Paul Burgwardt

NOTES [9/18 – 10/27]

SCHEME

Insert At Specific Location

(define (insertToSortedList element comparator lst)

(cond

[(or (empty? lst) (comparator element (first lst))) (cons element lst)]

[else (cons (first lst) (insertToSortedList element comparator (rest lst)))]))

(define (sort-demo comparator lst)

(cond

[(empty? lst) empty]

[else (insertToSortedList (first lst) comparator (sort-demo comparator (rest lst)))]))

USAGE: (sort-demo < (list 8 2 -7 6 33 2))

List Contains Element

(define (contains? item lst)

(cond

[(empty? lst) false]

[(cons? (first lst)) (or (contains? item (first lst)) (contains? item (rest lst)))]

[(= item (first lst)) true]

[else (contains? item (rest lst))]))

(contains? 3 (list 1 (list 2 8 ) (list 3)))

Filter

(define (isMember? element lst)

(cond

[(empty? lst) false]

[(= element (first lst)) true]

[else (isMember? element (rest lst))]))

(define (myFilter? element) (isMember? element (list 1 2 3 4)))

(define (filter-list lst)

(cond

[(empty? lst) empty]

[(myFilter? (first lst)) (filter-list (rest lst))]

[else (cons (first lst) (filter-list (rest lst)))]))

(filter-list (list 8 3 11 0 4 2 1))

Merge Sort

(define (merge-sort lstA lstB)

{cond

[(empty? lstA) lstB]

[(empty? lstB) lstA]

[(< (first lstA) (first lstB)) (cons (first lstA) (merge-sort (rest lstA) lstB))]

[else (cons (first lstB) (merge-sort lstA (rest lstB)))]})

(merge-sort (list 1 8 11) (list 2 12 13))

Find And Replace

(define (findIt listOfAtoms IndexToFind)

(cond

[(empty? listOfAtoms) empty]

[(= IndexToFind 1) (first listOfAtoms)]

[else (findIt (rest listOfAtoms) (sub1 IndexToFind))]))

(define (program listOfAtoms listOfIndex)

(cond

[(empty? listOfIndex) empty]

[else (cons (findIt listOfAtoms (first listOfIndex)) (program listOfAtoms (rest listOfIndex)))]))

(program '(a b c) '( 3 1 2)) // ‘(c a b)

Is A Number In List (#t,#f)

(define (num-in-list num lst)  
  (cond  
    [(empty? lst) false]  
    [(= num (first lst)) true]  
    [else (num-in-list num (rest lst))]))

Count List Matches

(define (list-matches lst1 lst2)

(cond

[(empty? lst1) 0]

[(num-in-list (first lst1) lst2) (+ 1 (list-matches (rest lst1) lst2))]

[else (list-matches (rest lst1) lst2)]))