

(Eventually I Will Get To A)

# Recap of Stanford Algorithms Course

(The Free Online Version)

Tracy Usher

Scientific Computing Applications

Department Meeting

May 4, 2012

# Can Old Dogs Learn New Tricks?

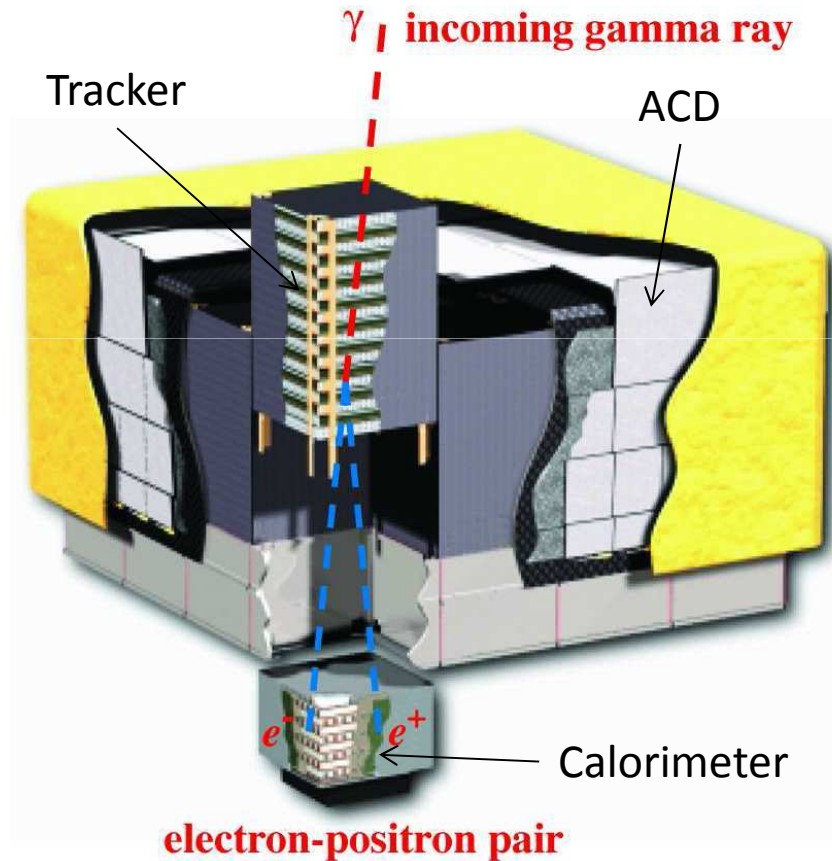
## Or... Why Am I Up Here?

- Current primary responsibility is the simulation and reconstruction software for the Fermi Large Area Telescope.
- Trained (a very long time ago) as a Physicist, not Computer Scientist
  - No formal education in Computer Science
    - No need - Physicists in training will have any needed programming skills simply materialize directly from the ether
  - First language was the then modern high level language of Fortran II
    - Thankfully, its possible to make C++ look like Fortran!
- Fast forward  $n$  decades (where  $n > 2$ ) to the era of faster computers, with more available memory, along with new techniques, etc., all making it possible to re-attack old problems in new ways
  - Lots of progress in Computer Science since I was a kid, I've been exposed to only to those pieces relevant to my applications – leaving lots of holes!
  - Perhaps worth taking some time to study what's out there?
- Start by spending a little time giving an overview of the problem we want to solve as motivation for why one might want to be interested in taking an online course in algorithms...

# Fermi Large Area Telescope

## Instant Instrument Overview

- Silicon Strip Tracker (TKR)
  - Tracks  $e^+e^-$  pair to determine incident photon direction
  - 36 planes of strips alternating x-y coordinates
  - 16 Towers
  - ~900,000 total strips
- CsI Calorimeter (CAL)
  - Measures incident photon energy
  - 8 layers of 12 crystals
  - 16 Towers
  - 1536 total crystals
- Anticoincidence Shield (ACD)
  - Reject charged particle background
  - Segmented array of 89 plastic scintillator tiles



# Fermi LAT Event Reconstruction

## “Standard” HEP Approach

In a nutshell, the Fermi LAT Reconstruction has three main steps:

- First Step - Primary event reconstruction:
  - Determine Energy from Calorimeter
    - Sum of energies in crystals above threshold
  - Find and Fit tracks in the Tracker
    - “Local” Pattern recognition utilizing a Kalman Filter finder-follower approach seeded by combinatoric trials
  - Combine tracks and energy to determine find gamma ray direction and energy
    - Standard vertex fit
- Second Step - Use Classification Tree Analyses to
  - Categorize as background (bringing in the ACD information)
  - Categorize quality of event
  - Provide best estimates of energy and direction
- Final Step
  - Output analysis ready summary information

# Connection to Computer Science

## Software Issues in the LAT Reconstruction

- LAT Reconstruction not necessarily a huge project in comparison to, say, ATLAS, when applying “standard” HEP techniques... still:
  - Average reconstruction time measured in 10’s of seconds
    - With a long tail on the total cpu time distribution
  - Fermi has “billions and billions” of events
  - Combination of two means recon needs to be cpu time aware:
    - Some perhaps less than efficient attempts to “optimize” key internal loops
    - Cutoffs inserted in key areas to prevent endless looping
    - Limits placed on size of events allowed to be reconstructed
- One must always ask the age old question: can we do better?
  - LAT on-orbit experience has indicated that current reconstruction, while certainly adequate, could stand improvement
    - Struggling to fully achieve initial LAT design goals in some energy ranges
      - e.g. resolution at very high energy due
    - Unexpected surprises have led to some band aid fixes
    - Always have concern that cutoffs may be biasing output data sets
  - Applying new reconstruction techniques could address these issues potentially further extending Fermi’s overall capabilities
- Provided, of course, that we implement them “correctly”

# Fermi Pass 8 Reconstruction

## Some Examples of “New Techniques” Applied to Fermi

- Tracker Reconstruction:
  - Develop a global pattern recognition approach known as “Tree Based Tracking”
    - Used crossed pairs from adjacent x-y planes to form 3D hits (forming “nodes”)
    - Link pairs together in adjacent layers (forming “edges” between “nodes”)
    - Link together links sharing a common 3D hit to form an “acyclic directed graph” (which we call a “Tree”)
    - Traverse the Tree to extract the tracks according to a “longest-straightest” metric
  - **Problem:** Ambiguities increase combinatorics geometrically
    - The 3D hits will include ambiguous combinations of x-y pairs
    - These create even more ambiguous links between adjacent layers
    - Ambiguities cannot be resolved until the final step of extracting tracks from all combinations
    - Running time of this approach is/can be VERY slow
- Calorimeter Reconstruction: CAL Clustering
  - Form “clusters” (in HEP sense) in the calorimeter by connecting hit crystals using a Minimum Spanning Tree approach
    - Exclude crystals from energy determination which are not part of the event (“ghost hits”)
    - Obtain a direction vector for incident gamma based solely on the calorimeter which can be compared to Tree solution
  - **Problem:** Performance trade off created by particular choice of cut parameter to separate into clusters
    - Cutoff too short – too many clusters leads to confusing environment
    - Cutoff too large – everything lumped into one cluster and back where we are now

# Motivation To Go Back To School...

## Finally Getting to the Real Story!

- Its clear that many of the problems we are encountering in developing the Pass 8 reconstruction already have solutions in other applications
  - We simply need to recognize what kind of problem we have
    - For Example:
      - building Trees is the same as building a directed graph
      - extracting tracks is the same as find the shortest path between two vertices
    - Someone with more formal training would recognize this right away
    - Perhaps taking some time to develop basic CS skills not a bad thing?
- Digging through the internet/self teaching not necessarily optimal way for someone like me to learn things
  - Can spend a fair bit of time running off on tangents
- Not enough time to take a “real” course on campus
  - Work schedules frequently get in the way of best intentions
- Would be ideal to find something in between self teaching and taking a “real” course
  - Flexible but structured and with some deadlines to force one to stay on track

# Stanford Online Algorithms Course

- This past Fall Stanford University (now in conjunction with several other universities) began offering “free online courses”
  - Online versions of courses taught on the campus
  - This Winter offered a course in the “Design and Analysis of Algorithms I”
- Why a course on algorithms?
  - Algorithms course provides the basic toolkit which underlies virtually everything else you might want to do.
    - Design techniques
    - Analysis
    - Applications
    - Data Structures
- Why the online course?
  - Structured course done on my time works very well for me
    - Watch lectures when time available, yet
    - Weekly assignments to keep pressure on to keep pace
    - Challenging programming assignments to get “hands on” experience
  - Video lectures often illustrated with real world examples which spark thought on how to transfer to our problems
    - e.g. breadth first search vs depth first search for finding different characteristics of our Tracker Trees



## Design and Analysis of Algorithms I

Tim Roughgarden, Associate Professor

In this course you will learn several fundamental principles of algorithm design: divide-and-conquer methods, graph algorithms, practical data structures, randomized algorithms, and more.



Go to Class!

Preview

Category

Computer Science

Duration

5 weeks

Next Session

TBA

7,964

Tweet

3.5k

>1

182

Share

6.4k

Like

### About the Course

In this course you will learn several fundamental principles of algorithm design. You'll learn the divide-and-conquer design paradigm, with applications to fast sorting, searching, and multiplication. You'll learn several blazingly fast primitives for computing on graphs, such as how to compute connectivity information and shortest paths. Finally, we'll study how allowing the computer to "flip coins" can lead to elegant and practical algorithms and data structures. Learn the answers to questions such as: How do data structures like heaps, hash tables, bloom filters, and balanced search trees actually work, anyway? How come QuickSort runs so fast? What can graph algorithms tell us about the structure of the Web and social networks? Did my 3rd-grade teacher explain only a suboptimal algorithm for multiplying two numbers?

### About the Instructor

**Tim Roughgarden** is an Associate Professor of Computer Science and (by courtesy) Management Science and Engineering at Stanford University, where he holds the Chambers Faculty Scholar development chair. At Stanford, he has taught the Design and Analysis of Algorithms course for the past eight years. His research concerns the theory and applications of algorithms, especially for networks, auctions and other game-theoretic applications, and data privacy. For his research, he has been awarded the ACM Grace Murray Hopper Award, the Presidential Early Career Award for Scientists and Engineers (PECASE), the Shapley Lectureship of the Game Theory Society, a Sloan Fellowship, INFORM's Optimization Prize for Young Researchers, and the Mathematical Programming Society's Tucker Prize.



### Textbooks

No books are required for the course. However, three books have significantly influenced the instructor's presentation and can be consulted for extra details. They are: Kleinberg & Tardos *Algorithm Design*, Dasgupta, Papadimitriou & Vazirani *Algorithms*, and Cormen, Leiserson, Rivest, & Stein *Introduction to Algorithms*.

No special software (e.g., development environment) is required to take this course.

### Frequently Asked Questions

- **What is the format of the class?**

The class consists of lecture videos, which are broken into small chunks, usually between eight and twelve minutes each. Some of these may contain integrated quiz questions. There will also be standalone quizzes that are not part of video lectures. There will be approximately two hours worth of video content per week.

- **What should I know to take this class?**

How to program in at least one programming language (like C, Java, or Python); familiarity with proofs, including proofs by induction and by contradiction; and some discrete probability, like how to compute the probability that a poker hand is a full house. At Stanford, a version of this course is taken by sophomore, junior, and senior-level computer science majors.

<https://www.coursera.org/course/algo>

# Algorithms I: Course Syllabus

5 week course presenting approximately 50% of the material presented in the Stanford CS161 course on campus (~ the first half)

**Note:** the syllabus may undergo minor revisions throughout the course.

**Abbreviations in suggested readings refer to the following textbooks:**

- CLRS - Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms (3rd edition)*
- DPV - Dasgupta, Papadimitriou, and Vazirani, *Algorithms*
- KT - Kleinberg and Tardos, *Algorithm Design*
- SW - Sedgewick and Wayne, *Algorithms (4th edition)*

## March 12-18 (click for overview)

### Topics (Video Sections

#### I,II,III)

- Introduction
- Merge Sort
- Asymptotic Notation
- Guiding Principles of Algorithm Analysis
- Divide & Conquer Algorithms

### Homework

- **Due Mar 25:**
- Problem Set #1: Divide & Conquer / Asymptotic Analysis
- Programming Assignment #1: Counting Inversions!

### Suggested Readings:

- CLRS: Chapter 2, 3, and 4 (though Section 4.2), and Sections 28.1 and 33.4
- DPV: Sections 0.3, 2.1, 2.3, 2.5
- KT: Sections 2.1, 2.2, 2.4, 5.1, and 5.3-5.5
- SW: Sections 1.4 and 2.2

## March 19-25 (click for overview)

### Topics

- Master Method
- QuickSort

### Homework

- **Due April 1:**
- Problem Set #2: QuickSort and the Master Method
- Programming Assignment #2: Counting Comparisons in Quick Sort

### Suggested Readings:

- CLRS Chapter 4 (Sections 4-6) and Chapter 7
- DPV Section 2.2
- KT Sections 5.2 and 13.5
- SW Section 2.3

## March 26-April 1 (click for overview)

### Topics

- Final Thoughts on Sorting & Searching
- Introduction to Graph Algorithms : Graph Representations & Min Cuts in Graph

### Homework

- **Due April 8**
- Problem Set #3: Randomized Selection & Min Cuts in Graphs
- Programming Assignment #3: Karger's Min Cut Algorithm

### Suggested Readings:

- CLRS Chapter 9, 22 (Only 22.1)
- SW Chapter 4, Section 4.1
- KT Chapter 13, Section 13.2,13.5
- DPV Chapter 3 (only 3.1)

## April 2-8 (click for overview)

### Topics

- Graph Search: Breadth-First Search, Depth-First Search
- Applications: Topological Sort, Connected Components

### Homework

- **Due April 15**
- Problem Set #4: Graphs, BFS, DFS, Topological Sort
- Programming Assignment #4: Computing SCCs

### Suggested Readings:

- CLRS Chapter 22
- SW Chapter 4, Section 4.1,4.2
- KT Chapter 3, Section 3.5, 3.6
- DPV Chapter 3

## April 9-15 (click for overview)

### Topics

- Dijkstra's Shortest-Path Algorithm
- Important data structures and how to use them
- Heaps
- Hash Tables/Maps & Applications

### Homework

- **Due April 22**
- Problem Set #5: Dijkstra, Heaps, Hash Maps
- Programming Assignment #5: 2 Sum Problem using HashTables

### Suggested Readings:

- CLRS Chapter 6,11, 24 (Sections 3,4)
- SW Chapter 3 (3.4), Chapter 4 (4.4)

## April 18-26 --- Click for Final Exam Details

Covers the basics: basic algorithm analysis, divide and conquer paradigm and applications to sorting algorithms, introduction to graphs and their analysis, introduction to basic data structures such as heaps, hash tables, balanced trees, etc.

# Course Logistics

## (From Course Website)

### Weekly Programming Assignments and Problem Sets

- The Programming Assignment and Problem set for each week will be available from Monday (12:00 AM PDT).
- The soft deadline is 14 days later (11:59 PM PDT on the second Sunday).
- The hard deadline is 11:59 PM PDT on April 22.
- Submissions are not allowed after the hard deadline.
- Submissions in between the soft deadline and hard deadline will receive 50% credit.
- For each problem set you are allowed a maximum of two attempts (we'll use the better score).
- For each programming assignment you're allowed a maximum of 5 attempts (we'll use the best score).

Weekly assignments to force you to keep pace

### Grading / Certificate of Accomplishment

- The grading will be 40% problem sets, 30% programming questions, and 30% for the final exam.
- Each programming assignment will be scaled to 6 points, every problem set will be scaled to 8 points
- Final exam will be worth 30 points
- Anybody getting over 70% of the maximum points will receive a certificate of accomplishment.

Grading actually helps to provide incentive to try!

### Theory Problems

- These are totally optional theory questions (no deadlines or credit given)
- We recommend that you attempt these questions and discuss them in the discussion forums to develop a deeper understanding of the design and analysis of algorithms.

Extra/Optional stuff to think about

### Video Lectures

- Lectures will be available weekly (for a total of five weeks).
- On the right side of the lectures you can find the lecture notes in .ppt, .pdf formats
- The subtitles are also available as a .txt file for your convenience
- You can download the .mov file format of the lectures for offline viewing


Video Lectures – the heart of the course

# Example of Problem Assignment Question

## Question 2

You are given functions  $f$  and  $g$  such that  $f(n) = O(g(n))$ . Is

$f(n) * \log_2(f(n)^c) = O(g(n) * \log_2(g(n)))$ ? (Here  $c$  is some constant  $> 0$ . You can assume that  $f$  and  $g$  are always bigger than 1.

Your Answer	Score	Explanation
<input checked="" type="radio"/> True	1.00 	That's correct! Roughly, because the constant $c$ in the exponent is inside a logarithm, it becomes part of the leading constant and gets suppressed by the big-Oh notation.
Total	1.00 / 1.00	

# Example of Programming Assignment

## Question 1

Download the text file [here](#). Zipped version [here](#). (Right click and save link as)

Links to data files to use as input to program

The file contains the edges of a directed graph. Vertices are labeled as positive integers from 1 to 875714. Every row indicates an edge, the vertex label in first column is the tail and the vertex label in second column is the head (recall the graph is directed, and the edges are directed from the first column vertex to the second column vertex). So for example, the 11<sup>th</sup> row looks like: "2 47646". This just means that the vertex with label 2 has an outgoing edge to the vertex with label 47646

Your task is to code up the algorithm from the video lectures for computing strongly connected components (SCCs), and to run this algorithm on the given graph.

Output Format: You should output the sizes of the 5 largest SCCs in the given graph, in decreasing order of sizes, separated by commas (avoid any spaces). So if your algorithm computes the sizes of the five largest SCCs to be 500, 400, 300, 200 and 100, then your answer should be "500,400,300,200,100". If your algorithm finds less than 5 SCCs, then write 0 for the remaining terms. Thus, if your algorithm computes only 3 SCCs whose sizes are 400, 300, and 100, then your answer should be "400,300,100,0,0".

434821,968,459,313,211

Input box for answer

Your Answer	Score	Explanation
434821,968,459,313,211	6.00	Instant Gratification
Total	6.00 / 6.00	

Actually interesting example... "natural" solution is to use recursion and this worked great on my test data set that I had made up. But... I blew my recursion stack when running on the full data set and, ultimately, was forced to find a non-recursive solution. That turned out to be the hardest problem to solve for the entire course!

# So... Was It Worth The Time?

- Bottom Line: For me the answer is YES
- Direct Results for the Pass 8 effort:
  - New techniques for constructing Tracker Trees
  - Much better techniques for providing ordered lists of Trees
  - New approaches for quickly traversing the trees depending on which information wanted
  - Idea for building a front end which significantly reduces combinatoric possibilities
  - All have led to significant performance gain for the Pass 8 Tracker code
    - Tree Based Tracking average cpu time performance now BETTER than the old approach
    - Worst case performance is still a bit worse than the old approach but have eliminated most of the “arbitrary” cutoffs
- Indirect/intangible
  - Many interesting topics
  - Examples presented in course are from current problems in a variety of other fields which are interesting to understand
  - You never know where a good idea might come from!

# Final Thoughts

- The Algorithms course is offered through a new joint project between several universities to offer online courses
  - A growing number of fields, from Computer Science to Humanities
  - All following more or less the same format
  - Typically starting every quarter and running 5-10 weeks
- Upcoming Computer Science courses include
  - Algorithms II, Computer Vision, Computer Architecture, Automata, Intro to Logic, Compilers, Machine Learning, etc.
- For those interested in the Algorithms course:
  - The Algorithms I course is scheduled to run again at the beginning of the Summer
  - Algorithms II is currently planned to run in the Fall
- Visit the Coursera website for more information!
  - [www.coursera.org](http://www.coursera.org)