The Story So Far....

- Introduction to Algorithms via Socks
- Decision Problems on Lists
- Analysis of Algorithms
  - Song Growth Rates
  - Big O Notation
  - Exponential Misuses
- Graphs
The Story So Far....

- Introduction to Algorithms via Socks
- Decision Problems on Lists

- Analysis of Algorithms
- Song Growth Rates
- Big O Notation
- Exponential Misuses
- Graphs
The Story So Far....

- Introduction to Algorithms via Socks
- Decision Problems on Lists

- Analysis of Algorithms
  - Song Growth Rates
  - Big O Notation
  - Exponential Misuses

- Graphs
The Story So Far....

- Introduction to Algorithms via Socks
- Decision Problems on Lists

- Analysis of Algorithms
  - Song Growth Rates
  - Big O Notation
  - Exponential Misuses

- Graphs

What? How Fast? Examples!
Still to Come...

- Graph Algorithms
- Sorting!
- Uncomputable Things
  - Halting Problem
- Huffman Codes
- Robots
Still to Come...

- Graph Algorithms
- Sorting!
- Uncomputable Things
- Halting Problem
- Huffman Codes
- Robots
Still to Come...

- Graph Algorithms
- Sorting!
- Uncomputable Things
  - Halting Problem
- Huffman Codes
- Robots

Examples!

Limits!
Still to Come...

- Graph Algorithms
- Sorting!
- Uncomputable Things
  - Halting Problem
- Huffman Codes
  - Robots
Graphical
Graphs Everywhere
Graphs Everywhere
Graphs Everywhere
Algorithms on Graphs

We can represent a graph in the computer by a list of nodes, and a function that, given a node \( i \), returns the list of nodes to which \( i \) is linked.
A node $j$ is reachable from a node $i$ if there is a path that begins at $i$ and ends at $j$.

Let’s list all the nodes reachable from $i$.

Any node that is reachable from a node that $i$ is linked to is also reachable.
Don’t Revisit!

- What goes wrong? Once we realize we can reach some node, we should mark it as “reached” and never pursue it again.
To-Do List Helps

• Keep track of what you need to do later once your current activity is done.

• Stack. (See ‘paintcan’ video.)
Breadth First Search
Breadth First Search

ToDo:
Breadth First Search

ToDo: A
Breadth First Search

ToDo: B
Breadth First Search

ToDo: C, D
Breadth First Search

ToDo: D,F
Breadth First Search

Todo: F, E
Breadth First Search

ToDo: E
Breadth First Search

ToDo:
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
Depth First
So, how does Google do it?

I. Web crawl: download known pages, collect links to other pages, repeat.

II. Indexing: Build a giant index that associates each word with a list of pages on which it appears.

III. Distributed search: Use lots and lots and lots of computers to do fast lookups.
Sorting Algorithms

• Another name for the lecture is “Google II”.
• Sorting is a great topic in CS:
  - relatively simple
  - extremely important
  - illustrates lots of different algorithms and analysis techniques

There’s more than one way to skin a cat.
What Can We Do?

• All the information is there, and we can sift through it.

• But, it’s slow and error-prone to skim through every page every time we want to find something.

• If there are $N$ words (total) on the web pages, how long would it take to sift through them each time? (Use “big O” notation.)

• How can we organize the data to simplify?
• Phonebook, look for a last name vs. look for a first name.

• “Is there a pair that sums to 86?” Don’t have to consider all pairs.

• Is there a repeated number in the list?

• Not to mention min, max, median.
Selection Sort

• Idea is quite simple. We go through the list one item at a time.

• We keep track of the smallest item we’ve found.

• When we’re through the list, we pull the smallest item out and add it to a list of sorted items.

• We repeat until all the items have been removed.
Selection Code

About a 2 1/2 min. to sort 100 items.
Selection Sort Analysis

• How many comparisons does Selection Sort do in the worst case? Assume the list is length $N$. Hint: What song is it like? You can use “big O” notation.

• Does it matter whether the list is sorted or not?
Other Sorting Approaches

• How else can you imagine sorting?
• Fewer comparisons than $O(N^2)$?
  - bubblesort
  - counting sort
  - insertion sort
  - Shell sort
Guess Who?

- Each player picks a character.
- Players take turns asking each other yes/no questions.
- First player to uniquely identify the other player’s character wins!
Mindreader: Set Cards
Mindreader: Set Cards

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Cross-Hatched?
Cross-Hatched?
Cross-Hatched?
Squiggle?
Squiggle?
Squiggle?
Insight

- Each question splits the remaining set of possibilities into two subsets (yes and no).
Insight

• Each question splits the remaining set of possibilities into two subsets (yes and no).

• We want to pick a question so that the larger of the two subsets is as small as possible.
Insight

- Each question splits the remaining set of possibilities into two subsets (yes and no).

- We want to pick a question so that the larger of the two subsets is as small as possible.

- Half!
Insight

• Each question splits the remaining set of possibilities into two subsets (yes and no).

• We want to pick a question so that the \textit{larger} of the two subsets is as \textit{small} as possible.

• Half!
Insight

• Each question splits the remaining set of possibilities into two subsets (yes and no).

• We want to pick a question so that the larger of the two subsets is as small as possible.

• Half!

• How many questions?
  • $n=1$, questions = 0
Insight

• Each question splits the remaining set of possibilities into two subsets (yes and no).

• We want to pick a question so that the larger of the two subsets is as small as possible.

• How many questions?
  • $n=1$, questions = 0
  • $n=2$, questions = 1

• Half!
Insight

- Each question splits the remaining set of possibilities into two subsets (yes and no).
- We want to pick a question so that the larger of the two subsets is as small as possible.
- Half!

How many questions?

- $n=1$, questions = 0
- $n=2$, questions = 1
- $n=4$, questions = 2
Insight

• Each question splits the remaining set of possibilities into two subsets (yes and no).

• We want to pick a question so that the larger of the two subsets is as small as possible.

• Half!

• How many questions?
  • $n=1$, questions = 0
  • $n=2$, questions = 1
  • $n=4$, questions = 2
  • $n=8$, questions = 3
Insight

- Each question splits the remaining set of possibilities into two subsets (yes and no).
- We want to pick a question so that the larger of the two subsets is as small as possible.
- Half!

How many questions?
- $n=1$, questions = 0
- $n=2$, questions = 1
- $n=4$, questions = 2
- $n=8$, questions = 3
- $n=16$, questions = 4
Insight

- Each question splits the remaining set of possibilities into two subsets (yes and no).

- We want to pick a question so that the larger of the two subsets is as small as possible.

- Half!

- How many questions?

  - $n=1$, questions = 0
  - $n=2$, questions = 1
  - $n=4$, questions = 2
  - $n=8$, questions = 3
  - $n=16$, questions = 4
  - $n$, questions = $\log_2 n$. 
Binary Search
Binary Search

- Let’s say we have a sorted list of $n$ items.
Binary Search

• Let’s say we have a sorted list of $n$ items.

• How many comparisons do we need to make to find where a new item belongs in the list?
Binary Search

- Let’s say we have a sorted list of \( n \) items.
- How many comparisons do we need to make to find where a new item belongs in the list?
- Can start at the bottom and compare until the new item is bigger.
Binary Search

- Let’s say we have a sorted list of \( n \) items.
- How many comparisons do we need to make to find where a new item belongs in the list?
- Can start at the bottom and compare until the new item is bigger.
- Maximum number of comparisons?
Binary Search

• Let’s say we have a sorted list of $n$ items.

• How many comparisons do we need to make to find where a new item belongs in the list?

• Can start at the bottom and compare until the new item is bigger.

• Maximum number of comparisons?

• One for each position: $n$. 
Binary Search

- Let’s say we have a sorted list of $n$ items.
- How many comparisons do we need to make to find where a new item belongs in the list?
- Can start at the bottom and compare until the new item is bigger.
- Maximum number of comparisons?
- One for each position: $n$.
- We can ask better questions: bigger than the halfway mark?
Binary Search

- Let’s say we have a sorted list of $n$ items.
- How many comparisons do we need to make to find where a new item belongs in the list?
- Can start at the bottom and compare until the new item is bigger.
- Maximum number of comparisons?
- One for each position: $n$.
- We can ask better questions: bigger than the halfway mark?
- That gets us: $\log (n+1)$!
Binary Search Sort

- Using $O(\lg N)$ comparisons, can find where to insert the next item.

- Since we insert $N$ items, comparisons is $O(N \lg N)$ in total.

- Can’t quite implement it that way, though: Once we find the spot, $O(N)$ to stick it in.

- However, other algorithms are really $O(N \lg N)$.

- Hillis mentions Quick Sort and Merge Sort.
Binary Sort Code

About a 1/2 min. to sort 100 items.