Lecture 9: Of On and Off Grounds Sorting

Coffee Bean Sorting in Guatemala

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Menu

- PS2
- Sorting
- PS3

Problem Sets

- Not just meant to review stuff you should already know
  - Get you to explore new ideas
  - Motivate what is coming up in the class
- The main point of the PSs is ***learning***, not ***evaluation***
  - Don't give up if you can't find the answer in the book
  - Discuss with other students

PS2: Question 3

Why is

(define (higher-card? card1 card2)
  (> (**card-rank** card1) (**card-rank** card2)

better than

(define (higher-card? card1 card2)
  (> (**car** card1) (**car** card2))

?

In this class, we won't worry too much about designing programs with good abstractions, since the programs we are dealing with are fairly small. For large programs, good abstractions are essential. That's what most of CS201(J) is about.

PS2: Question 8, 9

- Predict how long it will take
- Identify ways to make it faster

Much of this week, and later classes will be focused on how computer scientists predict how long programs will take, and on how to make them faster.

Can we do better?

```
(define (find-best-hand hole-cards community-cards)
   (car (sort higher-hand?
              (possible-hands hole-cards
                              community-cards))))
```

find-best-hand

```
(define (find-best-hand lst)
 (if (null? (cdr lst))
    (car lst)
    (let ((rest-best (find-best-hand (cdr lst))))
      (if (higher-hand? (car lst) rest-best)
          (car lst)
          rest-best))))
```


find-best-hand

```
(define (insertl lst f stopval)
  (if (null? lst)
     stopval
     (f (car lst) (insertl (cdr lst) f stopval))))
```

```
(define (find-best-hand lst)
  (insertl
   (lambda (hand1 hand2)
     (if (higher-hand? hand1 hand2) hand1 hand2))
   (cdr lst) ;; already used the car as stopval
   (car lst)))
```


find-best

```
(define (find-best cf lst)
  (insertl
    (lambda (c1 c2)
        (if (cf c1 c2) c1 c2))
    (cdr lst)
    (car lst)))
```

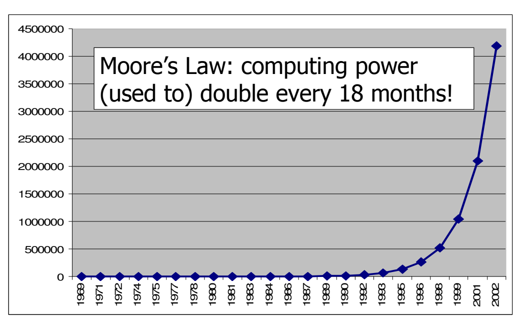


```
(define (find-best cf lst)
  (insertl
     (lambda (c1 c2) (if (cf c1 c2) c1 c2))
     (cdr lst)
     (car lst)))

(define (find-best-hand lst)
    (find-best higher-hand? lst))
```


How much work is find-best?

Why not just time it?



How much work is find-best?

```
(define (find-best cf lst)
  (insertl
   (lambda (c1 c2)
     (if (cf c1 c2) c1 c2))
   lst
   (car lst)))
```

- Work to evaluate (find-best f lst)?
  - Evaluate (insertl (lambda (c1 c2) …) lst)
  - ~~Evaluate lst~~
  - ~~Evaluate (car lst)~~

These don't depend on the length of the list, so we don't care about them.

Work to evaluate insertl

```
(define (insertl f lst stopval)
  (if (null? lst)
      stopval
      (f (car lst) (insertl f (cdr lst) stopval))))
```

- How many times do we evaluate *f* for a list of length n?

n

insertl is $\Theta(n)$ "Theta *n*"

If we double the length of the list, we amount of work required approximately doubles.

(We will see a more formal definition of Θ next class, and a more formal definition of "Amount of work" in November.)

Simple Sorting

- We know how to find-best
- How do we sort?
- Use (find-best lst) to find the best
- Remove it from the list
- Repeat until the list is empty

Simple Sort

```
(define (sort cf lst)
  (if (null? lst) lst
      (let ((best (find-best cf lst)))
        (cons
           best
           (sort cf
                 (delete lst best))))))
```


Sorting Hands

```
(define (sort-hands lst)
  (sort higher-hand? lst))
```


Sorting

```
(define (sort cf lst)
  (if (null? lst) lst
      (let ((most (find-best cf lst)))
        (cons
           most
           (sort cf
                 (delete lst best))))))
```

```
(define (find-best cf lst)
  (insertl
   (lambda (c1 c2)
     (if (cf c1 c2) c1 c2))
   lst
   (car lst)))
```

- How much work is sort?
- We measure work using *orders of growth*: How does work grow with problem size?

Sorting

```
(define (sort cf lst)
  (if (null? lst) lst
      (let ((best (find-best cf lst)))
        (cons best
           (sort cf (delete lst best))))))
```

- What grows?
  - n = the number of elements in lst
- How much work are the pieces?
  find-best is $\Theta(n)$ delete is $\Theta(n)$
- How many times does sort evaluate find-best and delete?

Sorting

```
(define (sort cf lst)
  (if (null? lst) lst
      (let ((best (find-best cf lst)))
        (cons best
           (sort cf (delete lst best))))))
```

- n = the number of elements in lst
- find-most is $\Theta(n)$ delete is $\Theta(n)$
- How many times does sort evaluate find-most and delete? n

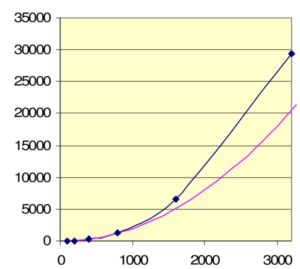
sort is $\Theta(n^2)$

If we double the length of the list, the amount of work *approximately* quadruples.

Timing Sort

```
> (time (sort < (revintsto 100)))
cpu time: 20 real time: 20 gc time: 0
> (time (sort < (revintsto 200)))
cpu time: 80 real time: 80 gc time: 0
> (time (sort < (revintsto 400)))
cpu time: 311 real time: 311 gc time: 0
> (time (sort < (revintsto 800)))
cpu time: 1362 real time: 1362 gc time: 0
> (time (sort < (revintsto 1600)))
cpu time: 6650 real time: 6650 gc time: 0
```


$\Theta(n^2)$



measured times

$= n^2/500$

Is our sort good enough?

Takes over 1 second to sort 1000-length list. How long would it take to sort 1 million items?

1s = time to sort 1000
4s ~ time to sort 2000

$\Theta(n^2)$

1M is 1000 * 1000

Sorting time is n^2
so, sorting 1000 times as many items will take 1000^2 times as long
= 1 million seconds ~ 11 days

Note: there are 800 Million VISA cards in circulation.
It would take 20,000 years to process a VISA transaction at this rate.

PS3: Lindenmayer System Fractals

L-Systems

CommandSequence ::= **(** *CommandList* **)**
CommandList ::= *Command CommandList*
CommandList ::=
Command ::= **F**
Command ::= **R***Angle*
Command ::= **O***CommandSequence*

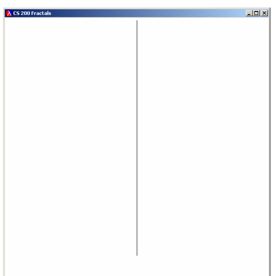
L-System Rewriting

CommandSequence ::= **(** *CommandList* **)**
CommandList ::= *Command CommandList*
CommandList ::=
Command ::= **F**
Command ::= **R***Angle*
Command ::= **O***CommandSequence*

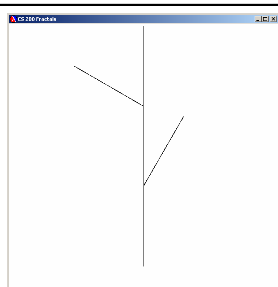
Start: (F)
Rewrite Rule:
F → (F O(R30 F) F O(R-60 F) F)

Work like BNF replacement rules, except replace all instances at once!

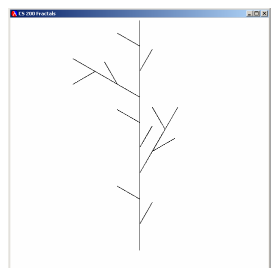
Why is this a better model for biological systems?



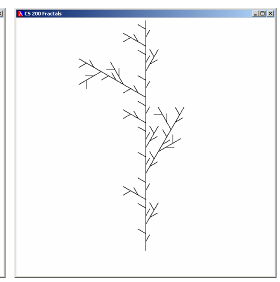
Level 0
Start: (F)
(F)



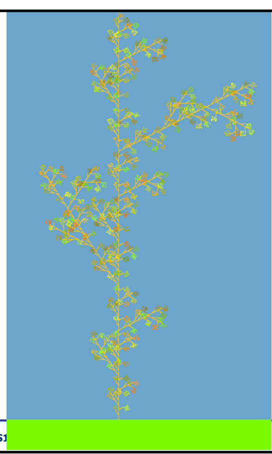
Level 1
F → (F O(R30 F) F O(R-60 F) F)
(F O(R30 F) F O(R-60 F) F)



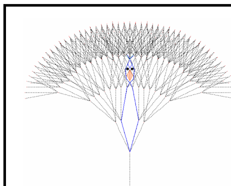
Level 2



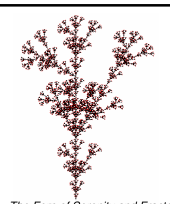
Level 3



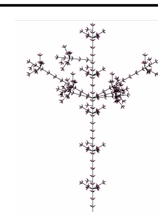
The Great Lambda Tree of Ultimate Knowledge and Infinite Power



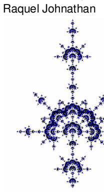
Proud Peacock, Marija Cvijetic and Raquel Johnathan



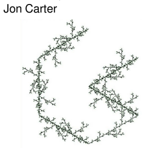
The Fern of Serenity and Fractal Frustration, Nicole Numbers and Jon Carter



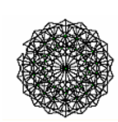
Pink Buds by Kate Morandi and Erika Vogel



Tie Dye by Bill Ingram



CS Logo, Lincoln Hamilton and Dan Nguyen



Spirograph by Leah Nylen

Charge

- Wednesday: faster ways of sorting
- Read Tyson's essay before Friday's class
  - How does it relate to $\theta(n^2)$
  - How does it relate to grade inflation
- PS3 due Monday, Sept 19: lots more code for you to write than PS2