

Branch: master ▾ [FP-150-Notebook / Lecture\\_7.md](#)

[Find file](#) [Copy path](#)

 SAMFYB make doctoc for lecture 7

4062060 3 days ago

1 contributor

156 lines (119 sloc) 5.61 KB

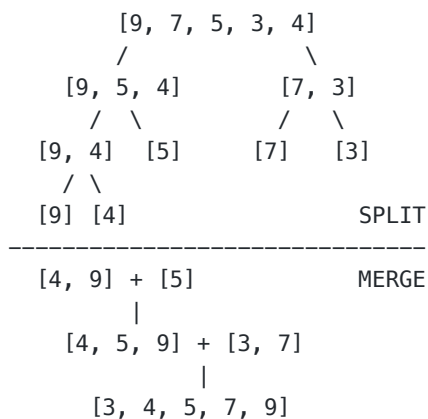
## Lecture 7 Sorting

- [Merge Sort Revisit](#)
- [Code on Merge Sort](#)
- [Analyzing the Work](#)
  - [The Work of the Helpers](#)
  - [The Work of msort](#)
- [The Concept of "Sorted" in Trees](#)
- [Inserting into a Tree](#)

### Merge Sort Revisit

Optimal complexity for sequential computation: **Merge Sort** (divide & conquer)

On parallel computation, merge sort speeds up a little on lists, and a lot on trees.



Note: It does not matter how to split as long as split evenly.

### Code on Merge Sort

```
(* msort : int list -> int list
 * REQ: true
```

```

* ENS: msort L is a sorted permutation of L
*)
(* Helpers
* split : int list -> int list * int list
* REQ: true
* ENS: split L ==> (A, B) such that (A @ B) is a permutation of L
*      such that |length A - length B| <= 1
* merge : int list * int list -> int list
* REQ: A & B are sorted
* ENS: merge (A, B) is a sorted permutation of (A @ B)
*)

fun msort ([] : int list) : int list = []
  | msort [x] = [x]
  | msort L =
    let
      val (A : int list, B : int list) = split L
    in
      merge (msort A, msort B)
    end

fun split ([] : int list) : int list * int list = ([], [])
  | split [x] = ([x], [])
  | split (x::y::xs) =
    let
      val (A, B) = split xs
    in
      (x::A, y::B)
    end

fun merge ([] : int list, B : int list) : int list = B
  | merge (A, []) = A
  | merge (x::xs, y::ys) =
    case Int.compare(x, y) of
      LESS => x::(merge (xs, y::ys))
    | GREATER => y::(merge (x::xs, ys))
    | EQUAL => x::y::(merge (xs, ys))

```

## Analyzing the Work

---

### The Work of the Helpers

```

fun split
  W(0) = c0
  W(1) = c1
  W(n) = c2 + W(n - 2)
        = c2 + c2 + W(n - 4)
        = c2 + c2 + c2 + W(n - 6)
        = c2 * (n / 2) + c1 or c0

```

Thus,  $W(n) \in O(n)$ .

Similarly,  $W_{merge} \in O(n)$  where  $n$  is the number of elements in the two lists.

Note: The **span** of the functions are also both  $O(n)$ . This is an intrinsic problem for lists. You have to walk through every elements of the list anyway.

## The Work of `msort`

$$W_{msort}(0) = c_0$$

$$W_{msort}(1) = c_1$$

$$W_{msort}(n) = c_3 + W_{msort}(n_1) + W_{msort}(n_2) + W_{split}(n) + W_{merge}(n)$$

Further,  $W_{split}(n) + W_{merge}(n)$  is linear, and  $n_1, n_2 \approx \frac{n}{2}$ .

Therefore,

$$W_{msort}(n) = c_3 + 2 \cdot W_{msort}(n/2) + c_4 \cdot (n)$$

We can consider the work with a tree analysis:

$$\begin{array}{c}
 c_4 n \\
 / \quad \backslash \\
 c_4(n/2) \quad c_4(n/2) \\
 / \quad \backslash \quad / \quad \backslash \\
 c_4(n/4) \quad c_4(n/4) \quad \dots
 \end{array}$$

At each level, the total work is exactly  $c_4 n$ , and there is  $\log n$  levels.

Therefore, the total amount of work is  $n \log n$ .

The **span** is the sum of the longest path along this tree, i.e.

$$c_4 \left( n + \frac{n}{2} + \dots + \frac{n}{2^d} \right) \approx 2c_4 n \in O(n)$$

## The Concept of "Sorted" in Trees

- Empty is sorted
- Node(L, x, R) is sorted iff
  - L is sorted &  $y \leq x$  for all  $y$  in L
  - R is sorted &  $z \geq x$  for all  $z$  in R

## Inserting into a Tree

```

(* Ins : int * tree -> tree
 * REQ: T is sorted
 * ENS: Ins (x, T) is a sorted tree consisting the elements of T & x
 *)
fun Ins (x : int, Empty : tree) : tree = Node(Empty, x, Empty)
| Ins (x, Node(L, y, R)) =
  case Int.compare(x, y) of
    GREATER => Node(L, y, Ins(x, R))
  | _ => Node(Ins(x, L), y, R)

```

