COMP250: More Recursion examples. Merge sort.

Jérôme Waldispühl
School of Computer Science
McGill University

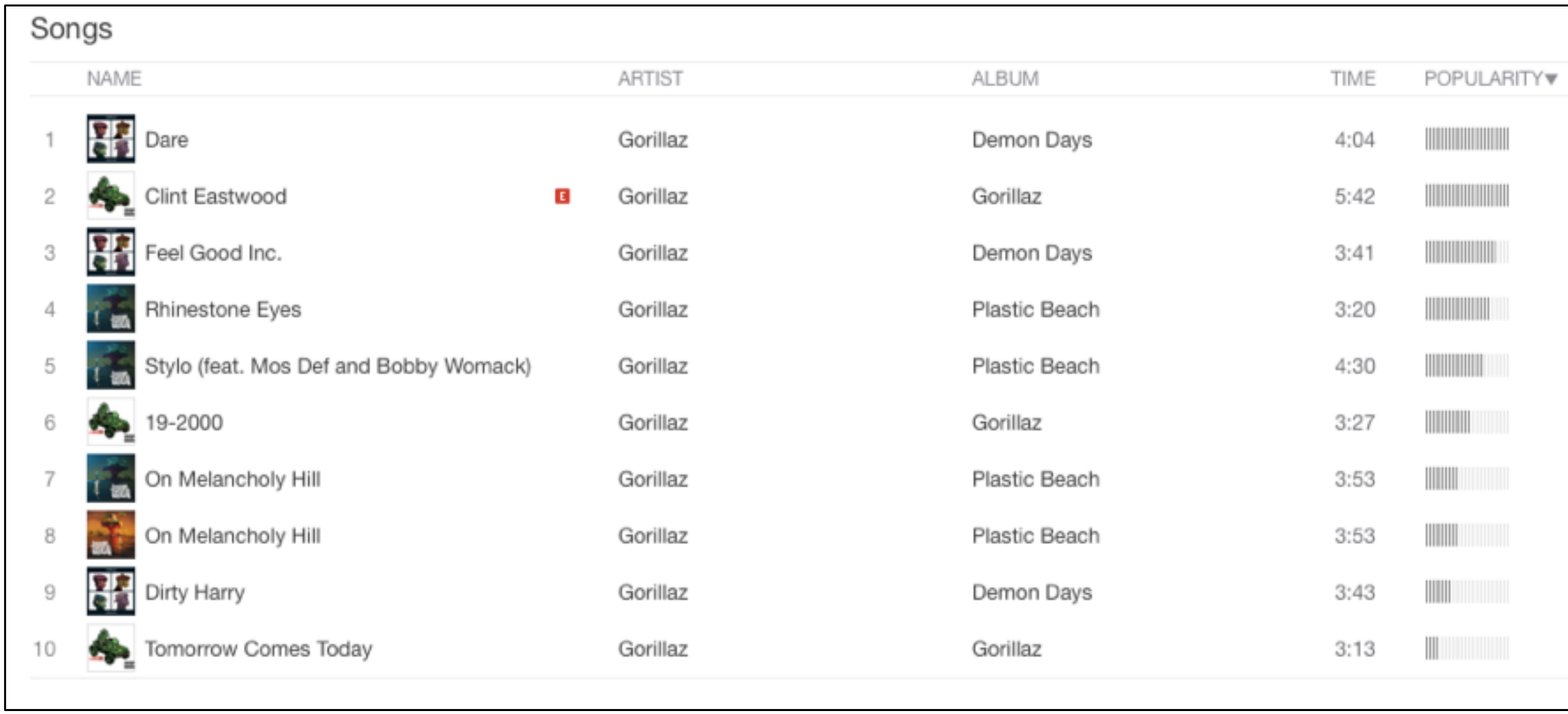
Based on slides from (Snoeyink,2004)

Designing recursive algorithms

- To write a recursive algorithm:
  - Find how the problem can be broken up in one or more smaller problems of the same nature
  - Remember the base case!
- Usually, better running times are obtained when the size of the sub-problems are approximately equal:
  - power(a,n) = a * power(a,n-1) $\Rightarrow$ O(n) operations
  - power(a,n) = (power(a,n/2))^2 $\Rightarrow$ O(log n) operations
  - Naïve Fibonacci $\Rightarrow$ O(ϕ^n) operations
  - Better Fibonacci $\Rightarrow$ O(log n) operations

Sorting problem

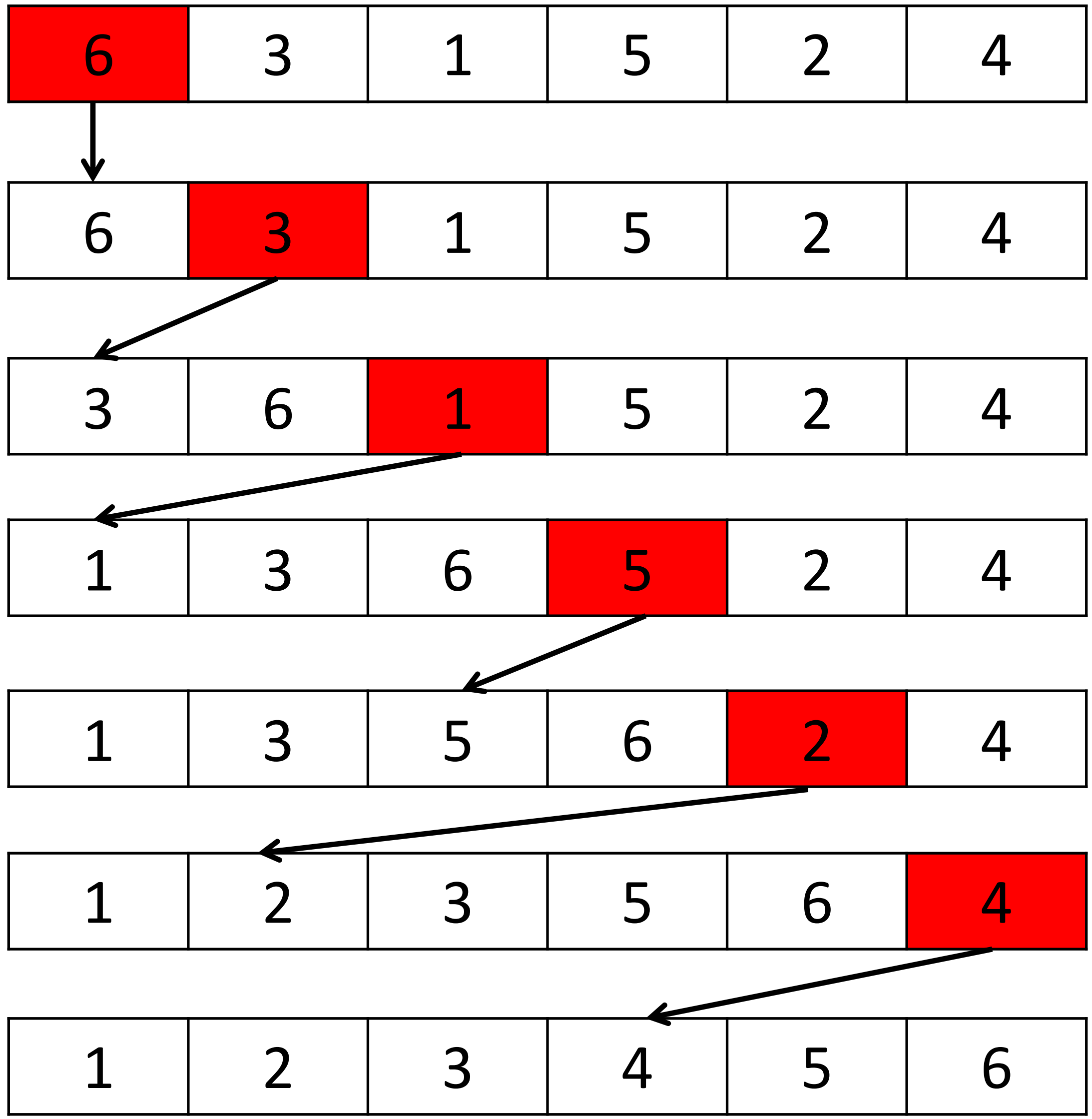
Problem: Given a list of *n* elements from a totally ordered universe, rearrange them in ascending order.


Songs

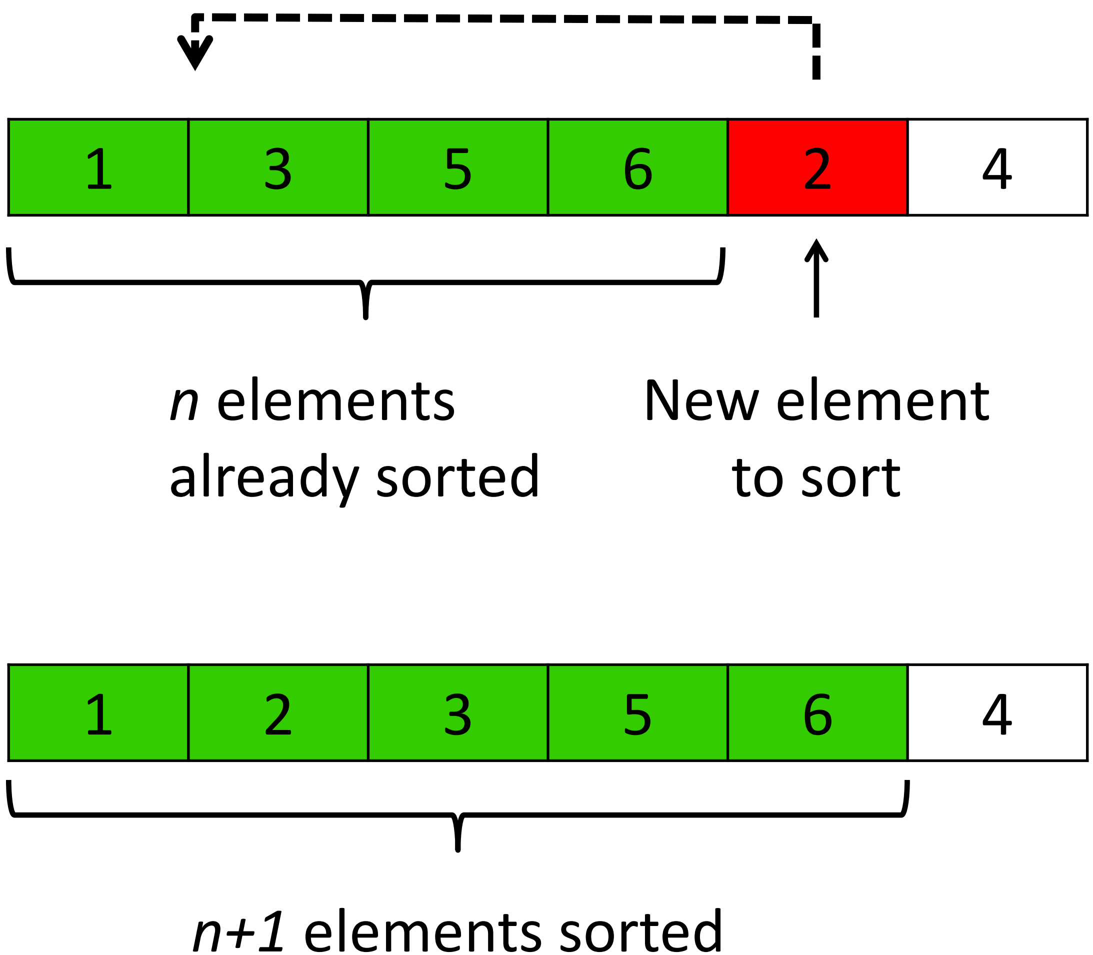
	NAME	ARTIST	ALBUM	TIME	POPULARITY
1	Dare	Gorillaz	Demon Days	4:04	
2	Clint Eastwood	Gorillaz	Gorillaz	5:42	
3	Feel Good Inc.	Gorillaz	Demon Days	3:41	
4	Rhinestone Eyes	Gorillaz	Plastic Beach	3:20	
5	Stylo (feat. Mos Def and Bobby Womack)	Gorillaz	Plastic Beach	4:30	
6	19-2000	Gorillaz	Gorillaz	3:27	
7	On Melancholy Hill	Gorillaz	Plastic Beach	3:53	
8	On Melancholy Hill	Gorillaz	Plastic Beach	3:53	
9	Dirty Harry	Gorillaz	Demon Days	3:43	
10	Tomorrow Comes Today	Gorillaz	Gorillaz	3:13	

Classical problem in computer science with many different algorithms (bubble sort, merge sort, quick sort, etc.)

Insertion sort



Insertion sort



Insertion sort

```
For i ← 1 to length(A) - 1
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
end for
```

- Iterative method to sort objects.
- Relatively slow, we can do better using a recursive approach!

Divide and Conquer

Recursive in structure

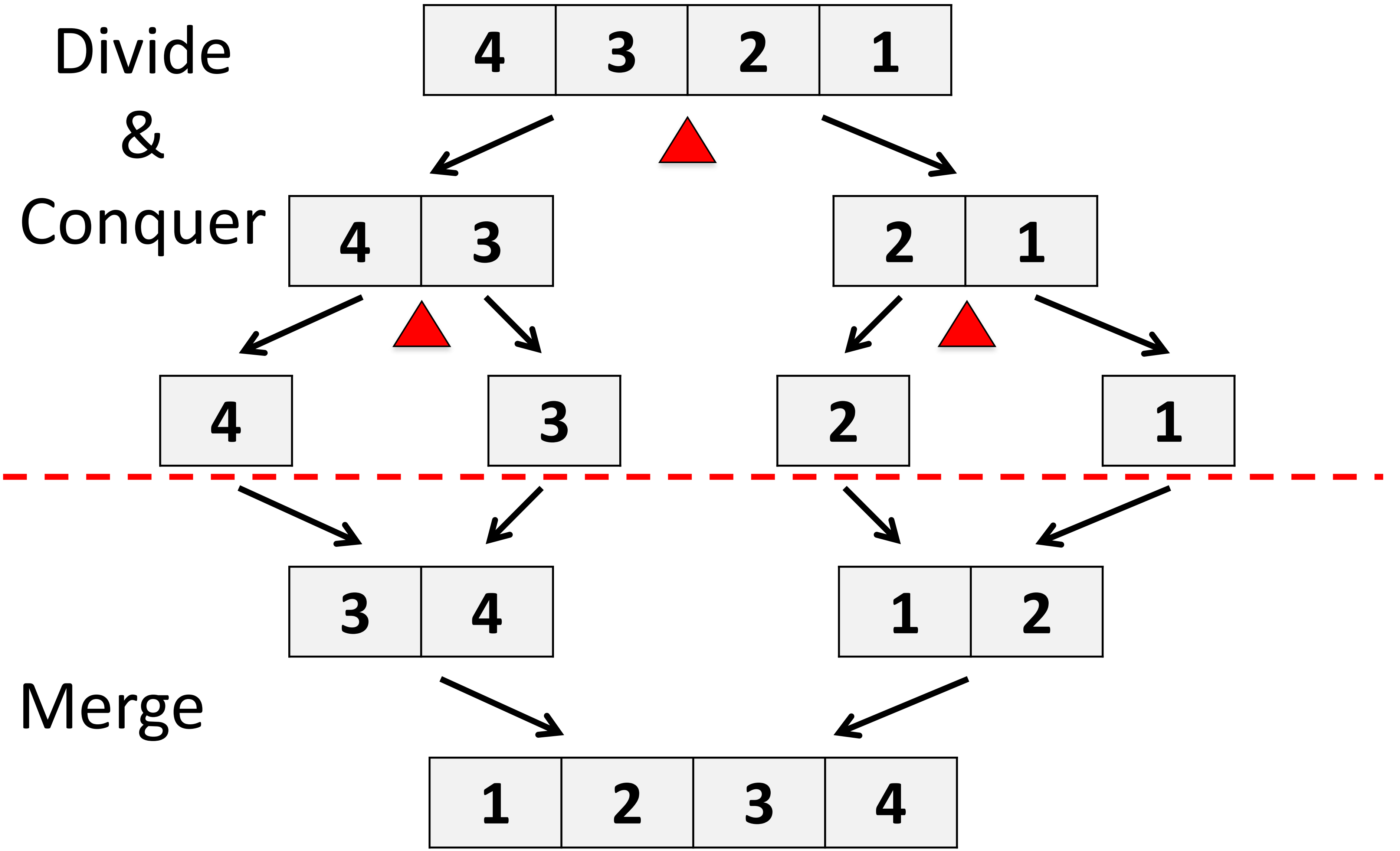
- ***Divide*** the problem into sub-problems that are similar to the original but smaller in size
- ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
- ***Combine*** the solutions to create a solution to the original problem

An Example: Merge Sort

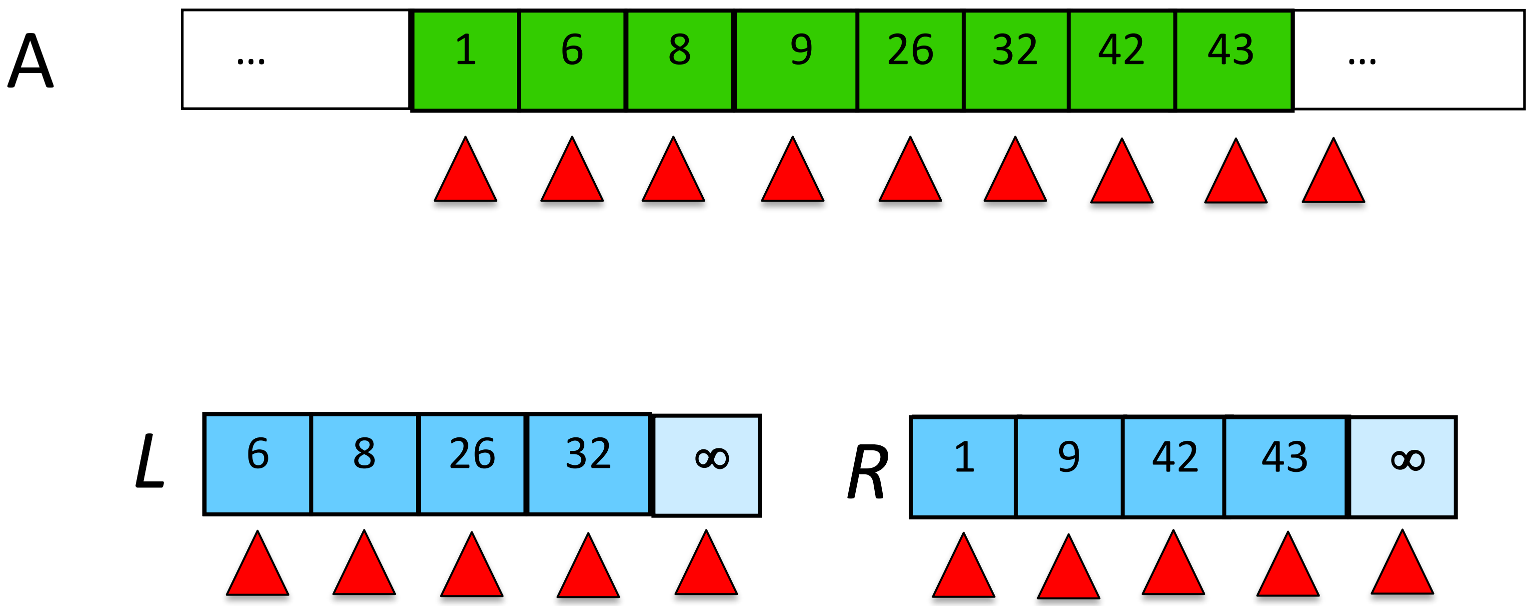
Sorting Problem**:** Sort a sequence of *n* elements into non-decreasing order.

- ***Divide*****:** Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each.
- ***Conquer:*** Sort the two subsequences recursively using merge sort.
- ***Combine*****:** Merge the two sorted subsequences to produce the sorted answer.

Merge Sort - Example



Merge/combine – Example



Idea: If we have 2 lists L and R already sorted, we can easily (i.e. quickly) build a sorted list A with all elements of L and R.

Merge sort (principle)

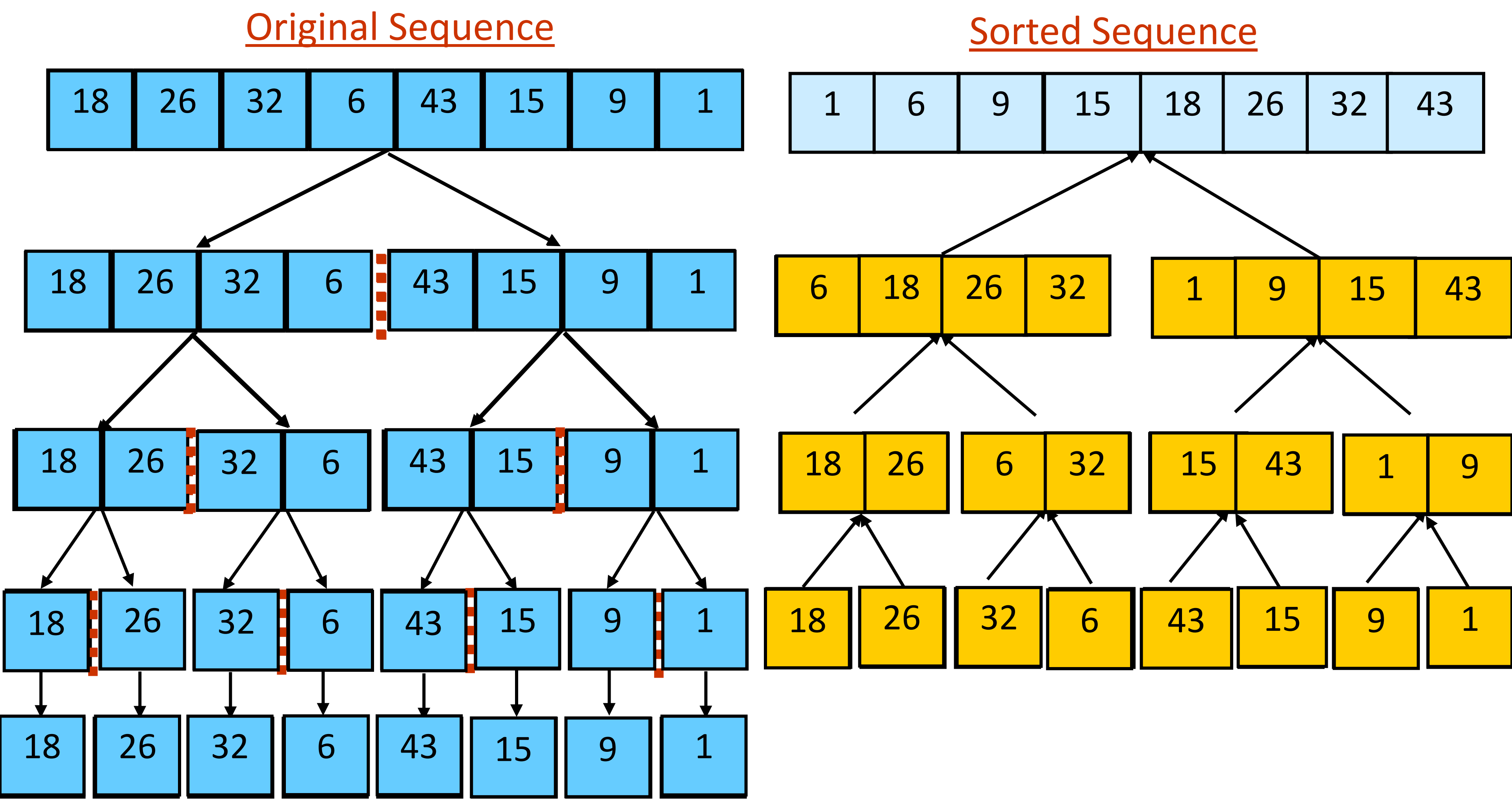
Recursive case

- Unsorted array A with *n* elements
- Split A in half $\rightarrow$ 2 arrays L and R with *n/2* elements
- Sort L and R
- Merge the two sorted arrays L and R

Base case: Stop the recursion when the array is of size 1.

Why? Because the array is already sorted!

Merge Sort – (bigger) Example



Merge-Sort (A, p, r)

INPUT: a sequence of *n* numbers stored in array A

OUTPUT: an ordered sequence of *n* numbers

MergeSort **(*A*, *p*, *r*)** // sort $A[p..r]$ by divide & conquer

if $p < r$

then $q \leftarrow \lfloor (p+r)/2 \rfloor$

MergeSort (A, p, q)

MergeSort $(A, q+1, r)$

Merge (A, p, q, r) // merges $A[p..q]$ with $A[q+1..r]$

Initial Call: MergeSort(A, 1, n)

Procedure Merge

```
Merge(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
    for i ← 1 to n1
        do L[i] ← A[p + i − 1]
    for j ← 1 to n2
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ← p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                 i ← i + 1
            else A[k] ← R[j]
                 j ← j + 1
```

Input: Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.

Output: Merged sorted subarray in $A[p..r]$.

Sentinels, to avoid having to check if either subarray is fully copied at each step.

Running time of Merge Sort

Running time $T(n)$ of Merge Sort:

- Base case: constant time c
- Divide: computing the middle takes constant time c'
- Conquer: solving 2 subproblems takes $2T(n/2)$
- Combine: merging n elements takes $k \cdot n$ (i.e. time proportional to the number of elements to merge)
- Total:

$$T(n) = c \qquad \text{if } n = 1$$
$$T(n) = 2T(n/2) + k \cdot n + c' \qquad \text{if } n > 1$$

Example: Let c=1, c'=1 and k=1

n	1	2	4	8	16	32	64	…	n
T(n)	1	5	15	39	95	223	511	…	?

Running time of Merge Sort