Class 12: Quickest Sorting

Insert Sort

\[
\text{insertsort (define (insertsort cf lst))}
\]

\[
\text{(define (insertel cf el lst))}
\]

\[
\text{(if (null? lst) null (insertel cf (car lst) (insertsort cf (cdr lst)))))
\]

\[
\text{insertsort is } \Theta(n^2)
\]

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

Experimental Confirmation

\[
\text{> (testgrowthappend)}
\]

\[
n = 10000, m = 1, time = 0 \quad n = 10000, m = 10000, time = 0
\]

\[
n = 20000, m = 1, time = 0 \quad n = 20000, m = 20000, time = 0
\]

\[
n = 40000, m = 1, time = 0 \quad n = 40000, m = 40000, time = 0
\]

\[
n = 80000, m = 1, time = 10 \quad n = 80000, m = 80000, time = 10
\]

\[
n = 160000, m = 1, time = 10 \quad n = 160000, m = 160000, time = 50
\]

\[
n = 320000, m = 1, time = 40 \quad n = 320000, m = 320000, time = 40
\]

\[
n = 640000, m = 1, time = 70 \quad n = 640000, m = 640000, time = 70
\]

\[
n = 1280000, m = 1, time = 150 \quad n = 1280000, m = 1280000, time = 150
\]

\[
n = 2560000, m = 1, time = 290 \quad n = 2560000, m = 2560000, time = 270
\]

\[
(1.0 4.0 1.79 2.14286 1.53333)
\]

\[
(5.0 0.8 1.75 2.142857143 1.8)
\]
Complexity of Insert Half Sort

\[
\text{define (insertelh cf el lst)} \\
\text{(if (null? lst) (list el))} \\
\text{(if (= (length lst) 1) (list (car lst) el))} \\
\text{(define (insertsorth cf lst))} \\
\text{(if (null? lst) null)} \\
\text{(insertelh cf (car lst))} \\
\text{(let ((fh (first-half lst)) (sh (second-half lst))) (append (insertelh cf el fh) sh) (append fh (insertelh cf el sh)))))}
\]

\[
n \text{ applications of insertelh} \\
\text{log } n \text{ applications of insertelh}
\]

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 \( \Theta(n) \)

Sorted Binary Trees

- A tree containing all elements \( x \) such that \((cf \ x \ el)\) is true
- A tree containing all elements \( x \) such that \((cf \ x \ el)\) is false

Tree Example

\[
\text{5} \\
\text{2} \quad \text{8} \\
\text{1} \quad \text{4} \quad \text{7} \\
\text{null} \quad \text{null}
\]

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))  
  (first tree))  
  (second 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))))))

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))))))

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())

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))))))

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 \( 2^b = a \)

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))))))

extract-elements

(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

\[ \Theta(n \log n) \]

\( n \) = number of elements in tree

\( \Theta(\log n) \)

\( n \) = number of elements in list

Comparing sorts

**\( \text{testgrowth simplesort} \)**

- \( n = 250, \text{time} = 110 \)
- \( n = 500, \text{time} = 371 \)
- \( n = 1000, \text{time} = 2363 \)
- \( n = 2000, \text{time} = 8162 \)
- \( n = 4000, \text{time} = 31757 \)

**\( \text{testgrowth insertsort} \)**

- \( n = 250, \text{time} = 40 \)
- \( n = 500, \text{time} = 180 \)
- \( n = 1000, \text{time} = 571 \)
- \( n = 2000, \text{time} = 2644 \)
- \( n = 4000, \text{time} = 11537 \)

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 \( \Theta(n^2) \) are there any harder problems?