Reminders

Final exam

- 2 hours (not 3)
- Comprehensive, but weighted towards end

Today's lecture

- What do computer scientists do?
    - What does this computer scientist do
- Overview of term

What Do
Computer Scientists Do?

What Do Computer Scientists Do?

They think computationally

Computational thinking will be a fundamental skill used by everyone in the world by the middle of the 21st Century.

Just like the three r's: reading, riting, and rithmetic.

- Ubiquitous computing and computers will enable the spread of computational thinking.

Computational Thinking: the Process

Identify or invent useful abstractions

Formulate solution to a problem as a computational experiment

Design and construct a sufficiently efficient implementation of experiment

Validate experimental setup (i.e., debug it)

Run experiment

Evaluate results of experiment

Repeat as needed

The Two A's of Computational Thinking

Abstraction

- Choosing the right abstractions
- Operating in terms of multiple layers of abstraction simultaneously
- Defining the relationships the between layers

Automation

- Think in terms of mechanizing our abstractions
- Mechanization is possible
  - Because we have precise and exacting notations and models
  - There is some "machine" that can interpret our notations

Examples of Computational Thinking

How difficult is this problem and how best can I solve it?

- Theoretical computer science gives precise meaning to these and related questions and their answers

Thinking recursively

- Reformulating a seemingly difficult problem into one which we know how to solve.
- Reduction, embedding, transformation, simulation

What One CS Does, My Research Group

Goals

- Help people live longer and better quality lives
  - In collaboration with clinicians
- Have fun pushing the frontiers of
  - Computer Science
  - Electrical Engineering
  - Medicine

Technical areas

- Machine learning and data mining
- Algorithm design
- Signal processing
- Software systems

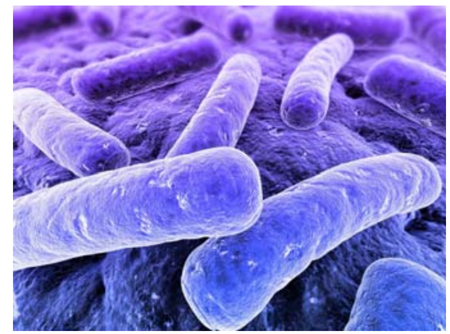
Healthcare-associated Infections

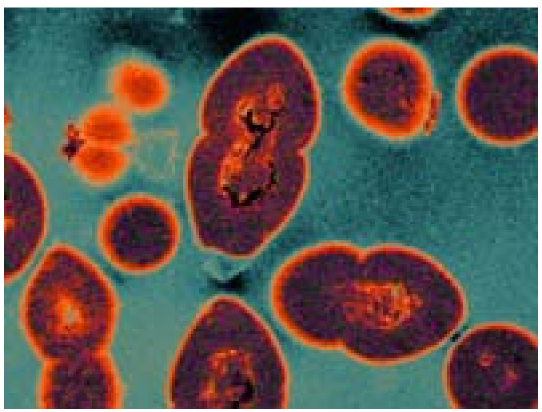
Approximately 1 in 20 hospital visits

Among top 10 leading causes of death in U.S.

Should be largely preventable

Studying data from >4.5M visits





Heart, brain, and connected anatomy

Some examples

- Predicting adverse cardiac events
- Detecting and responding to epileptic seizures

Example 1. Treating Epilepsy

Prevalence of ~1%; all ages

All countries

Characterized by recurrent seizures

Generated by abnormal electrical activity in brain

Acquired

Head Injury

Intracranial Hemorrhage

Infection

Stroke

Inherited

Ion Channelopathy

Defective Neural Organization

One Manifestation

Seizure Onset Seems Unpredictable

May result in injury

- Fractures, intracranial hematomas, burns, etc.

May result in death

- Mortality rate 2-3 times that of general population
- Accidents, aspiration, drowning, etc.
- SUDEP (annual risk estimated to be 1 per 100 for patients with symptomatic seizures)

Early Detection of Seizure Onset

Two onset times

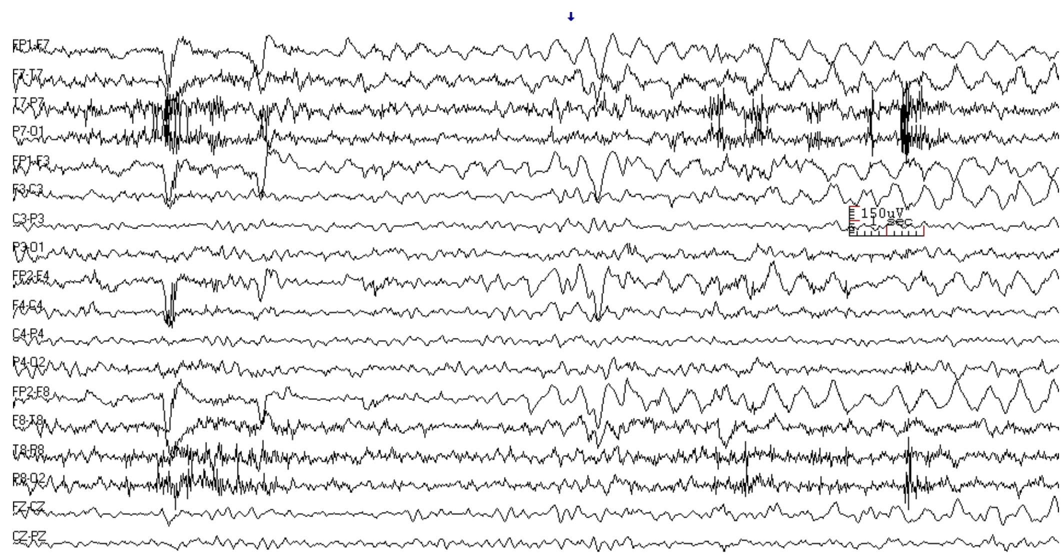
- Electrographic
- Clinical

Detecting electrographic onset

- Use scalp EEG

Therapeutic value

- Provide warning
- Summon help
- Fast acting drugs
- Neural stimulation



Not Easy

EEG varies greatly across patients
- Epileptics have abnormal baselines
- Generic detectors have not worked particularly well

Pretty consistent patterns for an individual

Use patient-specific detectors

Use machine learning to build patient-specific seizure onset detector. Highly successful retrospective studies

Turn on neural stimulator at start of seizure. Study in progress at MGH

Example 2: Predicting "Heart Attacks"

Acute coronary syndrome (ACS) common: ~1.25M/year in U.S.

15% - 20% of these people will suffer cardiac-related death within 4 years

Stratifying risk key to choosing treatments

Who gets a defibrillator?

Who should be treated aggressively with statins

Getting this right matters, a lot!

An Adverse Cardiac Event

Video of Hungarian soccer star Miklos Feher dying of sudden cardiac death on the field:

http://www.youtube.com/watch?v=52RJWyjogY0&feature=player_embedded#!

But, It Does Not Have to Be That Way

Another soccer player, Anthony Van Loo, collapsed during a match but is brought back to life by his Implantable Cardioverter Defibrillator (ICD):

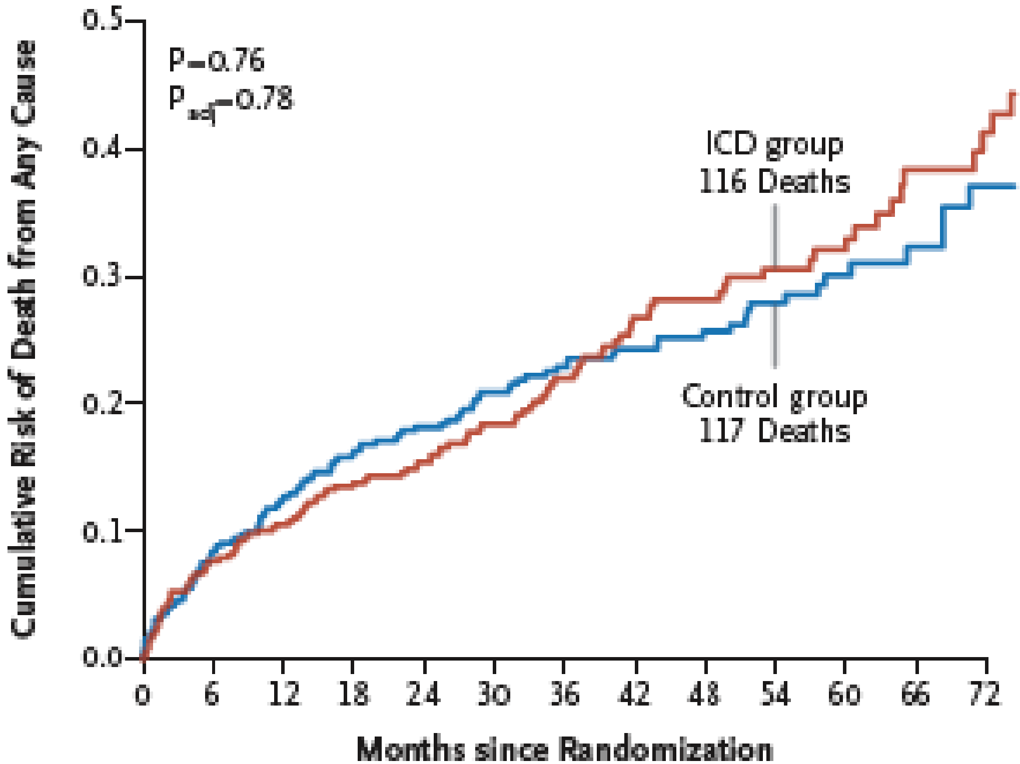
http://www.youtube.com/watch?v=DU_i0ZzIV5U

ICD's

Too many: Potentially risky, always expensive (~$50k)

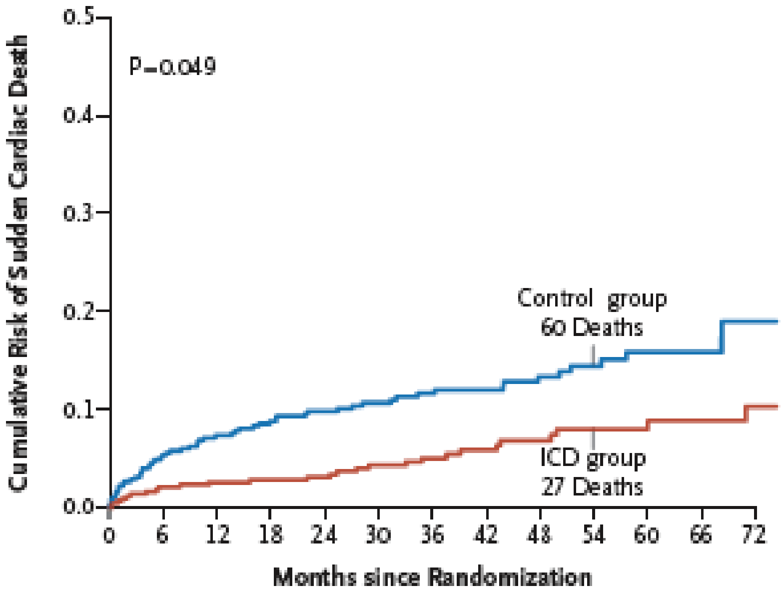
90% of recipients received < 0 medical benefit

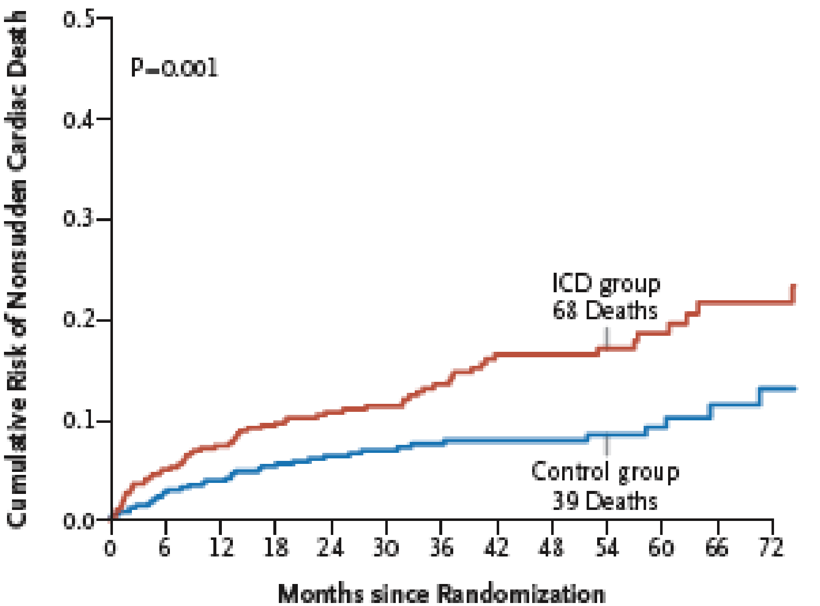
Too few: 100's of deaths/day potentially avoidable



NEJM Oct. 2009

Does This Mean that ICD's are "Useless?"





NEJM Oct. 2009

Approaches to Identifying High Risk Cases

Clinical characteristics

E.g., gender or high blood pressure

Biomarkers

E.g., cholesterol levels

Echocardiography

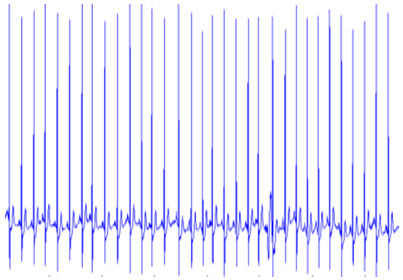
Ultrasound to measure, e.g., left ejection fraction

Electrocardiography (ECG)

Established methods, e.g., HRV and DC

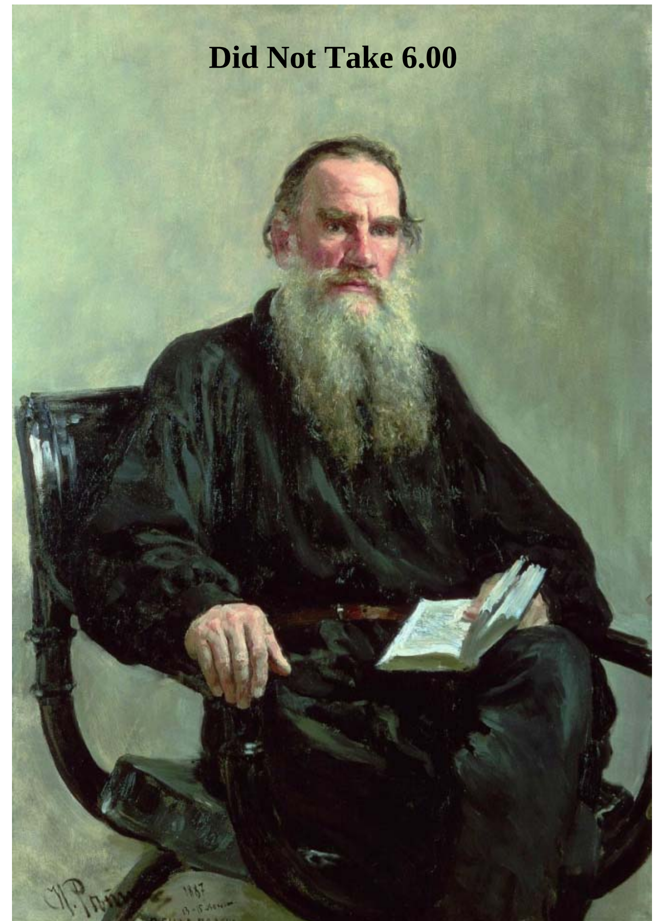
New method: Morphologic Variability (MV)

- Measures variability in shape of heart beats



The Tolstoy Approach to Risk Stratification

"Happy families are all alike;
every unhappy family is unhappy in its own way"



Looking for Outliers

“Happy hearts are all alike; every unhappy heart is unhappy in its own way”

Quantify differences between long symbolic sequences

Must account for differences in the shapes and frequencies of symbols

$$SM_{pq} = \sum_{a \in S_p} \sum_{b \in S_q} d(a,b) P_p[a] P_q[b]$$

$P_i[a]$: probability of symbol *a* in sequence *I*

$d(i,j)$: distance between symbols *i* and *j*

Use something called dynamic time warping to compute distances

Implemented using dynamic programming

Cluster patients and identify outliers

Outliers Post-ACS

Cluster	# of Patients	% Death (90 days)	% MI (90 days)	% Death/MI (90 days)
Anomaly Cluster A	53	3.77	1.89	5.66
Anomaly Cluster B	48	2.08	8.33	10.42
Anomaly Cluster C	22	18.18	4.55	22.73
Anomaly Cluster D	20	10.00	5.00	10.00
Anomaly Cluster E	12	0.00	16.67	16.67
Dominant Group	457	0.88	3.28	3.50

Six Major Topics

Learning a language for expressing computations – Python

Learning about the process of writing and debugging a program

Learning about the process of moving from a problem statement to a computational formulation of a method for solving the problem

Learning a basic set of recipes – algorithms

Learning how to use simulations to shed light on problems that don’t easily succumb to closed form solutions

Learning about how to use computational tools to help model and understand data

Why Python?

Relatively easy to learn and use

- Simple syntax
- Interpretive, which makes debugging easier
- Don’t have to worry about managing memory

Modern

- Supports currently stylish mode of programming, object-oriented

Increasingly popular

- Used in an increasing number of subjects at MIT and elsewhere
- Increasing use in industry
- Large and ever growing set of libraries

Writing, Testing, and Debugging Programs

Take it a step at time

- Understand problem
- Think about overall structure and algorithms independently of expression in programming language
- Break into small parts
- Identify useful abstractions (data and functional)
- Code and unit test a part at a time
- First functionality, then efficiency
- Start with pseudo code

Be systematic

- When debugging, think scientific method
- Ask yourself why program did what it did, not why it didn't do what you wanted it to do.

From Problem Statement to Computation

- **Break the problem into a series of smaller problems**
- **Try and relate problem to a problem you or somebody else have already solved**
  - E.g., can it be viewed as a knapsack problem
- **Think about what kind of output you might like to see, e.g., what plots**
- **Formulate as an optimization problem**
  - Find the min (or max) values satisfying some set of constraints
- **Think about how to approximate solutions**
  - Solve a simpler problem
  - Find a series of solutions that approaches (but may never reach) a perfect answer

Algorithms

Big O notation

- Orders of growth
  - Exponential, Polynomial, Linear, Log
- Amortized analysis

Kinds of Algorithms

- Exhaustive enumeration, Guess and check, Successive approximation, Greedy algorithms, Divide and conquer, Decision Trees, Dynamic programming

Specific algorithms

- E.g., Binary search, Merge sort

Optimization problems

- Knapsack, shortest path, dynamic programming

Modeling the World

Models always inaccurate

But often useful

Provide abstractions of reality

- Simulation models
  - Monte Carlo
  - Queuing network
- Statistical models (linear regression)
- Graph theoretic models

Making Sense of Data

Statistics
- Use and misuse

Plotting

Machine learning
- Supervised
  - Classification
- Unsupervised
  - Hierarchical clustering
  - K-means clustering
- Feature vectors
  - Abstract from data items to relevant properties
  - Scaling matters

Pervasive Themes

Power of abstraction

Systematic problem solving

What Next

Many of you have worked very hard

- TA's and I appreciate it

Only you know your return on investment

- Take a look at early problem sets
- Think about what you'd be willing tackle now

Remember that you can write programs to get answers

There are other CS courses you could take

- 6.01, 6.034, 6.005, 6.006

You could even major in Course VI or consider the new major "Computer Science and Molecular Biology"

You are qualified for interesting UROP's involving computation