

Lab: Merge Sort

Summary: In this laboratory, we consider merge sort, a more efficient technique for sorting lists of values.

Contents:

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Merging
 - Exercise 2: Reflecting on Merging
 - Exercise 3: Splitting
 - Exercise 4: Sorting
 - Exercise 5: Special Cases
 - Exercise 6: Is It Sorted?
- For Those with Extra Time
 - Extra 1: Splitting, Revisited

Exercises

Exercise 0: Preparation

a. Make a copy of mergesort.scm, my implementation of merge sort. Scan through the code and make sure that you understand all the procedures.

b. Start DrScheme.

Exercise 1: Merging

a. Write an expression to merge the lists (1 2 3) and (1 1.5 2.3).

b. Write an expression to merge two identical lists of numbers.

c. Write an expression to merge two lists of strings. (You may choose the strings yourself. Each list should have at least three elements.)

d. Assume that we represent names as lists of the form (*last-name first-name*). Write an expression to merge the following two lists

```
(define mathstats-faculty
  (list (list "Chamberland" "Marc")
        (list "French" "Chris")
        (list "Jager" "Leah")
        (list "Kornelson" "Keri")
        (list "Kuiper" "Shonda"))
```

```

(list "Moore" "Emily")
(list "Moore" "Tom")
(list "Mosley" "Holly")
(list "Romano" "David")
(list "Shuman" "Karen")
(list "Wolf" "Royce"))

(define more-faculty
  (list (list "Moore" "Ed")
        (list "Moore" "Jonathan")
        (list "Moore" "Peter")))

```

Exercise 2: Reflecting on Merging

- What will happen if you call `merge` with unsorted lists as the first two parameters?
- Verify your answer by experimentation.
- What will happen if you call `merge` with sorted lists of very different lengths as the first two parameters?
- Verify your answer by experimentation.

Exercise 3: Splitting

Use `split` to split:

- A list of numbers of length 6
- A list of numbers of length 5
- A list of strings of length 6

Exercise 4: Sorting

- Run `merge-sort` on a list you design of fifteen integers.
- Run `new-merge-sort` on a list you design of ten strings.
- Uncomment the lines in `new-merge-sort` that print out the current list of lists. Rerun `new-merge-sort` on a list you design of ten strings. Is the output what you expect?
- Rerun `new-merge-sort` on a list of twenty integers.

Exercise 5: Special Cases

- Run both versions of merge sort on the empty list.

- b. Run both versions of merge sort on a one-element list.
- c. Run both versions of merge sort on a list with duplicate elements.

Exercise 6: Is It Sorted?

As you've probably noticed, there are two key postconditions of a procedure that sorts lists: The result is a permutation of the original list and the result is sorted. We're fortunate that the unit test framework lets us test permutations (with `test-permutation!`). Hence, if we wanted to test merge sort in the unit test framework, we might write

```
(define some-list ...)
(test-permutation! (merge-sort some-list pred?) some-list)
```

However, we still need a way to make sure that the result is sorted, particularly if the result is very long.

Write a procedure, `(sorted? lst may-precede?)` that checks whether or not `lst` is sorted by `may-precede?`.

For example,

```
> (sorted? (list 1 3 5 7 9) <)
#t
> (sorted? (list 1 3 5 4 7 9) <)
#f
> (sorted? (list "alpha" "beta" "gamma") string<?)
```

Note that we can use that procedure in a test suite for merge sort with

```
(test! (sorted? (merge-sort some-list may-precede?) may-precede?) #t)

>
```

For Those with Extra Time

Extra 1: Splitting, Revisited

One of my colleagues prefers to define `split` something like the following

```
(define split
  (lambda (ls)
    (let kernel ((rest ls)
                  (left null)
                  (right null))
      (if (null? rest)
          (list left right)
          (kernel (cdr rest) (cons (car rest) right) left))))))
```

a. How does this procedure split the list?

b. Why might you prefer one version of split over the other?

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.