# Algorithms & Data Structures

## Dr Michele Sevegnani

*Joao Almeida-Domingues*[*]

*University of Glasgow*

*January 14th, 2020 – March 25th, 2020*

## Contents



These lecture notes were collated by me from a mixture of sources , the two main sources being the lecture notes provided by the lecturer and the content presented in-lecture. All other referenced material (if used) can be found in the *Bibliography* and *References* sections.

The primary goal of these notes is to function as a succinct but comprehensive revision aid, hence if you came by them via a search engine , please note that they're not intended to be a reflection of the quality of the materials referenced or the content lectured.

Lastly, with regards to formatting, the pdf doc was typeset in LaTeX, using a modified version of Stefano Maggiolo's class

[*]2334590D@student.gla.ac.uk

# 1 Efficient Sorting

## 1.1 *Merge Algorithm*

**1.1 definition. Sentinel** a number which is much larger than any other number in the array. It can be represented by `Integer.MAX_VALUE`

**1.2 remark.** A more general definition sees sentinel as any value which is interpreted as a condition to terminate the algorithm

**1.3 definition. Stable** an algorithm which preserves the input order of repeated elements

**1.4 remark.** note that this is only relevant if the repeated elements are somehow distinguishable (e.g deck of cards , (rank,value))

**1.5 definition. Key** for data being recorded as a tuple of values, the key is the value used to sort (e.g (name, surname) , K=S)

**1.6 definition. Merging Algorithm** an algorithm which takes two sorted sequences and returns a single combined sequence [4]

**Overview**

The merge algorithm takes an unsorted array and 3 indices, it then splits the array into two subarrays and copies each subarray into memory. Finally, it iterates over every element of the original array, for each iteration it compares the current elements of the 2 subarrays against each other, replacing the appropriate value into the original array and increasing the index of the subarray by 1

**Formal Definition**

**Algorithm 1:** Merge

```
   Data: Array A , indeces p, q, r
   Result: sorted subarray A[p..r]
 1 Merge(A,p,q,r)
 2 n_1 := q - p + 1                    /* initialize subarrays' size */
 3
 4 n_2 := r - q
 5 L[o..n_1] := A[p..q]                          /* copy to subarrays */
 6
 7 R[n_2..r] := A[q+1..r]
 8 L[n_1] := ∞   /* add the sentinel values to end of subarrays */
 9
10 R[n_2] := ∞
11 i,j := o
12 for k=p to r do
13 |
14 |  if L[i] ≤ R[j] then
15 |  |
16 |  |  A[k] := L[i]
17 |  |  i := i + 1
18 |  else
19 |  |
20 |  |  A[k] := R[j]
21 |  |  j := j + 1
22 end
```

**Properties**

- **Running Time :** $O(n)$ , since the initialization of the $L, R$ subarrays takes $O(n)$ and the loop is executed $n$ times and contains only constant-time operations
- **Stable**
- **Memory Requirements :** $O(n)$ to store $L, R$

## 1.2 *Merge-Sort*

**1.7 remark.** See informal definition in previous section

### Formal Definition

**Algorithm 2:** Merge-Sort Algorithm

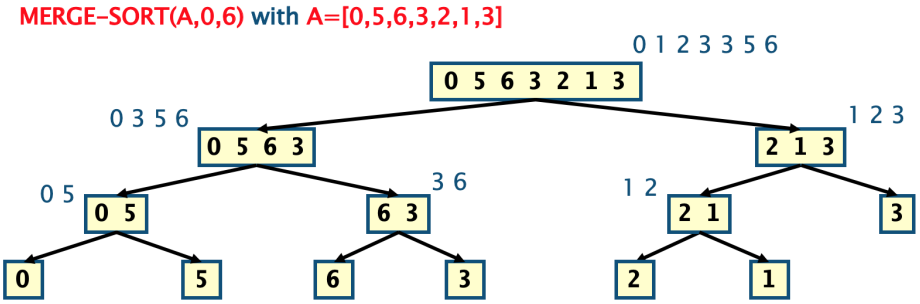
**Data:** Array $A$ , indeces $p, r$
**Result:** sorted array $A[p..r]$

```
1 Merge-Sort(A,p,r)
2 if p ¡ r then
3 |   q := (p + r) / 2
4 |   Merge-Sort (A,p,q)
5 |   Merge-Sort(A,q+1,r)
6 |   Merge-Sort 2A,p,q,r
```

### Execution & Properties

**1.8 remark.** to sort an array $A$ with $n$ elements, we would call `MERGE-SORT(A,0,n-1)`

Note how the algorithm gives rise to a binary recursive tree, since we split each subarray into 2 with each iteration until we hit the stopping condition (single element array). So, the first subarray is entirely sorted, before the second `MERGE-SORT` call. The two *half-subarrays* returned from each call are then merged by calling `MERGE`. The process is then repeated in the right branch




- **Stable**
- **Memory** $O(n)$
- **Running Time** $O(n \log n)$

## 1.3 *Recurrence Equations*

**1.9 definition. Recurrence Equation** describe the overall running time of a problem of size $n$ in terms of the running time on smaller inputs

Recursive algorithms can be described via *recurrence equations*. Take $t(n)$ to denote the *worst-case* running time of `MERGE-SORT(n)`, then we can characterize $t(n)$ by means of an equation where $t(n)$ is recursively expressed in terms of itself.

If `MERGE-SORT(n)` has running time $T(n)$ then, each recursive call will `MERGE-SORT(n/2)` will run on $T(n/2)$ time. Hence, we can define $T(n)$ recursively as follows

$$T(n) = \begin{cases} b & n \leq 1 \quad \text{(base case)} \\ 2T(n/2) + cn \end{cases}$$

**Iterative Method**

Though correct, a more informative definition will be one that does not involve $T(n)$ itself. From its *closed-form* characterization, we can define it in *big-Oh* terms. We do this, by iterative substitution on the RHS of the recurrence relation until the base case is reached

Continuing with the expression above, if we substitute $n$ by $n/2$ in the RHS we essentially double the children arrays, and we get

$$\begin{aligned} T(n) &= 2(2T((n/2)/2) + c(n/2)) + cn \\ &= 2^2 t(n/2^2) + 2(cn/2) + cn \\ &= 2^2 t(n/2^2) + 2cn \end{aligned}$$

If we repeat the process, assuming that $n$ is relatively large, then we find the general pattern

$$t(n) = 2^i t(n/2^i) + icn$$

In order to determine the base case, we look at the original definition, and observe that we need to find $t(n)$ , for $n \leq 1$. Given our general argument $\frac{n}{2^i}$ we need $2^i = n \iff i = \log n$. Hence, substituting $i$ by $\log n$ gives

$$\begin{aligned} t(n) &= 2^{\log n} t(n/2^{\log n}) + cn \log n \\ &= nt(1) + cn \log n \\ &= nb + cn \log n \end{aligned}$$

Lastly, since $b, c$ are constants, we have shown that $t(n)$ is $O(n \log n)$

**Master Method**

There is a general form of recurrence relation that arises in the analysis of the divide-and-conquer algorithms

$$T(n) = aT(n/b) + f(n) \qquad a, b \geq 1$$

Then, for $f(n) = \Omega(n^c)$ , the solution will be one of the following 3 cases

1. $\mathrm{c} < \log_{\mathrm{b}} \mathrm{a}$ then $\mathrm{T(n)} = \Theta\left(n^{\log_b a}\right)$
2. $\mathrm{c} = \log_{\mathrm{b}} \mathrm{a}$ then $\mathrm{T(n)} = \Theta(\mathrm{n} \leq \log \mathrm{n})$

3. c $>$ $\log_{b2}$ a then T(n) $= \Theta$(f(n))

For MERGE-SORT, we have $a = 2, b = 2, f(n) = \Omega(n), c = 1$ . Hence, the solution is given by case 2 , $\Omega(n^c \log n) = \Omega(n \log n)$

**Tree Method**

**1.10 remark.** See lecture 5 slides 52-60

## References

**CS2 Software Design & Data Structures** **calculating˙program˙running˙time**

*CS2 Software Design & Data Structures*. URL: https://opendsa-server.cs.vt.edu/ODSA/Books/CS2/html/AnalProgram.html.

**Goodrich et al.: Data Structures and Algorithms in Java, 6th Edition** **goodrich˙tamassia˙2014**

Michael Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java, 6th Edition*. John Wiley & Sons, 2014.

**Hochstein: What is tail recursion?** **hochsteinSO**

Lorin HochsteinLorin Hochstein. *What is tail recursion?* Aug. 1958. URL: https://stackoverflow.com/questions/33923/what-is-tail-recursion.

**Merging Algorithms** **merge**

*Merging Algorithms*. URL: http://www.cs.rpi.edu/~musser/gp/algorithm-concepts/merge-algorithms-screen.pdf.