COT 6405 Introduction to Theory of Algorithms

Topic 3. Divide and Conquer

General rule

To solve (an instance of) a problem P

IF (the instance of) P is “large enough” THEN

Divide P into smaller instances of the same problem

Recurse to solve the smaller instances

Combine solutions of the smaller instances to create a solution for the original instance

ELSE

Solve P directly

Time complexity

To solve (an instant of) a problem P

IF (the instant of) P is "large enough" n>s THEN

Divide P into smaller instances of the same problem at a cost of g(n)

Recurse to solve the smaller instances ≤ aT(n/b), if there are a instances of size ≤ n/b

Combine solutions of the smaller instances to create a solution for the original instance at a cost of h(n)

ELSE

Solve P directly at a cost of c_0

Time complexity (Cont'd)

- The total cost of a Divide-and-Conquer algorithm is governed by the recurrence relation

T(n) $\leq$ aT(n/b) + g(n) + h(n) if n > s, and

T(n) $\leq$ c_0 otherwise

Example 1: binary search

- Determine whether x is one of A[1], A[2], . . . , A[n] (and retrieve other information about x).
  – Assume that A is a sorted array

Binary search (cont'd)

```
function BINARYSEARCH(A, x, lower, upper)
  if lower < upper then
     mid  = floor((lower + upper)/2)
     if x  ≤ A[mid] then
         return BINARYSEARCH (A, x, lower, mid)
     else return BINARYSEARCH (A, x, mid + 1, upper)
  else
     if x = A[lower] then return lower
     else return 0
  end BINARYSEARCH
```

Number of comparisons

- If we count the number of comparisons, we can obtain T(n) $=$ T($\lceil n/2 \rceil$)+1

$$T(n) = T(\lceil n/2 \rceil) + 1$$
$$= [T(\lceil n/4 \rceil)+1] + 1$$
$$= \ldots\ldots$$
$$=T(\lceil n/2^i \rceil) + i \text{ as long as } n \geq 2^i$$

The recursion bottoms out when the array has only one element left. In this case, $n = 2^i$ and $i = \lg n$.

$$T(n) = T(1) + \lg n = 1 + \lg n,$$

where T(1) = 1 (only one comparison is made)

Example 2: merge sort

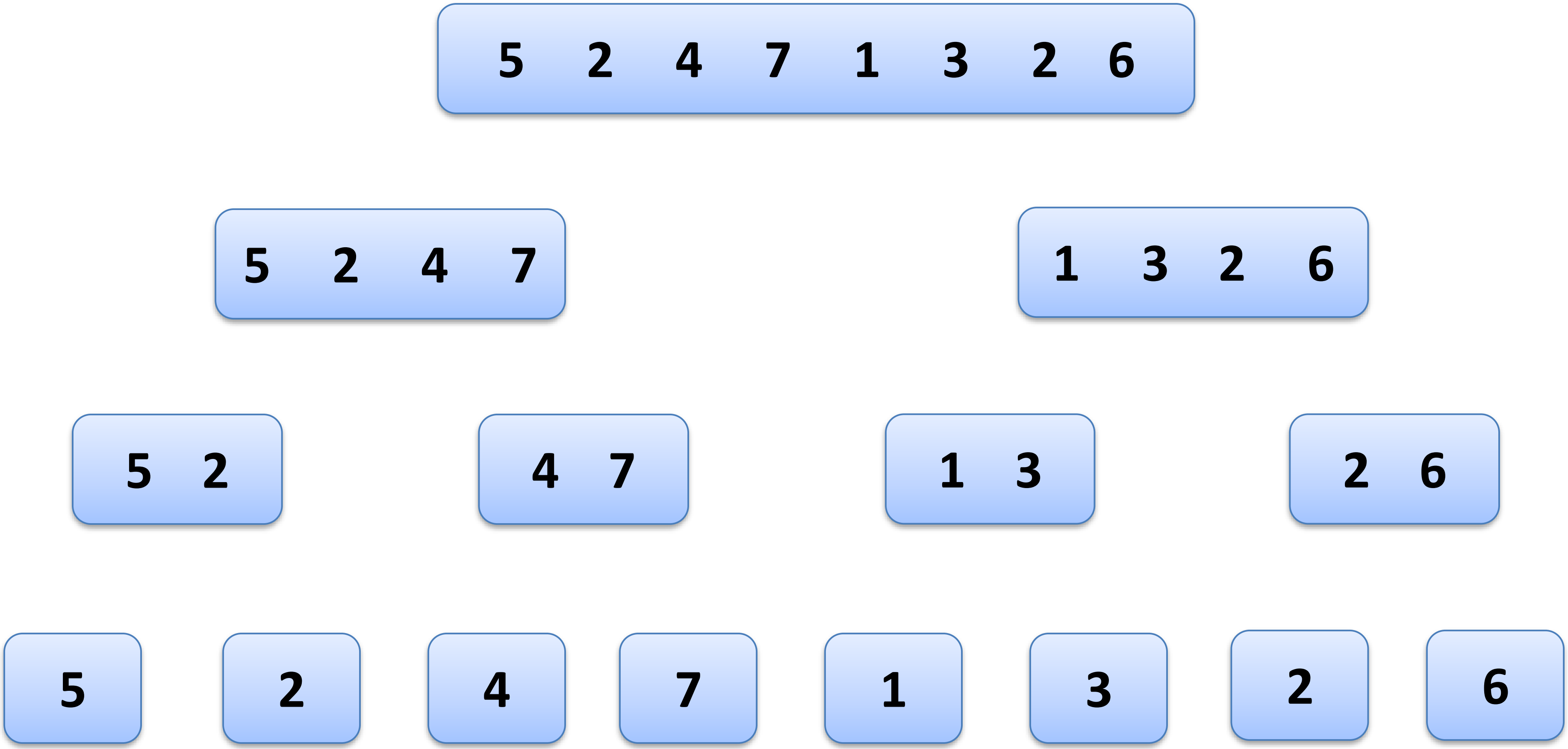
- The *merge sort* algorithm closely follows the divide-and-conquer paradigm
  - **Divide:** divide the n-element sequence to be sorted into two subsequences of n/2 elements each.
  - **Conquer:** sort the two subsequences recursively using merge sort
  - **Combine:** merge the two sorted subsequences to produce the sorted answer

The recursion "bottoms out" when the sequence to be sorted has length 1

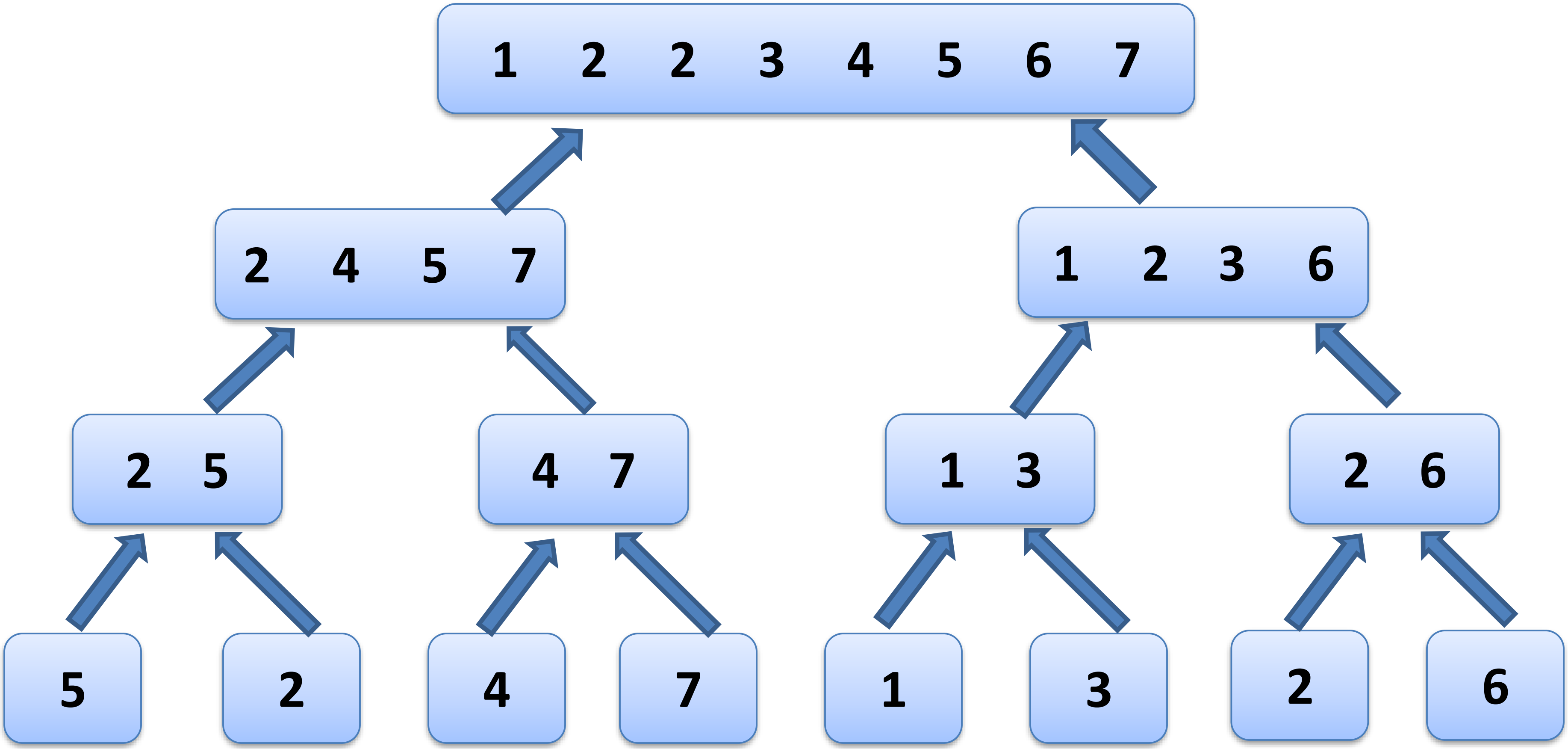
The general rule

- The length-n sequence is divided into n subsequence with one element only.
- The subsequences are repeatedly merged to form new sorted subsequences until there is only one subsequence remaining.

Merge sort: divide



Merge sort: concur



Merge sort (cont'd)

MERGE-SORT (A, p, r)

    if p < r

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

        MERGE-SORT (A, p, q)

        MERGE-SORT (A, q+1, r)

        MERGE (A, p, q, r)

Merging

MERGE (A, p, q, r)

$n_1 = q - p + 1$; $n_2 = r - q$

create arrays $L[1 \,.\,.\, (n_1 + 1)]$ and $R[1 \,.\,.\, (n_2 + 1)]$

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 $L[i] \le R[j]$

then $A[k] = L[i]$

$i = i + 1$

else $A[k] = R[j]$

$j = j + 1$

Merging (cont'd)

	8	9	10	11	12	13	14	15	16	17
A	**…**	**2**	**4**	**5**	**7**	**1**	**2**	**3**	**6**	**…**

	1	2	3	4	5
L	**2**	**4**	**5**	**7**	∞
	i				

	1	2	3	4	5
R	**1**	**2**	**3**	**6**	∞
	j				

MERGE(A, 9, 12, 16)

Merging (cont'd)

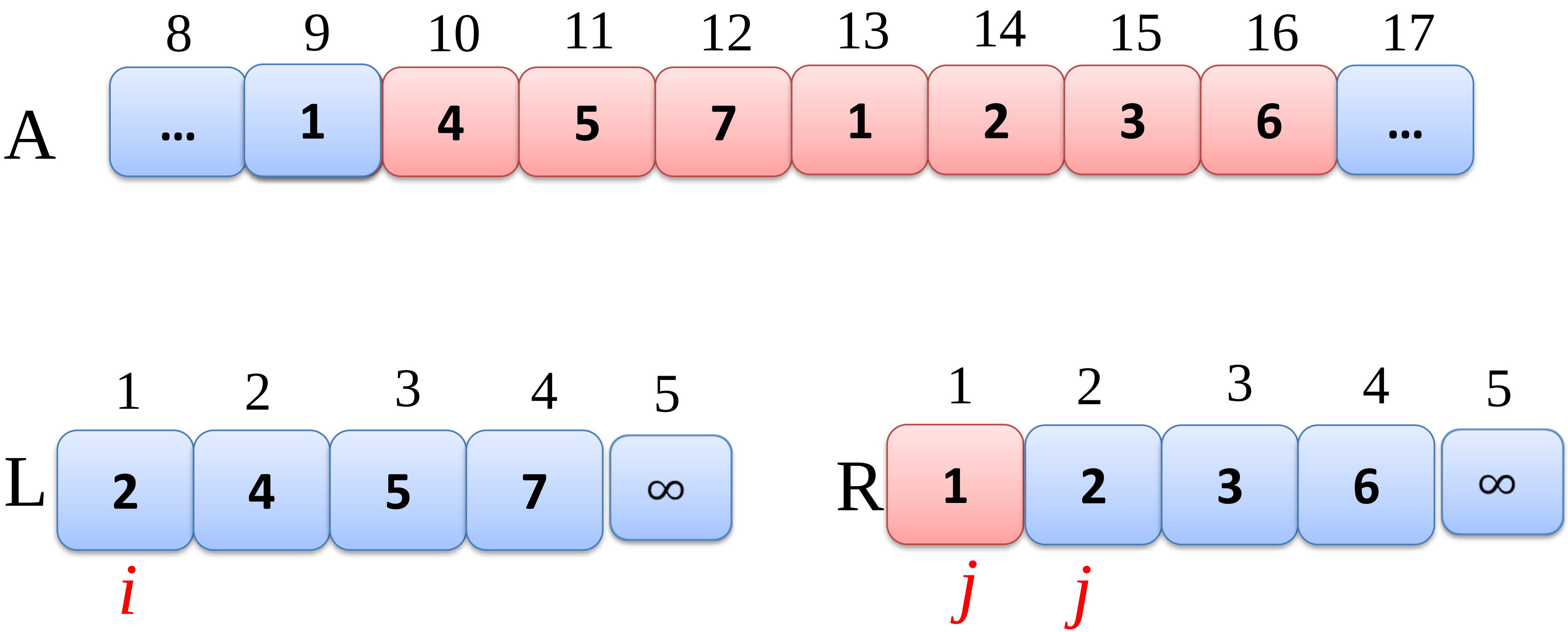
	8	9	10	11	12	13	14	15	16	17
A	…	**2**	**4**	**5**	**7**	**1**	**2**	**3**	**6**	…

	1	2	3	4	5
L	**2**	**4**	**5**	**7**	∞
	i				

	1	2	3	4	5
R	**1**	**2**	**3**	**6**	∞
	j				

MERGE(A, 9, 12, 16)

Merging (cont'd)



MERGE(A, 9, 12, 16)

Merging (cont'd)

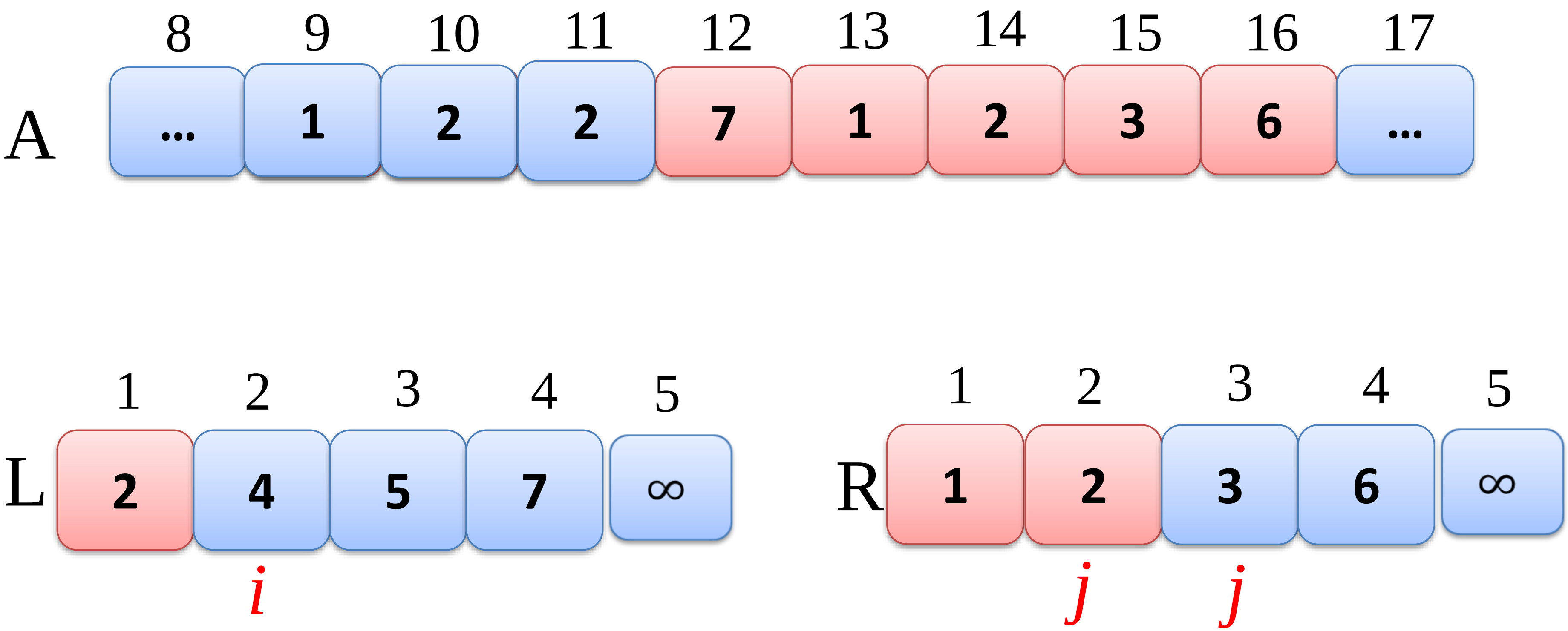
	8	9	10	11	12	13	14	15	16	17
A	…	**1**	**2**	**5**	**7**	**1**	**2**	**3**	**6**	…

	1	2	3	4	5
L	**2**	**4**	**5**	**7**	∞
	i	*i*			

	1	2	3	4	5
R	**1**	**2**	**3**	**6**	∞
		j			

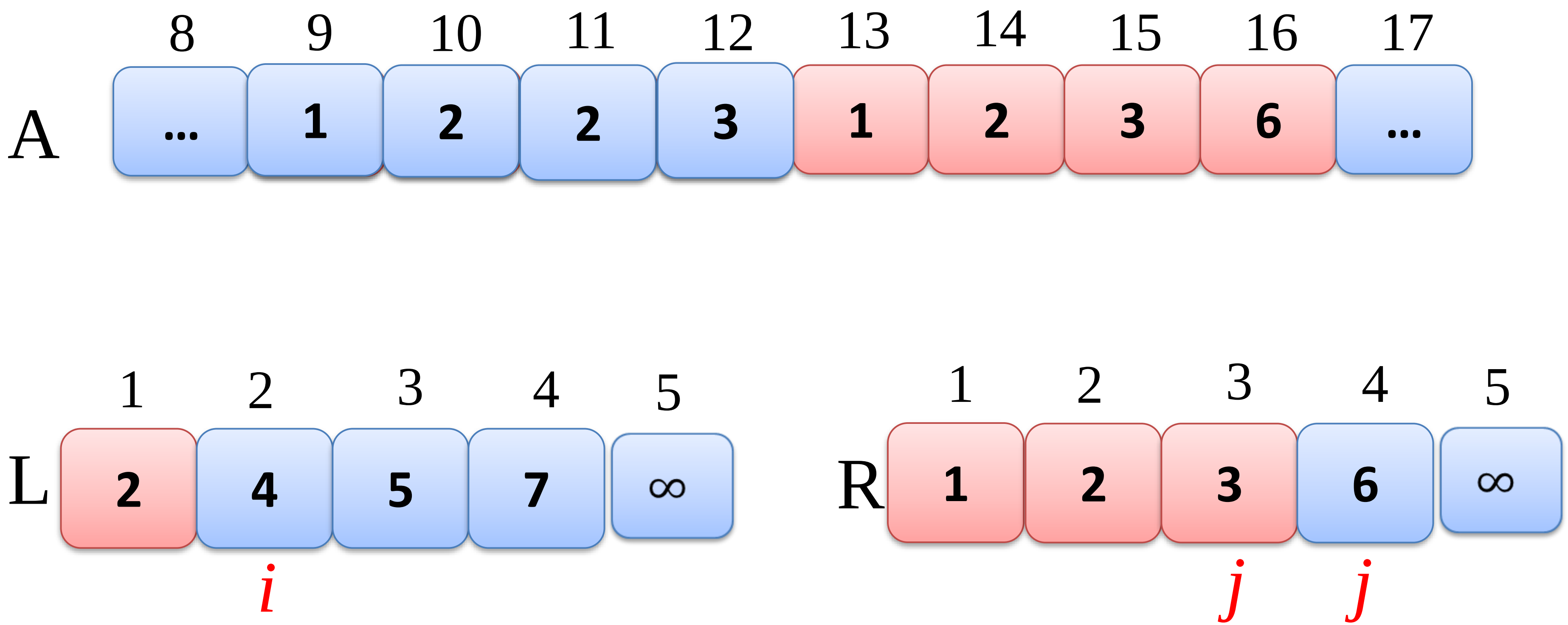
MERGE(A, 9, 12, 16)

Merging (cont'd)



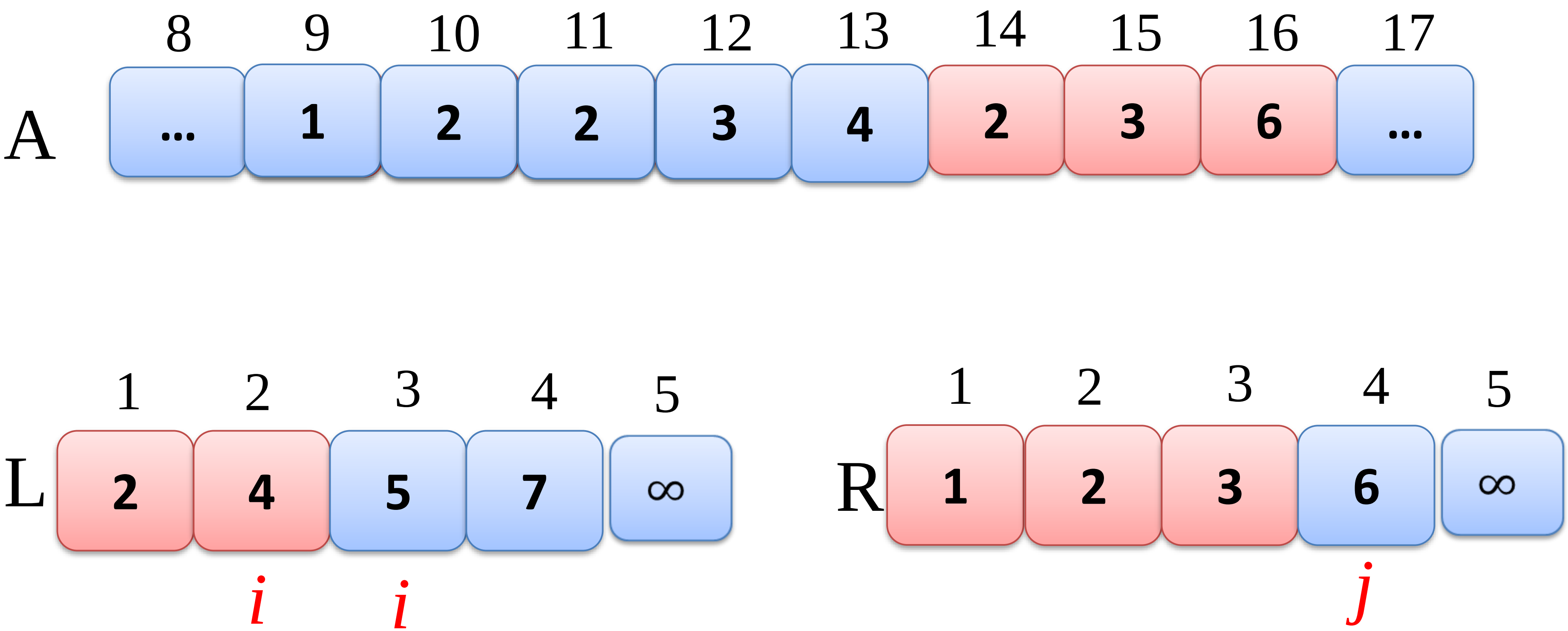
MERGE(A, 9, 12, 16)

Merging (cont'd)



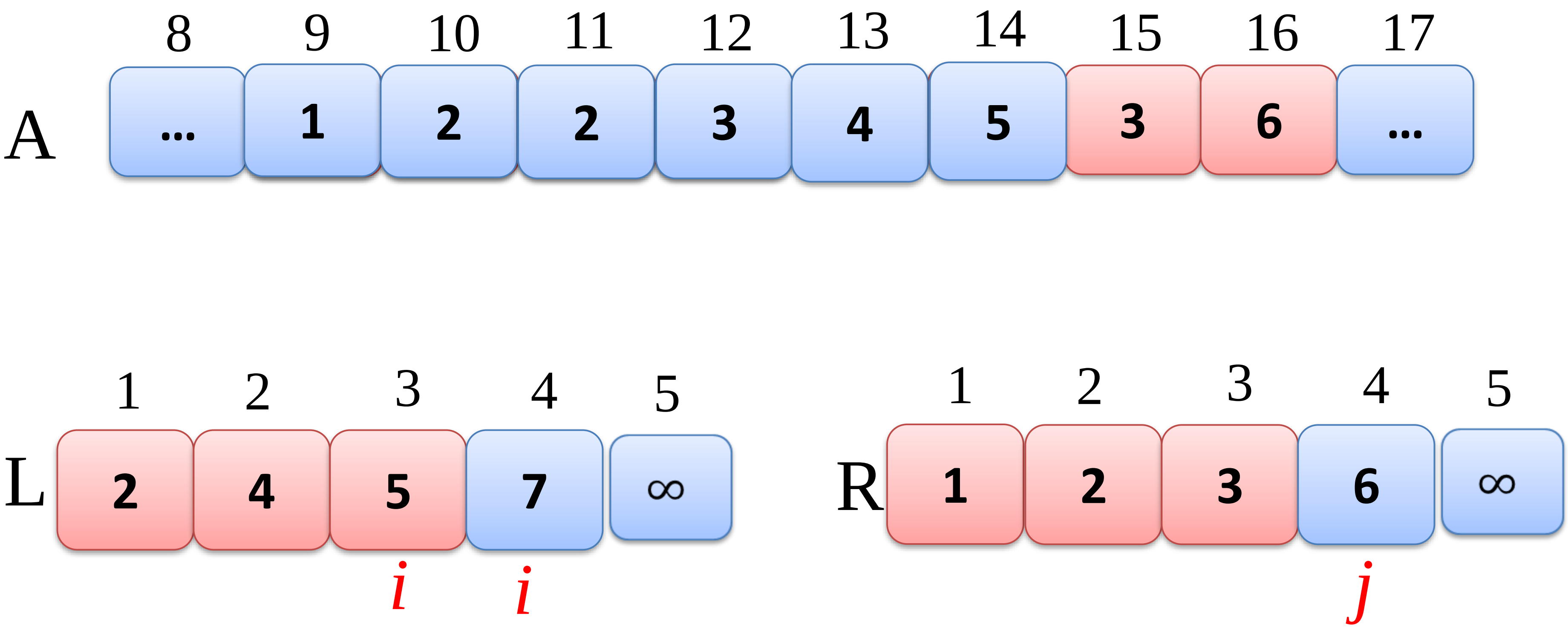
MERGE(A, 9, 12, 16)

Merging (cont'd)



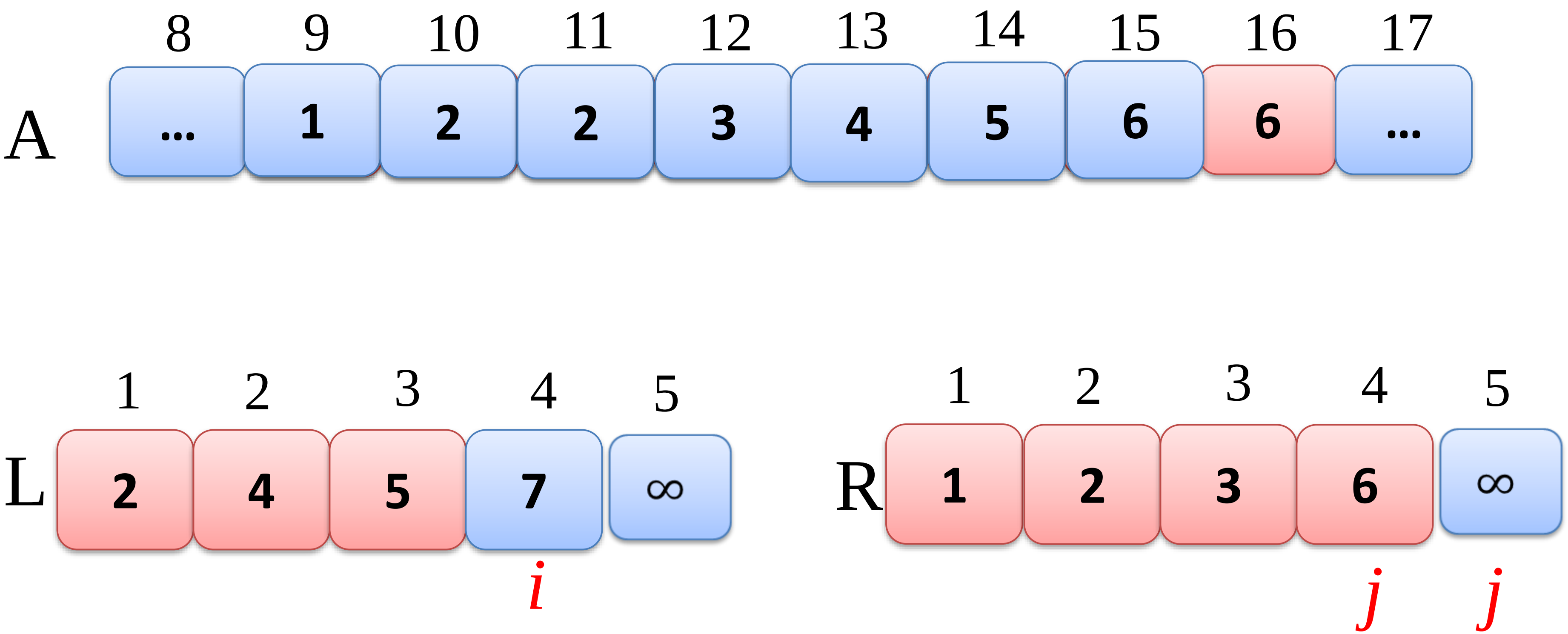
MERGE(A, 9, 12, 16)

Merging (cont'd)



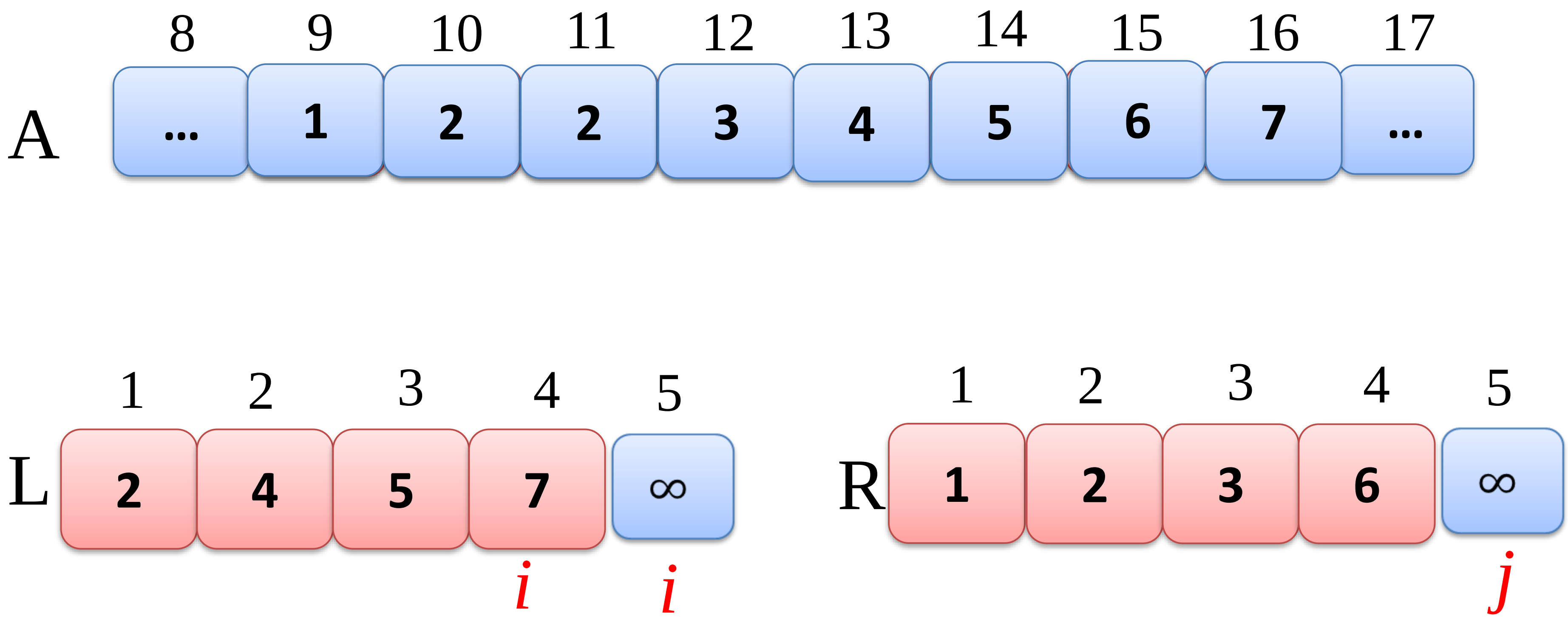
MERGE(A, 9, 12, 16)

Merging (cont'd)



MERGE(A, 9, 12, 16)

Merging (cont'd)



MERGE(A, 9, 12, 16)

Number of element comparisons

MERGE-SORT (A, p, r)

if p < r

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

MERGE-SORT (A, p, q) T(n/2)

MERGE-SORT (A, q+1, r) T(n/2)

MERGE (A, p, q, r) n

We count comparisons: T(n) = 2T(n/2) + n (n>1)
T(1) = ?

Comparisons (cont'd)

- Assume for simplicity that n is a power of two
- $T(n) = 2T(n/2) + n$

  $= 2(2T(n/4) + n/2) + n$

  $= 4T(n/4) + 2n$

  $= \ldots\ldots.$

  $= 2^i T(n/2^i) + i*n$ as long as $n \geq 2^i$

The recursion bottoms out when $n = 2^i$, and

$$T(n) = 2^i T(n/2^i) + i*n$$
$$= 2^i T(2^i/2^i) + i*n$$
$$= 2^i\, T(1) + \lg n * n = n\lg n$$