CIS 575. Introduction to Algorithm Analysis
Material for February 23, 2024

Merge Sort and Quicksort

The first topic of this note, *Merge Sort*, is covered in *Cormen*'s Section 2.3.
The second topic of this note, *Quicksort*, is covered in *Cormen*'s Section 7.1.

1 Divide & Conquer for Merge Sort

We have already seen (when motivating recurrences) one instance of the Divide & Conquer paradigm: **Merge Sort** whose workings we illustrated by an example:

$$| 18 | 14 | 12 | 27 | 20 | 28 | 10 | 11 |$$

1. We first decompose the problem, which just amounts to clarifying the boundary between the two partitions:

$$| 18 | 14 | 12 | 27 \,\|\| 20 | 28 | 10 | 11 |$$

2. We then recursively sort the two halves:

$$| 12 | 14 | 18 | 27 \,\|\| 10 | 11 | 20 | 28 |$$

3. Finally, we merge the two halves (into a new array):

$$| 10 | 11 | 12 | 14 | 18 | 20 | 27 | 28 |$$

Recall that the running time is described by the recurrence

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

which has solution $T(n) \in \Theta(n \lg(n))$.

2 Divide & Conquer for Quicksort

The *Quicksort* algorithm, discovered in 1959 by the British computer scientist Tony Hoare (now 90 years old), is another application of the Divide & Conquer paradigm. Whereas Merge Sort does non-trivial work only *after* having solved the subproblems, Quicksort does non-trivial work only *before* having solved the subproblems. To illustrate the workings of Quicksort, we shall again use the example input

$$| 18 | 14 | 12 | 27 | 20 | 28 | 10 | 11 |$$

1. The first step is to pick some *pivot*. How we do that is discussed in the next notes, but for now let us assume that we picked 20 as the pivot. We then *partition* the array so that all records with key less than 20 are left of the record(s) with key 20 which in turn is left of all records with key greater than 20. This may *for example* give the array

   $$\mid 18 \mid 14 \mid 12 \mid 11 \mid 10 \parallel 20 \parallel 28 \mid 27 \mid$$

   but it could also, depending on the algorithm we use for partitioning (discussed in Section 2.1) give

   $$\mid 14 \mid 10 \mid 12 \mid 11 \mid 18 \parallel 20 \parallel 27 \mid 28 \mid$$

   as well as numerous other possibilities.

2. We then recursively sort the leftmost and rightmost partition; this yields (no matter the partition)

   $$\mid 10 \mid 11 \mid 12 \mid 14 \mid 18 \parallel 20 \parallel 27 \mid 28 \mid$$

3. We are now done, and have a sorted array:

   $$\mid 10 \mid 11 \mid 12 \mid 14 \mid 18 \mid 20 \mid 27 \mid 28 \mid$$

2.1 Partitioning

To partition, we may use the Dutch National Flag algorithm:

- the **red** records are those with key less than the pivot
- the **white** records are those with key equal the pivot
- the **blue** records are those with key greater than the pivot.

Other algorithms can be used, for example the one given in Section 7.1 of *Cormen*, but observe that if keys may be duplicate it is not as powerful as the Dutch National Flag algorithm: if we start with say

$$\mid 10 \mid 15 \mid 12 \mid 14 \mid 18 \mid 15 \mid 19 \mid 15 \mid$$

and pick 15 as our pivot then

- the Dutch National Flag algorithm will yield something of the form (you may try to simulate it and see the actual output)

  $$\mid \ldots \mid \ldots \mid \ldots \parallel 15 \mid 15 \mid 15 \parallel \ldots \mid \ldots \mid$$

  so that the recursive calls will be made on arrays of size **3** and 2
- but the algorithm from *Cormen*'s Section 7.1 will not distinguish between keys less than 15 and keys equal 15, and will yield something of the form

  $$\mid \ldots \mid \ldots \mid \ldots \mid \ldots \mid \ldots \parallel 15 \parallel \ldots \mid \ldots \mid$$

  so that the recursive calls will be made on arrays of size **5** and 2.