

# Mergesort

## COMS10018 - Algorithms

Dr Christian Konrad

# Definition of the Sorting Problem

## Sorting Problem

- **Input:** An array  $A$  of  $n$  numbers
- **Output:** A reordering of  $A$  s.t.  $A[0] \leq A[1] \leq \dots \leq A[n-1]$

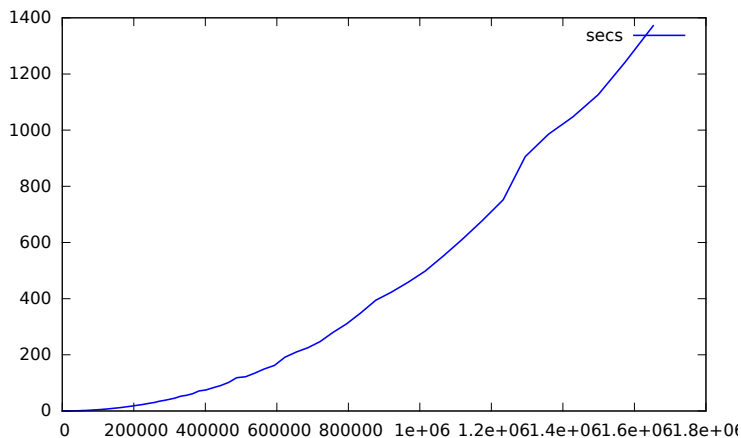
## Why is it important?

- Practical relevance: Appears almost everywhere
- Fundamental algorithmic problem, rich set of techniques
- There is a non-trivial lower bound for sorting (rare!)

## Insertion Sort

- Worst-case runtime  $O(n^2)$
- Surely we can do better?!

# Insertion sort in Practice on Worst-case Instances



$n$	46929	102428	364178	1014570
secs	1.03084	4.81622	61.2737	497.879

# Properties of a Sorting Algorithm

## **Definition** (in place)

A sorting algorithm is *in place* if at any moment at most  $O(1)$  array elements are stored outside the array

$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$	$a_9$	$a_{10}$
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------

--	--

 $O(1)$

**Example:** Insertion-sort is in place

## **Definition** (stability)

A sorting algorithm is *stable* if any pair of equal numbers in the input array appear in the same order in the sorted array

**Example:** Insertion-sort is stable

## Sorting Complex Data

- In reality, data that is to be sorted is rarely entirely numerical (e.g. sort people in a database according to their last name)
- A data item is often also called a **record**
- The **key** is the part of the record according to which the data is to be sorted
- Data different to the key is also referred to as **satellite data**

family name	first name	data of birth	role
<b>Smith</b>	Peter	02.10.1982	lecturer
<b>Hills</b>	Emma	05.05.1975	reader
<b>Jones</b>	Tom	03.02.1977	senior lecturer
...			

**Observe:** Stability makes more sense when sorting complex data as opposed to numbers

## Key Idea:

- Suppose that left half and right half of array is sorted
- Then we can merge the two sorted halves to a sorted array in  $O(n)$  time:

## Merge Operation

- Copy left half of  $A$  to new array  $B$
- Copy right half of  $A$  to new array  $C$
- Traverse  $B$  and  $C$  simultaneously from left to right and write the smallest element at the current positions to  $A$

## Example: Merge Operation

*A*

1	3	4	5	7	9	10	11
---	---	---	---	---	---	----	----

*B*

1	4	9	10
---	---	---	----

*C*

3	5	7	11
---	---	---	----

# Analysis: Merge Operation

## Merge Operation

- **Input:** An array  $A$  of integers of length  $n$  ( $n$  even) such that  $A[0, \frac{n}{2} - 1]$  and  $A[\frac{n}{2}, n - 1]$  are sorted
- **Output:** Sorted array  $A$

## Runtime Analysis:

- ① Copy left half of  $A$  to  $B$ :  $O(n)$  operations
- ② Copy right half of  $A$  to  $C$ :  $O(n)$  operations
- ③ Merge  $B$  and  $C$  and store in  $A$ :  $O(n)$  operations

**Overall:**  $O(n)$  time in worst case

**How can we establish that left and right halves are sorted?**

Divide and Conquer!



# Merge Sort: A Divide and Conquer Algorithm

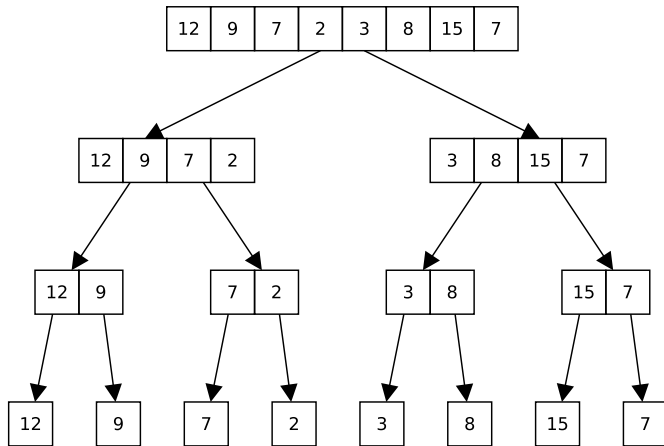
```
Require: Array  $A$  of  $n$  numbers  
if  $n = 1$  then  
    return  $A$   
 $A[0, \lfloor \frac{n}{2} \rfloor] \leftarrow \text{MERGESORT}(A[0, \lfloor \frac{n}{2} \rfloor])$   
 $A[\lfloor \frac{n}{2} \rfloor + 1, n-1] \leftarrow \text{MERGESORT}(A[\lfloor \frac{n}{2} \rfloor + 1, n-1])$   
 $A \leftarrow \text{MERGE}(A)$   
return  $A$ 
```

MERGESORT

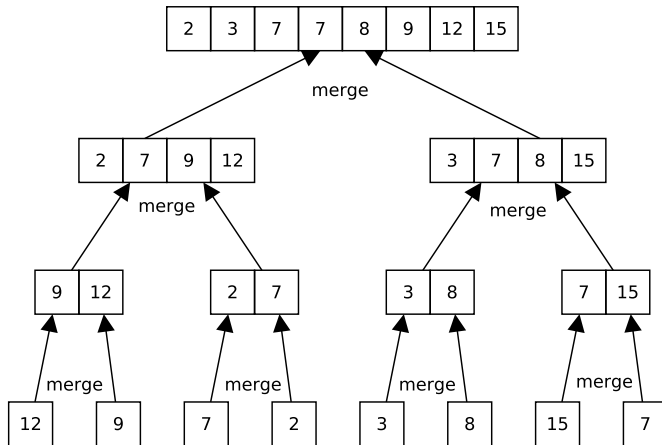
## Structure of a Divide and Conquer Algorithm

- **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
- **Conquer** the subproblems by solving them recursively. If the subproblems are small enough, just solve them in a straightforward manner.
- **Combine** the solutions to the subproblems into the solution for the original problem.

# Analyzing MergeSort: An Example



# Analyzing MergeSort: An Example



## Analysis Idea:

- We need to sum up the work spent in each node of the *recursion tree*
- The recursion tree in the example is a *complete binary tree*

**Definition:** A tree is a *complete binary tree* if every node has either 2 or 0 children.

**Definition:** A tree is a *binary tree* if every node has at most 2 children.

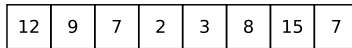
*(we will talk about trees in much more detail later in this unit)*

## Questions:

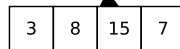
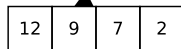
- How many levels?
- How many nodes per level?
- Time spent per node?

# Number of Levels

**Level 1**



**Level 2**



**Level 3**



**Level 4**



# Number of Levels (2)

## Level $i$ :

- $2^{i-1}$  nodes (at most)
- Array length in level  $i$  is  $\lceil \frac{n}{2^{i-1}} \rceil$  (at most)
- Runtime of merge operation for each node in level  $i$ :  $O(\frac{n}{2^{i-1}})$

## Number of Levels:

- Array length in last level  $l$  is 1:  $\lceil \frac{n}{2^{l-1}} \rceil = 1$

$$\frac{n}{2^{l-1}} \leq 1 \Rightarrow n \leq 2^{l-1} \Rightarrow \log(n) + 1 \leq l$$

- Array length in last but one level  $l - 1$  is 2:  $\lceil \frac{n}{2^{l-2}} \rceil = 2$

$$\frac{n}{2^{l-2}} > 1 \Rightarrow n > 2^{l-2} \Rightarrow \log(n) + 2 > l$$

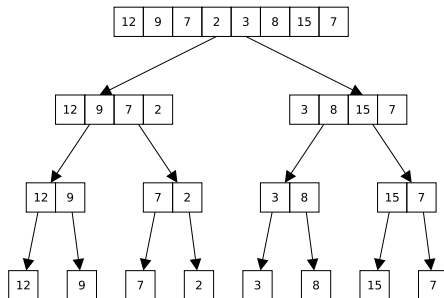
$$\log(n) + 1 \leq l < \log(n) + 2$$

Hence,  $l = \lceil \log n \rceil + 1$  .

# Runtime of Merge Sort

## Sum up Work:

- Levels:  
 $l = \lceil \log n \rceil + 1$
- Nodes on level  $i$ :  
at most  $2^{i-1}$
- Array length in level  $i$ :  
at most  $\lceil \frac{n}{2^{i-1}} \rceil$



## Worst-case Runtime:

$$\begin{aligned} \sum_{i=1}^{\lceil \log n \rceil + 1} 2^{i-1} O\left(\left\lceil \frac{n}{2^{i-1}} \right\rceil\right) &= \sum_{i=1}^{\lceil \log n \rceil + 1} 2^{i-1} O\left(\frac{n}{2^{i-1}}\right) \\ &= \sum_{i=1}^{\lceil \log n \rceil + 1} O(n) = (\lceil \log n \rceil + 1) O(n) = O(n \log n) . \end{aligned}$$

# Stability and In Place Property?

## Stability and In Place Property?

- Merge sort is stable
- Merge sort does not sort in place



## Divide and Conquer Algorithm:

Let **A** be a divide and conquer algorithm with the following properties:

- 1 **A** performs two recursive calls on input sizes at most  $n/2$
- 2 The conquer operation in **A** takes  $O(n)$  time

Then:

**A** has a runtime of  $O(n \log n)$  .