Quickest Sorting and Double Deltas

How much work is insert-sort?

```scheme
(define (insert-sort lst cf)
  (if (null? lst) null
      (insert-one (car lst) (insert-sort (cdr lst) cf) cf)))

(define (insert-one el lst cf)
  (if (null? lst) (list el)
      (if (cf el (car lst)) (cons el lst)
          (cons (car lst) (insert-one el (cdr lst) cf)))))
```

Running time of insert-sort is in $\Theta(n^2)$ worst case (reverse list), but is $\Theta(n)$ best case (sorted list).

One Slide Summary
- **Insert-sort** is $\Theta(n^2)$ worst case (reverse list), but is $\Theta(n)$ best case (sorted list).
- A recursive function that divides its input in **half** each time is often in $\Theta(\log n)$.
- If we could divide our input list in half rapidly, we could do a **quicker sort**: $\Theta(n \log n)$.
- **Sorted binary trees** are an efficient data structure for maintaining sorted sets.
- British codebreakers used **cribs** (guesses), brute force, and **analysis** to break the Lorenz cipher. Guessed wheel settings were likely to be correct if they resulted in a message with the right linguistic properties for German (e.g., repeated letters).

Which is better?

- Is insert-sort faster than best-first-sort?

Shuttle Rescue Mission
Monday Feb 23 and Wednesday Feb 25
MEC 205 until 5:30pm
http://shuttle.cs.virginia.edu:8080/
Build and program Lego Mindstorms robot to remotely sense and navigate a barren environment and retrieve a life pod from a crater.

Exam 1 Extra Credit: *either* show up and watch one day or write paragraph about how to do it

Outline
- Insert-sort
- Going half-sies
- Sorted binary trees
- Quicker-sort
- WWII Codebreaking

PS4 Written can be turned in during structured office hours.
> (insert-sort < (revintsto 20))
\[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]\nRequires 190 applications of <

> (insert-sort < (intsto 20))
\[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]\nRequires 19 applications of <

> (insert-sort < (rand-int-list 20))
\[0 11 16 19 23 26 31 32 32 34 42 45 53 63 64 81 82 84 84 92]\nRequires 104 applications of <

> (best-first-sort < (intsto 20))
\[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]\nRequires 210 applications of <

> (best-first-sort < (rand-int-list 20))
\[4 4 16 18 19 20 23 32 36 51 53 59 67 69 73 75 82 82 88 89]\nRequires 210 applications of <

---

**best-first-sort vs. insert-sort**

- Both are \(\Theta(n^2)\) worst case (reverse list)
- Both are \(\Theta(n^2)\) when sorting a randomly ordered list
  - But insert-sort is about twice as fast
- insert-sort is \(\Theta(n)\) best case (ordered input list)

---

**Can we do better?**

(quicker-insert < 88
\(\text{list } 1 2 3 5 6 23 63 77 89 90)\))

Suppose we had procedures
(first-half lst)
(second-half lst)
that quickly divided the list in two halves?

---

**quicker-insert using halves**

(\(\text{define (quicker-insert el lst cf)}\))
(if (null? lst) (list el) ;; just like insert-one
\(\text{(if (null? (cdr lst))}\)
\(\text{(if (cf el (car lst)) (cons el lst) (list (car lst) el))}\)
\(\text{(let ((front (first-half lst)) (back (second-half lst)))}\)
\(\text{(if (cf el (car back))}\)
\(\text{(append (quicker-insert el front cf) back))}\)
\(\text{(append front (quicker-insert el back cf))))))))\)

---

**Evaluating quicker-sort**

> (quicker-insert < 3 (list 1 2 4 5 7))
\[(\text{quicker-insert #<procedure:traced-<> 3 (1 2 4 5 7)})\]
\(\text{(define (quicker-insert el lst cf)}\))
\(\text{(if (null? lst) (list el))}\)
\(\text{(if (null? (cdr lst))}\)
\(\text{(if (cf el (car lst)) (cons el lst) (list (car lst) el))}\)
\(\text{(let ((front (first-half lst)) (back (second-half lst)))}\)
\(\text{(if (cf el (car back))}\)
\(\text{(append (quicker-insert el front cf) back))}\)
\(\text{(append front (quicker-insert el back cf))))}))\)

Every time we call quicker-insert, the length of the list is approximately halved!
How much work is quicker-sort?

Each time we call quicker-insert, the size of lst halves. So doubling the size of the list only increases the number of calls by 1.

<table>
<thead>
<tr>
<th>List Size</th>
<th># quicker-insert applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
</tr>
</tbody>
</table>

Liberal Arts Trivia:

• The argan tree, found primarily in Morocco, has a knobby, twisted trunk that allows these animals to climb it easily. The animals eat the fruit, which has an indigestible nut inside, which is collected by farmers and used to make argan oil: handy in cooking and cosmetics, but pricey at $45 per 500 ml.

Remembering Logarithms

\( \log_b n = x \) means \( b^x = n \)

What is \( \log_2 1024 \)?

What is \( \log_{10} 1024 \)?

Is \( \log_{10} n \) in \( \Theta(\log_2 n) \)?

Changing Bases

\( \log_b n = (1/\log_k b) \log_k n \)

If \( k \) and \( b \) are constants, this is constant

\( \Theta(\log_2 n) \equiv \Theta(\log_{10} n) \equiv \Theta(\log n) \)

No need to include a constant base within asymptotic operators.

Number of Applications

Assuming the list is well-balanced, the number of applications of quicker-insert is in \( \Theta(\log n) \) where \( n \) is the number of elements in the input list.
quicker-sort?

(define (quicker-sort lst cf)
  (if (null? lst) null
      (quicker-sort (car lst) (quicker-sort (cdr lst) cf) cf)))))

(define (quicker-insert el lst cf)
  (if (null? lst) (list el)
      (if (null? (cdr lst))
          (if (cf el (car lst))
              (cons el lst)
              (list (car lst) el))
          (let ((front (first-half lst))
                (back (second-half lst)))
              (if (cf el (car back))
                  (append (quicker-insert el front cf) back)
                  (append front
                      (quicker-insert el back cf)))))))

Is there a fast first-half procedure?

• No! (at least not on lists)
• To produce the first half of a list length \( n \), we need to cdr down the first \( n/2 \) elements
• So, first-half on lists has running time in \( \Theta(n) \)

Making it faster

We need to either:
1. Reduce the number of applications of insert-one in insert-sort
   Impossible – need to consider each element
2. Reduce the number of applications of quicker-insert in quicker-insert
   Unlikely… each application already halves the list
3. Reduce the time for each application of quicker-insert
   Need to make first-half, second-half and append faster than \( \Theta(n) \)

Sorted Binary Trees

A tree containing all elements \( x \) such that \( (cf x \text{ el}) \) is true

A tree containing all elements \( x \) such that \( (cf x \text{ el}) \) is false

“Nothing yet… How about you, Newton?”
Representing Trees

```scheme
(define (make-tree left el right)
  (cons el (cons left right)))

(define (tree-element tree)
  ... are trees
(null is a tree)
tree must be a non-null tree
tree must be a non-null tree
tree must be a non-null tree)
```

Representing Trees

```scheme
(define (insert-one-tree cf el tree)
  (if (null? tree)
      (make-tree null el null)
      (if (cf el (get-element tree))
          (make-tree
           (insert-one-tree cf el (get-left tree))
           (get-element tree)
           (get-right tree))
          (make-tree
           (get-left tree)
           (get-element tree)
           (insert-one-tree cf el (get-right tree))))))
```

How much work is insert-one-tree?

```
(define (insert-one-tree cf el tree)
  (if (null? tree)
      (make-tree null el null)
      (if (cf el (get-element tree))
          (make-tree
           (insert-tree cf el (get-left tree))
           (get-element tree)
           (insert-tree (get-right tree)))
          (insert-tree cf el (get-right tree)))))
```

The running time of insert-tree is in $\Theta(\log n)$ where $n$ is the number of elements in the input tree, which must be well-balanced.
quicker-insert-one

(define (quicker-insert-one cf lst)
  (if (null? lst) null
      (insert-one-tree
       cf (car lst)
       (quicker-insert-one cf (cdr lst))))))

No change (other than using insert-one-tree)... but evaluates to a tree not a list!
(((()) 1 ()) 2 ()) 5 (((()) 8 ())))

Liberal Arts Trivia: Classics
• This ancient Greek epic poem, traditionally attributed to Homer, is widely believed to be the oldest extant work of Western literature. It describes the events of the final year of the Trojan War. The plot follows Achilles and his anger at Agamemnon, king of Mycenae. It is written in dactylic hexameter and comprises 15,693 lines of verse. It begins:
  - μὴν ἄξιον ἦν ἡ Πηληϊάδεω χιλιῶν ἀχιλῆος
  - οὐλομένην, ἢ μυρί’ ἄχαιοις ἄλγε’ ἐθηκεν

Liberal Arts Trivia: Literature
• Name the author of the Age of Innocence (1920). The novel describes the upper class in New York city in the 1870s and questions the mores and assumptions of society. The title is an ironic comment on the polished outward manners of New York society, when compared to its inward machinations. The author was the first woman to win the Pulitzer Prize for Literature.

Code Breaking Intuition
• Suppose we are using a simple letter substitution cipher (i.e., replace every A with Q, etc.)
• You intercept these two messages:
• What does the first one say? What hints did you have?

12 wheels
501 pins
total (set to control wheels)
Work to break in Θ(ρ^n) so real
Lorenz is 4112/53 ~ 1 quintillion (10^18) times harder!
Breaking Fish
- Gov’t Communications HQ learned about first Fish link (Tunny) in May 1941
  - British codebreakers used “Fish” to refer to German teleprinter traffic
  - Intercepted unencrypted Baudot-encoded test messages
- August 30, 1941: Big Break!
  - Operator retransmits failed message with same starting configuration
  - Gets lazy and uses some abbreviations, makes some mistakes
    - SPRUCHNUMMER/SPRUCHNR (Serial Number)

“Two Time” Pad
- Allies have intercepted:
  \[ C_1 = M_1 \oplus K_1 \]
  \[ C_2 = M_2 \oplus K_1 \]
  Same key used for both (same starting configuration)
- Breaking message:
  \[ C_1 \oplus C_2 = (M_1 \oplus K_1) \oplus (M_2 \oplus K_1) \]
  \[ = (M_1 \oplus M_2) \oplus (K_1 \oplus K_1) \]
  \[ = M_1 \oplus M_2 \]

“Cribs”
- Know: C1, C2 (intercepted ciphertext)
  \[ C_1 \oplus C_2 = M_1 \oplus M_2 \]
- Don’t know M1 or M2
  - But, can make some guesses (cribs)
    - SPRUCHNUMMER
    - Sometimes allies moved ships, sent out bombers to help the cryptographers get good cribs
  - Given guess for M1, calculate M2
    \[ M_2 = C_1 \oplus C_2 \oplus M_1 \]
  - Once guesses that work for M1 and M2
    \[ K_1 = M_1 \oplus C_1 = M_2 \oplus C_2 \]

Intercepting Traffic
- Set up listening post to intercept traffic from 12 Lorenz (Fish) links
  - Different links between conquered capitals
  - Slightly different coding procedures, and different configurations
- 600 people worked on intercepting traffic

Breaking Traffic
- Knew machine structure, but a different initial configuration was used for each message
- Need to determine wheel setting:
  - Initial position of each of the 12 wheels
  - 1271 possible starting positions
  - Needed to try them fast enough to decrypt message while it was still strategically valuable

This is what you did for PS4 (except with fewer wheels)
Recognizing a Good Guess

- Intercepted Message (divided into 5 channels for each Baudot code bit)
  \[ Z_c = z_0 z_1 z_2 z_3 z_4 z_5 z_6 z_7 ... \]
  \[ z_{c,i} = m_{c,i} \oplus x_{c,i} \oplus s_{c,i} \]

- Look for statistical properties
  - How many of the \( z_{c,i} \)'s are 0? \( \frac{1}{2} \) (not useful)
  - How many of \( (z_{c,i+1} \oplus z_{c,i}) \) are 0? \( \frac{1}{2} \)

Double Delta

\[ \Delta Z_{c,i} = Z_{c,i} \oplus Z_{c,i+1} \]

Combine two channels:

\[ \Delta Z_{1,i} \oplus \Delta Z_{2,i} = \Delta M_{1,i} \oplus \Delta M_{2,i} > \frac{1}{2} \text{ Yippee!} \]
\[ \oplus \Delta X_{1,i} \oplus \Delta X_{2,i} = \frac{1}{2} \text{ (key)} \]
\[ \oplus \Delta S_{1,i} \oplus \Delta S_{2,i} > \frac{1}{2} \text{ Yippee!} \]

Why is \( \Delta M_{1,i} \oplus \Delta M_{2,i} > \frac{1}{2} \)
Message is in German, more likely following letter is a repetition than random
Why is \( \Delta S_{1,i} \oplus \Delta S_{2,i} > \frac{1}{2} \)
S-wheels only turn when M-wheel is 1

Actual Advantage

- Probability of repeating letters
  \[ \text{Prob}[\Delta M_{1,i} \oplus \Delta M_{2,i} = 0] \approx 0.614 \]
  3.3% of German digraphs are repeating

- Probability of repeating S-keys
  \[ \text{Prob}[\Delta S_{1,i} \oplus \Delta S_{2,i} = 0] \approx 0.73 \]
  \[ \text{Prob}[\Delta Z_{1,i} \oplus \Delta Z_{2,i} \oplus \Delta X_{1,i} \oplus \Delta X_{2,i} = 0] \]
  \[ = 0.614 \times 0.73 + (1-0.614) \times (1-0.73) \]
  \[ \Delta M \text{ and } S \text{ are } 0 \quad \Delta M \text{ and } S \text{ are } 1 \]
  \[ = 0.55 \text{ if the wheel settings guess is correct (0.5 otherwise)} \]

Using the Advantage

- If the guess of X is correct, should see higher than \( \frac{1}{2} \) of the double deltas are 0
- Try guessing different configurations to find highest number of 0 double deltas

Problem:

# of double delta operations to try one config
\[ \text{length of } Z \times \text{length of } X \]
\[ = \text{for 10,000 letter message} = 12 M \text{ for each setting} \times 7 \oplus \text{per double delta} \]
\[ = 89 M \oplus \text{operations} \]

Need a fast way to compute XOR!

Homework

- Problem Set 4 Due Today
- Study for Exam 1
  - Out on Monday