Lists
The definition of a list
The definition of a list

• The empty list is a list.
• A nonempty list list has two parts.
  • (car list) — the first element of the list.
  • (cdr list) — the rest of the list.
The definition of a list

- The empty list is a list.
- A nonempty list \( lst \) has two parts.
  - \((\text{car } lst)\) — the first element of the list.
  - \((\text{cdr } lst)\) — the rest of the list.

\text{car} is an element.

\text{cdr} is a list.
interleave

(interleave '(a b c) '(d e f))
interleave

(interleave '(a b c) '(d e f))

(car '(a b c))
  a
interleave

(interleave '(a b c) '(d e f))

(car '(a b c))  (cdr '(a b c))
  a              (b c)
interleave

(interleave '(a b c) '(d e f))

(car '(a b c))   (cdr '(a b c))
  a             (b c)

(interleave '(d e f) '(b c))
   (d b e c f)
interleave

(interleave '(a b c) '(d e f))

(car '(a b c))  (cdr '(a b c))
  a  (b c)

(interleave '(d e f) '(b c))
  (d b e c f)
  a
add-to-end

(add-to-end '(a b c d) 'x)
add-to-end

(add-to-end '(a b c d) 'x)

(car '(a b c d))

a
add-to-end

(add-to-end '(a b c d) 'x)

(car '(a b c d)) (cdr '(a b c d))
  a  (b c d)
add-to-end

(add-to-end '(a b c d) 'x)

(car '(a b c d))  (cdr '(a b c d))
  a  (b c d)

(add-to-end '(b c d) 'x)
  (b c d x)
add-to-end

(add-to-end '(a b c d) 'x)

(car '(a b c d))     (cdr '(a b c d))
  a                   (b c d)

(add-to-end '(b c d) 'x)

(b c d x)

  a
(define add-to-end
  (lambda (lst elt)
    (if (null? lst)
        (cons elt '())
        (cons (car lst)
              (add-to-end (cdr lst) elt))))

What is the efficiency of add-to-end?
What is the efficiency of `add-to-end`?

\[ \Theta(n) \]
(define my-reverse
  (lambda (lst)
    (if (null? lst)
      ()
      (add-to-end (my-reverse (cdr lst)) (car lst)))))

What is the efficiency of my-reverse?
What is the efficiency of my-reverse?

$$\Theta(n^2)$$
(reverse-onto ' (a b c) ' (1 2 3))
  (c b a 1 2 3)
(reverse-onto ' (a b c) ' (1 2 3))
(c b a 1 2 3)

(car '(a b c))
a
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)

(car '(a b c))
a

(cdr '(a b c))
(b c)
(reverse-onto ' (a b c) ' (1 2 3))
   (c b a 1 2 3)

(car ' (a b c))  (cdr ' (a b c))
   a                (b c)

(cons ' a ' (1 2 3))
   (a 1 2 3)
(reverse-onto '(a b c) '(1 2 3))
  (c b a 1 2 3)

(car '(a b c))  (cdr '(a b c))
  a  (b c)

(cons 'a '(1 2 3))
  (a 1 2 3)

(reverse-onto '(b c) '(a 1 2 3))
  (c b a 1 2 3)
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1) (cons (car lst1) lst2))))

(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
(reverse-onto ' (a b c) ' (1 2 3))
  (c b a 1 2 3)

(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1) (cons (car lst1) lst2)))))
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)

(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst1
        lst2)))
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
                      (cons (car lst1) lst2))))

(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
          (cons (car lst1) lst2))))
)
(reverse-onto '(a b c) '(1 2 3))
(c b a 1 2 3)

(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
                      (cons (car lst1) lst2))))

What is the efficiency of reverse-onto?
(define reverse-onto
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (reverse-onto (cdr lst1)
                       (cons (car lst1) lst2)))))

What is the efficiency of reverse-onto?

Θ(n)
;; Efficient version of reverse
(define your-reverse
;;; Efficient version of reverse
(define your-reverse
  (lambda (lst)
    (define reverse-onto
      (lambda (lst1 lst2)
        (if (null? lst1)
            lst2
            (reverse-onto (cdr lst1)
                          (cons (car lst1) lst2))))
    (reverse-onto lst '())))
;; Efficient version of reverse
(define your-reverse
  (lambda (lst)
    (define reverse-onto
      (lambda (lst1 lst2)
        (if (null? lst1)
            lst2
            (reverse-onto (cdr lst1)
                          (cons (car lst1) lst2)))))))
;; Efficient version of reverse
(define your-reverse
  (lambda (lst)
    (define reverse-onto
      (lambda (lst1 lst2)
        (if (null? lst1)
            lst2
            (reverse-onto (cdr lst1)
                          (cons (car lst1) lst2)))))
    (reverse-onto lst '())))
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)

(car '(2 4 6 8))
2
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)

(car '(2 4 6 8))  (car '(1 3 5 8 9))
2 1
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)

(car '(2 4 6 8))  (car '(1 3 5 8 9))
2 1
(cdr '(1 3 5 8 9))
(3 5 8 9)
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)

(car '(2 4 6 8)) (car '(1 3 5 8 9))
2 1

(cdr '(1 3 5 8 9))
(3 5 8 9)

(merge '(2 4 6 8) '(3 5 8 9))
(2 3 4 5 6 8 9)
(merge '(2 4 6 8) '(1 3 5 8 9))
(1 2 3 4 5 6 8 9)

(car '(2 4 6 8))   (car '(1 3 5 8 9))
2                  1

(cdr '(1 3 5 8 9))
(3 5 8 9)

(merge '(2 4 6 8) '(3 5 8 9))
(2 3 4 5 6 8 9)
(odd-part '(g i r a f f e))
  (g r f e)
(odd-part '(g i r a f f e))
  (g r f e)

(car '(g i r a f f e))
  g
(odd-part '(g i r a f f e))
(g r f e)

(car '(g i r a f f e))
g

(cdr '(g i r a f f e))
(i r a f f e)
(odd-part '(g i r a f f e))
  (g r f e)

(car '(g i r a f f e))
  g

(cdr '(g i r a f f e))
  (i r a f f e)

(even-part '(i r a f f e))
  (r f e)
(odd-part '(g i r a f f e))
  (g r f e)

(car '(g i r a f f e))
  g

(cdr '(g i r a f f e))
  (i r a f f e)

(even-part '(i r a f f e))
  (r f e)
A child at the county fair wins 5 tickets.

The redemption store carries the following items, priced in tickets:

a — apples, 3 tickets each
b — balls, 3 tickets each
c — cookies, 2 tickets each
d — dolls, 1 ticket each
e — ear muffs, 1 ticket each
A child at the county fair wins 5 tickets.

The redemption store carries the following items, priced in tickets:

a — apples, 3 tickets each
b — balls, 3 tickets each
c — cookies, 2 tickets each
d — dolls, 1 ticket each
e — ear muffs, 1 ticket each

In how many ways can the child spend her tickets?
a b c d e
(3 3 2 1 1)
a b c d e
(3 3 2 1 1)

ac
add
ade
ae
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} & \quad \text{e} \\
(3 & \quad 3 & \quad 2 & \quad 1 & \quad 1) \\
\text{ac} & \quad \text{bc} \\
\text{add} & \quad \text{bdd} \\
\text{ade} & \quad \text{bde} \\
\text{ae} & \quad \text{bee}
\end{align*}
CoSc 450: Programming Paradigms

\[ a \ b \ c \ d \ e \]
\[ (3 \ 3 \ 2 \ 1 \ 1) \]

\[
\begin{array}{ccc}
ac & bc & ccd \\
add & bdd & cce \\
ade & bde & cdddd \\
aee & bee & cdde \\
aee & bee & cdee \\
aee & bee & ceee
\end{array}
\]
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac</td>
<td></td>
<td></td>
<td>ccd</td>
<td></td>
<td>ddd</td>
</tr>
<tr>
<td>add</td>
<td></td>
<td></td>
<td>cce</td>
<td></td>
<td>dddd</td>
</tr>
<tr>
<td>ade</td>
<td></td>
<td></td>
<td>cddd</td>
<td></td>
<td>ddee</td>
</tr>
<tr>
<td>aee</td>
<td></td>
<td></td>
<td>cdde</td>
<td></td>
<td>ddee</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cdee</td>
<td></td>
<td>deee</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ceee</td>
<td></td>
<td>deee</td>
</tr>
</tbody>
</table>

(3 3 2 1 1)
<table>
<thead>
<tr>
<th>ac</th>
<th>bc</th>
<th>ccd</th>
<th>ddddd</th>
<th>eeeee</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>bdd</td>
<td>cce</td>
<td>dddde</td>
<td></td>
</tr>
<tr>
<td>ade</td>
<td>bde</td>
<td>cddd</td>
<td>dddee</td>
<td></td>
</tr>
<tr>
<td>aee</td>
<td>bee</td>
<td>cdde</td>
<td>ddeee</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cdee</td>
<td>deeee</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ceee</td>
<td>deeeee</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>ac</td>
<td>bc</td>
<td>ccd</td>
<td>ddddd</td>
<td>eee</td>
</tr>
<tr>
<td>add</td>
<td>bdd</td>
<td>cce</td>
<td>dddde</td>
<td></td>
</tr>
<tr>
<td>ade</td>
<td>bde</td>
<td>cddd</td>
<td>dddee</td>
<td></td>
</tr>
<tr>
<td>aee</td>
<td>bee</td>
<td>cdde</td>
<td>ddeee</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cdee</td>
<td>deeee</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ceee</td>
<td>deee</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>deee</td>
<td></td>
</tr>
</tbody>
</table>

(count-combos '(3 3 2 1 1) 5)
20
(count-combos '(3 3 2 1 1) 5)
(count-combos '(3 3 2 1 1) 5)

Pick first item
(count-combos '(3 3 2 1 1) 5)

Pick first item

(count-combos '(3 3 2 1 1) 2)
(count-combos '(3 3 2 1 1) 5)

Pick first item

Do not pick first item

(count-combos '(3 3 2 1 1) 2)
(count-combos '(3 3 2 1 1) 5)

Pick first item

(count-combos '(3 3 2 1 1) 2)

(count-combos '(3 2 1 1) 5)

Do not pick first item
(33211)₅
$(33211)_2 \rightarrow (33211)_5$
\[(33211)_{5}\]

\[(33211)_{2}\]

0

\[(33211)_{-1}\]

\[(3211)_{2}\]
(33211)5

(33211)2

(33211)−1 (3211)2

0

(3211)−1 (211)2

4

(211)0 (11)2

3

(11)1 (1)2

1

(11)0 (1)1

1

(1)0 ( )1
The diagram illustrates a tree structure with nodes labeled with sequences of digits. The tree starts with the sequence (33211)5 at the root. The tree branches down with each node labeled with a sequence of digits, and the branches are labeled with integers. The diagram continues to branch out with the sequences (33211)2, (3211)2, (3211)-1, (211)2, (211)0, (11)2, (11)1, (11)0, (1)2, (1)1, (1)0, and (1)1. The tree structure continues to expand with more sequences as it branches further.
\[(33211)_5\]

- \[(33211)_2\]
  - \[(33211)_{-1}\]
    - \[(3211)_{-1}\]
      - \[(211)_{-1}\]
        - \[(11)_{-1}\]
          - \[(1)_{-1}\]
            - \[(0)_{-1}\]
              - \[\cdot\cdot\cdot\]
    - \[(211)_{2}\]
      - \[(11)_{2}\]
        - \[(1)_{2}\]
          - \[(1)_{2}\]
            - \[(0)_{2}\]
              - \[\cdot\cdot\cdot\]
  - \[(3211)_{2}\]
    - \[(3211)_{4}\]
      - \[(0)_{4}\]
        - \[\cdot\cdot\cdot\]

- \[(3211)_{-1}\]
  - \[(211)_{2}\]
    - \[(11)_{2}\]
      - \[(1)_{2}\]
        - \[(0)_{2}\]
          - \[\cdot\cdot\cdot\]
- \[(3211)_{5}\]
  - \[(33211)_{5}\]
    - \[(4)_{5}\]
      - \[(4)_{5}\]
        - \[\cdot\cdot\cdot\]

- \[(3211)_{5}\]
  - \[(3211)_{5}\]
    - \[(4)_{5}\]
      - \[(4)_{5}\]
        - \[\cdot\cdot\cdot\]
(define count-combos
  (lambda (prize-list amount)
    .
    .
    (+ (count-combos prize-list (- amount (car prize-list)))
      (count-combos (cdr prize-list) amount))))

(count-combos '(3 3 2 1 1) 5)

(define count-combos
  (lambda (prize-list amount)
(define count-combos
  (lambda (prize-list amount)
    .
    .
    .
    (+
      (count-combos prize-list (- amount (car prize-list)))
      (count-combos (cdr prize-list) amount))))

(count-combos '(3 3 2 1 1) 5)
(define count-combos
  (lambda (prize-list amount)
    .
    .
    .
    (+ (count-combos prize-list (- amount (car prize-list))))
)

(count-combos '(3 3 2 1 1) 5)
(count-combos '(3 3 2 1 1) 5)

(define count-combos
  (lambda (prize-list amount)
    .
    .
    .
    (+ (count-combos prize-list (- amount (car prize-list)))
      (count-combos (cdr prize-list) amount)))))