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.
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.

Three Pages of Words
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
9 a  4 courses  3 frameset  9 i  3 noframes  3 noresize  3 this
4 add  9 courses  9 from  9 in  9 include  9 not
4 address  4 cs  9 google  9 googleblackout4 index  4 of
9 and  9 cs  9 googletest  9 index  9 insights
9 any  9 did  9 great  9 is
4 apache  4 differences  9 other
9 at  4 directory  4 h  9 old
9 b  9 discovery  9 h  9 on
9 b  4 doctype  9 has  9 own
3 babes  9 documents  3 head  9 is
3 banner  9 does  4 head  9 its
3 body  3 doesn  3 header
4 body  4 dtd  9 header  9 january
3 browser  4 edu  9 history  3 left
3 but  9 edu  9 homepage
4 c  4 en  9 how  9 match
3 cols  4 exception  9 href  4 ml
9 computer  9 explicitly  9 href  4 mllttman
9 concatenating  4 final  3 htm  9 mllttman
9 concoced  9 find  3 html  9 primary
9 consists  9 for  4 html  9 public
3 contents  3 frame  9 html  4 nim
4 at  4 directory  4 h  9 is
9 at  9 discovery  9 h  9 its
9 b  9 discovery  9 h  9 it
9 b  4 doctype  9 has  9 january
3 babes  9 documents  3 head  9 is
3 banner  9 does  4 head  9 its
3 body  3 doesn  3 header
4 body  4 dtd  9 header  9 january
3 browser  4 edu  9 history  3 left
3 but  9 edu  9 homepage
4 c  4 en  9 how  9 match
3 cols  4 exception  9 href  4 ml
9 computer  9 explicitly  9 href  4 mllttman
9 concatenating  4 final  3 htm  9 mllttman
9 concoced  9 find  3 html  9 primary
9 consists  9 for  4 html  9 public
3 contents  3 frame  9 html  4 nim
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.

Code

```python
def Selection(l):
    sorted = []
    while len(l) > 0:
        (smallest, rest) = findSmallest(l)
        sorted = sorted + [smallest]
        l = rest
    return sorted

def findSmallest(l):
    smallest = l[0]
    rest = []
    for i in range(1,len(l)):
        if l[i] < smallest:
            rest = rest + [smallest]
            smallest = l[i]
        else:
            rest = rest + [l[i]]
    return (smallest, rest)
```
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
Cross-Hatched?

Squiggle?
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.

**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.
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).

Code

def Quicksort(l):
    if len(l) <= 1: return l
    pivot = l[randint(0,len(l)-1)]
    (left,equal,right) = partition(l,pivot)
    return Quicksort(left) + equal + Quicksort(right)

def partition(l,pivot):
    left = []
    right = []
    equal = []
    for item in l:
        if item < pivot: left = left + [item]
        if item > pivot: right = right + [item]
        if item == pivot: equal = equal + [item]
    return (left,equal,right)
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 list 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.
• Length doubles each time.
• Initially, \(L = N\) lists of length 1 each.
• After \(\lg N\) merging passes, 1 list of length \(N\).
• Total comparisons: \(O(N \lg N)\).
Lower Bound

- We’ve shown that we can sort in $O(N \lg N)$ 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 $N \lg N$ 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: $120 = 5 \times 4 \times 3 \times 2$

- $N$: $N! = N \times (N-1) \times (N-2) \times ... \times 2 \times 1$

- 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

$$N! = 1 \times 2 \times 3 \times ... \times N/2 \times (N/2 + 1) \times ... \times N$$

- # 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 $N/2$
- $\lg N/2$
- $= N/2 \lg N/2$
- or, essentially $N \lg N$. $O(N \lg N)$ 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.