algorithms
- Scheduling and Resource Allocation

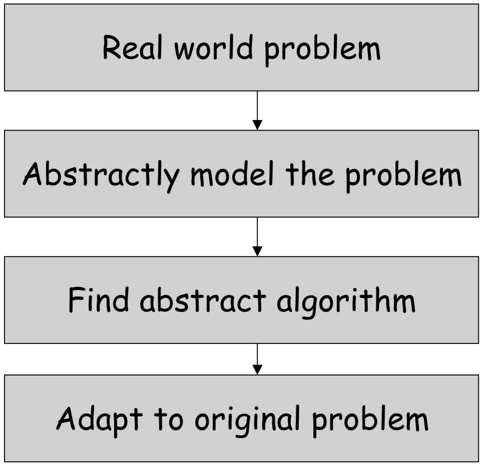
Other

- Interested in suggestions, feedback, constructive criticism. (E.g., topics/depth vs. breadth)
- Ask lots of questions! Participate! Offer opinions!

Plan For Today

- Introduction - Skiena, chapter 1
- Cool lectures from CMU
  - intro to “great theoretical ideas in computer science”
  - the stable marriage problem

Applied Algorithm Scenario



Real world problem

Abstractly model the problem

Find abstract algorithm

Adapt to original problem

Modeling

- What kind of algorithm is needed
  - Sorting or Searching
  - Graph Problem
  - Linear Programming
  - Dynamic Programming
  - Clustering
  - Algebra
- Can I find an algorithm or do I have to invent one

Algorithm Design Goals

- Correctness
- Efficiency
- Simple, if possible.

Evaluating an algorithm

Mike: My algorithm can sort 10^6 numbers in 3 seconds.

Bill: My algorithm can sort 10^6 numbers in 5 seconds.

Mike: I've just tested it on my new Pentium IV processor.

Bill: I remember my result from my undergraduate studies (1985).

Mike: My input is a random permutation of $1..10^6$.

Bill: My input is the sorted output, so I only need to verify that it is sorted.

Types of complexity

- We should analyze separately 'good' and 'bad' inputs.
- Processing time is surely a bad measure!!!
- We need a 'stable' measure, independent of the implementation.

* A complexity function is a function $T: N \rightarrow N$.
  T(n) is the number of operations the algorithm does on an input of size n.

* We can measure three different things.

- Worst-case complexity
- Best-case complexity
- Average-case complexity

The RAM Model of Computation

- Each simple operation takes 1 time step.
- Loops and subroutines are not simple operations.
- Each memory access takes one time step, and there is no shortage of memory.

For a given problem instance:

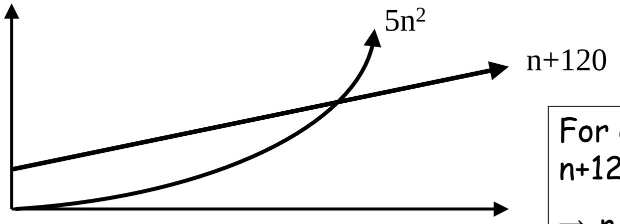
- Running time of an algorithm = # RAM steps.
- Space used by an algorithm = # RAM memory cells

useful abstraction $\Rightarrow$ allows us to analyze algorithms in a machine independent fashion.

Big O Notation

- Goal :
  - A stable measurement independent of the machine.
- Way:
  - ignore constant factors.
- f(n) = O(g(n)) if c·g(n) is upper bound on f(n)

⇔ There exist c, N, s.t. for any $n \geq N$, $f(n) \leq c \cdot g(n)$



For all $n \geq 5$
$n+120 \leq 5n^2$

$\Rightarrow n+120 = O(n^2)$

Ω, Θ Notation

- f(n) = Ω(g(n)) if c·g(n) is lower bound on f(n)

⇔ There exist c, N, s.t. for any $n \geq N$, $f(n) \geq c \cdot g(n)$

- f(n) = Θ(g(n)) if f(n) = O(g(n)) and f(n) = Ω(g(n))

⇔ There exist c_1, c_2, N, s.t. for $n \geq N$,

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Examples:

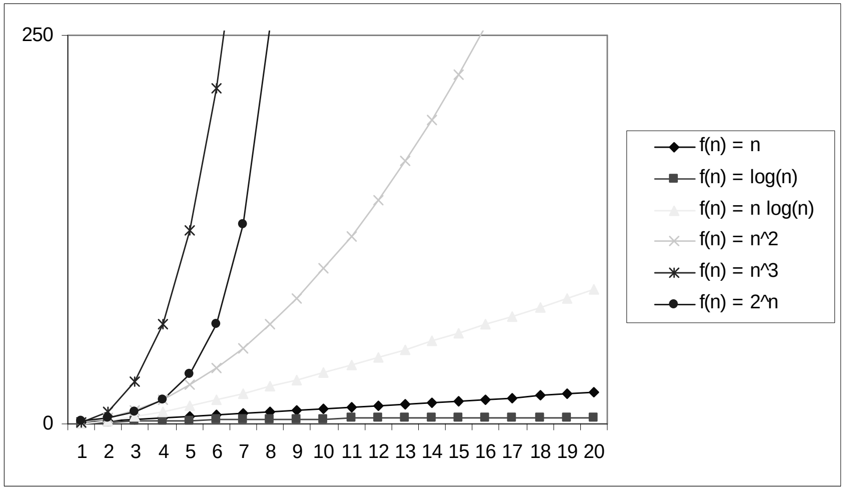
Any positive f is Ω(1)

$4x^2+100 = O(x^2)$	$4x^2+100 \neq \Theta(x^3)$	$123400 = O(1)$
$4x^2+100 = \Omega(x^2)$	$4x^2+100 = O(x^3)$	$4x^2 + x\log x = O(x^2)$
$4x^2+100 = \Theta(x^2)$	$4x^2+100 = \Omega(x)$	$4x^2 - 100 = O(x^2)$

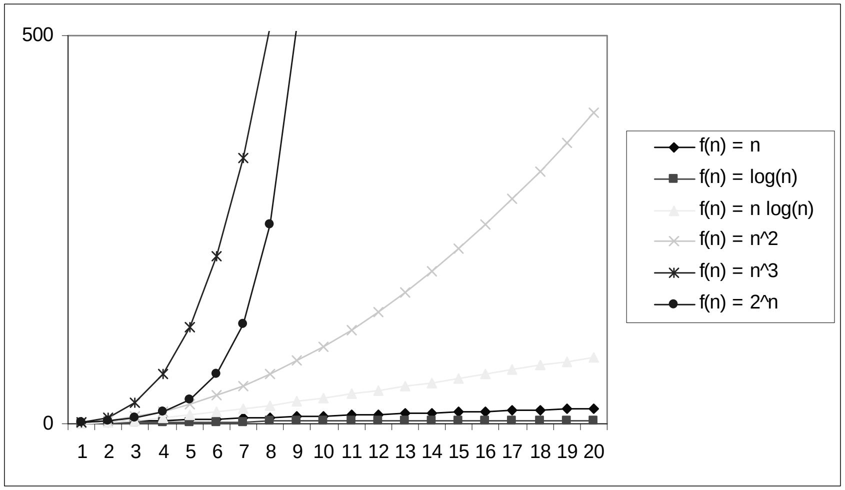
Growth Rates

- Even by ignoring constant factors, we can get an excellent idea of whether a given algorithm will be able to run in a reasonable amount of time on a problem of a given size.
- The “big O” notation and worst-case analysis are tools that greatly simplify our ability to compare the efficiency of algorithms.

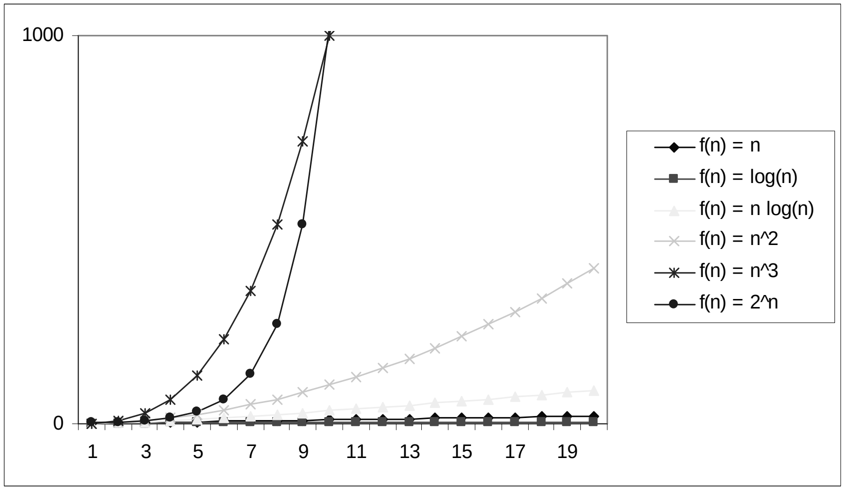
Practical Complexity



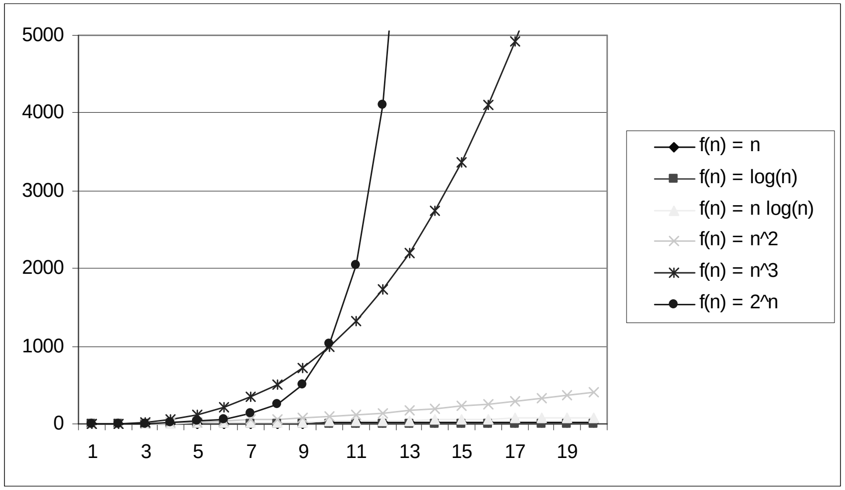
Practical Complexity



Practical Complexity



Practical Complexity



Big O Fact

- A polynomial of degree k is $O(n^k)$
- Proof:
  - Suppose $f(n) = b_k n^k + b_{k-1} n^{k-1} + \dots + b_1 n + b_0$
    - Let $a = \max_i \{b_i\}$
  - $f(n) \leq an^k + an^{k-1} + \dots + an + a$

    $\leq kan^k \leq cn^k$ (for c=ka).

Next Bunch of Slides Taken From:

Great Theoretical Ideas In Computer Science

CS 15-251

Professors
Steven Rudich and Bruce Maggs
Carnegie Mellon University

The future belongs to the computer scientist who has

- Content: An up-to-date grasp of fundamental problems and solutions
- Method: Principles and techniques to solve the vast array of unfamiliar problems that arise in a rapidly changing field

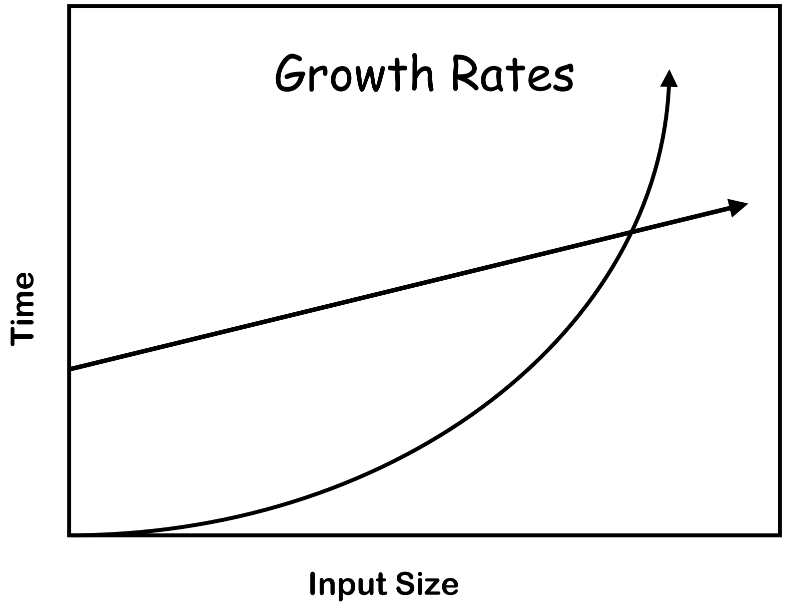
Course Content

- A survey of fundamental theoretical ideas in Computer Science as they occur in a variety of important applications.
- An introduction to discrete mathematics, graph algorithm, and other theoretical tools.
- Effective problem solving techniques.

A survey of fundamental theoretical ideas in Computer Science as they occur in a variety of important applications

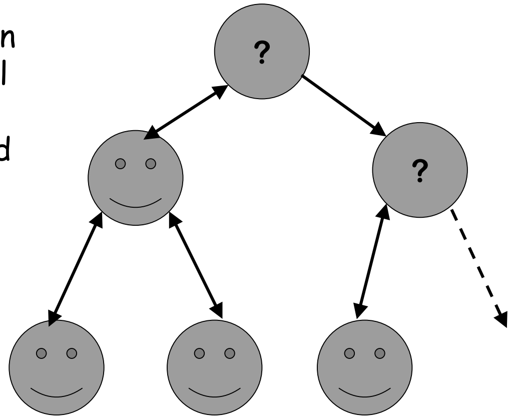
For example . . .

Growth Rates

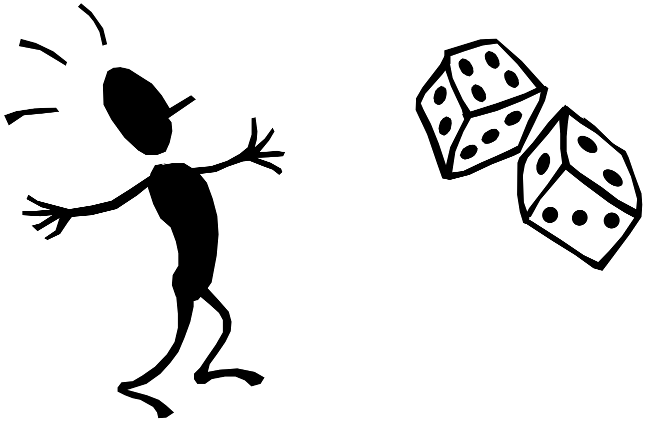


Recursion

Recursion is an important tool in both developing and analyzing of algorithms



Randomization



Use of Randomization: Example

We are given a binary array of length 2n. Either all the 2n entries are ‘1’s, or n are ‘0’ and n are ‘1’.

How can we distinguish between the two cases?

Solution 1: Read the entries, until you cross a ‘0’ or until you see n+1 ‘1’s.

Worst case analysis: In the first case, or if all the ‘0’s are in the second part of the array: n steps.

1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

Use of Randomization: Example

Solution 2: Randomly select two entries. If they are both '1' answer **all '1'**. Otherwise answer **half-half**.

								1								0		**half-half**
					0								0					**half-half**
					1								1					**All 1**
1	0	1	1	0	1	0	0	0	0	1	1	0	1	1	1	0	1	**Could be wrong**

Repeat more than once to increase your confidence.

					1				1				1				1	**All 1**

Can still be wrong but with lower probability

Use of Randomization: Example

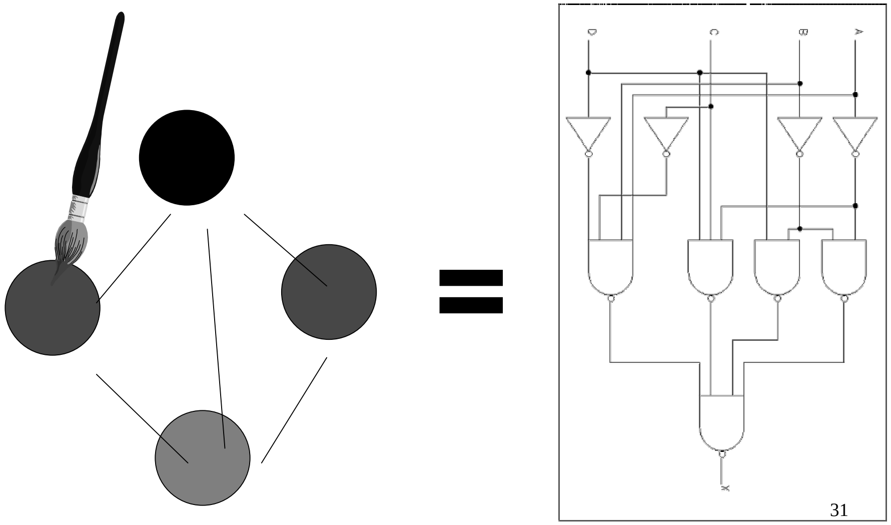
Q: Why is it different than answering 'All 1' after seeing 2,4,6 or more '1's in solution 1?

A: We are 'playing' against an adversary who knows our algorithm and can pick the array

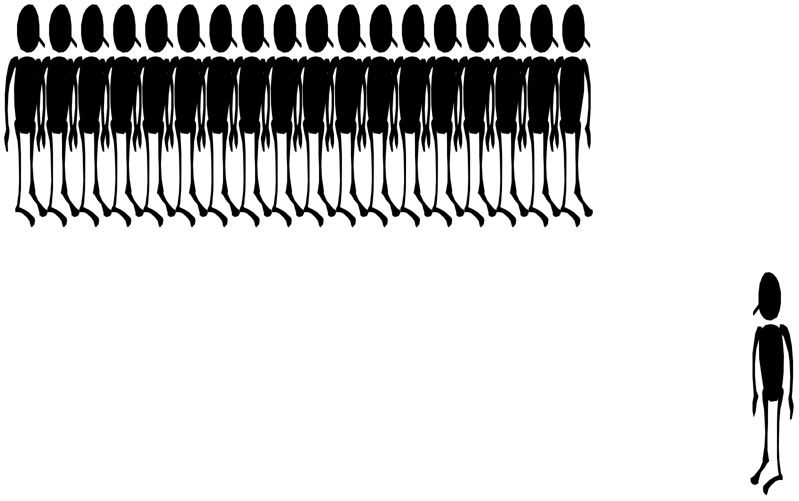
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

to fool us. An adversary who knows our randomized algorithm can not pick a 'bad' input in advance.

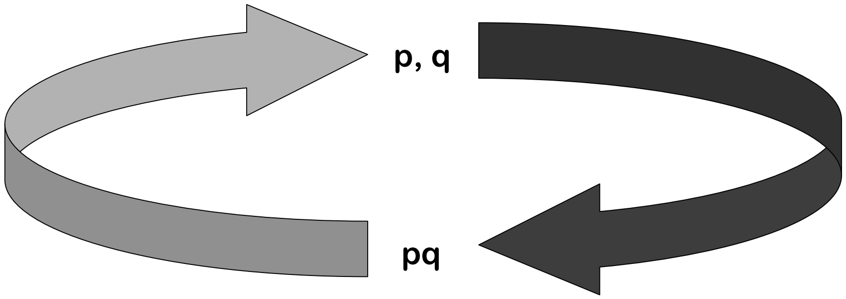
Reduction



Parallel Versus Sequential Work



Factoring / Multiplication

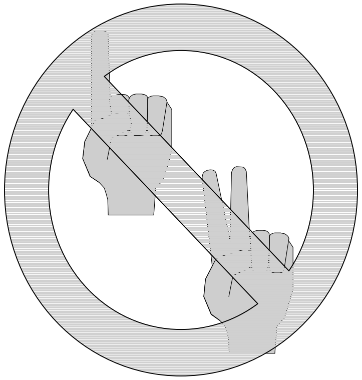


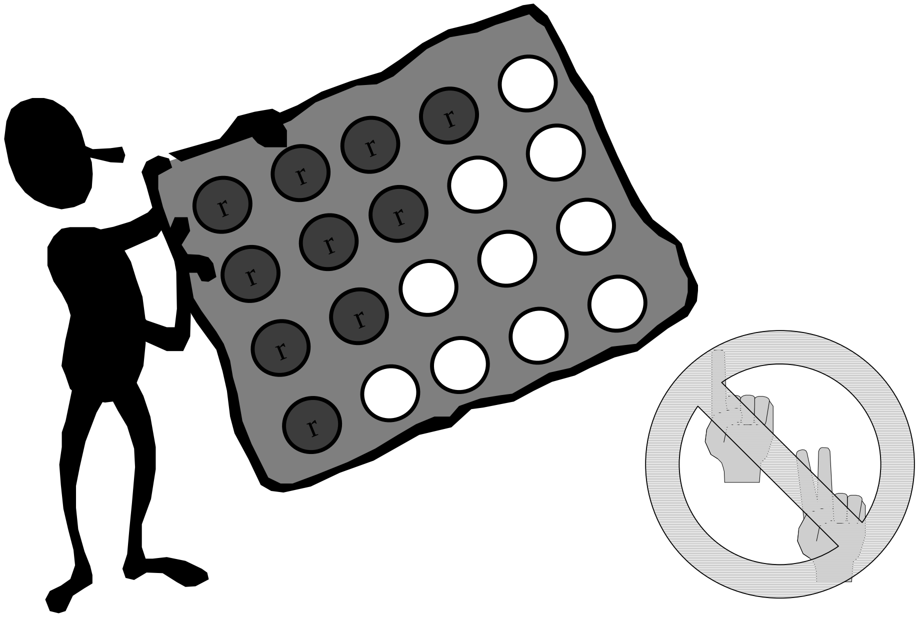
It is easy to multiply two primes.

Very difficult to retrieve the primes composing the result.

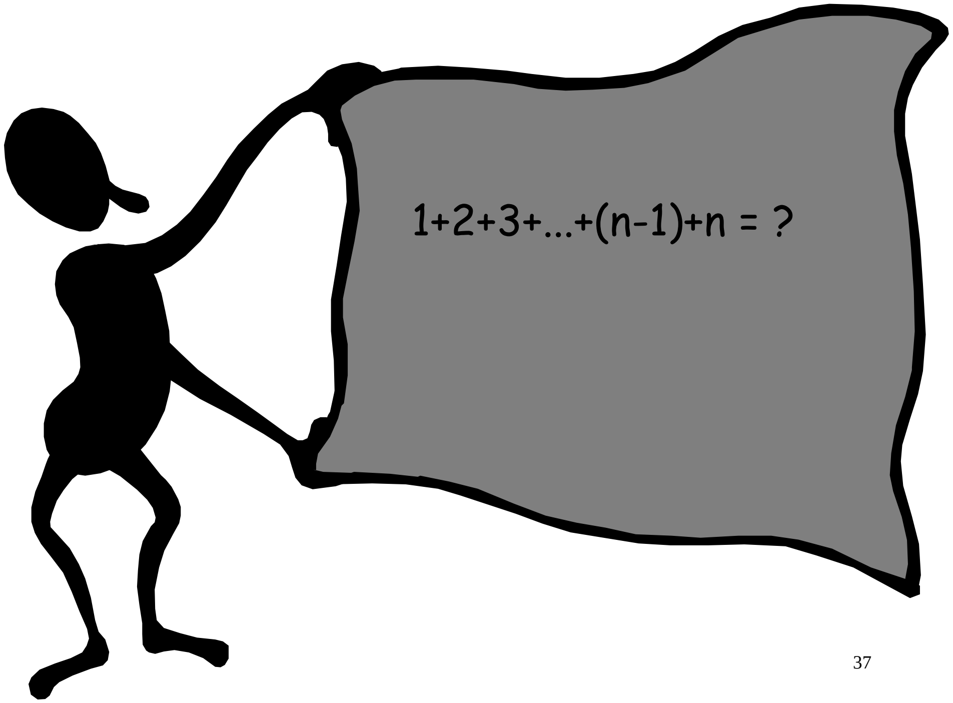
An introduction to discrete mathematics

Count without counting: Estimate, Calculate, Analyze





How many red dots on this page?



$1+2+3+\ldots+(n-1)+n = ?$

$$\begin{array}{cccccccccccccc}
1 & + & 2 & + & 3 & + & \ldots & + & n-1 & + & n & = & S \\
n & + & n-1 & + & n-2 & + & \ldots & + & 2 & + & 1 & = & S \\
\hline
(n+1) & + & (n+1) & + & (n+1) & + & \ldots & + & (n+1) & + & (n+1) & = & 2S \\
 & & & & & & & & & & n(n+1) & = & 2S
\end{array}$$

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

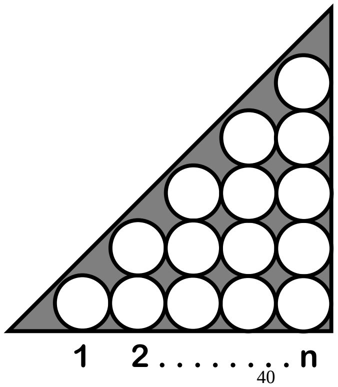
$$\left.\begin{array}{cccccccccccccc}
1 & + & 2 & + & 3 & + & \ldots & + & n-1 & + & n & = & S \\
n & + & n-1 & + & n-2 & + & \ldots & + & 2 & + & 1 & = & S \\
 & & & & & & & & & & & & \\
(n+1) & + & (n+1) & + & (n+1) & + & \ldots & + & (n+1) & + & (n+1) & = & 2S
\end{array}\right\}\text{Algebraic argument}$$

$$n(n+1) = 2S$$

Let's restate this argument using a *geometric* representation

$\boxed{1 + 2 + 3 + \ldots + n-1 + n = S}$ = number of white dots.

$$\begin{array}{cccccccccccccc}
n & + & n-1 & + & n-2 & + & \ldots & + & 2 & + & 1 & = & S \\
 & & & & & & & & & & & & \\
(n+1) & + & (n+1) & + & (n+1) & + & \ldots & + & (n+1) & + & (n+1) & = & 2S \\
 & & & & & & & & & & n(n+1) & = & 2S
\end{array}$$

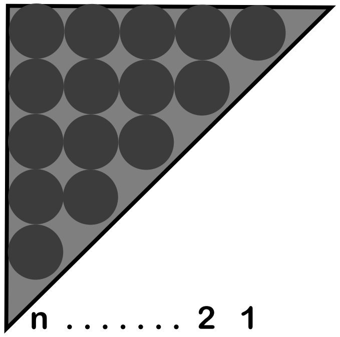


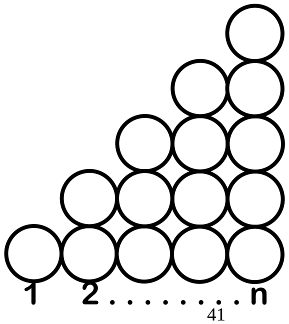
$1 + 2 + 3 + \ldots + n\text{-}1 + n = S$ = number of white dots

$n + n\text{-}1 + n\text{-}2 + \ldots + 2 + 1 = S$ = number of red dots

$(n+1) + (n+1) + (n+1) + \ldots + (n+1) + (n+1) = 2S$

$n(n+1) = 2S$





$1 + 2 + 3 + \ldots + n\text{-}1 + n = S$ = number of white dots

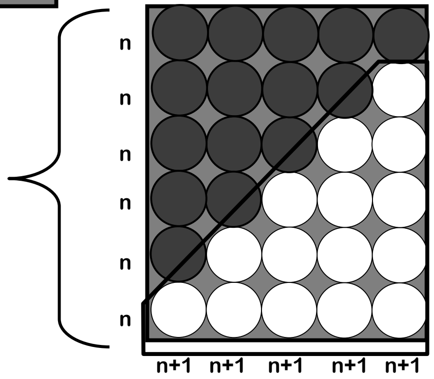
$n + n\text{-}1 + n\text{-}2 + \ldots + 2 + 1 = S$ = number of yellow dots

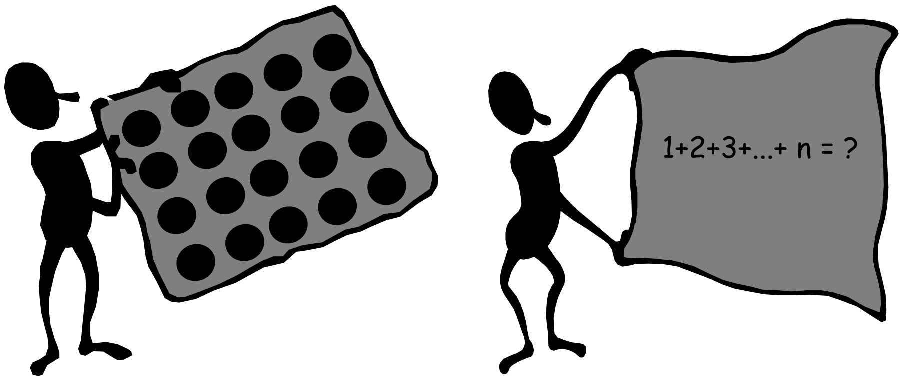
$(n+1) + (n+1) + (n+1) + \ldots + (n+1) + (n+1) = 2S$

$n(n+1) = 2S$

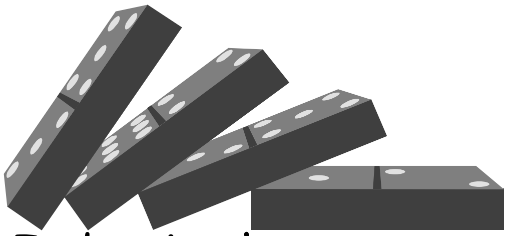
There are n(n+1) dots in the grid

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





At first glance, these problems appeared unrelated, but a representational change revealed a correspondence. An extension of the similarity principle is that when we come to understand that seemingly unrelated things *are* related, we are making intellectual progress.



Induction has many appearances.

- Formal Arguments
- Loop Invariants
- Recursion
- Algorithm Design
- Recurrences

Review: Induction

- Suppose
  - S(k) is true for fixed constant k
    - Often k = 0
  - S(n) → S(n+1) for all n >= k
- Then S(n) is true for all n >= k

Proof By Induction

- Claim:S(n) is true for all n >= k
- Base:
  - Show S(n) is true for n = k
- Inductive hypothesis:
  - Assume S(n) is true for an arbitrary n
- Step:
  - Show that S(n) is then true for n+1

Induction Example: Geometric Closed Form

- Prove $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for all $a \neq 1$
  - Basis: show that $a^0 = (a^{0+1} - 1)/(a - 1)$:
    $a^0 = 1 = (a^1 - 1)/(a - 1)$
  - Inductive hypothesis:
    - Assume $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$
  - Step (show true for n+1):
    $a^0 + a^1 + \ldots + a^{n+1} = a^0 + a^1 + \ldots + a^n + a^{n+1}$
    $= (a^{n+1} - 1)/(a - 1) + a^{n+1} = (a^{n+1+1} - 1)/(a - 1)$

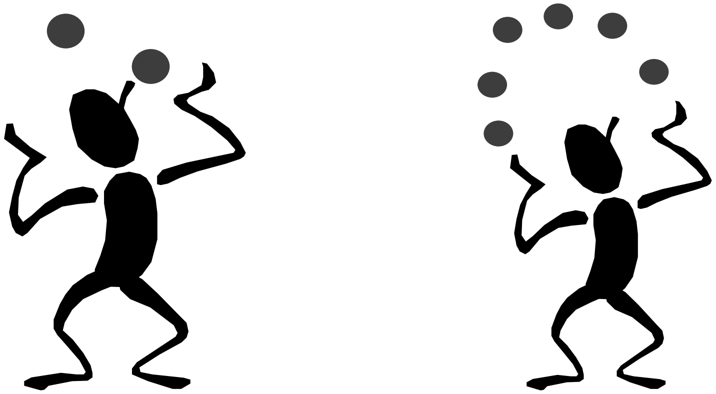
Induction

- Another variation:
  - Basis: show S(0), S(1)
  - Hypothesis: assume S(n) and S(n+1) are true
  - Step: show S(n+2) follows

Effective problem solving:

Exemplification:

Try out a problem or solution on small examples.



Representation:

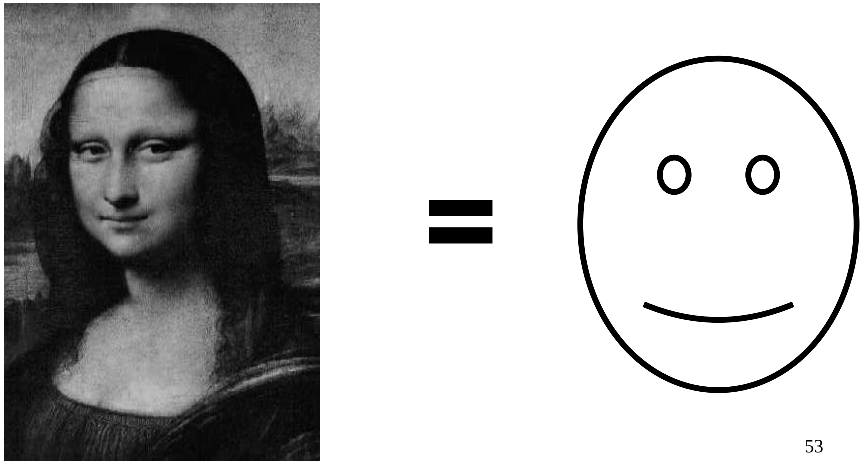
Understand the relationship between different representations of the same information or idea

1	●
2	● ●
3	● ● ●

Modularity:

Decompose a complex problem into simpler subproblems

Abstraction: Abstract away the inessential features of a problem



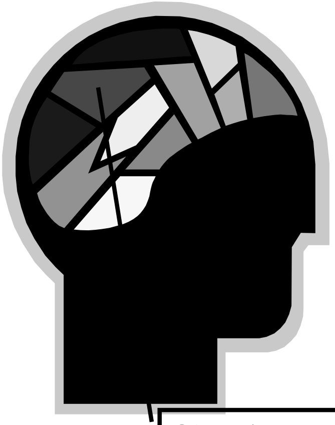
Refinement: The best solution comes from a process of repeatedly refining and inventing alternative solutions

Build your <u>toolbox</u> of abstract structures and concepts. Know the capacities and limits of each tool.

Appreciate Alternative Solutions To The Same Problem

When presented with multiple solutions don't remember only the one that you find easiest to understand. The expert learner takes the opportunity to think through the similarities and differences between the approaches. Such meditations lay the foundations for effective learning and problem solving.

Scanning the brains of master problem solvers

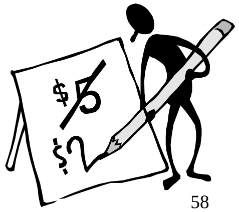


- The better the problem solver, the less brain activity is evident. The real masters show almost no brain activity!

Simple and to the point

The Master Programmer

- The master seeks an algorithm that will use as small an amount of computer resources (e.g., time, space, communication) as possible. Most "expert" programmers (black belts) will miss the best program because they did not have the patience to refine their solutions further.



A case study.

Anagram Programming Task.

You are given a 70,000 word dictionary. Write an anagram utility that given a word as input returns all anagrams of that word appearing in the dictionary.

Examples

- Input: CAT
- Output: ACT, CAT, TAC

- Input: SUBESSENTIAL
- Output: SUITABLENESS

Impatient Hacker
(Novice Level Solution)

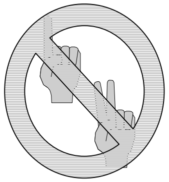
- Loop through all possible ways of rearranging the input word
  - Use binary search to look up resulting word in dictionary.
  - If found, output it.

Performance Analysis:

On input "microphotographic" the loop will run 17! >> $3* 10^{14}$ iterations.

Even at one microsecond (10^{-6}) per iteration this will take $3*10^{8}$ seconds.

This is about a decade.



"Expert" Hacker
(Black Belt Level)

- Subroutine ANAGRAM(X,Y) returns **TRUE** exactly when X and Y are anagrams. It works by sorting the letters in X and Y

- Input X
- Loop through all dictionary words Y
  - If ANAGRAM(X,Y) output Y

Comparing an input word with each of 70,000 dictionary entries takes about 15 seconds.

The hacker is satisfied and reflects no further

The master keeps trying to refine the solution.

- The master's program runs in less than 1/1000 seconds.

Master Solution

- Don't keep the dictionary in sorted order!
- Rearranging the dictionary into anagram classes will make the original problem simple.

Suppose the dictionary was the list below.

- ASP
- DOG
- LURE
- GOD
- NICE
- RULE
- SPA

After each word, write its "signature" (sort its letters)

• ASP	APS
• DOG	DGO
• LURE	ELRU
• GOD	DGO
• NICE	CEIN
• RULE	ELRU
• SPA	APS

Sort by the signatures

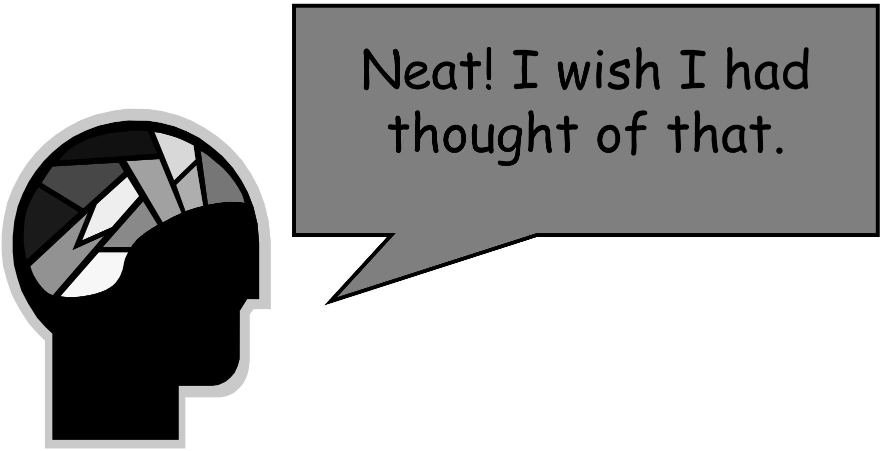
ASP	APS
SPA	APS
NICE	CEIN
DOG	DGO
GOD	DGO
LURE	ELRU
RULE	ELRU

Master Program

- Input word W (e.g., CAT)
- X := signature of W (ACT)
- Use binary search to find the anagram class of W and output it.

About $\log_2(70000) \times 25$ microseconds $\approx$ .0004 seconds.

Right, it takes about 30 seconds to create the dictionary, but it is perfectly fair to think of this as programming time. The building of the dictionary is a one-time cost that is part of writing the program.



Learning Advice

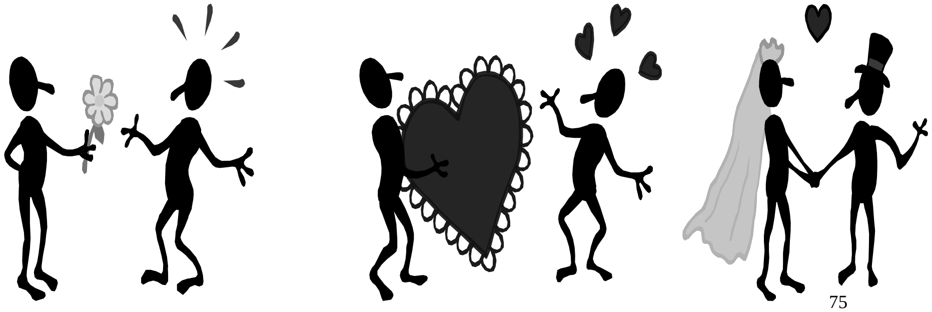
- Whenever you see something you wish you had thought of, try and formulate the minimal and most general lesson that will insure that you will not miss the same thing the next time. Name the lesson to make it easy to remember.

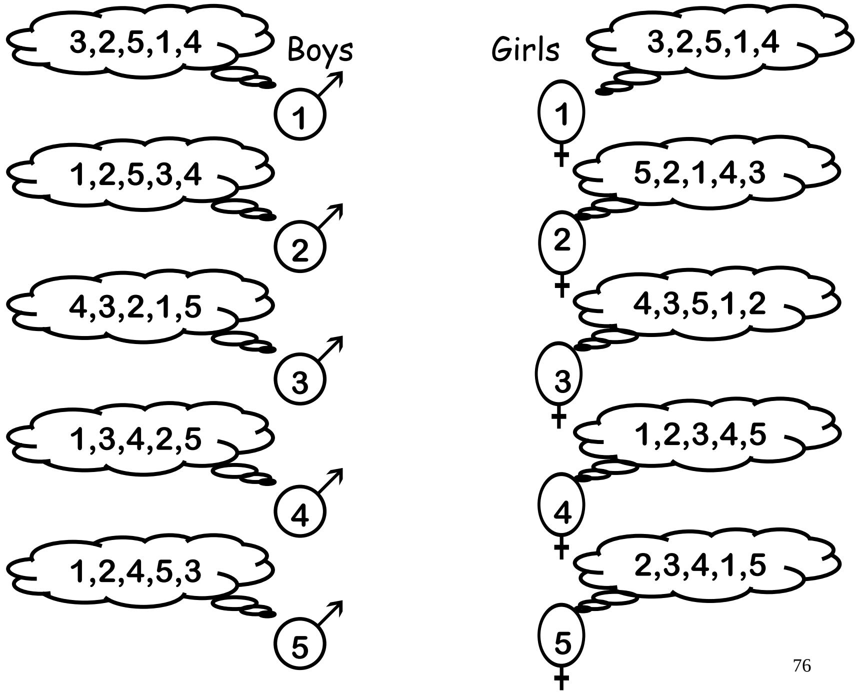
NAME: Preprocessing

- It is sometimes possible to pay a reasonable, one-time preprocessing cost to reorganize your data in such a way as to use it more efficiently later. The extra time required to preprocess can be thought of as additional programming effort.

Great Theoretical Ideas In Computer Science

The Mathematics Of 1950's Dating: Who wins the battle of the sexes?





Boys

3,2,5,1,4 ♂ 1

1,2,5,3,4 ♂ 2

4,3,2,1,5 ♂ 3

1,3,4,2,5 ♂ 4

1,2,4,5,3 ♂ 5

Girls

3,2,5,1,4 ♀ 1

5,2,1,4,3 ♀ 2

4,3,5,1,2 ♀ 3

1,2,3,4,5 ♀ 4

2,3,4,1,5 ♀ 5

Dating Scenario

- There are n boys and n girls
- Each girl has her own ranked preference list of all the boys
- Each boy has his own ranked preference list of all the girls
- The lists have no ties

Question: How do we pair them off?

Which criteria come to mind?

What is considered a “good” pairing?

- Maximizing total satisfaction
  - How is the ‘average’ match relative to his/her rank.
- Maximizing the minimum satisfaction
  - How deep in his/her list is the coupe of the most unsatisfied participant?
- Minimizing the maximum difference in mate ranks
  - Everybody is more or less equally satisfied
- Maximizing the number of people who get their first choice
  - Barbie and Ken Land

Rogue Couples

- Suppose we pair off all the boys and girls. Now suppose that some boy and some girl prefer each other to the people to whom they are paired. They will be called a rogue couple.

Why be with them when we can be with each other?

Stable Pairings

- A pairing of boys and girls is called stable if it contains no rogue couples.

Stability is primary.

- Any list of criteria for a good pairing must include **stability**. (A pairing is doomed if it contains a rogue couple.)

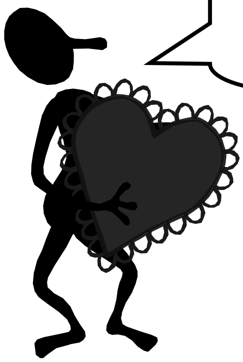
- Any reasonable list of criteria must contain the stability criterion.

The study of stability will be the subject of the entire lecture.

- We will:
  - Analyze various mathematical properties of an algorithm that looks a lot like 1950's dating
  - Discover the naked mathematical truth about which sex has the romantic edge.
  - Learn how the world's largest, most successful dating service operates.

Given a set of preference lists, how do we find a stable pairing?

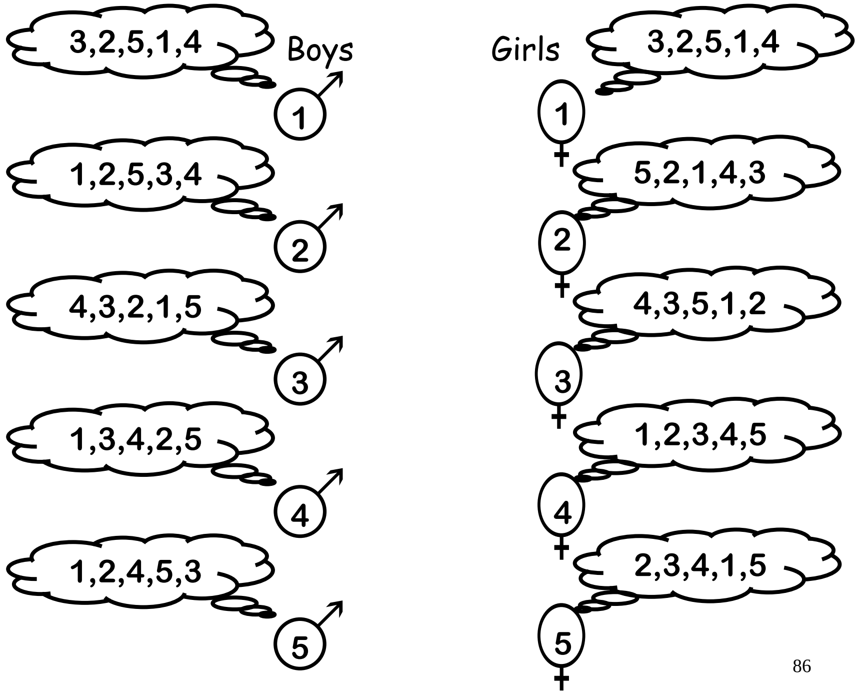
Given a set of preference lists, how do we find a stable pairing?

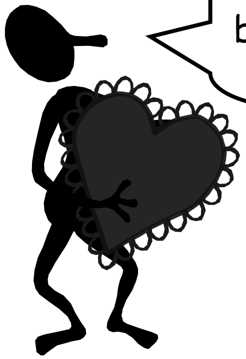


Wait! There is a more primary question!

The Existence Question:

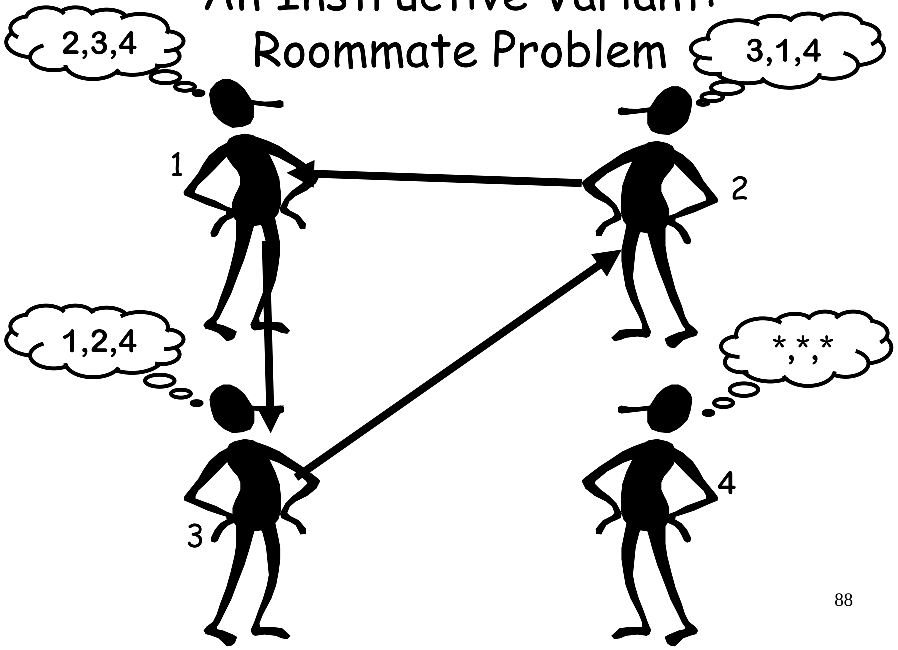
Does every set of preferences lists have at least one stable pairing???





Can you argue that the couples will not continue breaking up and reforming forever?

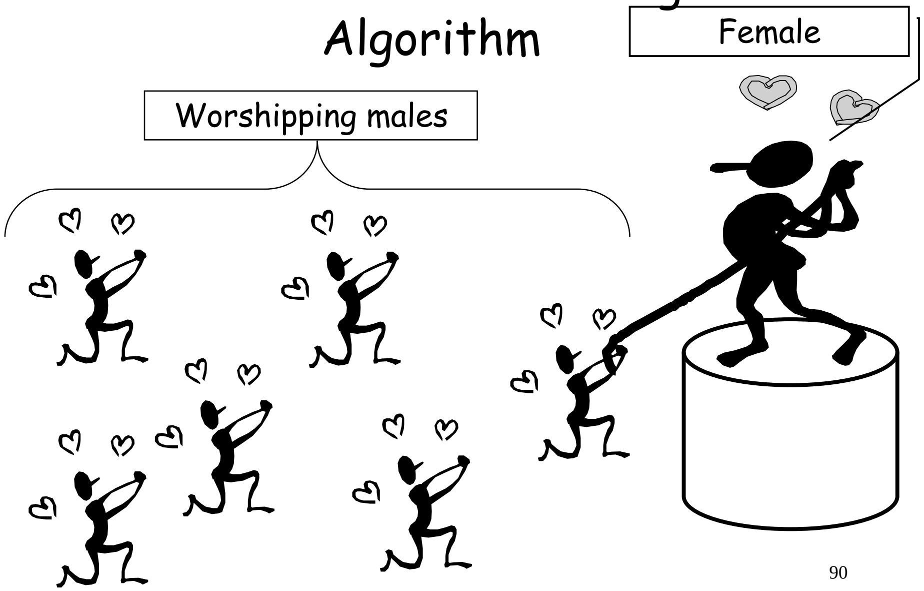
An Instructive Variant: Roommate Problem



Insight

- Any proof that couples do not break up and reform forever must contain a step that fails in the case of the roommate problem.
- If you have a proof idea that works equally well in the marriage problem and the roommate problem, then your idea is not adequate to show the couples eventually stop.

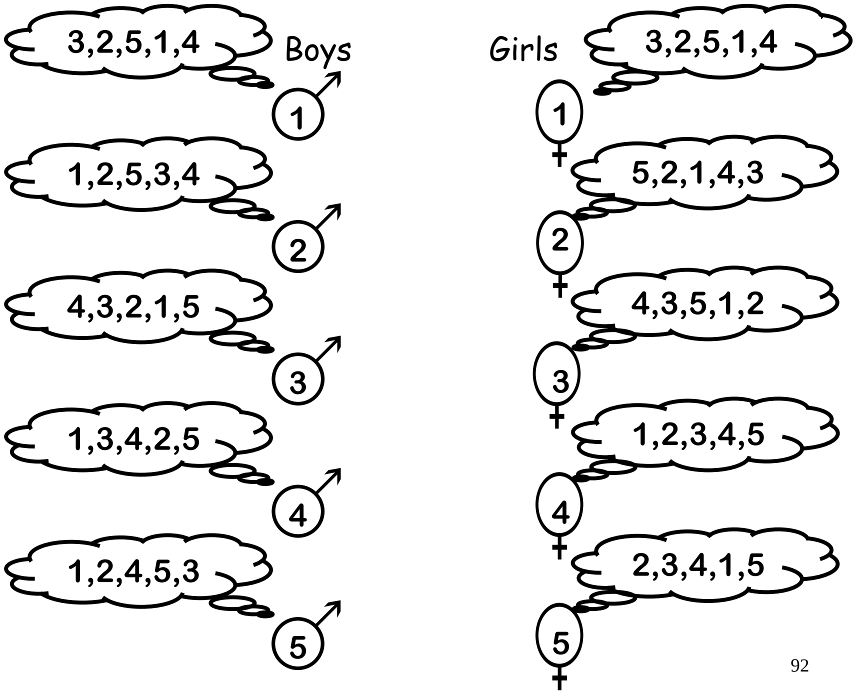
The Traditional Marriage Algorithm



Traditional Marriage Algorithm

- For each day that some boy gets a “**No**” do:
  - **Morning**
    - Each girl stands on her balcony
    - Each boy proposes under the balcony of the best girl whom he has not yet crossed off
  - **Afternoon (for those girls with at least one suitor)**
    - To today’s best suitor: **“Maybe, come back tomorrow”**
    - To any others: **“No, I will never marry you”**
  - **Evening**
    - Any rejected boy crosses the girl off his list.

Each girl marries the boy to whom she just said “maybe”



Boys

1: 3,2,5,1,4

2: 1,2,5,3,4

3: 4,3,2,1,5

4: 1,3,4,2,5

5: 1,2,4,5,3

Girls

1: 3,2,5,1,4

2: 5,2,1,4,3

3: 4,3,5,1,2

4: 1,2,3,4,5

5: 2,3,4,1,5

Traditional Marriage Algorithm

- Example (see slide 89 for preferences lists)

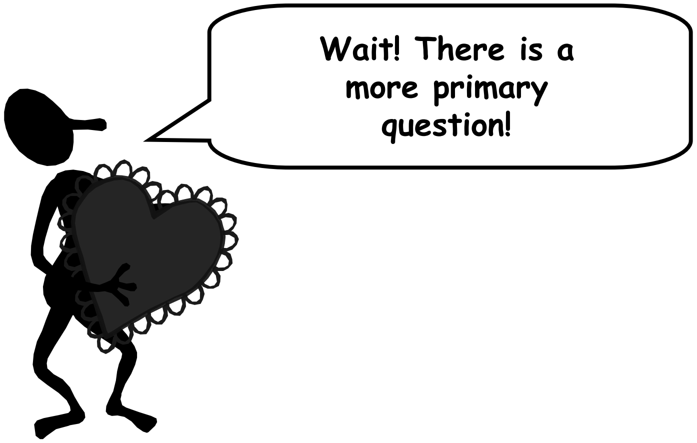
girl	Day 1	Day 2
1	②,4,5	②
2		⑤
3	①	1,④
4	③	③
5		

○ = come tomorrow

In class exercise:

Complete the execution.

Does the Traditional Marriage Algorithm always produce a <u>stable</u> pairing?



Does TMA always terminate?

- It might encounter a situation where the algorithm does not specify what to do next (core dump error).
- It might keep on going for an infinite number of days.

<u>Lemma</u>: No boy can be rejected by all the girls

- Proof by contradiction.
- Suppose Bob is rejected by all the girls. At that point:
  - Each girl must have a suitor other than Bob (Once a girl has a suitor she will always have at least one)
  - But there are n girls and only n-1 boys besides Bob.

Contradiction

Theorem: The TMA always terminates in at most n^2 days

- Consider the "master list" containing all the boy's preference lists of girls. There are n boys, and each list has n girls on it, so there are a total of $n \times n = n^2$ girls' names in the master list.
- Each day that at least one boy gets a "No", at least one girl gets crossed off the master list.
- Therefore, the number of days is bounded by the original size of the master list.

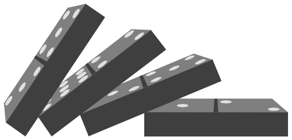
Great! We know that TMA will terminate and produce a pairing.

But is it stable?

MAYBE Lemma: In TMA if on day i a girl says "maybe" to boy b, she is guaranteed to marry a husband that she likes at least as much as b.

- She would only let go of him in order to "maybe" someone better
- She would only let go of that guy for someone even better
- She would only let go of that guy for someone even better
- AND SO ON . . . .

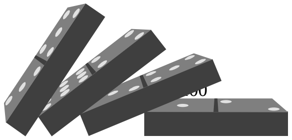
Informal Induction



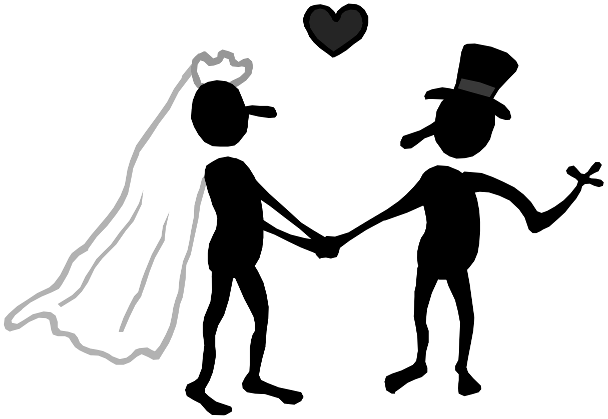
MAYBE Lemma: In TMA if on day i a girl says "maybe" to boy b, she is guaranteed to marry a husband that she likes at least as much as b.

- (*) For all $k \geq 0$, on day i+k the girl will say "maybe" to a boy she likes as much as b.
- Base: k=0 (true by assumption)
- Assume (*) is true for k-1. Thus she has a boy as good as b on day i+k-1. The next day she will either keep him or reject him for some better. Thus (*) is true for k.

Formal Induction



Corollary: Each girl will marry her absolute favorite of the boys who visit her during the TMA.



Theorem: The pairing produced by TMA is stable.

- Proof by contradiction:
  Suppose Bob and Mia are a rogue couple.

Alice
Bob
Mia
Luke

- This means Bob likes Mia more than his wife, Alice.
- Thus, Bob proposed to Mia before he proposed to Alice.
- Mia must have rejected Bob for someone she preferred.
- By the Maybe lemma, she must like her husband Luke more than Bob.

Contradiction!

Opinion Poll

Who is better off in traditional dating, the boys or the girls?

Forget TMA for a moment

- How should we define what we mean when we say “the optimal girl for Bob”?

Flawed Attempt:
“The girl at the top of Bob’s list”

The Optimal Girl

- A boy's optimal girl is the highest ranked girl for whom there is some stable pairing in which the boy gets her.

- She is the best girl he can conceivably get in a stable world. Presumably, she might be better than the girl he gets in the stable pairing output by TMA.

The Pessimal Girl

- A boy's pessimal girl is the lowest ranked girl for whom there is some stable pairing in which the boy gets her.

- She is the worst girl he can conceivably get in a stable world.

Dating Heaven and Hell

- A pairing is male-optimal if every boy gets his optimal mate. This is the best of all possible stable worlds for all the boys simultaneously.

- A pairing is male-pessimal if every boy gets his pessimal mate. This is the worst of all possible stable worlds for all the boys simultaneously.

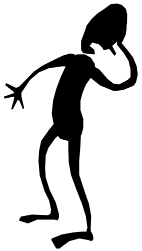
Dating Heaven and Hell

- A pairing is female-optimal if every girl gets her optimal mate. This is the best of all possible stable worlds for every girl simultaneously.

- A pairing is female-pessimal if every girl gets her pessimal mate. This is the worst of all possible stable worlds for every girl simultaneously.

The Naked Mathematical Truth!

- The Traditional Marriage Algorithm always produces a male-optimal, female-pessimal pairing.



Theorem: TMA produces a male-optimal pairing

- Suppose not: i.e. that some boy gets rejected by his optimal girl during TMA.
- In particular, let's say Bob is the first boy to be rejected by his optimal girl Mia: Let's say she said "maybe" to Luke, whom she prefers.
- Since Bob was the only boy to be rejected by his optimal girl so far, Luke must like Mia at least as much as his optimal girl.

We are assuming that Mia is Bob's optimal girl. Mia likes Luke more than Bob. Luke likes Mia at least as much as his optimal girl.

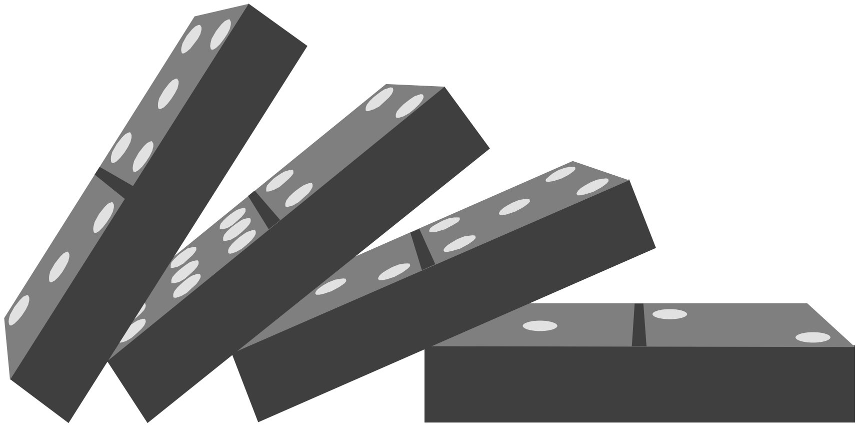
- We'll show that any pairing S in which Bob marries Mia cannot be stable (for a contradiction).
- Suppose S is stable:
  - Luke likes Mia more than his wife in S
    - Luke likes Mia at least as much as his best possible girl, but he does not have Mia in S
  - Mia likes Luke more than her husband Bob in S

Contradiction

We are assuming that Mia is Bob's optimal girl. Mia likes Luke more than Bob. Luke likes Mia at least as much as his optimal girl.

- We've shown that any pairing in which Bob marries Mia cannot be stable.
- Thus, Mia cannot be Bob's optimal girl (since he can never marry her in a stable world).
- So Bob never gets rejected by his optimal girl in the TMA, and thus the TMA is male-optimal.

What proof technique did we just use?



<u>Theorem</u>: The TMA pairing, T, is female-pessimal.

- Suppose there is a stable pairing S where some girl Alice does worse than in T.
- Let Luke be her mate in T.
  Let Bob be her mate in S.
  - By assumption, Alice likes Luke better than her mate Bob in S.
  - Luke likes Alice better than his mate in S.
    - We already know that Alice is his optimal girl (remember, T is male-optimal).
  - Therefore, S is not stable.

A contradiction

References

•D. Gale and L. S. Shapley, *College admissions and the stability of marriage*, American Mathematical Monthly 69 (1962), 9-15

•Dan Gusfield and Robert W. Irving, *The Stable Marriage Problem: Structures and Algorithms*, MIT Press, 1989

“The Match”: Doctors and Medical Residencies

- Each medical school graduate submits a ranked list of hospitals where he/she wants to do a residency.
- Each hospital submits a ranked list of newly minted doctors.
- A computer runs TMA (extended to handle Mormon marriages).
- Until recently, it was hospital-optimal

History

- 1900
  - Idea of hospitals having residents (then called “interns”)
- Over the next few decades
  - Intense competition among hospitals for an inadequate supply of residents
    - Each hospital makes offers independently
    - Process degenerates into a race. Hospitals steadily advancing date at which they finalize binding contracts

History

- 1944 Absurd Situation. Appointments being made 2 years ahead of time!
  - All parties were unhappy
  - Medical schools stop releasing any information about students before some reasonable date
  - **Offers were made at a more reasonable date, but new problems developed**

History

- 1945-1949
  - Hospitals started putting time limits on offers
  - Time limit gets down to 12 hours
  - Lots of unhappy people
  - Many instabilities resulting from lack of cooperation

History

- 1950 Centralized System
  - Each hospital ranks residents
  - Each resident ranks hospitals
  - National Resident Matching Program produces a pairing

Whoops! The pairings were not always stable. By 1952 the algorithm was the TMA (hospital-optimal) and therefore stable.