Chapter 5: Algorithms and Heuristics

CS105: Great Insights in Computer Science
Last Time...

- Selection Sort
  - Mentioned Bubble Sort
- Binary Search Sort
  - Based on $\lg(n)$
- QuickSort
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
Cross-Hatched?
Cross-Hatched?
<table>
<thead>
<tr>
<th>Cross-Hatched?</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image-url" alt="Images" /></td>
</tr>
</tbody>
</table>
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 \textit{larger} of the two subsets is as \textit{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 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 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.
- Half!

- How many questions?
  - $n=1$, questions $= 0$
  - $n=2$, questions $= 1$
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 = $\log n$. 

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: \( \lg (n+1)! \)
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.
Quicksort

- **quicksort**: Another sorting algorithm.
- **Idea**: Break the list of $n+1$ elements into the median and two lists of $n/2$. The two lists are those smaller than the median and those larger than the median.
- Sort the two lists separately.
- Glue them together: All $n$ are sorted.
Quicksort Example

- Original list:
  - [56, 80, 66, 64, 37, 36, 91, 48, 17, 20, 86, 89, 41, 1, 96, 12, 74]
- Median is 56; smaller: [37, 36, 48, 17, 20, 41, 1, 12]
  - bigger: [80, 66, 64, 91, 86, 89, 96, 74]
- Sort each; smaller: [1, 12, 17, 20, 36, 37, 41, 48]
  - bigger: [64, 66, 74, 80, 86, 89, 91, 96]
- Glue:
  - [1, 12, 17, 20, 36, 37, 41, 48, 56, 64, 66, 74, 80, 86, 89, 91, 96]
But...

- If we could find the median, the whole sorting process would be pretty easy.

- Sufficient to split anywhere in the middle half at least half the time: Still $O(n \log n)$.

- Pick a random list element. 25% of the time, it will be in the 1st quarter of the sorted list, 25% of the time in the last quarter, and 50% in the middle half.
But...

- If we could find the median, the whole sorting process would be pretty easy.

- Sufficient to split anywhere in the middle half at least half the time: Still $O(n \log n)$.

- Pick a random list element. 25% of the time, it will be in the 1st quarter of the sorted list, 25% of the time in the last quarter, and 50% in the middle half.
Quicksort’s Flow

• Pick an item, any item (the “pivot”).
• Partition the list as to less (left) or greater than (right) pivot.
• Sort the two halves (recursively!).
Web Search, Again

• We’ve seen two of the major steps needed to implement a web search engine:
  • gather up pages using graph search
  • index the words using sorting

• In a later lecture, we’ll talk about the last step: using more than one computer to respond quickly to millions of queries a day.
Heuristics

• Hillis makes a distinction between:
  - **algorithms**: “fail safe” procedures. They are guaranteed to do the job.
  - **heuristics**: rules of thumb. They might get pretty close to the right answer much of the time.

• In fact, we often refer to both categories as “algorithms”, but that’s ok.
Let’s return to the (known hard) problem of subset set.

Is there a subset that sums to 433?

One algorithm is to list all $2^n$ subsets and check each one.

If we’re happy just getting close, we could use hillclimbing.
Hill Climbing Example

48, 41, 14, 46, 31, 2, 27, 12, 22, 71, 44, 63, 33, 64, 83, 28, 96, 87, 52

• Target = 433

• Start with a random subset: [41, 46, 2, 27, 12, 71, 44, 33, 64, 28], sum = 368.

• What number can we add to the set to get closer to the target? Including 63 gets us a sum of 431.

• Can’t get any closer to 433 by adding or removing a single number. Stuck at the top of a hill. Can start again with a different subset.
Hill Climbing Limitations

- Can get trapped in local minimum: need to go further to get closer.
- No guarantee that a maximum will be found.
- Can even be slow to find a local minimum!
- With the wrong scoring function, finding the right answer can be like a needle in a haystack.
Chapter 4: How Universal Are Turing Machines?

CS105: Great Insights in Computer Science
Philosophy

- Some fields are ignored by philosophers (civil engineering, physical chemistry).
- Computer Science is not so lucky.
  - What does it mean to “know” something?
  - What is “intelligence”?
  - Do objects “compute” their movements?
Great Humiliations?

- **Freud:**
  - Astronomy (Galileo): Our world is just another world.
  - Natural History (Darwin): Our species is just another species.
  - Psychoanalysis (Freud): Consciousness is just another mental process.

- **Hillis:**
  - Computer Science (Turing?): Our mind is just another computing device.
To fight this idea, philosophers are fond of finding things that computers can’t do.

- No one gets wet from a computer simulation of a hurricane (Dennett).

- Translate human languages (Dreyfus).

- Be self reflective or conscious (Lucas).
Kinds of Computation

Von Neumann architecture:
finite-state machine +
memory
Kinds of Computation

Turing machine:
finite-state machine + memory tape

Von Neumann architecture:
finite-state machine + memory
Kinds of Computation

Turing machine:
finite-state machine + memory tape

Post Correspondence:
string extension rules

Von Neumann architecture:
finite-state machine + memory
Kinds of Computation

Turing machine:
finite-state machine +
memory tape

Church’s lambda calculus:
mathematical function
notation

Von Neumann architecture:
finite-state machine +
memory

Post Correspondence:
string extension rules
Kinds of Computation

- Turing machine: finite-state machine + memory tape
- Church’s lambda calculus: mathematical function notation
- Post Correspondence: string extension rules
- Von Neumann architecture: finite-state machine + memory
- Chomsky’s context-sensitive grammars: string rewrite rules
Kinds of Computation

- Turing machine: finite-state machine + memory tape
- Church’s lambda calculus: mathematical function notation
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines
Kinds of Computation

- Church’s lambda calculus: mathematical function notation
  - can simulate
- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines
Kinds of Computation

- Turing machine: finite-state machine + memory tape
- Church’s lambda calculus: mathematical function notation
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines

Can simulate
Kinds of Computation

Church’s lambda calculus: mathematical function notation
Von Neumann architecture: finite-state machine + memory
Chomsky’s context-sensitive grammars: string rewrite rules
Post Correspondence: string extension rules
Turing machine: finite-state machine + memory tape

programming languages: instructions, loops, subroutines

can simulate
Kinds of Computation

- Church’s lambda calculus: mathematical function notation
  - can simulate
- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines
Kinds of Computation

- **Church’s lambda calculus**: mathematical function notation
- **Turing machine**: finite-state machine + memory tape
- **Von Neumann architecture**: finite-state machine + memory
- **Post Correspondence**: string extension rules
- **Chomsky’s context-sensitive grammars**: string rewrite rules
- **programming languages**: instructions, loops, subroutines

*can simulate*
Kinds of Computation

- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Church’s lambda calculus: mathematical function notation
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines

Church’s lambda calculus can simulate Turing machines.
Kinds of Computation

- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Church’s lambda calculus: mathematical function notation
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules

Programming languages: instructions, loops, subroutines
Kinds of Computation

- Church’s lambda calculus: mathematical function notation
- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines

(can simulate)
Kinds of Computation

- Church’s lambda calculus: mathematical function notation
- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines

can simulate
Kinds of Computation

- Church’s lambda calculus: mathematical function notation
- Turing machine: finite-state machine + memory tape
- Von Neumann architecture: finite-state machine + memory
- Post Correspondence: string extension rules
- Chomsky’s context-sensitive grammars: string rewrite rules
- Programming languages: instructions, loops, subroutines

Turing machine and Von Neumann architecture can simulate each other.
Jonesing for Mario
Jonesing for Mario
Jonesing for Mario
Jonesing for Mario

Nestopia
Jonesing for Mario

Nestopia
Jonesing for Mario

Nestopia
Jonesing for Mario

Nestopia
Jonesing for Mario

Nestopia

VirtualBox
Jonesing for Mario

Nestopia

VirtualBox
Church-Turing Thesis

- Usually taken to mean that any machine computation can be carried out on a Turing machine.
- Sometimes expanded to mean that any physical system can be simulated on a Turing machine.
- And, since brains are physical systems, our minds must be equivalent to Turing machines!