

Lab: Merge Sort

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Merging
 - Exercise 2: Reflecting on Merging
 - Exercise 3: Splitting
 - Exercise 4: Splitting, Revisited
 - Exercise 5: Sorting
 - Exercise 6: Special Cases
 - Exercise 7: Sorting Students
 - Exercise 8: Verifying Sorts
 - Exercise 9: Comparing Sorts

Exercises

Exercise 0: Preparation

- a. Make a copy of `mergesort.ss`, my implementation of merge sort. Scan through the code and make sure that you understand all the procedures.
- b. The `merge-sort` procedure has different parameters than the one given in the corresponding reading. *Make sure that you understand and can explain the difference.*
- c. 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 lists that contain the same values.
- c. Write an expression to merge two lists of words. (You may choose the words yourself. Each list should have at least three elements. You can represent names as strings.)
- 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 cs-faculty
  (list (list "Gum" "Ben")
        (list "Rebelsky" "Samuel")
        (list "Stone" "John")
        (list "Walker" "Henry")))

(define young-cs-kids
  (list (list "Rebelsky" "Jonathan")
        (list "Rebelsky" "William")))
```

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
- A length-4 list of lists (each sublist should have length 2 or more).

Exercise 4: 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)
          (values left right)
          (kernel (cdr rest) (cons (car rest) right) left))))))
```

- How does this procedure split the list?
- Why might you prefer one version of `split` over the other?

Exercise 5: Sorting

- a. Run merge sort on a list you design of fifteen integers.
- b. Run merge sort on a list you design of twenty strings.

Exercise 6: Special Cases

- a. Run merge sort on the empty list.
- b. Run merge sort on a one-element list.
- c. Run merge sort on a list with duplicate elements.

Exercise 7: Sorting Students

Assume that we represent students with a list of the form

```
(lastname firstname id major)
```

- a. Create a list of ten or more students.
- b. Write an expression to sort that list by first name.
- c. Write an expression to sort that list by id number.
- d. Write an expression to sort that list so that students are arranged alphabetically by major and then alphabetically by last name within each major.

Exercise 8: Verifying Sorts

- a. Write a procedure, `verify-sort`, that verifies the postconditions of `merge-sort`.
- b. Use that procedure to verify that `merge-sort` correctly sorts lists of 1000 "random" numbers.

Exercise 9: Comparing Sorts

- a. Using DrScheme's built-in timing mechanism (you may have to look through the online help to find information about that mechanism), make a table of the running time of insertion sort and merge sort on inputs of size 0, 1, 10, 100, 500, 1000, 2000, and 5000.
- b. Graph your data.
- c. Based on your data, what can you say about the relative speeds of the two sorting methods?