

CH-231-A

**Algorithms and Data Structures**

ADS

**Lecture 8**

Dr. Kinga Lipskoch

Spring 2022

# Divide & Conquer

- ▶ Divide & Conquer (Latin Divide et Impera) is one concept/method/technique that can produce faster algorithms.
- ▶ It is based on three steps:
  - ▶ **Divide** the given problem into smaller subproblems.
  - ▶ **Conquer** the subproblems by solving them recursively.
  - ▶ **Combine** the solutions of the subproblems.
- ▶ **Example:** Sort recursively?

# Merge Sort Idea

**MERGE-SORT**  $A[1 \dots n]$

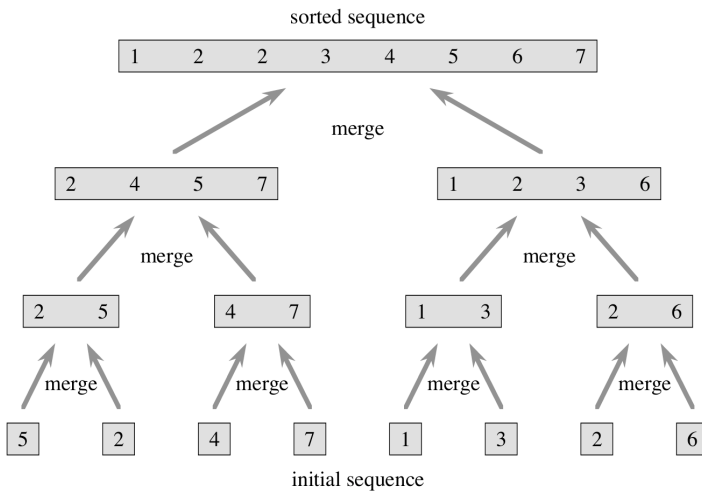
1. If  $n = 1$ , done.
2. Recursively sort  $A[1 \dots \lceil n/2 \rceil]$   
and  $A[\lceil n/2 \rceil + 1 \dots n]$ .
3. “*Merge*” the 2 sorted lists.

## Merge Sort as Divide & Conquer

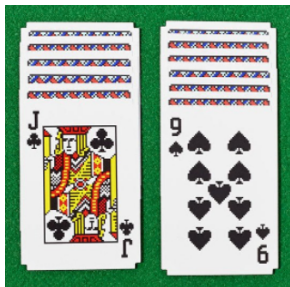
```
MERGE-SORT( $A, p, r$ )  
  if  $p < r$                                 // check for base case  
     $q = \lfloor (p + r)/2 \rfloor$                     // divide  
    MERGE-SORT( $A, p, q$ )                      // conquer  
    MERGE-SORT( $A, q + 1, r$ )                  // conquer  
    MERGE( $A, p, q, r$ )                       // combine
```

Initial call: Merge-Sort( $A, 1, A.length$ )

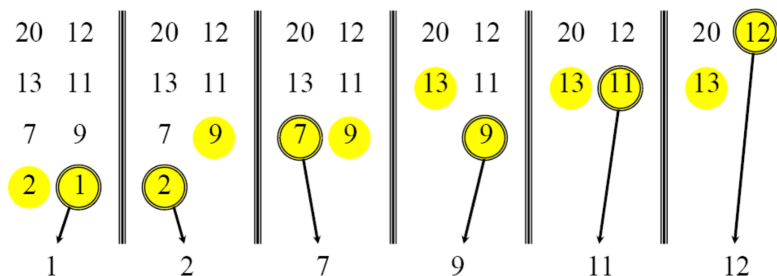
## Merge Sort: Example



## How Is Merging Done? (1)



## How Is Merging Done? (2)



# Merge

```
MERGE( $A, p, q, r$ )  
   $n_1 = q - p + 1$   
   $n_2 = r - q$   
  let  $L[1 \dots n_1 + 1]$  and  $R[1 \dots n_2 + 1]$  be new arrays  
  for  $i = 1$  to  $n_1$   
     $L[i] = A[p + i - 1]$   
  for  $j = 1$  to  $n_2$   
     $R[j] = A[q + j]$   
   $L[n_1 + 1] = \infty$   
   $R[n_2 + 1] = \infty$   
   $i = 1$   
   $j = 1$   
  for  $k = p$  to  $r$   
    if  $L[i] \leq R[j]$   
       $A[k] = L[i]$   
       $i = i + 1$   
    else  $A[k] = R[j]$   
       $j = j + 1$ 
```



## Correctness of Merging

Loop Invariant:

- ▶ At the start of each iteration of the for  $k$  loop, the subarray  $A[p..k-1]$  contains the  $k-p$  smallest elements of  $L$  and  $R$  in sorted order.
- ▶ Moreover,  $L[i]$  and  $R[j]$  are the smallest elements of their arrays which have not been copied back into  $A$ .

# Asymptotic Analysis (1)

- ▶ Merging step?  
Computation time is  $\Theta(n)$ .
- ▶ What is the overall computation time of Merge Sort?

## Asymptotic Analysis (2)

$T(n)$	<b>MERGE-SORT</b> $A[1 \dots n]$
$\Theta(1)$	1. If $n = 1$ , done.
$2T(n/2)$	2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$ .
$\Theta(n)$	3. <b><i>Merge</i></b> the 2 sorted lists

## Asymptotic Analysis (3)

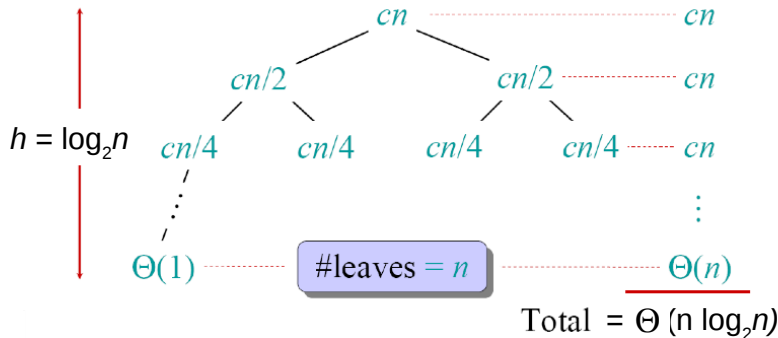
- ▶ The overall running time for Merge Sort is given by the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

- ▶ The base case may be omitted, if it is obvious that it can be neglected in the asymptotic analysis.

# Recursion Tree

Solve  $T(n) = 2T(n/2) + cn$ , where  $c > 0$  is constant.



## Merge Sort vs. Insertion Sort

- ▶  $\Theta(n \log_2 n)$  grows more slowly than  $\Theta(n^2)$ .
- ▶ Merge Sort asymptotically beats Insertion Sort (in the average case and in the worst case).
- ▶ In practice, Merge Sort beats Insertion Sort for  $n > 30$  or so.

# Sorting Algorithms' Complexities

Algorithm	Data structure	Time complexity:Best	Time complexity:Average	Time complexity:Worst	Space complexity:Worst
Quick sort	Array	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(n)$
Merge sort	Array	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
Heap sort	Array	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
Smooth sort	Array	$O(n)$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble sort	Array	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	Array	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	Array	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$

Here the notation  $\log(n)$  is mathematically  $\log_2 n$ .