

**Reading:** GT 4.1.1 fragment, 4.1.2, pp. 223–224 [up to 15 minutes]

The book gives two different complexity analyses for MERGE-SORT. You should understand both. In the lecture we will cover the more difficult 4.1.2.

**Ingredients:** Space usage of MERGE-SORT, running time of MERGE-SORT, practitioner's corner

---

## Space Usage

**Question.** How much additional memory MERGE-SORT uses?

**Question.** Is this memory requirement bigger or smaller than INSERTION-SORT or SELECTION-SORT?

## Running Time

**Question.** Does the running time of MERGE-SORT depend on the kind of input? What is the perverse input? The ideal input?

Let us now try to analyze the running time.

The number of comparisons when sorting  $n$  elements, denoted  $t(n)$ , is roughly

$$\begin{cases} t(n) = 2t\left(\frac{n}{2}\right) + cn \\ t(1) = b \end{cases} \quad (1)$$

Such a recursive system is often called a *recurrence equation* or a *recurrence relation*. We would like to find a *closed form* solution for this system, i.e. one that would not involve  $t$  in the right hand side.

For simplicity assume that  $n$  is a power of 2. Let us expand the inductive part several times:

$$\begin{aligned} t(n) &= 2t\left(\frac{n}{2}\right) + cn = 2\left(2t\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn = 2^2t\left(\frac{n}{2^2}\right) + 2cn = \\ &= 2^2\left(2t\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right) + 2cn = 2^3t\left(\frac{n}{2^3}\right) + 3cn = \dots = 2^i t\left(\frac{n}{2^i}\right) + icn \quad (2) \end{aligned}$$

**Q.** How many times can we continue expanding? How large  $i$  can be?

Remember that we switch to the basic case when  $n = 1$  (and  $t(1) = b$ ), where  $b > 0$  is a constant. This will occur if  $\frac{n}{2^i} = 1$ , so when  $n = 2^i$ , so  $i = \log n$ . So

$$\begin{aligned} t(n) &= 2^{\log n} t\left(\frac{n}{2^{\log n}}\right) + (\log n) cn = nt\left(\frac{n}{n}\right) + cn \log n = \\ &= nt(1) + cn \log n = bn + cn \log n = O(n \log n). \quad (3) \end{aligned}$$

Since we have agreed that the running time of MERGE-SORT is independent of the input, we can actually conclude that MERGE-SORT is  $\Theta(n \log n)$ .  $\square$

**An Important General Rule:** An algorithm that divides the problem in half using constant time and then recursively solves both halves separately spending linear time to merge the results always runs in time proportional to  $n \log n$ .

## Practitioner's Corner

MERGE-SORT accesses elements in order, which implies

- that it can be used to sort files off-memory (on disk, flash memory, etc)
- that it can be used to sort linked lists (no other navigation than sequential is possible in linked lists).
- that it interacts well with modern memory architecture (multilevel caches)

MERGE-SORT is easy to implement.

MERGE-SORT is slightly slower than QUICK-SORT on average and uses more memory. However if degenerate cases for QUICK-SORT are likely to occur, or if stability is required then MERGE-SORT is the algorithm of choice

INSERTION-SORT is faster on presorted sequences, uses less memory and is also stable.

According to the Wikipedia article on MERGE-SORT (last seen on September 14, 2006) this is the default sorting algorithm in Perl since version 5.8 (before then QUICK-SORT was used).

---

The standard Java library uses MERGE-SORT to sort (`Arrays.sort`) arrays of objects. Arrays of simple types are sorted with QUICK-SORT. See for example

<http://java.sun.com/j2se/1.5.0/docs/api/java/util/Array.html>.