INT102
Algorithmic Foundations
And
Problem Solving

Divide and Conquer

Dr Yushi Li

Department of Computer Science

西交利物浦大学
Xi'an Jiaotong-Liverpool University

Divide and Conquer …

Learning outcomes

- Understand how divide and conquer works and able to analyze complexity of divide and conquer methods by solving recurrence
- See examples of divide and conquer methods

Learning outcomes

- Understand how divide and conquer works and able to analyze complexity of divide and conquer methods by solving recurrence
- See examples of divide and conquer methods

Divide and Conquer

One of the **best-known** algorithm design techniques.

Idea:

- A problem instance is ***divided*** into several **smaller** instances of the same problem, ideally of about same size
- The smaller instances are **solved,** typically **recursively**
- The solutions for the smaller instances are ***combined*** to get a solution to the original problem

Binary Search

Recall that we have learnt binary search:

Input: a sequence of n **sorted** numbers $a_0, a_1, \ldots, a_{n-1}$; and a number X

Idea of algorithm:

- compare X with number in the **middle**
- then focus on only the first **half** or the second half (depend on whether X is smaller or greater than the middle number)
- reduce the amount of numbers to be searched by half

Binary Search (2)

we first work on n numbers, from a[0]..a[n-1]

3 7 11 12 15 19 24 33 41 55

24

then we work on n/2 numbers,
from a[n/2]..a[n-1]

19 24 33 41 55

24

further reduce by half

19 24

24

Recursive Binary Search

```
RecurBinarySearch(A, first, last, X)
begin
    if (first > last) then
        return false
    mid = ⌊(first + last)/2⌋
    if (X == A[mid]) then
        return true
    if (X < A[mid]) then
        return RecurBinarySearch(A, first, mid-1, X)
    else
        return RecurBinarySearch(A, mid+1, last, X)
end
```

invoke by calling
RecurBinarySearch(A, 0, n-1, X)
return true if X is found,
false otherwise

Recursive Binary Search (RBS)

To find 24

3 7 11 12 15 19 24 33 41 55

RBS(A, 0, 9, 24)

- if 0 > 9? No; mid = 4; if 24 == A[4]? No; if 24 < A[4]? No

  RBS(A, 5, 9, 24)

  - if 5 > 9? No; mid = 7; if 24 == A[7]? No; if 24 < A[7]? Yes

    RBS(A, 5, 6, 24)

    - if 5 > 6? No; mid = 5; if 24 == A[5]? No; if 24 < A[5]? No

      RBS(A, 6, 6, 24)

      - if 6 > 6? No; mid = 6; if 24 == A[6]? YES; **return true**

      RBS(A, 6, 6, 24) is done, **return true**

    RBS(A, 5, 6, 24) is done, **return true**

  RBS(A, 5, 9, 24) is done, **return true**

RBS(A, 0, 9, 24) is done, **return true**

Recursive Binary Search (RBS)

To find 23

3 7 11 12 15 19 24 33 41 55

RBS(A, 0, 9, 23)

– if 0 > 9? No; mid = 4; if 23 == A[4]? No; if 23 < A[4]? No

RBS(A, 5, 9, 23)

• if 5 > 9? No; mid = 7; if 23 == A[7]? No; if 23 < A[7]? Yes

RBS(A, 5, 6, 23)

– if 5 > 6? No; mid = 5; if 23 == A[5]? No; if 23 < A[5]? No

RBS(A, 6, 6, 23)

» if 6 > 6? No; mid = 6; if 23 == A[6]? No; if 23 < A[5]? No

RBS(A, 7, 6, 23): if 7 > 6? Yes; **return false**

RBS(A, 7, 6, 23) is done, **return false**

RBS(A, 6, 6, 23) is done, **return false**

RBS(A, 5, 6, 23) is done, **return false**

RBS(A, 5, 9, 23) is done, **return false**

RBS(A, 0, 9, 23) is done, **return false**

Time complexity

Let T(n) denote the time complexity of binary search algorithm on n numbers.

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$$

We call this formula a **<u>recurrence</u>**.

Recurrence

A recurrence is an equation or inequality that describes a function in terms of ***its value on smaller inputs***.

E.g.,

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

To solve a recurrence is to derive *asymptotic bounds* on the solution

Substitution method

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$$

Make a guess, $T(n) \le 2 \log n$

Base case? When n=1, statement is FALSE!

L.H.S = $T(1) = 1$ R.H.S $= c \log 1 = 0 <$ L.H.S

Yet, when n=2,

L.H.S = $T(2) = T(1)+1 = 2$

R.H.S = $2 \log 2 = 2$

L.H.S $\le$ R.H.S

Substitution method

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$$

Make a guess, $T(n) \leq 2 \log n$

Assume true for all $n' < n$ [assume $T(n/2) \leq 2 \log (n/2)$]

$T(n) = T(n/2) + 1$

$\leq 2 \log (n/2) + 1$ ← *by hypothesis*

$= 2(\log n - 1) + 1$ ← ***log(n/2) = log n – log 2***

$< 2\log n$

$\boxed{\text{i.e., } T(n) \leq 2 \log n}$

Example

Prove that $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n/2) + 1 & \text{otherwise} \end{cases}$ is $O(n)$

Guess: $T(n) \le 2n - 1$

More Example

Prove that $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n/2) + n & \text{otherwise} \end{cases}$ is O(n log n)

Guess: $T(n) \leq 2\, n \log n$

Summary

Depending on the recurrence, we can guess the order of magnitude

$T(n) = T(n/2)+1$ $T(n)$ is $O(\log n)$

$T(n) = 2T(n/2)+1$ $T(n)$ is $O(n)$

$T(n) = 2T(n/2)+n$ $T(n)$ is $O(n \log n)$

Learning outcomes

- ✓ Understand how divide and conquer works and able to analyze complexity of divide and conquer methods by solving recurrence
- ➢ See examples of divide and conquer methods

Merge Sort …

Merge sort

- using divide and conquer technique
- divide the sequence of n numbers into two halves
- **recursively** sort the two halves
- **merge** the two sort halves into a single sorted sequence

51, 13, 10, 64, 34, 5, 32, 21

we want to sort these 8 numbers,
divide them into two halves

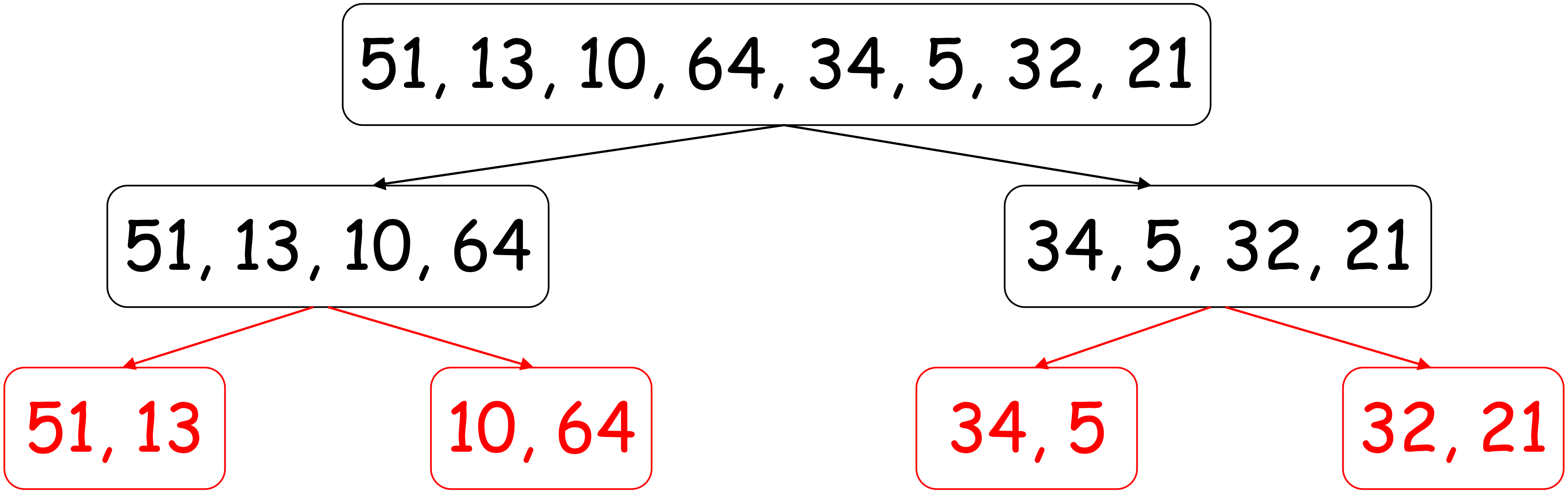
51, 13, 10, 64, 34, 5, 32, 21

51, 13, 10, 64

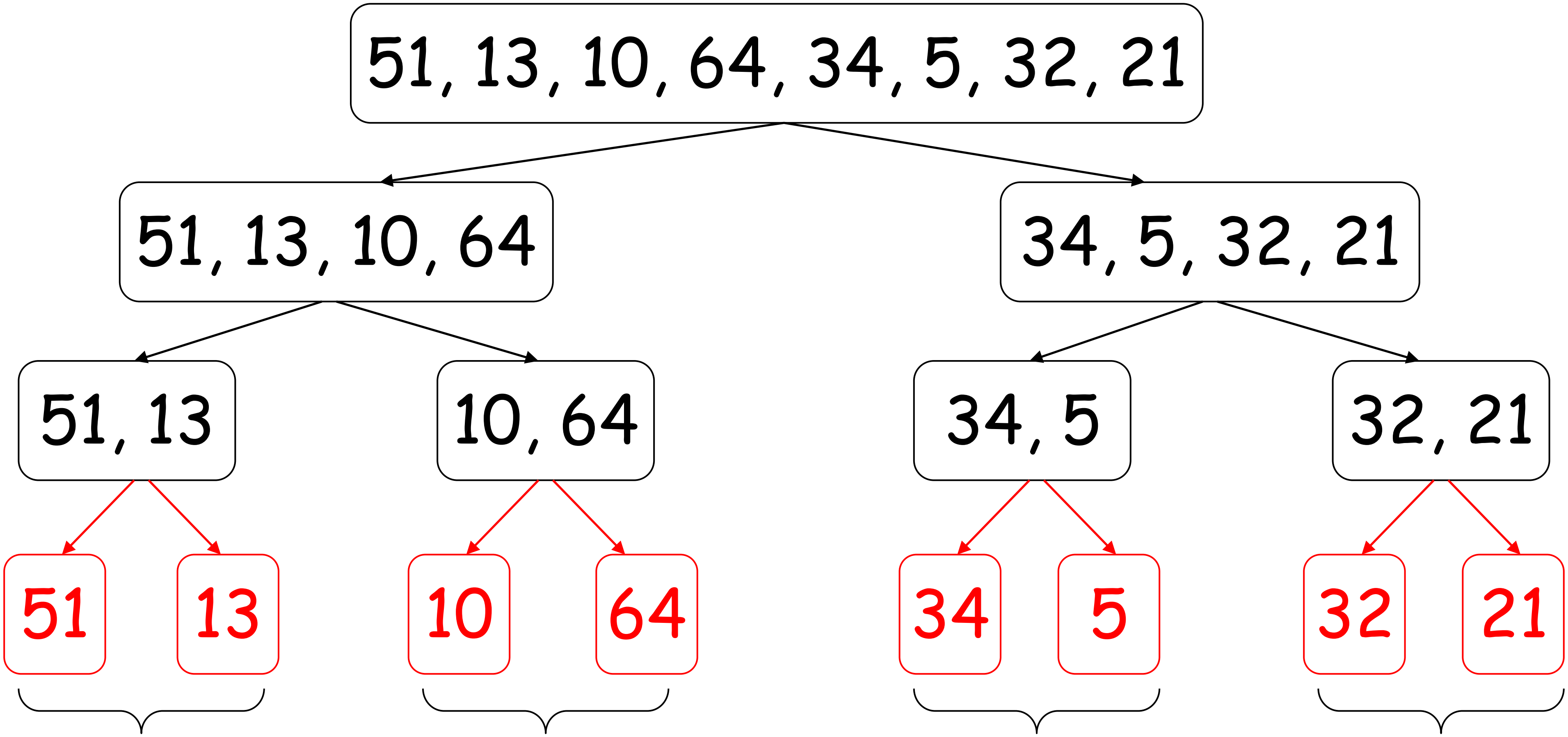
34, 5, 32, 21

divide these 4 numbers into halves

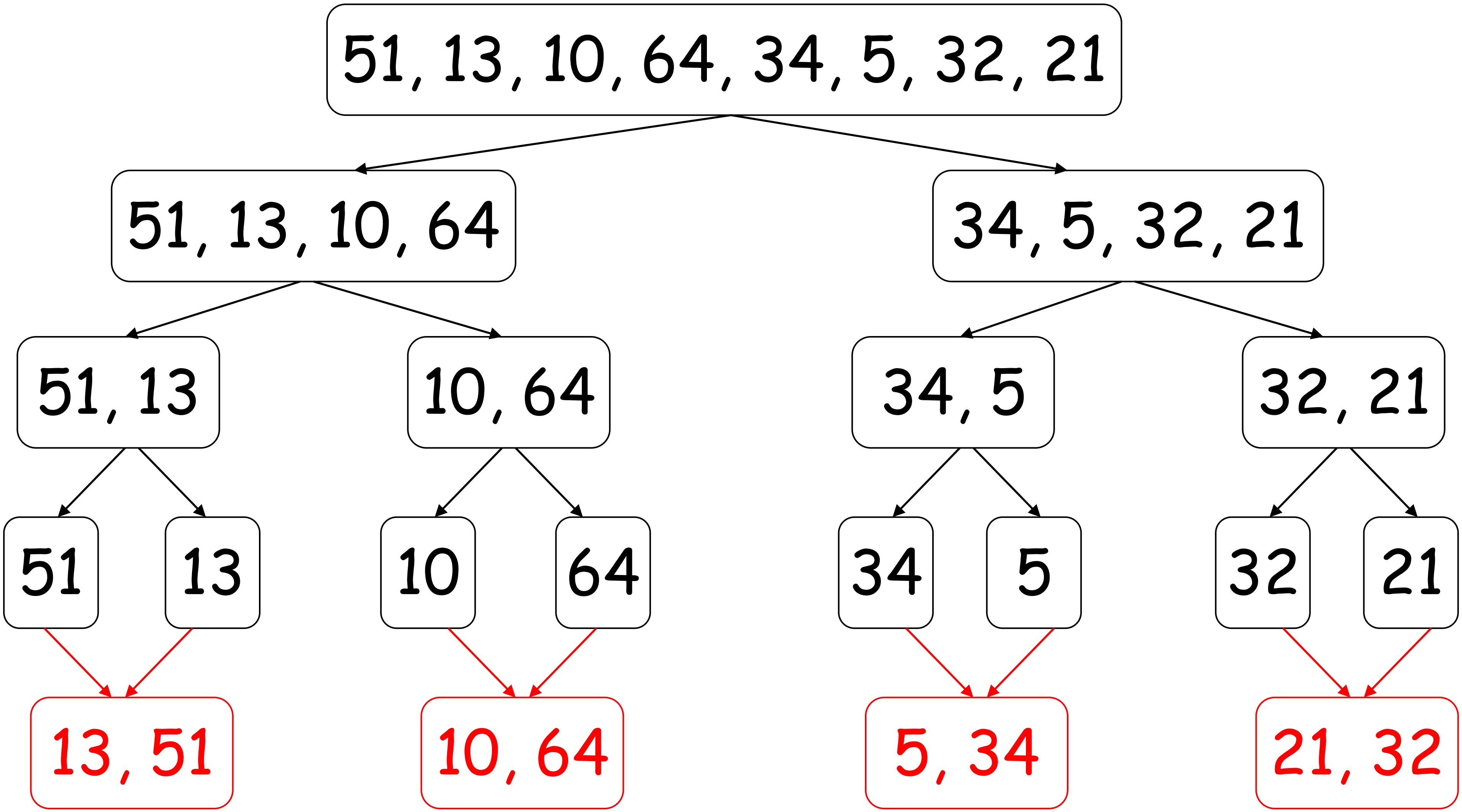
similarly for these 4



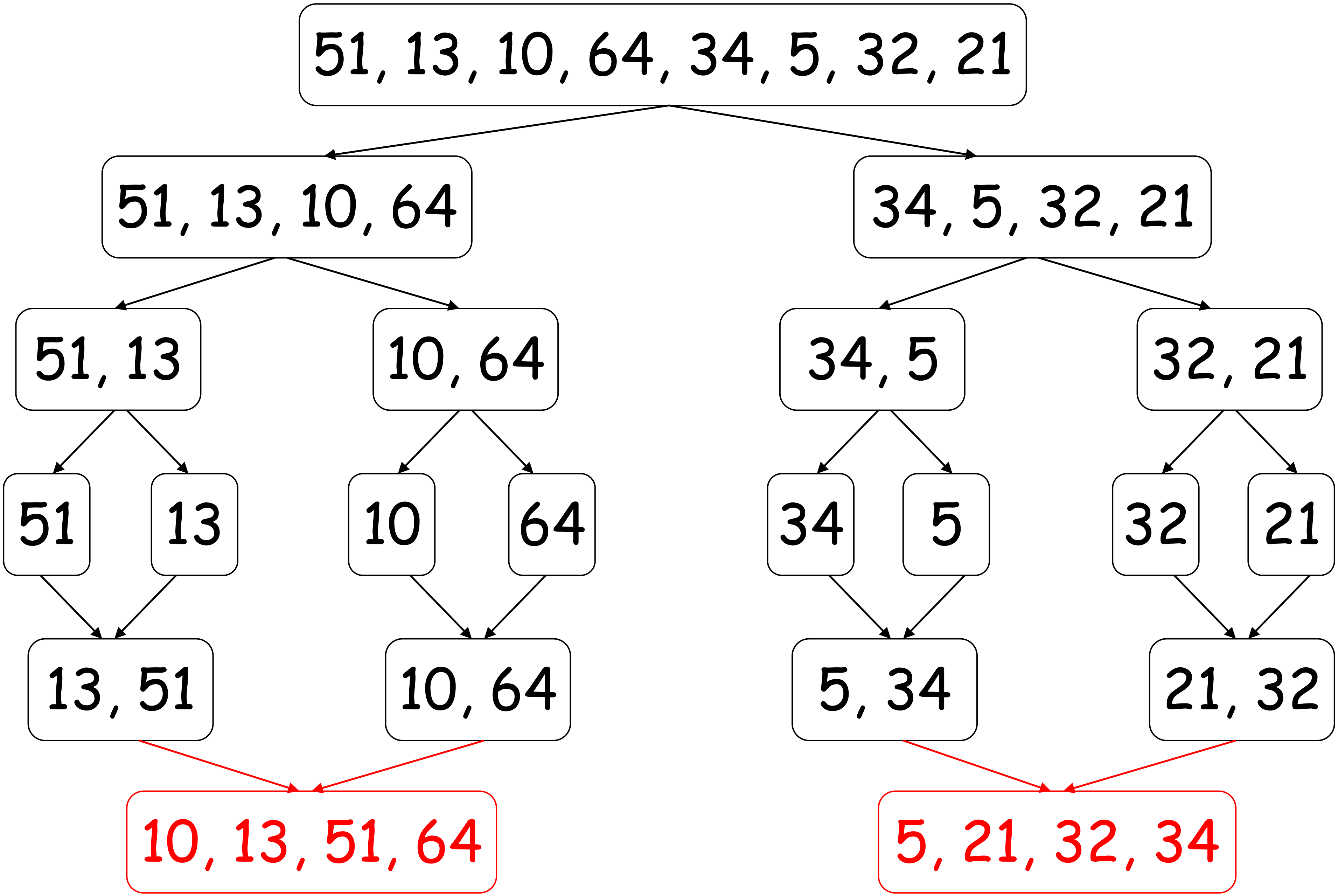
further divide each shorter sequence …
until we get sequence with only **1** number



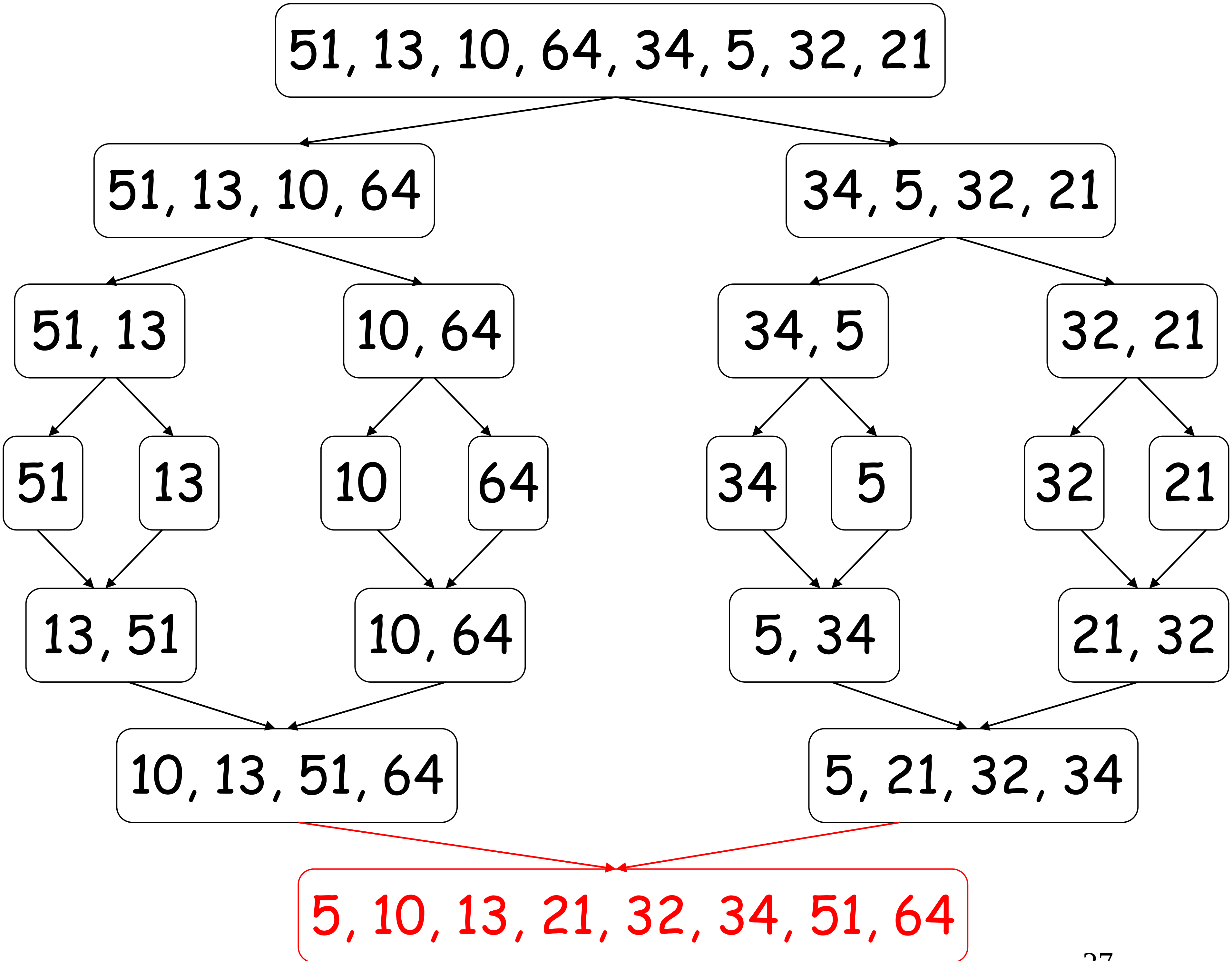
merge pairs of single number into a sequence of 2 sorted numbers



then **merge** again into sequences of 4 sorted numbers



one more merge give the **final** sorted sequence


51, 13, 10, 64, 34, 5, 32, 21
51, 13, 10, 64
34, 5, 32, 21
51, 13
10, 64
34, 5
32, 21
51
13
10
64
34
5
32
21
13, 51
10, 64
5, 34
21, 32
10, 13, 51, 64
5, 21, 32, 34
5, 10, 13, 21, 32, 34, 51, 64

Summary

Divide

– dividing a sequence of n numbers into two smaller sequences is straightforward

Conquer

– merging two sorted sequences of **total length n** can also be done easily, at most **n-1** comparisons

10, 13, 51, 64

5, 21, 32, 34

Result:

To merge two sorted sequences,
we keep two **pointers**, one to each sequence

Compare the two numbers pointed,
copy the **smaller** one to the result
and **advance** the corresponding pointer

10, 13, 51, 64

5, 21, 32, 34

Result: 5,

Then compare again the two numbers pointed to by the pointer; copy the smaller one to the result and advance that pointer

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10,

Repeat the same process ...

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13

Again …

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, 21

and again …

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, 21, 32

...

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, 21, 32, 34

When we reach the **end** of one sequence, simply copy the **remaining** numbers in the other sequence to the result

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, 21, 32, 34, 51, 64

Then we obtain the final sorted sequence

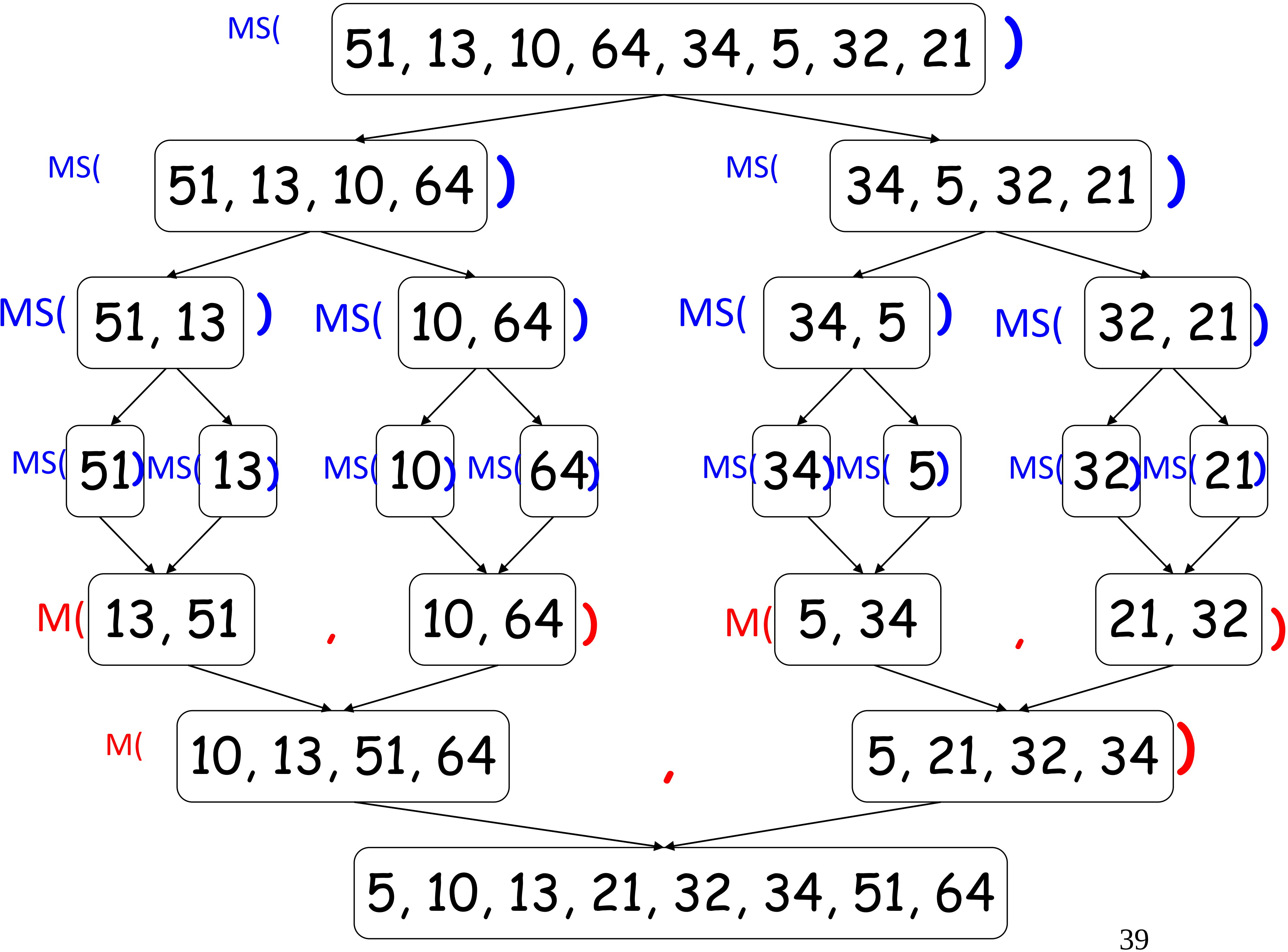
Pseudo code

```
Algorithm Mergesort(A[0..n-1])
 if n > 1 then begin
  copy A[0..⌊n/2⌋-1] to B[0..⌊n/2⌋-1]
  copy A[⌊n/2⌋..n-1] to C[0..⌈n/2⌉-1]
  Mergesort(B[0..⌊n/2⌋-1])
  Mergesort(C[0..⌈n/2⌉-1])
  Merge(B, C, A)
 end
```

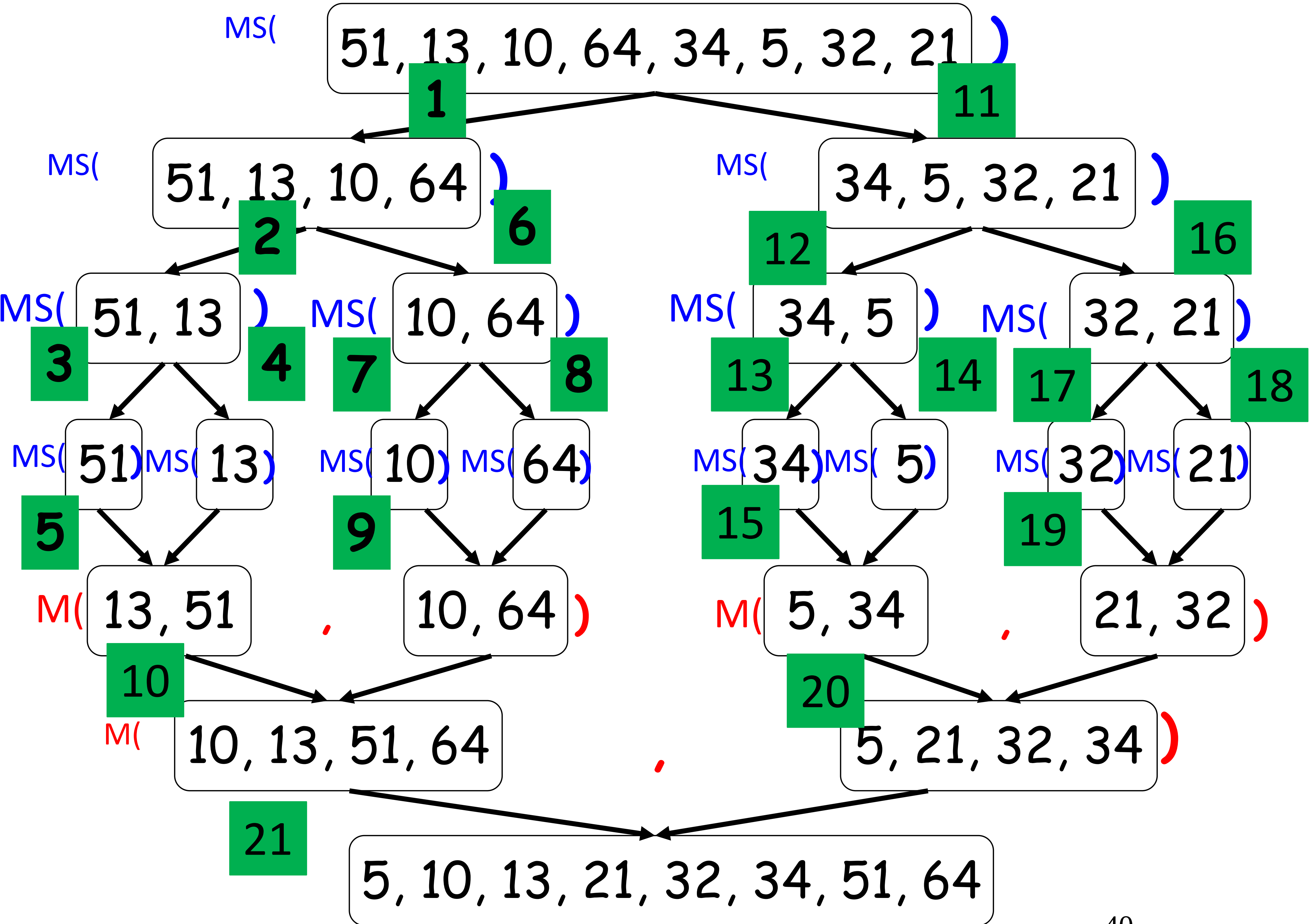
```
Algorithm Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])
    Set i=0, j=0, k=0
    while i<p and j<q do
    begin
        if B[i]≤C[j] then set A[k]=B[i] and increase i
        else set A[k] = C[j] and increase j
        k = k+1
    end
    if i==p then copy C[j..q-1] to A[k..p+q-1]
    else copy B[i..p-1] to A[k..p+q-1]
```

Pseudo code

```
Algorithm Mergesort(A[0..n-1])
 if n > 1 then begin
  copy A[0..⌊n/2⌋-1] to B[0..⌊n/2⌋-1]
  copy A[⌊n/2⌋..n-1] to C[0..⌈n/2⌉-1]
  Mergesort(B[0..⌊n/2⌋-1])
  Mergesort(C[0..⌈n/2⌉-1])
  Merge(B, C, A)
 end
```



MS(51, 13, 10, 64, 34, 5, 32, 21)
MS(51, 13, 10, 64)
MS(34, 5, 32, 21)
MS(51, 13)
MS(10, 64)
MS(34, 5)
MS(32, 21)
MS(51)
MS(13)
MS(10)
MS(64)
MS(34)
MS(5)
MS(32)
MS(21)
M(13, 51 , 10, 64)
M(5, 34 , 21, 32)
M(10, 13, 51, 64 , 5, 21, 32, 34)
5, 10, 13, 21, 32, 34, 51, 64


MS(51, 13, 10, 64, 34, 5, 32, 21)
1
11
MS(51, 13, 10, 64)
6
MS(34, 5, 32, 21)
2
12
16
MS(51, 13)
3
4
MS(10, 64)
7
8
MS(34, 5)
13
14
MS(32, 21)
17
18
MS(51) MS(13)
5
MS(10) MS(64)
9
MS(34) MS(5)
15
MS(32) MS(21)
19
M(13, 51 , 10, 64)
10
M(5, 34 , 21, 32)
20
M(10, 13, 51, 64 , 5, 21, 32, 34)
21
5, 10, 13, 21, 32, 34, 51, 64

Pseudo code

```
Algorithm Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])
    Set i=0, j=0, k=0
    while i<p and j<q do
    begin
        if B[i]≤C[j] then set A[k]=B[i] and increase i
        else set A[k] = C[j] and increase j
        k = k+1
    end
    if i==p then copy C[j..q-1] to A[k..p+q-1]
    else copy B[i..p-1] to A[k..p+q-1]
```

p=4

B: 10, 13, 51, 64

q=4

C: 5, 21, 32, 34

	i	j	k	A[ ]
Before loop	0	0	0	empty
End of 1st iteration	0	1	1	5
End of 2nd iteration	1	1	2	5, 10
End of 3rd	2	1	3	5, 10, 13
End of 4th	2	2	4	5, 10, 13, 21
End of 5th	2	3	5	5, 10, 13, 21, 32
End of 6th	2	4	6	5, 10, 13, 21, 32, 34
				5, 10, 13, 21, 32, 34, 51, 64

Time complexity

Let $T(n)$ denote the time complexity of running merge sort on n numbers.

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n/2) + n & \text{otherwise} \end{cases}$$

See Slide #16, $T(n) = O(n \log n)$

Learning outcomes

- ✓ Understand how divide and conquer works able to analyze complexity of divide and conquer methods by solving recurrence
- ✓ See examples of divide and conquer methods