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/cs150

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)))
```

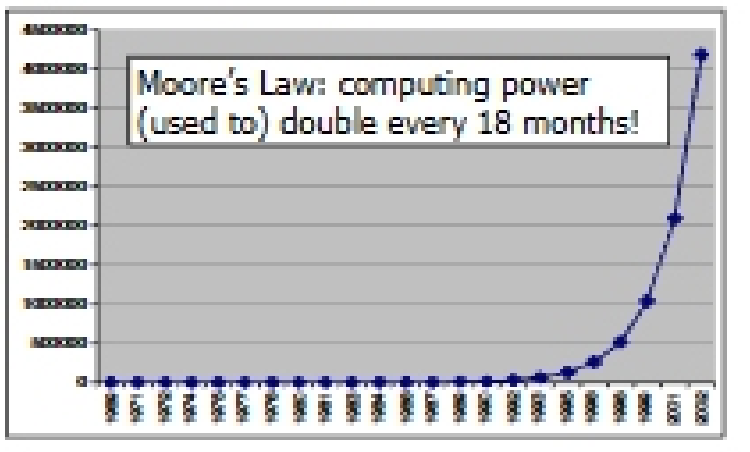


```
(define (find-best cf lst)
  (insertl
     (lambda (c1 c2) (if (cf c1 c2) c1 c2))
     (cdr lst)
     (car lst)))

(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?