Merge sort

Goldsmiths Computing

Motivation

Merge sort: a straightforward, efficient sorting algorithm, with some additional useful properties:

- stability
- genericity (linked lists and vectors)

Definition

To sort a sequence using merge sort: sort two half-length subsequences, then combine the results.

Merge (vector)

```
Require: a,b :: Vector
  function MERGE(a,b)
      al ← LENGTH(a); bl ← LENGTH(b); cl ← al + bl
      c ← new Vector(cl)
      ai ← bi ← ci ← 0
      while ci < cl do
          if ai = al then
              c[ci] ← b[bi]; bi ← bi + 1
          else if bi = bl ∨ a[ai] ≤ b[bi] then
              c[ci] ← a[ai]; ai ← ai + 1
          else
              c[ci] ← b[bi]; bi ← bi + 1
          end if
          ci ← ci + 1
      end while
      return c
  end function
```

Merge (linked list)

```
Require: a,b :: Linked List
  function MERGE(a,b)
      if NULL?(a) then
          return b
      else if NULL?(b) then
          return a
      else if FIRST(a) ≤ FIRST(b) then
          return CONS(FIRST(a), MERGE(REST(a), b))
      else
          return CONS(FIRST(b), MERGE(a, REST(b)))
      end if
  end function
```

Mergesort

```
function MERGESORT(s)
    sl ← LENGTH(s)
    if sl ≤ 1 then
        return s
    else
        mid ← ⌊sl/2⌋
        left ← MERGESORT(s[0...mid))
        right ← MERGESORT(s[mid...sl))
        return MERGE(left,right)
    end if
end function
```

Complexity analysis

Time complexity: merge

- each iteration:
  - two compares
  - two memory read/writes
  - one addition
- exactly LENGTH(a) + LENGTH(b) iterations

$$\Longrightarrow \Theta(N_A + N_B)$$

Time complexity: mergesort

$$T(N) = 2 \times T\left(\frac{N}{2}\right) + \Theta(N)$$

$$\Longrightarrow \ldots?$$