Department of
Computer Engineering

Data Structure & Algorithms

Divide and Conquer

Divide and Conquer

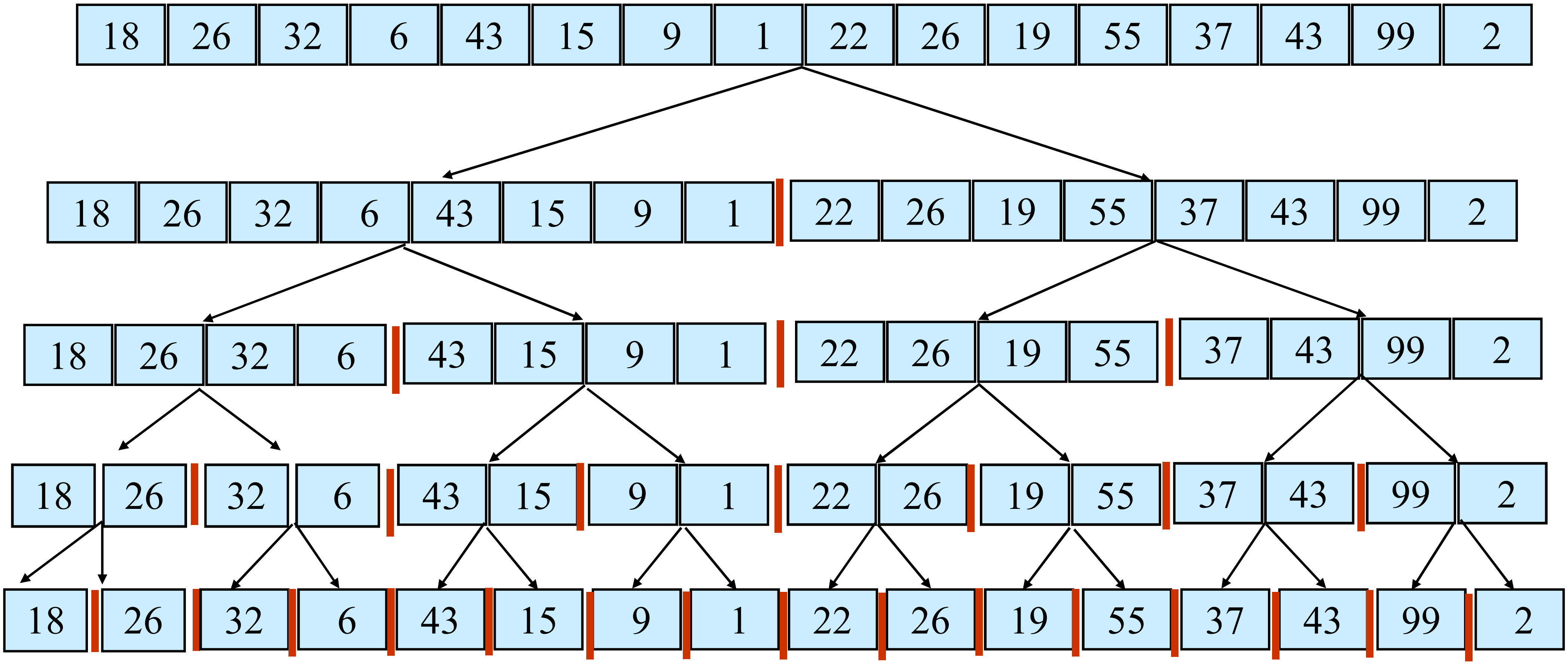
- Recursive in structure
  - ***Divide*** the problem into subproblems that are similar to the original but smaller in size
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create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

An Example: Merge Sort

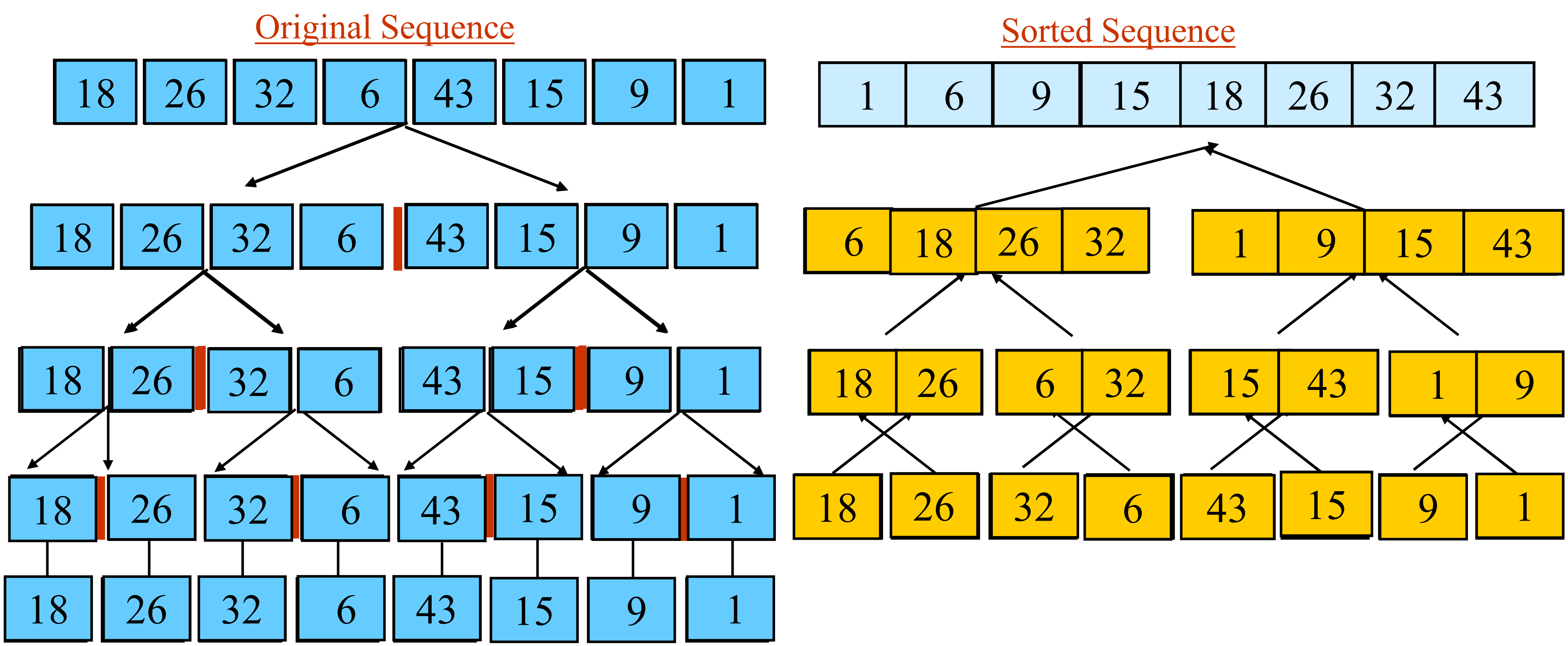
Sorting Problem: Sort a sequence of n elements into non-decreasing order.

- ***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 Sort – Example



Merge-Sort (A, p, r)

INPUT: a sequence of *n* numbers in an 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 ← ⌊(p+r)/2⌋
        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

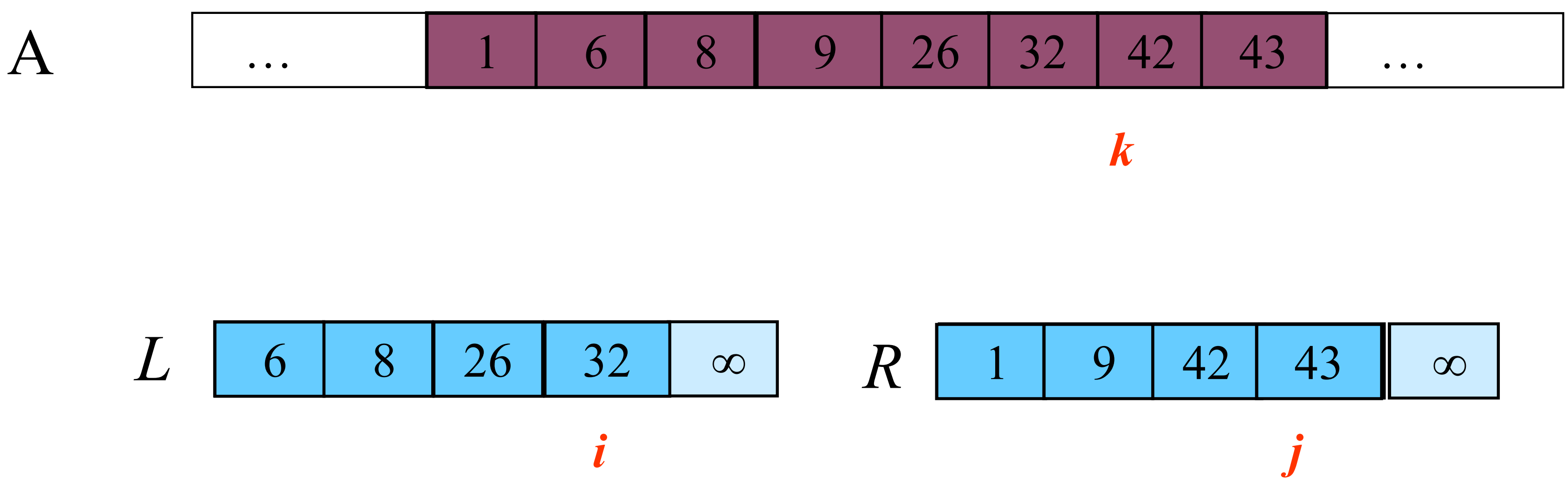
```
Merge(A, p, q, r)
n1 ← q – p + 1
n2 ← r – q
for i ← 1 to n1
    do L[i] ← A[p + i – 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1+1] ← ∞
R[n2+1] ← ∞
i ← 1
j ← 1
for k ←p to r
    do if L[i] ≤ R[j]
        then A[k] ← L[i]
             i ← i + 1
        else A[k] ← R[j]
             j ← j + 1
```

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

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

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

Merge – Example



Correctness of 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
```

Loop Invariant for the *for* loop

At the start of each iteration of the for loop:

Subarray $A[p..k-1]$ contains the $k-p$ smallest elements of L and R in sorted order.

$L[i]$ and $R[j]$ are the smallest elements of L and R that have not been copied back into A.

Initialization:

Before the first iteration:

- $A[p..k-1]$ is empty.
- $i = j = 1$.
- $L[1]$ and $R[1]$ are the smallest elements of L and R not copied to A.

Correctness of Merge

```
Merge(A, p, q, r)
1  n1 ← q – p + 1
2  n2 ← r – q
3  for i ← 1 to n1
4      do L[i] ← A[p + i – 1]
5  for j ← 1 to n2
6      do R[j] ← A[q + j]
7  L[n1+1] ← ∞
8  R[n2+1] ← ∞
9  i ← 1
10 j ← 1
11 for k ←p to r
12     do if L[i] ≤ R[j]
13         then A[k] ← L[i]
14              i ← i + 1
15         else A[k] ← R[j]
16              j ← j + 1
```

Maintenance:

Case 1: $L[i] \le R[j]$

- By LI, A contains $p - k$ smallest elements of L and R in sorted order.
- By LI, $L[i]$ and $R[j]$ are the smallest elements of L and R not yet copied into A.
- Line 13 results in A containing $p - k + 1$ smallest elements (again in sorted order).

Incrementing i and k reestablishes the LI for the next iteration.

Similarly for $L[i] > R[j]$.

Termination:

- On termination, $k = r + 1$.
- By LI, A contains $r - p + 1$ smallest elements of L and R in sorted order.
- L and R together contain $r - p + 3$ elements. All but the two sentinels have been copied back into A.

Analysis of Merge Sort

- Running time $T(n)$ of Merge Sort:
- Divide: computing the middle takes $\Theta(1)$
- Conquer: solving 2 subproblems takes $2T(n/2)$
- Combine: merging n elements takes $\Theta(n)$
- Total:

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

$\Rightarrow T(n) = \Theta(n \lg n)$