Lecture 11:
Sorting Algorithms

CS442: Great Insights in Computer Science
Michael L. Littman, Spring 2006

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
Google...

- Last time, I said Google does its thing in a couple of very significant steps:

I. Collect pages from the web (graph search).

II. Index them.

III. Respond to queries.
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?

Sort, Remove Duplicates

| 4 a  | 9 course | 3 frames | 9 http | 3 no  | 4 python | 9 there |
| 4 a  | 4 courses | 3 frameset | 9 i | 3 noframes | 3 noresize | 3 this |
| 4 add | 9 courses | 9 from | 9 in | 9 not | 9 noth | 9 this |
| 4 address | 4 cs | 9 google | 9 include | 9 index | 9 odf | 3 rows |
| 9 and | 9 cs | 9 googleblackout index | 9 insights | 4 old | 4 of | 3 rutgers |
| 9 any | 9 did | 9 googletest | 9 index | 9 on | 3 rutgers | 3 title |
| 4 apache | 4 differences | 9 great | 9 is | 9 other | 3 science | 3 title |
| 4 at | 4 directory | 4 h | 9 its | 9 own | 9 to | 3 title |
| 4 b | 9 discovery | 9 h | 9 it | 9 own | 3 sc | 3 title |
| 9 b | 4 doctype | 9 h | 9 its | 9 owner | 3 scrol | 3 science |
| 3 babes | 9 documents | 3 head | 3 january | 3 p | 4 server | 9 sever |
| 3 banner | 3 head | 3 has | 3 left | 9 p | 3 socks | 3 search |
| 3 body | 3 h | 3 has | 3 left | 3 page | 3 socks | 4 server |
| 4 body | 4 h | 3 history | 3 main | 9 page | 9 page | 3 server |
| 3 browser | 4 edu | 9 history | 9 match | 9 pages | 9 page | 9 server |
| 3 but | 9 edu | 9 homepage | 9 ml | 9 parent | 9 page | 3 server |
| 4 c | 4 en | 9 how | 9 ml | 9 port | 9 page | 3 server |
| 3 cols | 4 exception | 4 href | 4 ml | 4 pdf | 3 target | 9 sever |
| 9 computer | 9 explicitly | 9 href | 4 mlittman | 4 port | 9 target | 9 sever |
| 9 concatenating | 4 final | 3 htm | 4 mlittman | 4 port | 9 target | 9 server |
| 9 consecutive | 9 find | 3 html | 3 monica | 4 public | 9 target | 9 server |
| 9 consists | 9 for | 4 html | 3 name | 4 purpose | 9 target | 9 server |
| 3 contents | 3 frame | 9 html | 4 nim | 4 py | 3 target | 9 server |
| 3 frame | 9 html | 4 nim | 4 py | 3 them | 9 server | 9 server |
Sort by Birthdate

- Pick 8 people.
- Can only ask a pair of people who has the later birthdate (month and day).
- insertionSort(namesClass)

Insertion Sort

- Idea is quite simple.
- We go through the list one item at a time.
- We keep a sorted list of everything we’ve gone through so far.
- When we need to put a new item into the list, we go through the sorted list in order to figure out where it fits.
## Insertion Sort Analysis

- **How many comparisons does Insertion Sort do in the worst case?** Assume the list is length $N$. Hint: What song is it like? You can use “big O” notation.
- **What kind of list would force it to do this many comparisons?**
- **How many comparisons does Insertion Sort do in the best case?** Ditto.

## Other Sorting Approaches

- **How else can you imagine sorting?**
- **Fewer comparisons than $O(N^2)$?**
  - bubblesort
  - counting sort
  - selection 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
**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**

- 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.

```plaintext
mergeSort(names)
```

**Merge Sort**

- View all the items as separate sorted lists.
- Pick the two shortest lists and combine them into a single sorted list:
  - Compare the first items. Move smaller one to end of the combined list.
- Repeat until one listed is empty.
- Repeat until only a single list is left.
Merge Sort Analysis

- To merge two lists of length $N$ requires at most $2N$ comparisons.
- If there are $N$ items in $L$ lists of length $N/L$, after one merging pass, $N$ comparisons, $L/2$ lists of length $2N/L$.
- Length doubles each time.
- Initially, $L = N$ lists of length 1 each.
- After $\log N$ merging passes, 1 list of length $N$.
- Total comparisons: $O(N \log N)$.

Lower Bound

- We’ve shown that we can sort in $O(N \log N)$ comparisons.
- What if someone comes along and does it better?
- We need to protect ourselves and prove a “lower bound”: that is, to show that nothing less than $N \log N$ will suffice.
- Let’s return to “Guess Who?”. 
**Sorting Lower Bound**

- If we are asking yes/no questions to uniquely identify one item out of $n$, how many questions do we need in the worst case?
- Might be as many as $\lg n$, since each question cannot exclude more than half.
- Sorting $N$ elements identifies the correct ordering using just yes/no questions.

**Counting Orderings**

- How many ways to order $N$ elements?
  - 1: 1
  - 2: 2
  - 3: $6 = 3 \times 2$
  - 4: $24 = 4 \times 3 \times 2$
  - 5: $120 = 5 \times 4 \times 3 \times 2$
- $N$: $N! = N \times (N-1) \times (N-2) \times \ldots \times 2 \times 1$
  - Known as the *factorial* function.
  - Thus, sorting must find the unique sorted ordering from a set of $N!$ possibilities using just yes/no questions.
A Little Math

\[ N! = \]
\[ 1 \times 2 \times 3 \times \ldots \times \frac{N}{2} \times (\frac{N}{2} + 1) \times \ldots \times N \]
\[ > (\frac{N}{2} + 1) \times \ldots \times N \]
\[ > \frac{N}{2} \times \ldots \times \frac{N}{2} \]
\[ = \frac{N}{2}^{\frac{N}{2}} \]

Number of comparisons to sort \( N \) items

- \# of yes/no questions to pick one out of \( N! \)
- \# of yes/no questions to pick one out of \( \frac{N}{2}^{N/2} \)
- \( \log \frac{N}{2}^{N/2} \)

= \( \frac{N}{2} \log \frac{N}{2} \)

or, essentially \( N \log N \). \( O(N \log N) \) wins!

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.
Next Time

- NP Completeness.
- Still in Hillis, Chapter 5.