Binary Twees

CS21b: Structure and Interpretation of Computer Programs
Spring 2004
Binary trees with data at the leaves

(define (leaf x) x)  ;Value: leaf

(define (tree L R) (list L R))  ;Value: tree

(define (leafval L) L)  ;Value: leafval

(define left car)  ;Value: left

(define right cadr)  ;Value: right

(define (leaf? T) (not (pair? T)))  ;Value: leaf?

(define nonleaf? pair?)  ;Value: nonleaf?

(leaf 3)  ;Value: 3

(tree (leaf 3) (tree (leaf 10) (leaf 40)))  ;Value: (3 (10 40))

(define (leaves T)
  (if (leaf? T)
      (list (leafval T))
      (append (leaves (left T)) (leaves (right T))))))

;Value: leaves

(leaves (tree (tree (leaf 21) (leaf 30)) (tree (leaf 14) (leaf 22))))
;Value: (21 30 14 22)
Binary trees with data also at internal nodes

(define (node x L R) (list x L R))
;Value: node

(define null '())
;Value: null

(define (leaf x) (node x null null))
;Value: leaf

(define root car)
;Value: root

(define left cadr)
;Value: left

(define right caddr)
;Value: right

(define (leaf? T) (and (null? (left T)) (null? (right T))))
;Value: leaf?

(define (not-leaf? T) (not (leaf? T)))
;Value: not-leaf?

(leaf 3)
;Value: (3 () ())

(node 3 (node 1 (leaf 4) (leaf 5)) (node 2 (leaf 3) (leaf 9)))
;Value: (3 (1 (4 () ()) (5 () ()))) (2 (3 () ()) (9 () ())))
Treesort: sorting based on “inorder” trees

(define (insert x T)
  (if (null? T)
    (leaf x)
    (if (< x (root T))
      (node (root T) (insert x (left T)) (right T))
      (node (root T) (left T) (insert x (right T))))))

;Value: insert

(leaf 3)
;Value: (3 () ())

(insert 4 (insert 5 (insert 1 (leaf 3))))
;Value: (3 (1 () ()) (5 (4 () ()) ()))

(define (leaves T)
  (if (null? T)
    T
    (append (leaves (left T)) (cons (root T) (leaves (right T))))))

;Value: leaves

(leaves (insert 4 (insert 5 (insert 1 (leaf 3))))))
;Value: (1 3 4 5)

(define (insert* L)
  (if (null? L)
    null
    (if (null? (cdr L))
      (leaf (car L))
      (insert (car L) (insert* (cdr L))))))

;Value: insert*

(define (treesort L) (leaves (insert* L)))
;Value: treesort

(treesort (list 748 197 917 819 930 653 682 792 523 284 420 851 450 372 308 91 342 501 725 744))
;Value: (91 197 284 308 342 372 420 450 501 523 653 682 725 744 748 792 819 851 917 930)
Heapsort: build a heap, then select root and reheap...

```
(define (order a b c)              ; returns (x y z) where x <= y and x <= z
  (if (and (<= a b) (<= a c))
    (list a b c)
    (if (and (<= b a) (<= b c))
      (list b a c)
      (list c a b))))
;Value: order

(define (build-heap L)
  (if (null? L)
    null
    (if (null? (cdr L))
      (leaf (car L))
      (let ((first (car L))
        (L (heapify (build-heap (odds (cdr L)))))
        (R (heapify (build-heap (evens (cdr L)))))
        (heapify (node first L R))))))
;Value: build-heap

(define (heapify T)
  (if (or (null? T) (leaf? T))
    T
    (if (null? (left T))
      (node (min (root T) (root (right T)))
        (heapify (node (max (root T) (root (right T)))
          (left (right T)))
        (right (right T))))
      (left T))
    (if (null? (right T))
      (heapify (node (root T) (right T) (left T)))
      (let ((three (order (root T) (root (left T)) (root (right T))))
        (node (car three)
          (heapify (node (cadr three) (left (left T)) (right (left T))))
          (heapify (node (caddr three) (left (right T)) (right (right T)))))))
    )))
;Value: heapify

(build-heap '(3 1 4 1 2 6 2 7 1 8 2 8 10 20 40 3 99))
;Value: (1 (1 (3 (3 (7 () ()) ()) (8 () ())) (6 (8 () ())) (20 () ())) (20 () ())) (2 (4 (99 () ()))) (10 () ())) (2 (2 () ())) (40 () ())))
(define (pop T)
  (if (null? T)
      (error "You cannot pop a null tree")
    (if (leaf? T)
        (cons (root T) null)
      (if (null? (left T))
          (cons (root T) (right T))
        (if (null? (right T))
            (cons (root T) (left T))
          (if (< (root (left T)) (root (right T)))
              (let ((L (pop (left T))))
                (cons (root T)
                  (node (car L) (cdr L) (right T))))
            (let ((R (pop (right T))))
                (cons (root T)
                  (node (car R) (left T) (cdr R)))))))))

;Value: pop

(define (pop* T)
  (if (null? T)
      '()
    (let ((U (pop T)))
      (cons (car U) (pop* (cdr U)))))

;Value: pop*

(define (heapsort L) (pop* (build-heap L)))

;Value: heapsort

(heapsort (list
  748 197 917 819 930 653 682 792 523 284 420 851 450 372 308 91 342 501 725 744))

;Value: (91 197 284 308 342 372 420 450 501 523 653 682 725 744 748 792 819 851 917 930)