• Sorting a list
• List abbreviations
• Lists containing lists
• Dictionaries and association lists
• Lists of lists as 2D data
• Processing two lists simultaneously
• Consuming a list and a number

Lists, part II
Readings

• HtDP, sections 11, 12, 13 (Intermezzo 2).
Sorting a list
Sorting a list

When writing a function to consume a list, we may find that we need to create an helper function to do some of the work. The helper function may or may not be recursive itself.

Sorting a list of numbers provides a good example; in this case the solution follows easily from the templates and design process.

In this course and CS 136, we will see several different sorting algorithms.
Starting with the list template

;;; (sort lon) sorts the elements of lon in non-decreasing order
;;; sort: (listof Num) -> (listof Num)

(check-expect (sort (cons 2 (cons 0 (cons 1 empty))))
             (cons 0 (cons 1 (cons 2 empty))))

(define (sort lon)
  (cond
   [(empty? lon) ...]
   [else (... [X-template (first lon))
             (sort (rest lon))]])))

If the list lon is empty, so is the result.

Otherwise, the template suggests doing something with the first element of the list, and the sorted version of the rest.
Filling in the list template

;;; (sort lon) sorts the elements of lon in non-decreasing order

;;; sort: (listof Num) -> (listof Num)

(check-expect (sort (cons 2 (cons 0 (cons 1 empty))))
  (cons 0 (cons 1 (cons 2 empty))))

(define (sort lon)
  (cond
   [(empty? lon) ...]
   [else (insert
       (first lon)
       (sort (rest lon)))]))

insert is a recursive helper function that consumes a number and a sorted list, and inserts the number to the sorted list.
A condensed trace of sort and insert

(sort (cons 2 (cons 4 (cons 3 empty))))
=> (insert 2 (sort (cons 4 (cons 3 empty))))
=> (insert 2 (insert 4 (sort (cons 3 empty))))
=> (insert 2 (insert 4 (insert 3 (sort empty))))
=> (insert 2 (insert 4 (insert 3 empty)))
=> (insert 2 (insert 4 (cons 3 empty)))
=> (insert 2 (cons 3 (cons 4 empty)))
=> (cons 2 (cons 3 (cons 4 empty)))
The helper function insert

We again use the list template for insert.

;; (insert n slon) inserts the number n into the
;; sorted list slon so that the resulting list is
;; also sorted.

;; insert: Num (listof Num) -> (listof Num)
;; requires: slon is sorted in non-decreasing order

(define (insert n slon)
 (cond
  [(empty? slon) ...]
  [else (... [else (...]
    (X-template (first slon))
    (insert n (rest slon)))]))
The helper function insert

\[
\text{define (insert n slon)}
\text{(cond}
\text{[[(empty? slon) ...]}
\text{[else (...}
\text{(X-template (first slon))}
\text{(insert n (rest slon)))]])}
\]

If slon is empty, the result is the list containing just n.

n: 5, slon: empty
=> (cons 5 empty)
The helper function `insert`

```scheme
(define (insert n slon)
  (cond
    [(empty? slon) (cons n empty)]
    [else (... (X-template (first slon))
               (insert n (rest slon)))]))
```

n: 5, slon: empty => (cons 5 empty)

If slon is empty, the result is the list containing just n.
The helper function \texttt{insert}

\begin{verbatim}
(define (insert n slon)
  (cond
    [(empty? slon) (cons n empty)]
    [else (\ldots
      (X-template (first slon))
      (insert n (rest slon)))]))
\end{verbatim}

\texttt{n: 5, slon: (cons 10 \ldots)} => \texttt{(cons 5 (cons 10 \ldots))}

If \texttt{n} is less than or equal to the first number in \texttt{slon}, it will become the first number in the result.
The helper function `insert`

```scheme
(define (insert n slon)
  (cond
    [(empty? slon) (cons n empty)]
    [(<= n (first slon)) (cons n slon)]
    [else (... (X-template (first slon))
               (insert n (rest slon)))]))
```

n: 5, slon: (cons 10 ...)
=> (cons 5 (cons 10 ...))

If n is less than or equal to the first number in slon, it will become the first number in the result.
The helper function insert

(define (insert n slon)
  (cond
    [(empty? slon) (cons n empty)]
    [(<= n (first slon)) (cons n slon)]
    [else (...)
      (X-template (first slon))
      (insert n (rest slon)))])))

n: 5, slon: (cons 2 ...)
=> (cons 2...))

Otherwise, the first number in the result is the first number in slon, and the rest of the result is what we get when we insert n into (rest slon).
The helper function insert

\[
\text{(define (insert n slon)}
\text{(cond}
\text{[(empty? slon) (cons n empty)]}
\text{[(<= n (first slon)) (cons n slon)]}
\text{[else (cons)}
\text{(first slon)}
\text{(insert n (rest slon))))]})
\]

\[
n: 5, \text{ slon: (cons 2 ...)}
\Rightarrow \text{(cons 2...)}
\]

Otherwise, the first number in the result is the first number in slon, and the rest of the result is what we get when we insert n into (rest slon).
A condensed trace of insert

(insert 4 (cons 1 (cons 2 (cons 5 empty))))
=> (cons 1 (insert 4 (cons 2 (cons 5 empty))))
=> (cons 1 (cons 2 (insert 4 (cons 5 empty))))
=> (cons 1 (cons 2 (cons 4 (cons 5 empty))))

Our sort with the helper-function insert are together known as insertion sort.