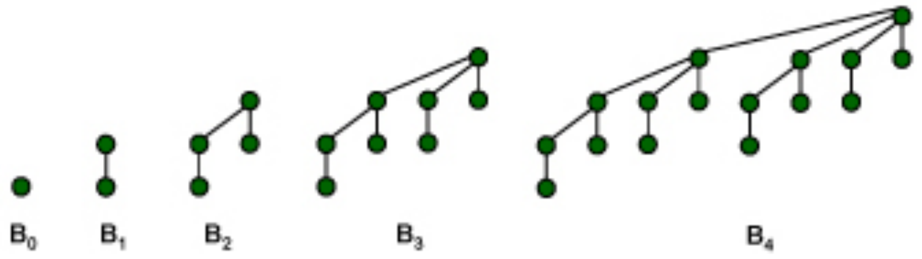
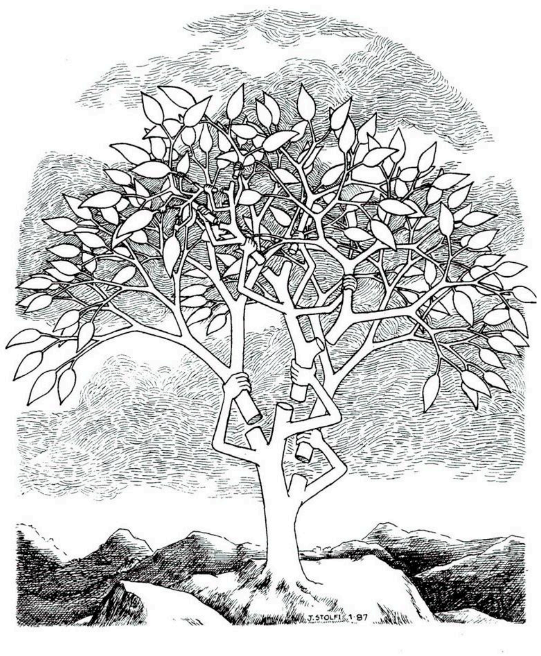
Binomial Priority Queues

CS21b: Structure and Interpretation of Computer Programs
Spring 2012





What is a priority queue?

A *priority queue* is a data structure for dynamically maintaining a *set of elements* (think: positive integers) supporting the following operations:

`(make-queue)`	create an empty queue
`(insert x Q)`	insert integer `x` into the queue
`(find-max Q)`	find the largest element in the queue
`(remove-max Q)`	remove largest queue element
`(merge Q1 Q2)`	merge two queues

We implement a priority queue using a *forest* of *binomial trees.*

Goal: implement these operations, on an n-node queue, in O(log n) time.

Why are we learning about this?

It’s an interesting and complicated example of list structures and recursion.

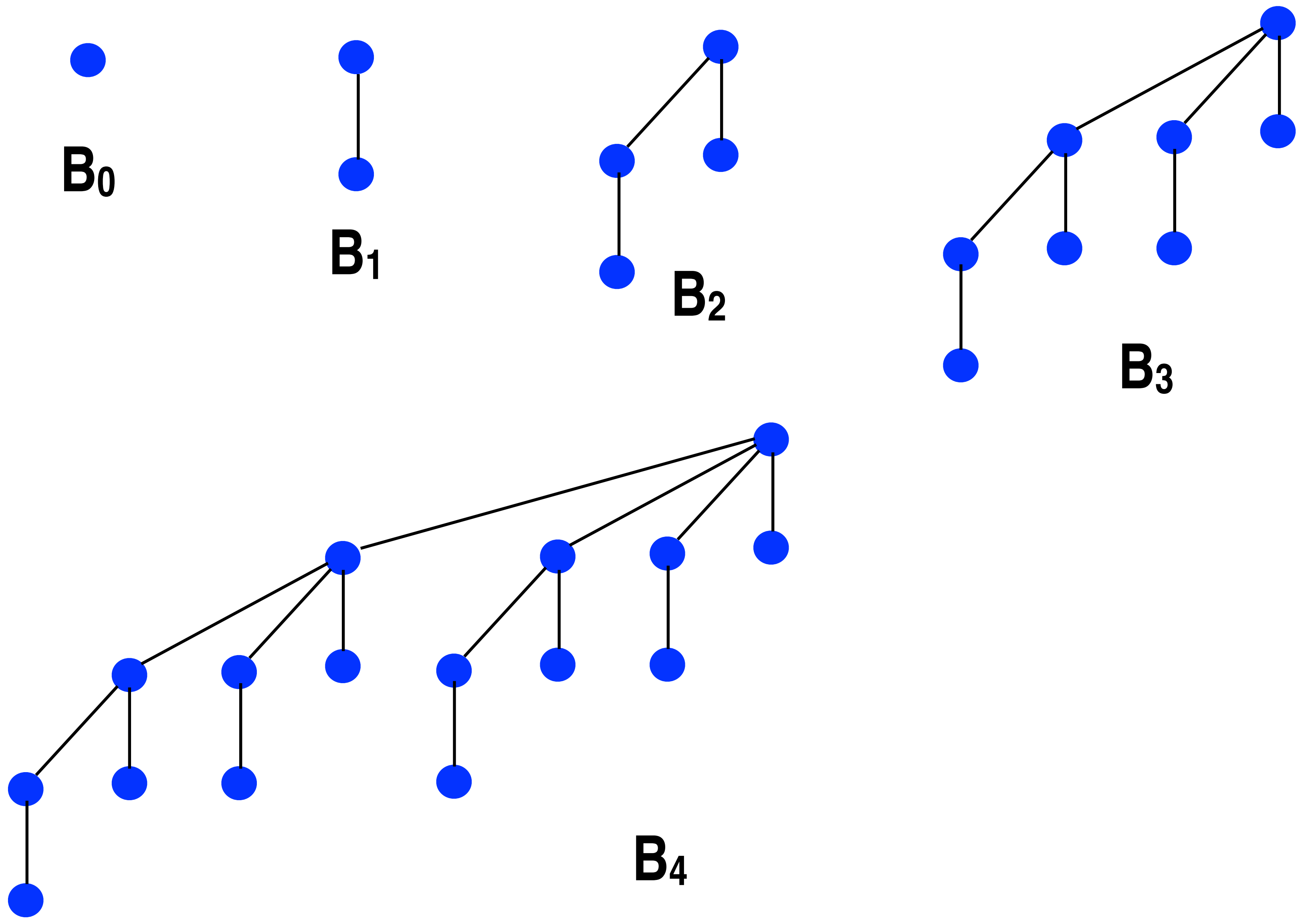
It’s a nice algorithm.

It has something to do with addition.

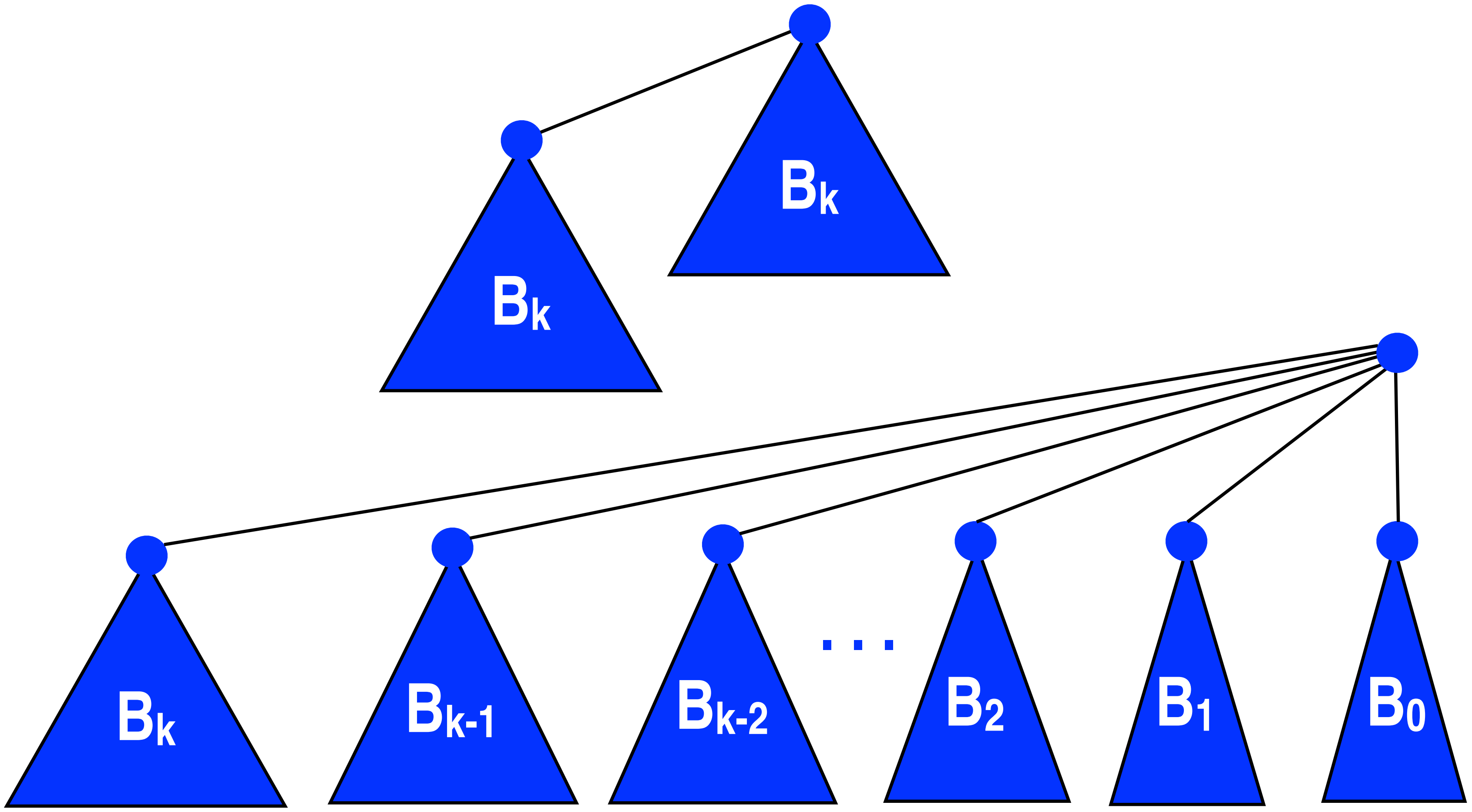
It’s serious.

It’s not boring.

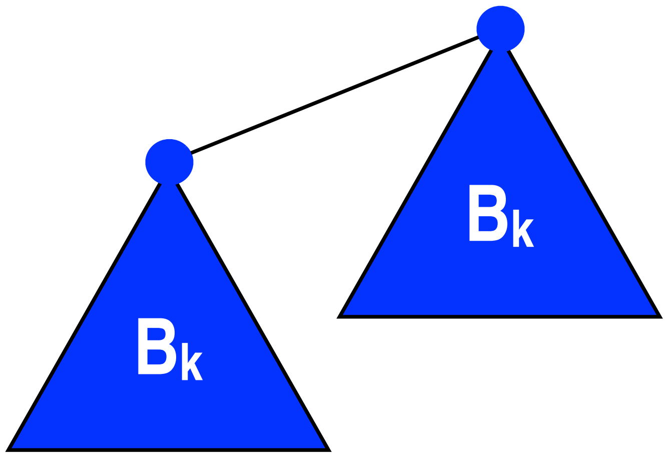
What is a binomial tree?

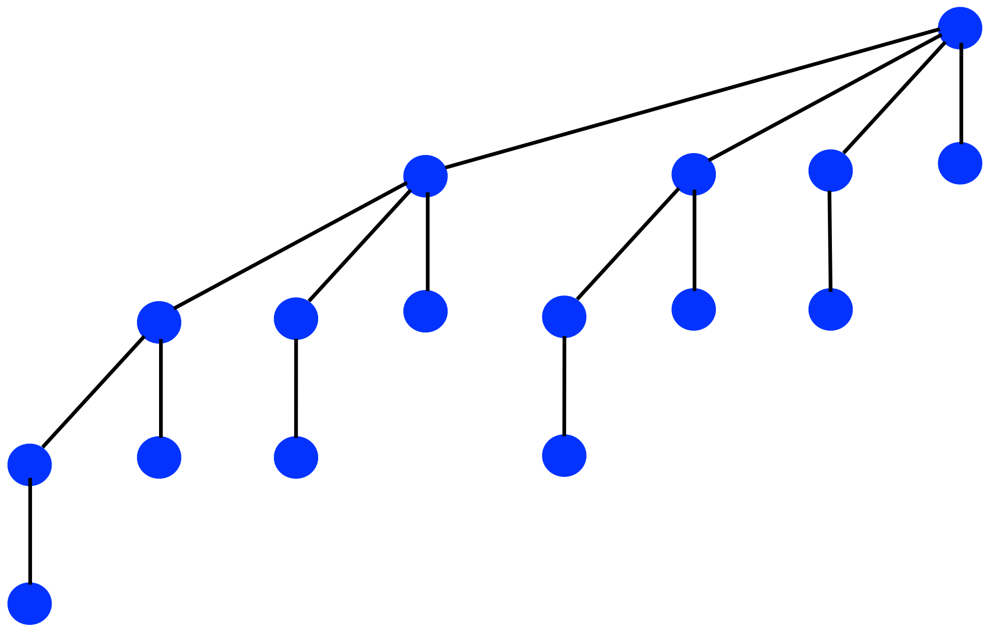


Two ways of thinking about binomial trees: B_{k+1} is either...



B_{k+1} =



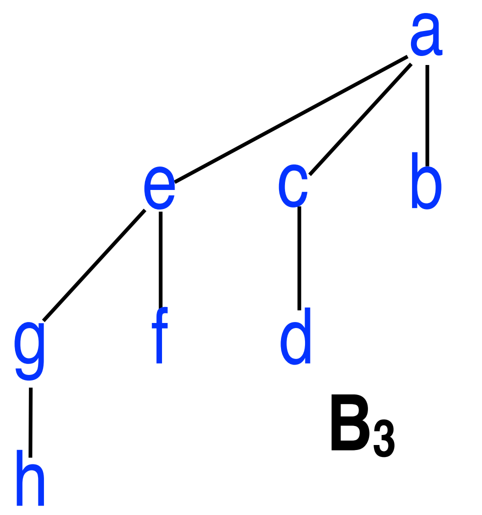


B_k has 2^k nodes with $\binom{k}{j}$ nodes at the *j*-th level--recall that

$$\sum_{0 \le j \le k} \binom{k}{j} = 2^k$$

(***binomial*** coefficients)

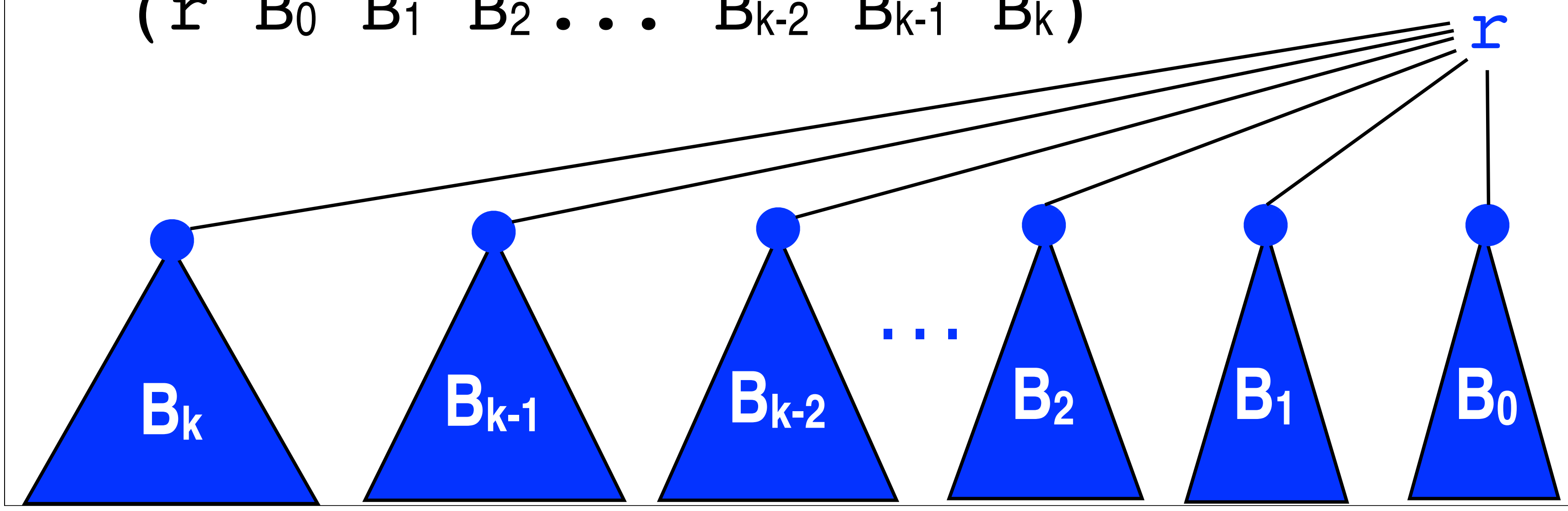
Representing a binomial tree as a list structure



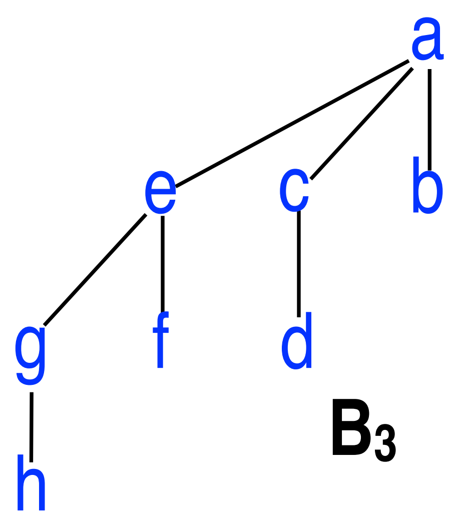
B_3

`(a b (c d) (e f (g h)))`

`(r` B_0 B_1 B_2 `...` B_{k-2} B_{k-1} B_k`)`



Representing a binomial tree of integers as a list structure



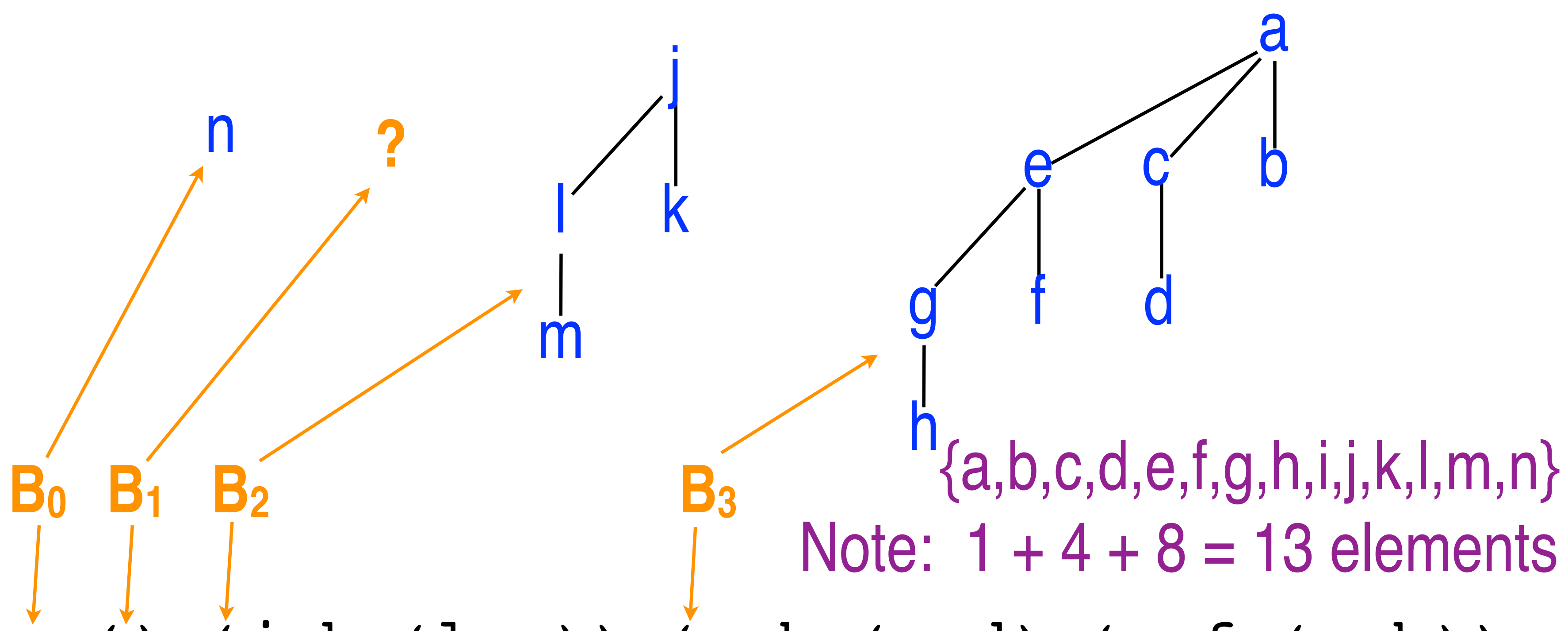
```
(a b (c d) (e f (g h))
```

(Integer) elements are
heap ordered:

```
a > b, c, e
c > d
e > f, g
g > h
```

A binomial queue is a *forest* of binomial *trees*

where no two trees are the same size



{a,b,c,d,e,f,g,h,i,j,k,l,m,n}

Note: 1 + 4 + 8 = 13 elements

```
(n () (j k (l m)) (a b (c d) (e f (g h))
```

Claim: Regardless of the queue on a set of integers (not unique!), the maximum element is always the root of some tree in the forest.

Insertion into the queue

```
(define (make-queue) ‘())

(define (tack x lst)
  (if (null? lst)
      (list x)
      (cons (car lst) (tack x (cdr lst)))))

(define (root e)
  (if (number? e)
      e
      (car e)))

(define (meld e1 e2)
;; for combining two equal-sized binomial trees
  (if (and (number? e1) (number? e2))
      (list (max e1 e2) (min e1 e2))
      (if (< (root e1) (root e2))
        (tack e1 e2)
        (tack e2 e1))))

(define (insert e q)
  (if (null? q)
      (list e)
      (if (null? (car q))
          (cons e (cdr q))
          (cons '()
            (insert (meld e (car q))
                    (cdr q))))))
```

```
(tack 10 ‘(0 2 4 6 8))
;Value: (0 2 4 6 8 10)

(meld 5 10)
;Value: (10 5)

(meld '(15 0) '(10 5))
;Value: (15 0 (10 5))

(insert 10 (make-queue))
;Value: (10)

(insert 5 (insert 10 (make-queue)))
;Value: (() (10 5))

(insert 15 (insert 5
  (insert 10 (make-queue))))
;Value: (15 (10 5))

(insert 0 (insert 15 (insert 5
  (insert 10 (make-queue)))))
;Value: (() () (15 0 (10 5)))
```

Insertion takes time *logarithmic* in the queue size---*why?*

Why is this procedure like adding 1?

Merging two queues

```
(define (merge q1 q2)
  (cond ((null? q1) q2)
        ((null? q2) q1)
        ((null? (car q1))
         (cons (car q2)
               (merge (cdr q1)
                      (cdr q2))))
        ((null? (car q2))
         (cons (car q1)
               (merge (cdr q1)
                      (cdr q2))))
        (else
         (cons '()
               (insert
                 (meld (car q1)
                       (car q2))
                 (merge (cdr q1)
                        (cdr q2))))))))
```

```
(merge
  '(1
    ()
    ()
    (9 8 (7 6) (5 4 (3 2))))
  '(10
    ()
    (14 13 (12 11))
    (22 21
        (20 19)
        (18 17 (16 15)))))

;Value:
(()
 (10 1)
 (14 13 (12 11))
 ()
 (22 21
     (20 19)
     (18 17 (16 15))
     (9 8 (7 6) (5 4 (3 2)))))
```

1001 + 1101= 10110

Why is queue merging like binary addition?

Finding the maximum element in a queue

```
(define (max-elt e)
  (if (null? e) 0 (if (number? e) e (car e))))

(define (find-max q)
  (apply max (map max-elt q)))
```

Cleaning up a queue by removing large, empty subtrees (like leading zeroes in a binary number)

```
(define (slinkyleft leftlist rightlist)
  (if (null? rightlist)
      leftlist
      (slinkyleft
        (cons (car rightlist) leftlist)
        (cdr rightlist))))

(define (clean lst)
  (if (null? lst)
      '()
      (if (null? (car lst))
          (clean (cdr lst))
          lst)))

(define (cleanup q)
  (slinkyleft
    '()
    (clean (slinkyleft '() q))))
```

```
(cleanup
  '(1
    ()
    ()
    (9 8 (7 6) (5 4 (3 2)))
    ()
    ()
    ()
    ()
    ()
    ()))
;Value:
   (1
    ()
    ()
    (9 8 (7 6) (5 4 (3 2))))
```

Removing the maximum element in a queue

```scheme
(define (select q maxval)
; returns (tree with max root . rest of queue)
  (if (null? q)
      (error
        "Cannot select from empty queue")
      (if (= maxval (max-elt (car q)))
          (cons (car q)
                (cleanup
                  (cons '() (cdr q))))
          (let ((v (select (cdr q) maxval)))
            (cons (car v)
                  (cleanup
                    (cons (car q)
                          (cdr v)))))))))

(define (remove-max q)
  (let ((q (select q (find-max q))))
    (if (number? (car q))
        (cdr q)
        (merge (cdar q) (cdr q)))))

(define (removes q n)
  (if (= n 0)
      q
      (removes (remove-max q) (- n 1))))
```

```scheme
(define qq
 '(() () () ()
   (16 15 (14 13)
       (12 11 (10 9))
       (8 7 (6 5)
         (4 3 (2 1))))))
;Value: qq

(removes qq 1)
;Value: (15 (14 13) (12 11 (10 9))
(8 7 (6 5) (4 3 (2 1))))

(removes qq 2)
;Value: (() (14 13) (12 11 (10 9))
(8 7 (6 5) (4 3 (2 1))))

(removes qq 3)
;Value: (13 () (12 11 (10 9))
(8 7 (6 5) (4 3 (2 1))))

(removes qq 4)
;Value: (() () (12 11 (10 9))
(8 7 (6 5) (4 3 (2 1))))

(removes qq 5)
;Value: (11 (10 9) ()
(8 7 (6 5) (4 3 (2 1))))
```

Removing the maximum element in a queue

```
(define qq
'(() () () ()
  (16 14 (13 12) (15 11 (10 9))
      (8 4 (6 5) (7 3 (2 1))))))
;Value: qq

(removes qq 1)
;Value: (14 (13 12) (15 11 (10 9))
            (8 4 (6 5) (7 3 (2 1))))

(removes qq 2)
;Value: (() (14 11) (13 12 (10 9))
            (8 4 (6 5) (7 3 (2 1))))

(removes qq 3)
;Value: (11 () (13 12 (10 9))
            (8 4 (6 5) (7 3 (2 1))))

(removes qq 4)
;Value: (() () (12 11 (10 9))
            (8 4 (6 5) (7 3 (2 1))))

(removes qq 5)
;Value: (11 (10 9) ()
            (8 4 (6 5) (7 3 (2 1))))

(removes qq 6)
;Value: (() (10 9) ()
            (8 4 (6 5) (7 3 (2 1))))

(removes qq 7)
;Value: (9 () () (8 4 (6 5) (7 3 (2 1))))
```

```
(removes qq 8)
;Value: (() () () (8 4 (6 5) (7 3 (2 1))))

(removes qq 9)
;Value: (4 (6 5) (7 3 (2 1)))

(removes qq 10)
;Value: (() (4 3) (6 5 (2 1)))

(removes qq 11)
;Value: (5 () (4 3 (2 1)))

(removes qq 12)
;Value: (() () (4 3 (2 1)))

(removes qq 13)
;Value: (3 (2 1))

(removes qq 14)
;Value: (() (2 1))

(removes qq 15)
;Value: (1)

(removes qq 16)
;Value: ()
```