(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
        (cons (car xs) ys))))

(define reverse (xs) (revapp xs '()))
(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
              (cons (car xs) ys)))))

(define reverse (xs) (revapp xs '()))
(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
              (cons (car xs) ys)))
)

(define reverse (xs) (revapp xs '()))

(reverse '(1 2 3))
(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
               (cons (car xs) ys))))

(define reverse (xs) (revapp xs '()))

reverse '(1 2 3)

revapp '(1 2 3) '()

revapp '(2 3) '(1)
(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
               (cons (car xs) ys))))

(define reverse (xs) (revapp xs '()))

reverse '(1 2 3)

revapp '(1 2 3) '()

revapp '(2 3) '(1)

revapp '(3) '(2 1)
(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
               (cons (car xs) ys))))

(define reverse (xs) (revapp xs '()))

reverse '(1 2 3)

revapp '(1 2 3) '()

revapp '(2 3) '(1)

revapp '(3) '(2 1)

revapp '() '(3 2 1)
(define revapp (xs ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
              (cons (car xs) ys)))))

(define reverse (xs) (revapp xs '()))
(define revapp (xs ys)
    (if (null? xs)
        ys
        (revapp (cdr xs)
            (cons (car xs) ys)))))

(define reverse (xs) (revapp xs '()))

reversing '(1 2 3)

Number of revapp steps is length of xs.
Each step allocates 1 cons cell.
Therefore, cost is linear in length of xs.

reverse '(1 2 3)

revapp '(1 2 3) '()