Puzzles, Costs and Sneezewort

532. Achillea Ptarmica L.

Sneezewort

One-Slide Summary

- We can solve a game by **recursively enumerating** all legal moves from a position, stopping when we're left with a winning board or no more possible moves.
- The basic recursive computation of Fibonacci can take quite a while. **There are faster ways**.
- We can formally measure and evaluate the cost of a computer program. We abstract away details such as processor speed and instead measure how **the solving time increases as the input increases**.

Outline

- That Cursed Pegboard!
  - Jumps, legal moves, winning, ...
- Sneezewort and Fibonacci
- **Cost** of computing Fibonacci
- **Cost** of sorting

Example Board: "Grey"

```
(define grey-rows 5)
(define grey-holes (list (make-position 1 1) (make-
   position 2 1) (make-position 2 2) (make-position 3 2))
(define grey-board (make-board grey-rows grey-holes))
```

```
          1,1
        2,1  2,2
      3,1  3,2  3,3
    4,1  4,2  4,3  4,4
  5,1  5,2  5,3  5,4  5,5
```

Solving the Peg Board Game

- Try all possible moves on the board
- Try all possible moves from the positions you get after each possible first move
- Try all possible moves from the positions you get after trying each possible move from the positions you get after each possible first move
- ...

Filter Remove

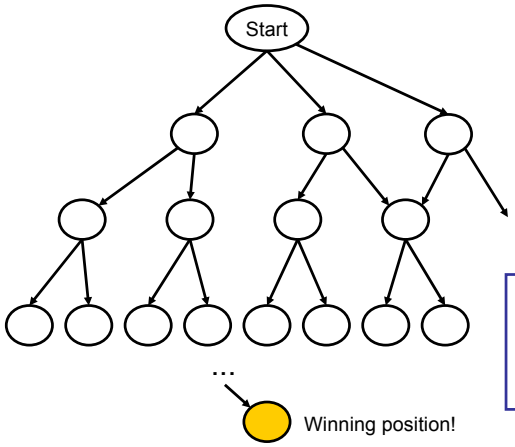
```
(define (filter pred lst)
  (if (null? lst)
      null
      (if (pred (car lst)) ; pred is true, keep it
          (cons (car lst) (filter proc (cdr lst)))
          (filter pred (cdr lst))))) ; pred is false, drop it
```

```
(define (remove-hole lst posn)
  (filter (lambda (pos)
            (not (same-position pos posn)))
          lst))
```

Jumps

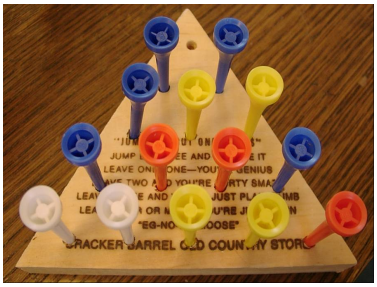
```
;;; move creates a list of three positions: a start (the posn
;;; that the jumping peg starts from), a jump (the posn
;;; that is being jumped over), and end (the posn that
;;; the peg will end up in)

(define (make-move start jump end) (list start jump end))
(define (get-start move) (first move))
(define (get-jump move) (second move))
(define (get-end move) (third move))

;;; execute-move evaluates to the board after making move
;;; move on board.
(define (execute-move board move)
  (add-peg (remove-peg (remove-peg board (get-start move))
                       (get-jump move))
           (get-end move)))
```

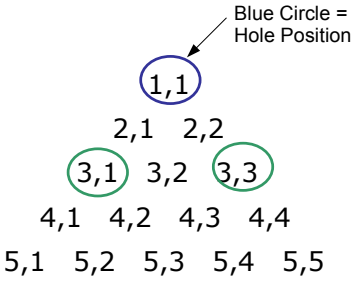

Finding a Winning Strategy



How is winning 2-person games (e.g., chess, poker) different?

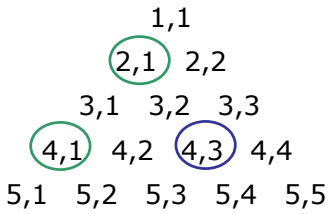
Pegboard Puzzle





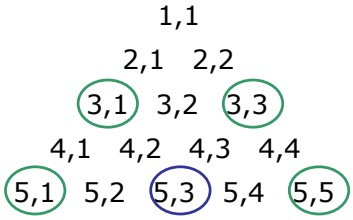
How do we find all possible jumps that land in a given target hole?

Pegboard Puzzle



How do we find all possible jumps that land in a given target hole?

Pegboard Puzzle



How do we find all possible jumps that land in a given target hole?

All Moves into Target

```
;;; generate-moves evaluates to all possible moves that move a peg into
;;; the position target, even if they are not contained on the board.
(define (generate-moves target)
   (map (lambda (hops)
          (let ((hop1 (car hops)) (hop2 (cdr hops)))
            (make-move
              (make-position (+ (get-row target) (car hop1))
                             (+ (get-col target) (cdr hop1)))
              (make-position (+ (get-row target) (car hop2))
                             (+ (get-col target) (cdr hop2)))
              target)))
        (list (cons (cons 2 0) (cons 1 0))    ;; right of target, hopping left
              (cons (cons -2 0) (cons -1 0))  ;; left of target, hopping right
              (cons (cons 0 2) (cons 0 1))    ;; below, hopping up
              (cons (cons 0 -2) (cons 0 -1))  ;; above, hopping down
              (cons (cons 2 2) (cons 1 1))    ;; above right, hopping down-left
              (cons (cons -2 2) (cons -1 1))  ;; above left, hopping down-right
              (cons (cons 2 -2) (cons 1 -1))  ;; below right, hopping up-left
              (cons (cons -2 -2) (cons -1 -1)))))  ;; below left, hopping up-right
```

All Possible Moves

```
(define (all-possible-moves board)
  (append-all
    (map generate-moves (board-holes holes))))

(define (append-all lst)
   (if (null? lst) null
      (append (car lst) (append-all (cdr lst)))))
```

But…only legal if: start and end are positions on the board containing pegs!

Note: could use (apply append ...) instead of append-all.

Legal Move

```
(define (legal-move? move)
  ;; A move is valid if:
  ;;    o the start and end positions are on the board
  ;;    o there is a peg at the start position
  ;;    o there is a peg at the jump position
  ;;    o there is not a peg at the end position
  (and (on-board? board (get-start move))
       (on-board? board (get-end move))
       (peg? board (get-start move))
       (peg? board (get-jump move))
       (not (peg? board (get-end move)))))
```


All Legal Moves

```
(define (all-possible-moves board)
  (append-all
     (map generate-moves
          (board-holes holes))))
```

```
(define (legal-move? move)
  ;; A move is valid if:
  ;;    o the start and end positions are on the board
  ;;    o there is a peg at the start position
  ;;    o there is a peg at the jump position
  ;;    o there is not a peg at the end position
  (and (on-board? board (get-start move))
       (on-board? board (get-end move))
       (peg? board (get-start move))
       (peg? board (get-jump move))
       (not (peg? board (get-end move)))))
```

```
(define (legal-moves board)
  ???
```

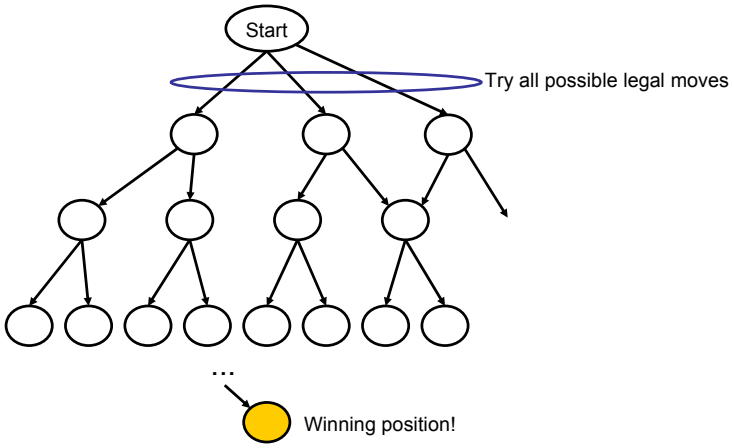

All Legal Moves

```
(define (all-possible-moves board)
  (append-all
     (map generate-moves
          (board-holes holes))))
```

```
(define (legal-move? move)
  ;; A move is valid if:
  ;;    o the start and end positions are on the board
  ;;    o there is a peg at the start position
  ;;    o there is a peg at the jump position
  ;;    o there is not a peg at the end position
  (and (on-board? board (get-start move))
       (on-board? board (get-end move))
       (peg? board (get-start move))
       (peg? board (get-jump move))
       (not (peg? board (get-end move)))))
```

```
(define (legal-moves board)
  (filter legal-move? (all-possible-moves board)))
```


Becoming a "Genius"!



Winning Position

How do we tell if a board is in a winning position?

is-winning-position?

```
(define (board-squares board)
  (count-squares (board-rows board)))

(define (count-squares nrows)
  (if (= nrows 1) 1
      (+ nrows (count-squares (- nrows 1)))))

(define (is-winning-position? board)
  (= (length (board-holes board))
     (- (board-squares board) 1)))
```


Solve Pegboard

```
(define (solve-pegboard board)
  (find-first-winner board (legal-moves board)))

(define (find-first-winner board legal-moves)
  ;; returns a list of moves that will solve the given board
  (if (null? legal-moves)
      (if (is-winning-position? board)
          null ;; Found winning game, no moves needed
          #f)  ;; A losing position, no more moves
      (let ((result (solve-pegboard
                      (execute-move board (car legal-moves)))))
        (if result ;; winner (not #f)
            (cons (car legal-moves) result) ;; this move wins!
            (find-first-winner board (cdr legal-moves))))))  ;; try rest
```


All Cracker Barrel Games

(starting with peg 2 1 missing)

Pegs Left	Number of Ways	Fraction of Games	Cracker Barrel IQ Rating
1	1550	0.01	*"You're Genius"*
2	20686	0.15	*"You're Purty Smart"*
3	62736	0.46	*"Just Plain Dumb"*
4	46728	0.33	
5	5688	0.04	*"Just Plain Eg-no-ra-moose"*
6	374	0.0027	
7	82	0.00058	
10	2	0.00001	

All Cracker Barrel Games

(starting with peg 2 1 missing)

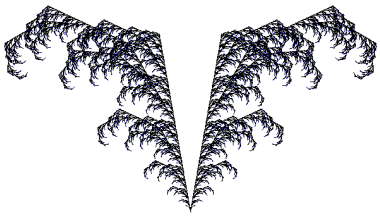
Pegs Left	Number of Ways	Fraction of Games	Cracker Barrel IQ Rating
1	1550	0.01	*"You're Genius"*
2	20686	0.15	*"You're Purty Smart"*
3	62736	0.46	*"Just Plain Dumb"*
4	46728	0.33	
5	5688	0.04	*"Just Plain Eg-no-ra-moose"*
6	374	0.0027	
7	82	0.00058	
10	2	0.00001	Leaving 10 pegs requires much more brilliance than leaving 1!?

Liberal Arts Trivia: History

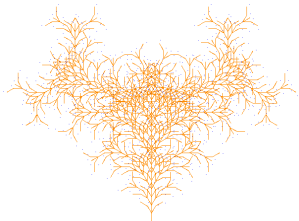
- This 20th-century American inventor is credited with the phonograph, the carbon telephone transmitter, the practical electric light, and the phrase "Genius is one percent inspiration, ninety-nine percent perspiration." He fought against Nikola Tesla's alternating current in the so-called War of the Currents.

Liberal Arts Trivia: Physics

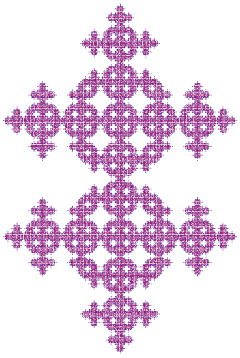
- Count Alessandro Antonio Anastasio Volta was a 19th-century Italian physicist. Volta studied what we now call capacitance, developing separate means to study both electrical potential *V* and charge *Q*, and discovering that for a given object they are proportional. His experiments in "animal electricity", in which two different metals were connected in series with frog's legs, eventually led to his most famous discovery. What was it?



"V" shrubbery
by Andrew Jesien, Becky Elstad



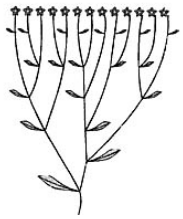
Robot Cav Man
by Jamie Jeon & Walter Borges



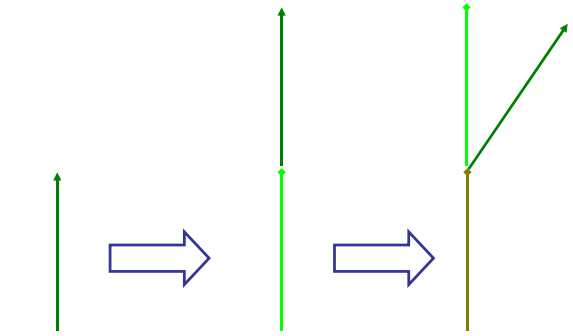
After the Incident
by Ben Morrison and Liz Peterson

Sneezewort

- Achillea ptarmica is real.
- It is "moste efficacious in the inflaming of the braine, and [is] therefore much used in Confusing and Befuddlement Draughts, where the wizard is desirous of producing hot-headedness and recklessness."
  - Order of the Phoenix, p.18
- Sneezewort's pattern of development displays the Fibonacci sequence.



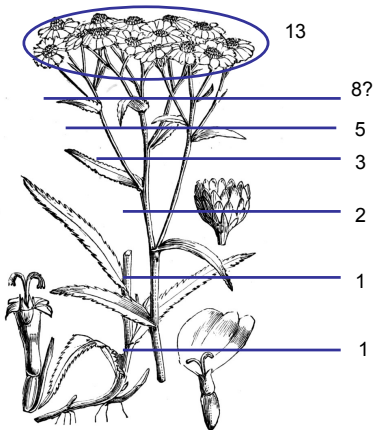
Sneezewort Growth

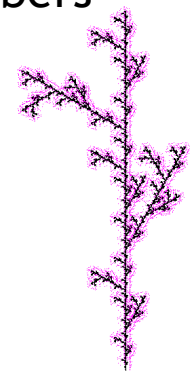


First Time Unit Second Time Unit Offshoot

Could we model Sneezewort with PS3 code?

Sneezewort Numbers

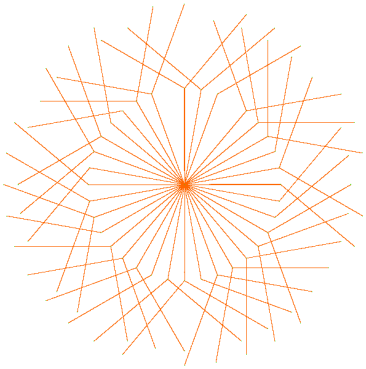




pink
by Jessica Geist, Ellen Clarke

Fibo Results

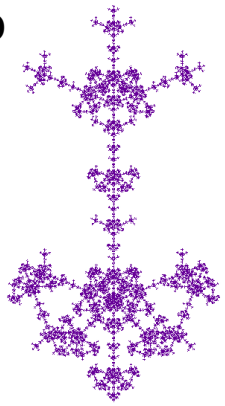
```
> (fibo 2)
1
> (fibo 3)
2
> (fibo 4)
3
> (fibo 10)
55
> (fibo 60)
```
Still working…



At least we finished.
by Dmitriy Semenov and Sara Alspaugh

Tracing Fibo

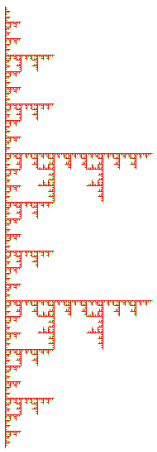
```
> (require-library "trace.ss")
> (trace fibo)
(fibo)
> (fibo 3)
|(fibo 3)
|       (fibo 2)
|       1
|       (fibo 1)
|       1
|2
2
```



Purple Arrow
by Rachel Lathbury and Andrea Yoon

```
> (fibo 5)
|(fibo 5)
| (fibo 4)
| |(fibo 3)
| | (fibo 2)
| | 1
| | (fibo 1)
| | 1
| |2
| |(fibo 2)
| |1
| 3
| (fibo 3)
| |(fibo 2)
| |1
| |(fibo 1)
| |1
| 2
|5
5
```

A right-wing Christmas - awwwwww.....
by Andrew Baker & Emily Lam



To calculate (fibo 5) we calculated:

(fibo 4)	1 time	
(fibo 3)	2 times	
(fibo 2)	3 times	5 times total
(fibo 1)	2 times	

= 8 calls to fibo = (fibo 6)

How many calls to calculate (fibo 60)?

fast-fibo

```
(define (fast-fibo n)
  (define (fib-helper a b left)
    (if (<= left 0)
        b
        (fib-helper b (+ a b) (- left 1))))
  (fib-helper 1 1 (- n 2)))
```


Fast-Fibo Results

```
> (fast-fibo 10)
55
> (time (fast-fibo 61))
cpu time: 0 real time: 0 gc time: 0
2504730781961
```

So original fibo would take at least 2.5 Trillion applications
2.5 GHz computer does 2.5 *Billion* simple operations per second, so 2.5 Trillion applications operations take ~1000 seconds.
Each application of fibo involves hundreds of simple operations…

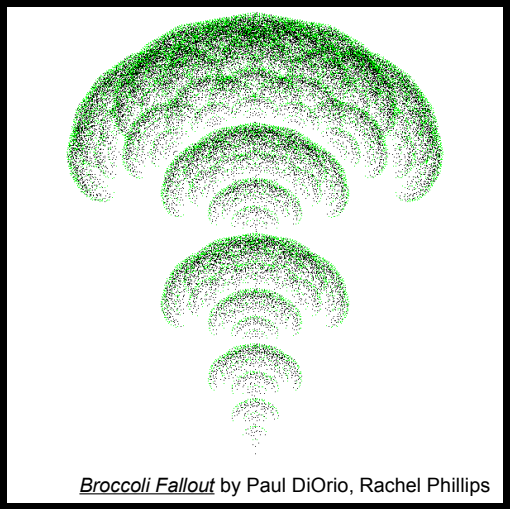

```
  ;;; The Earth's mass is 6.0 x 10^24 kg
> (define mass-of-earth (* 6 (expt 10 24)))
  ;;; A typical rabbit's mass is 2.5 kilograms
> (define mass-of-rabbit 2.5)
> (/ (* mass-of-rabbit (fast-fibo 60)) mass-of-earth)
6.450036483e-013
> (/ (* mass-of-rabbit (fast-fibo 120)) mass-of-earth)
2.2326496895795693
```

According to Bonacci's model, after less than 10 years, rabbits would out-weigh the Earth!


```
  ;;; The Earth's mass is 6.0 x 10^24 kg
> (define mass-of-earth (* 6 (expt 10 24)))
  ;;; A typical rabbit's mass is 2.5 kilograms
> (define mass-of-rabbit 2.5)
> (/ (* mass-of-rabbit (fast-fibo 60)) mass-of-earth)
6.450036483e-013
> (/ (* mass-of-rabbit (fast-fibo 120)) mass-of-earth)
2.2326496895795693
```

According to Bonacci's model, after less than 10 years, rabbits would out-weigh the Earth!

Beware the Bunnies!! Beware the Sneezewort!!

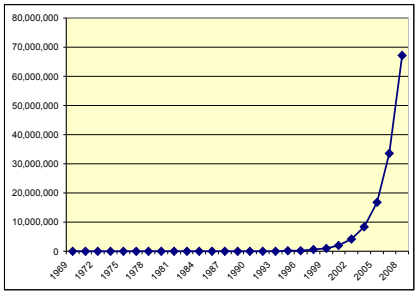


Broccoli Fallout by Paul DiOrio, Rachel Phillips

Evaluation Cost

Actual running times vary according to:

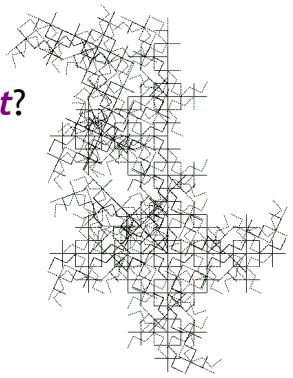
- How fast a processor you have
- How much memory you have
- Where data is located in memory
- How hot it is
- What else is running
- etc...



Moore's "Law" – computing power doubles every 18 months

Measuring Cost

- How does the cost scale with the ***size of the input***?
- If the input size increases by one, how much longer will it take?
- If the input size *doubles*, how much longer will it take?



Untitled
Nokomis McCaskill
Chris Hooe

Cost of Fibonacci Procedures

```
(define (fibo n)
        (if (or (= n 1) (= n 2))
    1 ;;; base case
    (+ (fibo (- n 1))
       (fibo (- n 2)))))
```

```
(define (fast-fibo n)
  (define (fib-helper a b left)
    (if (= left 0)
        b
        (fib-helper b (+ a b) (- left 1))))
  (fib-helper 1 1 (- n 2)))
```

Input	fibo	fast-fibo
m	q	mk
$m+1$		$(m+1)k$
$m+2$	at least q^2	$(m+2)k$

Cost of Fibonacci Procedures

```
(define (fibo n)
        (if (or (= n 1) (= n 2))
    1 ;;; base case
    (+ (fibo (- n 1))
       (fibo (- n 2)))))
```

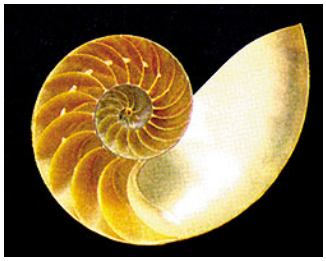
```
(define (fast-fibo n)
  (define (fib-helper a b left)
    (if (= left 0)
        b
        (fib-helper b (+ a b) (- left 1))))
  (fib-helper 1 1 (- n 2)))
```

Input	fibo	fast-fibo
m	q	mk
$m+1$	$q*\Phi$	$(m+1)k$
$m+2$	at least q^2	$(m+2)k$

Φ = (/ (+ 1 (sqrt 5)) 2) = "The Golden Ratio" ~ 1.618033988749895...
~ (/ (fast-fibo 61) (fast-fibo 60)) = 1.618033988749895

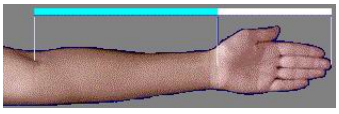
The Golden Ratio

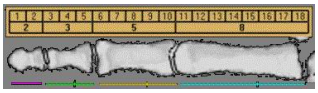
Parthenon



Nautilus Shell

More Golden Ratios





http://www.fenkefeng.org/essaysm18004.html
by Oleksiy Stakhov

Liberal Arts Trivia: Medicine

- Nicolae Paulescu was a 20th century physiologist and professor of medicine. He is considered the true discoverer of hormone that causes most of the body's cells to take up glucose from the blood. His first experiments involved an aqueous pancreatic extract which, when injected into a diabetic dog, proved to have a normalizing effect on blood sugar levels. Name the hormone.

Liberal Arts Trivia: Sailing

- Name the collection of apparatus through which the force of the wind is transferred to the ship in order to propel it forward – this includes the masts, yardarms, sails, spars and cordage.

PS2 Question

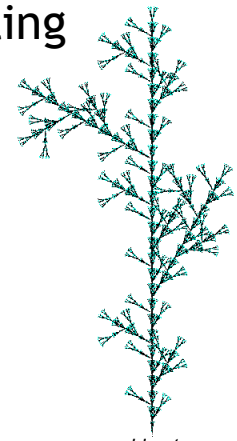
```
(define (find-best-hand hands)
  (car (sort hands higher-hand?)))
```

```
(define (find-best lst cf)
  (if (= 1 (length lst)) (car lst)
      (pick-better cf (car lst) (find-best (cdr lst) cf))))
(define (pick-better cf num1 num2)
  (if (cf num1 num2) num1 num2))
(define (find-best-hand hands)
  (find-best hands higher-hand?))
```

Which is better and by how much?

Simple Sorting

- Can we use find-best to implement sort?
  - Yes!
- Use (find-best lst) to find the best
- Remove it from the list
  - Adding it to the answer
- Repeat until the list is empty



crazy blue tree
by Victor Malaret, Folami Williams

Simple Sort

```
;; cf = comparison function
(define (sort lst cf)  ;; simple sort
  (if (null? lst) lst
      (let ((best (find-best lst cf)))
        (cons
          best
          (sort (delete lst best) cf)))))
;; delete lst x = (filter ... (not (eq? x ...
```


Sorting Hands

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

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

Sorting

```
(define (sort lst cf)
  (if (null? lst) lst
      (let ((best (find-best lst cf)))
        (cons best (sort (delete lst best) cf)))))
(define (find-best lst cf)
  (if (= 1 (length lst)) (car lst)
      (pick-better cf (car lst) (find-best (cdr lst) cf))))
(define (pick-better cf num1 num2)
  (if (cf num1 num2) num1 num2))
```

How much work is sort?

Sorting Cost

- What grows?
  - n = the number of elements in lst
- How much work are the pieces?
  - find-best:
  - delete:

Sorting Cost

- What grows?
  - n = the number of elements in lst
- How much work are the pieces?
  - find-best: work scales as n (increases by one)
  - delete: work scales as n (increases by one)
- How many times does sort evaluate find-best and delete?

Sorting Cost

- What grows?
  - n = the number of elements in lst
- How much work are the pieces?
  - find-best: work scales as n (increases by one)
  - delete: work scales as n (increases by one)
- How many times does sort evaluate find-best and delete? n
- Total cost: scales as

Sorting Cost

- What grows?
  - n = the number of elements in lst
- How much work are the pieces?
  - find-best: work scales as n (increases by one)
  - delete: work scales as n (increases by one)
- How many times does sort evaluate find-best and delete? n
- Total cost: scales as n^2

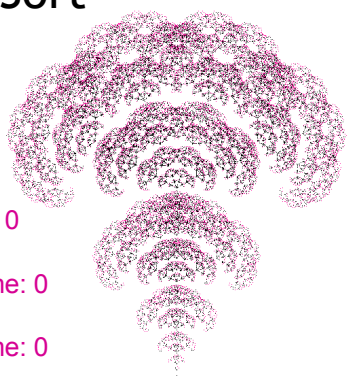
Sorting Cost

```
(define (sort lst cf)
  (if (null? lst) lst
      (let ((best (find-best lst cf)))
        (cons best (sort (delete lst best) cf)))))
(define (find-best lst cf)
  (if (= 1 (length lst)) (car lst)
      (pick-better cf (car lst) (find-best (cdr lst) cf))))
```

If we double the length of the list, the amount of work *approximately* quadruples: there are twice as many applications of find-best, and each one takes twice as long

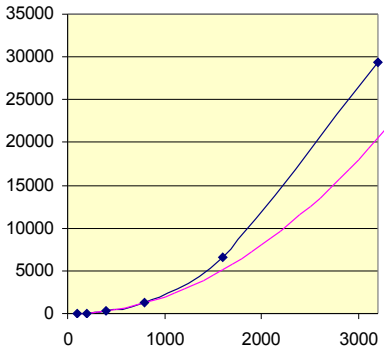
Timing Sort

```
> (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
```



Cherry Blossom
by Ji Hyun Lee, Wei Wang

Timing Sort

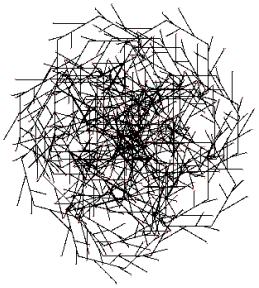


measured times

= $n^2/500$

Homework

- Read Course Book Chapter 7 before Wednesday
  - Has a formal notation for this kind of analysis!
- Problem Set 3 due Wednesday



The Mask
by Zachary Pruckowski, Kristen Henderson