6.001
A. Meyer

Fall 1997
Lecture: 10/7/97

Representing Sets

$$S = \underbrace{\{1, 7, 5, 3, 11, 9\}}_{\text{order does not matter!}}$$

- As a **list**

```
(1 7 5 3 11 9)
```

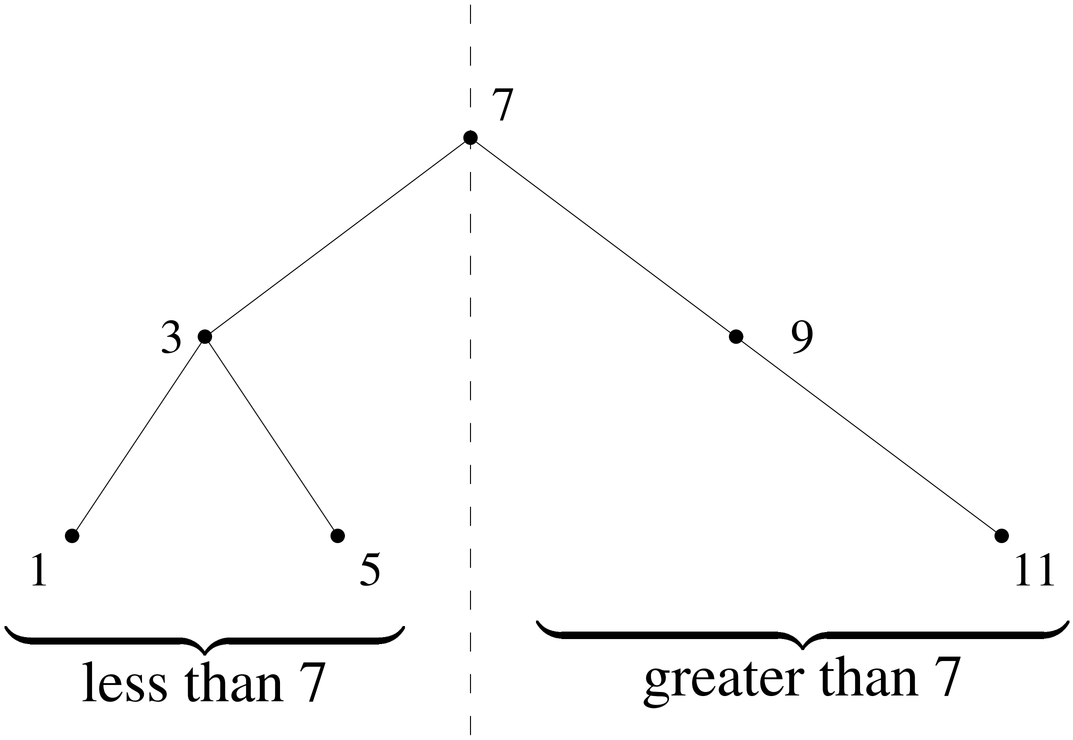
- As a **list with repeats**

```
(1 7 1 5 3 11 9 5)
```

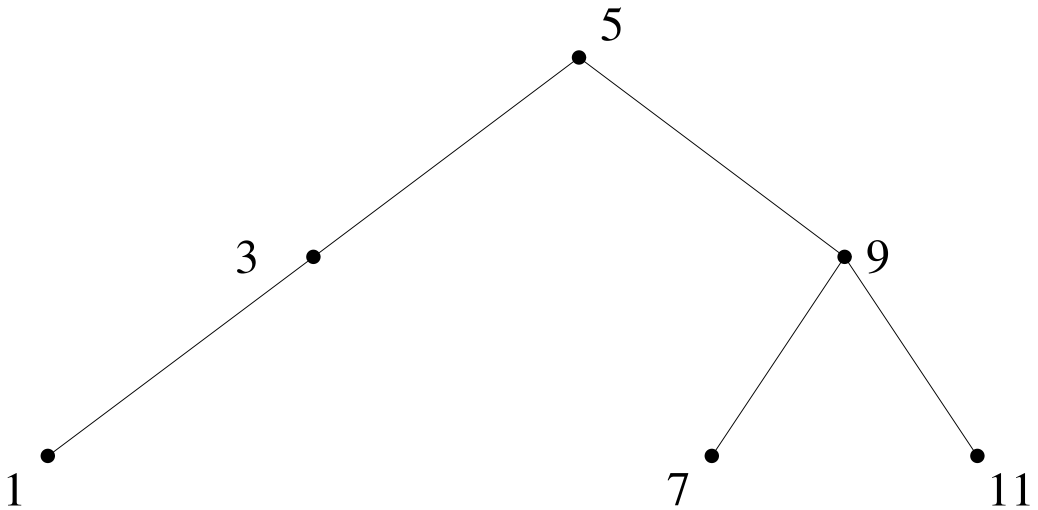
- As a **sorted list**

```
(1 3 5 7 9 11)
```

- As a **labelled tree**:



Or



- As a **membership predicate**

```
(lambda (el) (and (integer? el)
                  (odd? el)
                  (< el 13)
                  (> el 0)))
```

Operations on Sets

```
(element-of-set? el S)

(adjoin-set el S)
```

— returns representation of {`el`} ∪ `S`

```
(intersection-set S1 S2)

(union-set S1 S2)

(empty-set? S)

the-empty-set
```

Further operations

```
(set=? S1 S2)
```

```
(size-set S)
```

—what if `S` is infinite?

```
(some-element S)
```

```
(least-element S)
```

```
(list-of-all-elements S)
```

—what if `S` is infinite?

Implementing Union

```
; List with repeats

(define (union S1 S2)
  (append S1 S2))
```

$$\mathrm{Time}, \mathrm{Space} = \Theta(\mathrm{length}(S_1))$$

but $\mathrm{length}(S_1)$ **may be** $\gg \mathrm{size}(S_1)$

```
; Sorted lists
(define union merge)
```

$$\mathrm{Time}, \mathrm{Space} = \Theta(\mathrm{size}(S_1) + \mathrm{size}(S_2))$$

```
(define (merge l1 l2)
  (cond
   ((null? l1) l2)
   ((null? l2) l1)
   (else
    (let ((e1 (car l1))
          (e2 (car l2)))
      (cond ((< e1 e2)
             (cons e1
                   (merge (cdr l1) l2)))
            ((< e2 e1)
             ...   )
            (else                  ; e1 = e2
             (cons e1
                   (merge (cdr l1)
                          (cdr l2)))))))))
```

```
; Membership predicates

(define (union S1 S2)
  (lambda (el) (or (S1 el) (S2 el))))
```

$$\text{Time}, \text{Space} = \Theta(1)$$

```
; List without repeats
```

Time for union is $\Theta(\text{size}(\texttt{S1}))$ times the time to **adjoin** an element to `S2` (really some subset of $\texttt{S1} \cup \texttt{S2}$), because:

```
(define (union S1 S2)
  (accumulate adjoin-set S2 S1))


(define (adjoin-set el S)
  (if (element-of-set? el S)
      S
      (cons el S)))
```

Time for `(adjoin-set el S)` when `el` is not in `S` is Time for `(element-of-set? el S)` is $\Theta(\text{size}(\texttt{S}))$.

So Time for union of two disjoint sets of size n is $\Theta(n^2)$. (Compare to $\Theta(n)$ for Ordered lists.)

`set=?`

- is **easily definable** for **list** and **tree** representations
- is **not** definable using **membership predicates**, even when the sets represented are finite, because there is no way in general to define a test that two procedures always return the same values.

Procedures are black-box abstractions!

Issues of Multiple Representations

- Which one? (Cost trade-offs)
- Convert from one to another
  - How? (and how costly?)
  - When?
- Use more than one
  - How to keep track of which rep? (Different list reps may look alike)
  - "Generic" union operation works on any, even mixed reps

Next Week: Handling Multiple Reps