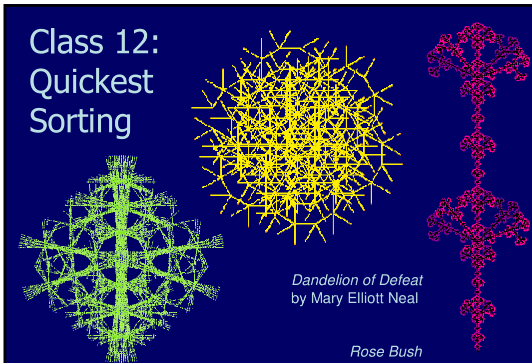
Class 12: Quickest Sorting



Dandelion of Defeat
by Mary Elliott Neal

by Kristina Caudle

Rose Bush
by Jacintha Henry & Rachel Kay

CS150: Computer Science
University of Virginia
Computer Science

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

Insert Sort

```
(define (insertsort cf lst)
  (if (null? lst)
      null
      (insertel cf
        (car lst)
        (insertsort cf
                (cdr lst)))))
```

```
(define (insertel cf el lst)
  (if (null? lst)
      (list el)
      (if (cf el (car lst))
           (cons el lst)
           (cons (car lst)
                (insertel cf el
                      (cdr lst))))))
```

insertsort is $\Theta(n^2)$

Divide and Conquer

- Both simplesort and insertsort divide the problem of sorting a list of length n into:
  - Sorting a list of length n-1
  - Doing the right thing with one element
- Hence, there are always n steps
  - And since each step is $\theta(n)$, they are $\theta(n^2)$
- To sort more efficiently, we need to divide the problem more evenly each step

Insert Halves

```
(define (insertsorth cf lst)
  (if (null? lst) null
    (insertelh cf
       (car lst)
       (insertsorth cf (cdr lst)))))
```

Same as insertsort except uses insertelh

```
(define (insertelh cf el lst)
  (if (null? lst)
    (list el)
    (let ((fh (first-half lst))
          (sh (second-half lst)))
      (if (cf el (car fh))
          (append (cons el fh) sh)
          (if (null? sh)
              (append fh (list el))
              (if (cf el (car sh))
                  (append (insertelh cf el fh) sh)
                  (append fh (insertelh cf el sh))))))))
```

```
(if (= (length lst) 1)
   (if (cf el (car lst))
       (cons el lst)
       (list (car lst) el))
   (let ((fh …
```


append can't be $O(1)$

- Needs to make a new list of length $n + m$ where n is the length of the first list, and m is the length of the second list
- Making a list of length k requires k conses, so append must be $\Theta(\mathrm{n} + \mathrm{m})$

Experimental Confirmation

```
> (testgrowthappend)
n = 10000, m = 1, time = 0
n = 20000, m = 1, time = 0
n = 40000, m = 1, time = 0
n = 80000, m = 1, time = 10
n = 160000, m = 1, time = 10
n = 320000, m = 1, time = 40
n = 640000, m = 1, time = 70
n = 1280000, m = 1, time = 150
n = 2560000, m = 1, time = 290
(1.0 4.0 1.75 2.14286 1.9333)
```

```
n = 10000, m = 10000, time = 0
n = 20000, m = 20000, time = 0
n = 40000, m = 40000, time = 0
n = 80000, m = 80000, time = 10
n = 160000, m = 160000, time = 50
n = 320000, m = 320000, time = 40
n = 640000, m = 640000, time = 70
n = 1280000, m = 1280000, time = 150
n = 2560000, m = 2560000, time = 270
(5.0 0.8 1.75 2.142857142857143 1.8)
```

Complexity of Insert Half Sort

```
(define (insertelh cf el lst)
  (if (null? lst) (list el)
    (if (= (length lst) 1)
      (if (cf el (car lst)) (cons el lst)
        (list (car lst) el))
      (let ((fh (first-half lst))
            (sh (second-half lst)))
        (if (cf el (car sh))
            (append (insertelh cf el fh) sh)
            (append fh (insertelh cf el sh)))))))
```

```
(define (insertsorth cf lst)
  (if (null? lst) null
    (insertelh cf (car lst)
      (insertsorth cf
        (cdr lst)))))
```

n applications of insertelh

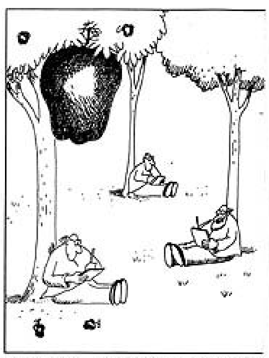
$\log n$ applications of insertelh

Each one is $\Theta(n)$ work since append is $\Theta(n)$

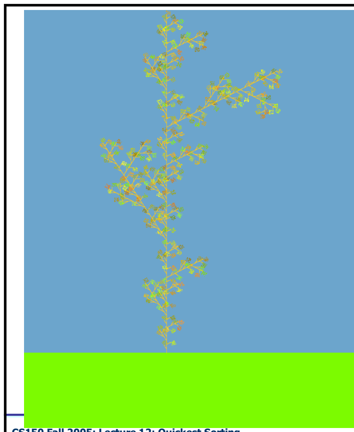
Complexity is: $\Theta(n^2 \log n)$

Making it faster

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

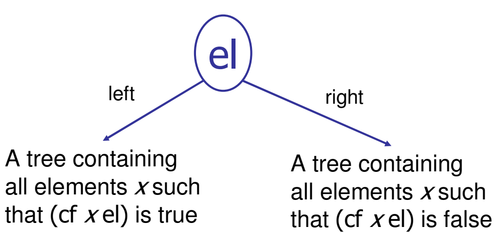


"Nothing yet. ...How about you, Newton?"

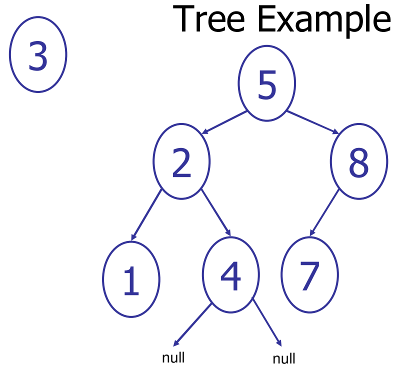


The Great Lambda Tree of Ultimate Knowledge and Infinite Power

Sorted Binary Trees



Tree Example



cf: <

Representing Trees

```
(define (make-tree left el right)
  (list left el right))
```

left and right are trees (**null** is a tree)

```
(define (get-left tree)
  (first tree))
```

tree must be a non-null tree

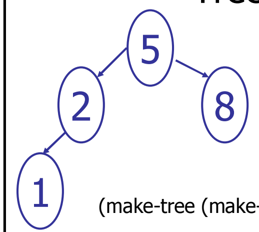
```
(define (get-element tree)
  (second tree))
```

tree must be a non-null tree

```
(define (get-right tree)
  (third tree))
```

tree must be a non-null tree

Trees as Lists




```
(define (make-tree left el right)
  (list left el right))

(define (get-left tree)
  (first tree))
(define (get-element tree)
  (second tree))
(define (get-right tree)
  (third tree))
```

```
(make-tree (make-tree (make-tree null 1 null)
                      2
                      null)
           5
           (make-tree null 8 null))
```


insertel-tree

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

If the tree is null, make a new tree with el as its element and no left or right trees.

Otherwise, decide if el should be in the left or right subtree. insert it into that subtree, but leave the other subtree unchanged.

How much work is insertel-tree?

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

Each time we call insertel-tree, the size of the tree. So, doubling the size of the tree only increases the number of calls by 1!

insertel-tree is $\theta(\log_2 n)$

$\log_2 a = b$ means $2^b = a$

insertsort-tree

```
(define (insertsort cf lst)
  (if (null? lst) null
      (insertel cf (car lst)
                (insertsort cf (cdr lst)))))

(define (insertsort-worker cf lst)
    (if (null? lst) null
        (insertel-tree cf (car lst)
                   (insertsort-worker cf (cdr lst)))))
```

No change…but insertsort-worker evaluates to a tree not a list!

((((() 1 ()) 2 ()) 5 (() 8 ()))

extract-elements

We need to make a list of all the tree elements, from left to right.

```
(define (extract-elements tree)
  (if (null? tree) null
      (append (extract-elements (get-left tree))
              (cons (get-element tree)
                 (extract-elements (get-right tree))))))
```

Complexity of insertsort-tree

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

n = number of elements in tree

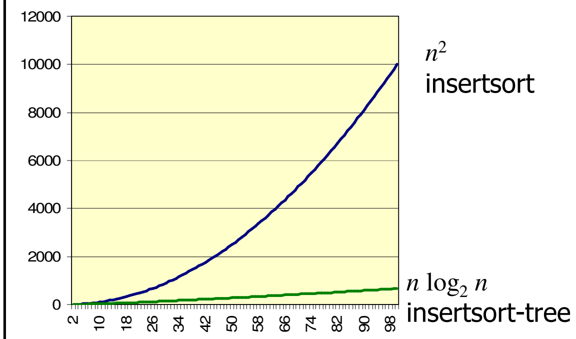
$\theta(\log n)$

```
(define (insertsort-tree cf lst)
  (define (insertsort-worker cf lst)
    (if (null? lst) null
        (insertel-tree cf (car lst)
                (insertsort-worker cf (cdr lst)))))
  (extract-elements (insertsort-worker cf lst)))
```

n = number of elements in lst

$\theta(n \log n)$

Growth of time to sort random list



Comparing sorts

```
> (testgrowth simplesort)
n = 250, time = 110
n = 500, time = 371
n = 1000, time = 2363
n = 2000, time = 8162
n = 4000, time = 31757
(3.37 6.37 3.45 3.89)
> (testgrowth insertsort)
n = 250, time = 40
n = 500, time = 180
n = 1000, time = 571
n = 2000, time = 2644
n = 4000, time = 11537
(4.5 3.17 4.63 4.36)
```

```
> (testgrowth insertsorth)
n = 250, time = 251
n = 500, time = 1262
n = 1000, time = 4025
n = 2000, time = 16454
n = 4000, time = 66137
(5.03 3.19 4.09 4.02)
> (testgrowth insertsort-tree)
n = 250, time = 30
n = 500, time = 250
n = 1000, time = 150
n = 2000, time = 301
n = 4000, time = 1001
(8.3 0.6 2.0 3.3)
```


Can we do better?

- Making all those trees is a lot of work
- Can we divide the problem in two halves, without making trees?

Continues in Lecture 13

Charge

- PS4 due Monday (1 week only!)
- Next class:
  - Finishing quicksort
  - Understanding the universe is $\Theta(n^3)$ are there any harder problems?