Announcements

• The midterm is on Monday, March 4, from 7:00 to 8:50 PM, and covers material up to and including end of Module 06.

• There will be a Midterm Help Session on Saturday, March 2, from 1:30 to 3:30 PM in MC 1058.

Goals of this tutorial

You should be able to...

• merge two sorted lists.
• perform a condensed traced of insertion sort.
• understand the difference between structural and accumulative recursion.
• write functions using accumulative recursion.
Group Problem - N-increasing?

We sometimes see sequences of numbers like: \{8, 7, 6, 5, 4, \ldots \} or \{8, 8, 7, 5, \ldots \}. If \(a_{i+1} - a_i \leq 0\) for all \(i \geq 1\), then we call such a sequence a non-increasing sequence:

;; A N-increasing Sequence is one of:
;; empty
;; (cons Num empty)
;; (cons n s) where s is a non-empty N-increasing Sequence
;; n is a Num, n \geq (first s)

Write a function, `set-merge`, that consumes 2 N-increasing sequences and merges the 2 N-increasing sequences into one N-increasing sequence without duplicate elements. You should first write a helper function, `remove-dup` which removes duplicates from a N-increasing sequence.

Insertion Sort Trace

We will perform a condensed trace of an insertion sort:

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

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

(sort (list 5 3 9 2 5 7 1 4))
```
Insertion Sort Trace
(sort (list 5 3 9 2 5 7 1 4))

```
5 3 9 2 5 7 1 4
```

Insertion Sort Trace
(sort (list 5 3 9 2 5 7 1 4))
=> (insert 5 (sort (list 3 9 2 5 7 1 4)))

```
5 3 9 2 5 7 1 4
```

Insertion Sort Trace
(sort (list 5 3 9 2 5 7 1 4))
=> (insert 5 (insert 3 (sort (list 9 2 5 7 1 4))))

```
5 3 9 2 5 7 1 4
```
Insertion Sort Trace

= > (insert 5 (sort (list 3 9 2 5 7 1 4)))
= > (insert 5 (insert 3 (sort (list 9 2 5 7 1 4))))
= > (insert 5 (insert 3 (insert 9 (sort (list 2 5 7 1 4)))))

5 3 9 2 5 7 1 4
Insertion Sort Trace

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{sort } (\text{list } 7 \ 1 \ 4)))))\))

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{sort } (\text{list } 1 \ 4)))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{sort } (\text{list } 4))))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{insert } 4 \ (\text{sort } \text{empty}))))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{insert } 4 \ (\text{insert empty})))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{insert } 4 \ (\text{insert empty})))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{insert } 4 \ (\text{insert empty})))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{insert } 4 \ (\text{insert empty})))))\))\))\)

$=> (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \\
(\text{insert } 5 \ (\text{insert } 7 \ (\text{insert } 1 \ (\text{insert } 4 \ (\text{insert empty})))))\))\))\)

\[5 \ 3 \ 9 \ 2 \ 5 \ 7 \ 1 \ 4\]
Insertion Sort Trace

= > (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (insert 1 (insert 4 (sort empty))))))))))

= > (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (insert 1 (insert 4 empty))))))))

5 3 9 2 5 7 1 4

= > (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (insert 1 (list 4))))))))

= > (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (list 1 4)))))))

5 3 9 2 5 7

1 4

= > (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (list 1 4)))))))

5 3 9 2 5 7 1 4
Insertion Sort Trace

=> (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (list 1 4)))))))

=> (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (list 1 4 7)))))

5 3 9 2 5

1 4 7

Insertion Sort Trace

=> (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (list 1 4)))))))

=> (insert 5 (insert 3 (insert 9 (insert 2
(list 1 4 5 7))))

5 3 9 2

1 4 5 7

Insertion Sort Trace

=> (insert 5 (insert 3 (insert 9 (insert 2
(list 1 4 5 7))))

=> (insert 5 (insert 3 (insert 9 (list 1 2 4 5 7))))

5 3 9

1 2 4 5 7
Insertion Sort Trace

\[ \Rightarrow (\text{insert } 5 (\text{insert } 3 (\text{list } 1 2 4 5 7))) \]

\[ \Rightarrow (\text{insert } 5 (\text{insert } 3 (\text{list } 1 2 4 5 7 9))) \]

\[ 5 \quad 3 \]

\[ 1 \quad 2 \quad 4 \quad 5 \quad 7 \quad 9 \]

\[ \Rightarrow (\text{insert } 5 (\text{list } 1 2 3 4 5 7 9)) \]

\[ 5 \]

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 7 \quad 9 \]

\[ \Rightarrow (\text{list } 1 2 3 4 5 5 7 9) \]

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 5 \quad 7 \quad 9 \]
Clicker Question: Insertion Sort

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

(define (insert n slon)
  (cond [(empty? slon) (cons n empty)]
        ;; Insert condition here
        [else (cons (first slon) (insert n (rest slon)))]))
```

What conditional statement would be the most appropriate to sort a list in strictly decreasing order?

- A: `(< n slon) (cons n slon)]`
- B: `(< n (first slon)) (cons (first slon) (rest slon)]`
- C: `(< n (first slon)) (cons n (rest slon)))]`
- D: `(< n (first slon)) (cons n slon)]`
- E: `(< n (first slon)) (cons (first slon) (rest slon)]`

Group Problem: Structural Recursion

Write a function called `sum-lon` that consumes a (listof Num) and produces the sum of the numbers in the list. This function should use pure structural recursion.
Group Problem: Accumulative Recursion

Write a function called `sum-lon2` that consumes a `(listof Num)` and produces the sum of the numbers in the list. This function should use accumulative recursion.

Group Problem - Structural Recursion

Write a structurally recursive function called `member-count` that consumes a `(listof Sym)` and a `Sym`. It should produce the number of times that `Sym` appears in the `(listof Sym)`.

```scheme
;; (member-count losym item) Counts the number of times item appears in the losym
;; member-count: (listof Sym) Sym → Nat
;; Examples:
;; (check-expect (member-count empty 'help) 0)
;; (check-expect (member-count (list 'I 'knew 'you 'were 'trouble 'when 'you 'walked 'in 'trouble 'trouble 'trouble) 'trouble) 4)
```

Group Problem (Optional) - is-prefix-of?

Write a function, `is-prefix-of?`, that consumes a list of `Sym` called `pattern`, and another list of `Sym` called `word`. It produces `true` if the symbols in `pattern` occur in the same order at the start of `word`, and `false` otherwise.

```scheme
;; Tests
(check-expect (is-prefix-of? (list 'yellow 'green 'purple 'pineapple) 'help) 0)
```