

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2004

Recitation 7
HOPs, Trees, and Programming

Problems

```
(define (make-node val left right)
  (lambda (p)
    (p val left right)))

(get-val (make-node 3 #f #f))
;Value: 3

(get-right (make-node 3 (make-node 2 #f #f) #f))
;Value: #f

(get-left (make-node 3 #f (make-node 5 #f #f)))
;Value: #f

(node->list (make-node 3 #f (make-node 5 #f #f)))
;Value: (3 #f (5 #f #f))
```

1. Write `get-val`.

```
(define (get-val node)
```

2. Write `get-right`.

```
(define (get-right node)
```

3. Write `get-left`.

```
(define (get-left node)
```

4. Write `node->list`.

```
(define (node->list node)
```

5. Write `leaf?`.

```
(define (leaf? node)
```

```
(define (new-right node right)
  (node (lambda (val left oldright) (lambda (p) (p val left right))))))
```

```
(define (new-left node left)
  (node (lambda (val oldleft right) (lambda (p) (p val left right))))))
```

6. Write `insert-tree`.

```
(define (insert-tree val tree)
```

7. Write `in-order-read`.

```
(define (in-order-read tree)
```

8. Write `mysort` using `insert-tree` and `in-order-read`.

```
(define (mysort tree)
```

Order of growth in time? Space?

9. Rewrite `in-order-read` to use `tree-accum`.

```
(define (tree-accum op leafop tree)
  (cond ((not tree) #f)
        ((leaf? tree)
         (leafop (get-val tree)))
        (else
         (op (get-val tree)
              (tree-accum op leafop (get-left tree))
              (tree-accum op leafop (get-right tree))))))

(define (in-order-read tree)
```