DTS203TC
Design and Analysis of Algorithms

Lecture 3: Divide and Conquer

Dr. Qi Chen
School of AI and Advanced Computing

Acknowledgment: Some slides are adapted from ones by Prof. Prudence Wong

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

Divide and Conquer

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

- Idea:
  - Divide: A problem instance is ***divided*** into several **smaller** instances of the same problem, ideally of about same size
  - Conquer: The smaller instances are **solved**, typically **recursively**

Recursively finding the sum

12	34	2	9	7	5

Recursively finding the sum

- **Algorithm RecSum(A[1..n])**
- **begin**
- **if (n > 1) then**
- **begin**
- **copy A[1..$\left\lfloor \frac{n}{2} \right\rfloor$] to B[1..$\left\lfloor \frac{n}{2} \right\rfloor$]**
- **copy A[$\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right)$..n] to C[1..$\left\lceil \frac{n}{2} \right\rceil$]**
- **sum1 = RecSum(B[1..$\left\lfloor \frac{n}{2} \right\rfloor$])**
- **sum2 = RecSum(C[1..$\left\lceil \frac{n}{2} \right\rceil$])**
- **return sum1 + sum2**
- **end**
- **else return A[1]**
- **end**

Finding maximum

- Algorithm RecMax(A[1..n])
- begin
- if (n > 1) then
- begin
- copy A[1..$\left\lfloor \frac{n}{2} \right\rfloor$] to B[1..$\left\lfloor \frac{n}{2} \right\rfloor$]
- copy A[$\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right)$..n] to C[1..$\left\lceil \frac{n}{2} \right\rceil$]
- answer1 = RecMax(B[…])
- answer2 = RecMax(C[…])
- return larger of answer1 and answer2
- end
- else return A[1]
- end

Binary Search

- Recall that we have learnt binary search:

- Input: a sequence of n **sorted** numbers $a_1, a_2, \ldots, a_n$; 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

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

- 3 7 11 12 15 19 24 33 41 55

  24

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

- 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, 1, n, X)
return true if X is found,
false otherwise

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 sorted 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

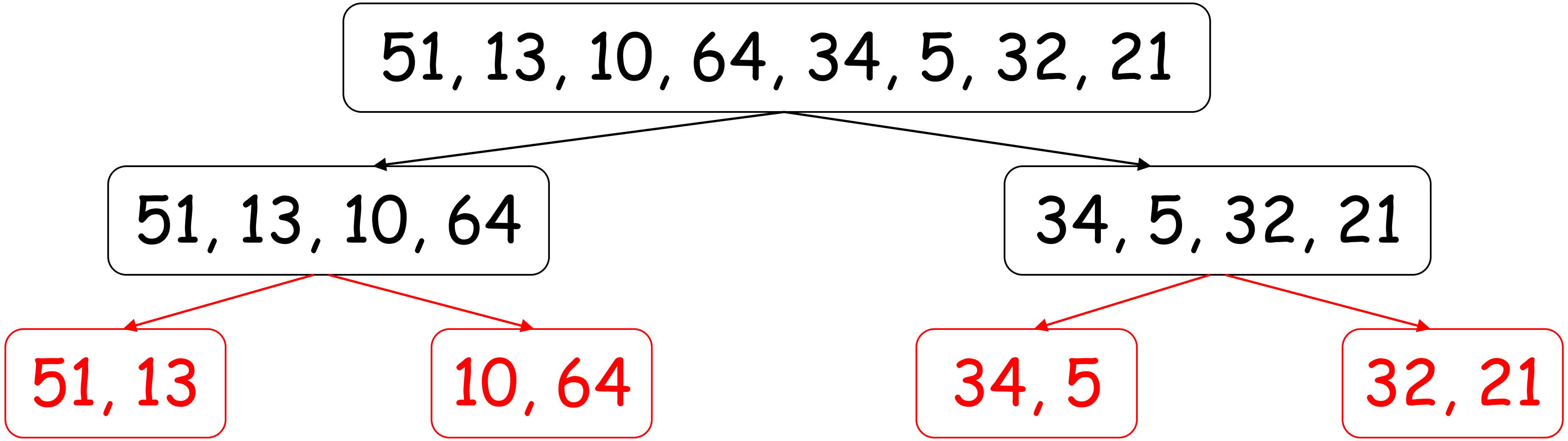
51, 13, 10, 64, 34, 5, 32, 21

51, 13, 10, 64

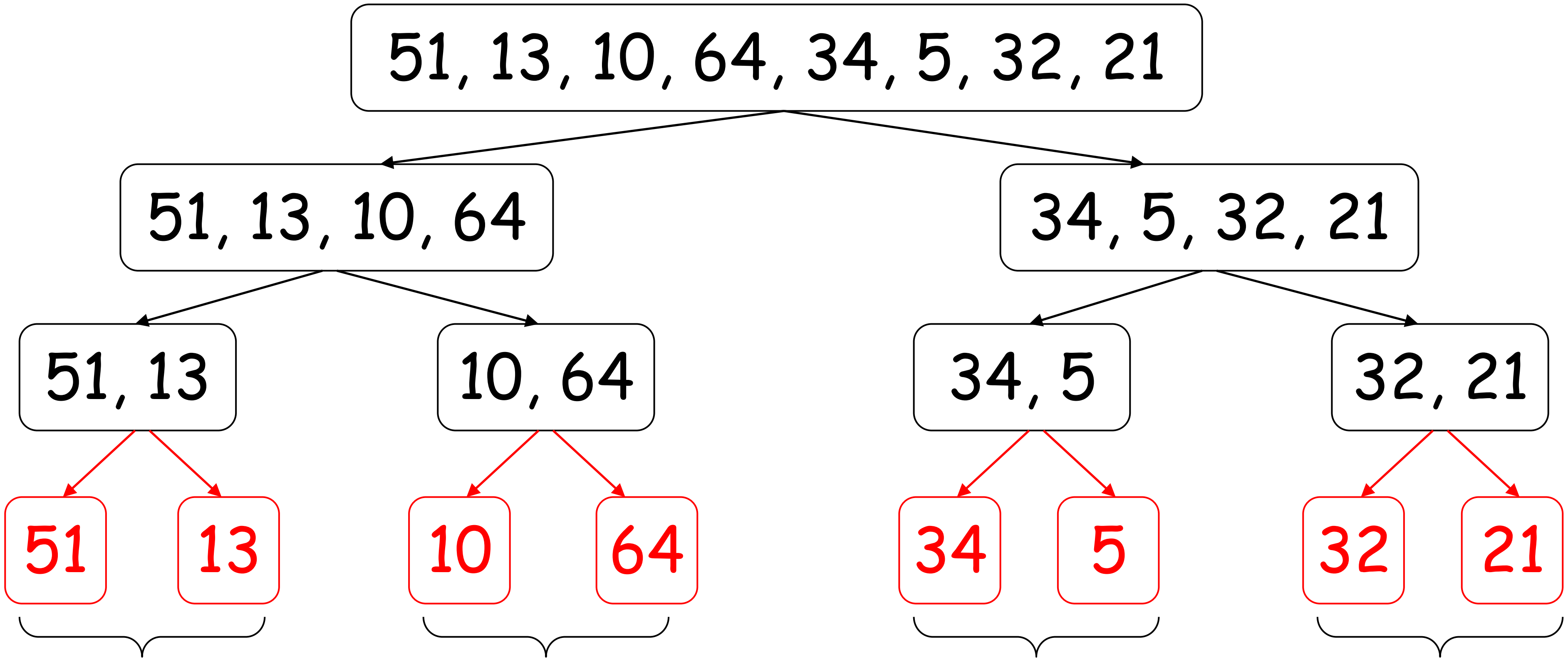
34, 5, 32, 21

divide these 4 numbers into halves

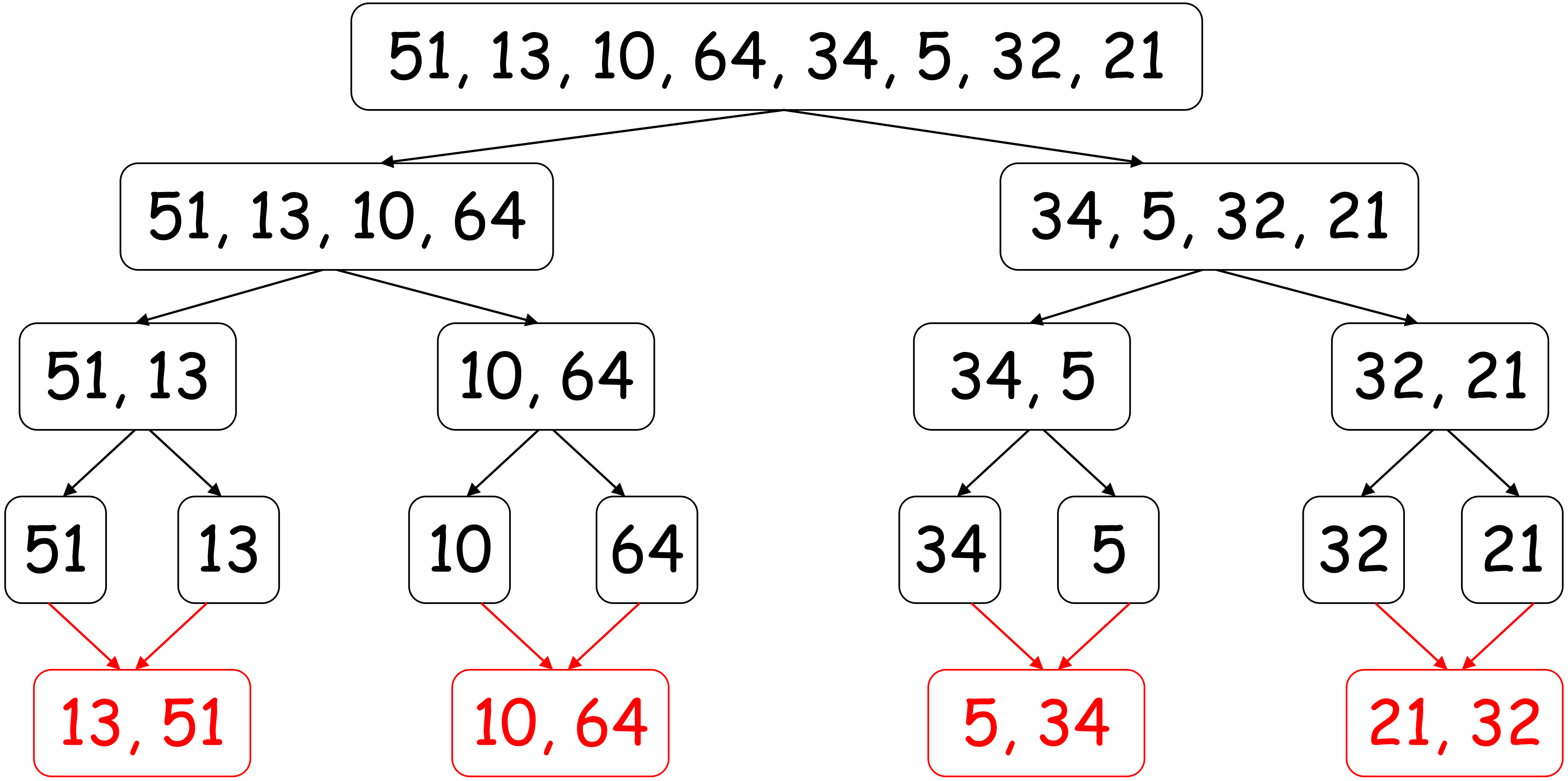
similarly for these 4



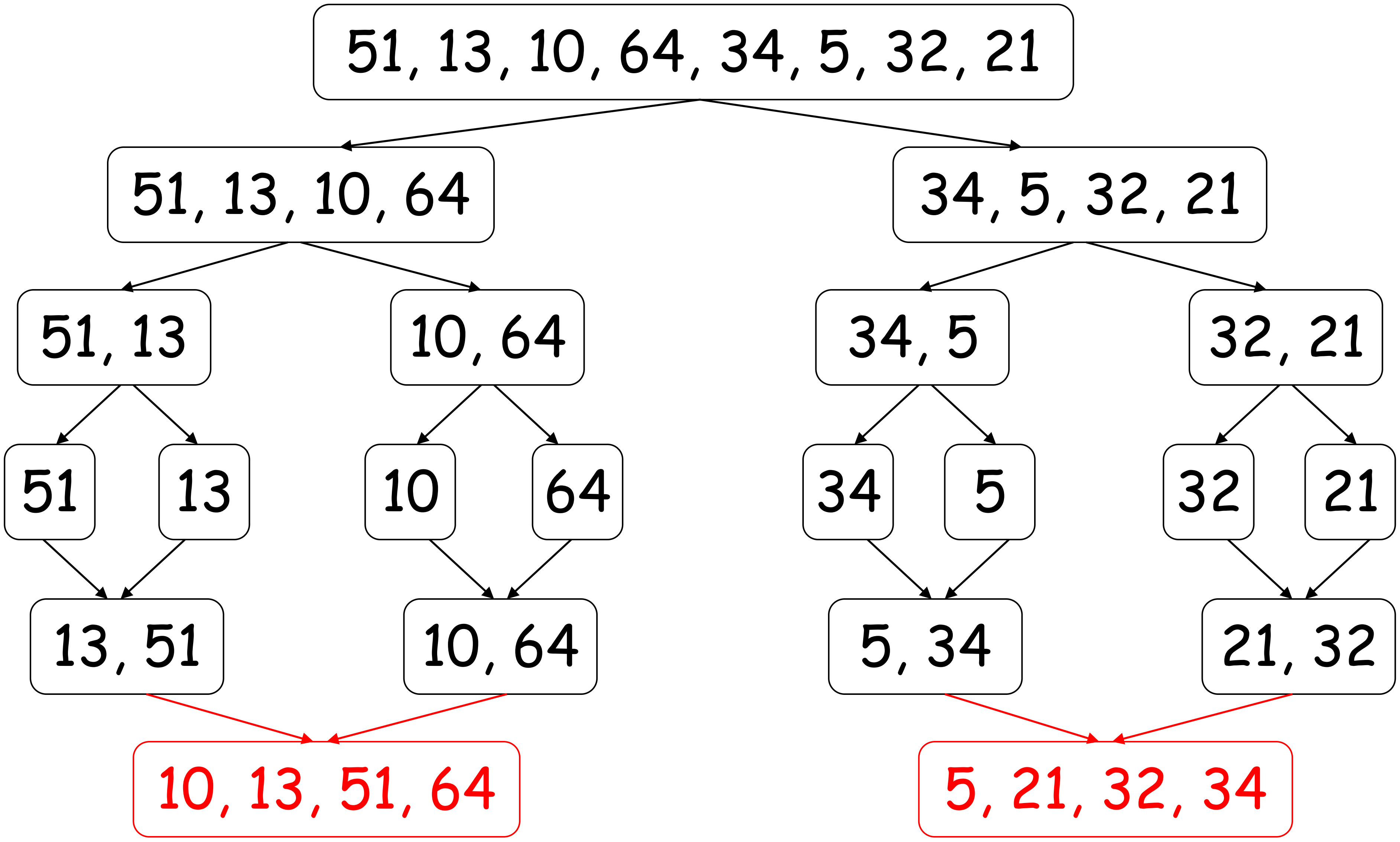
further divide each shorter sequence …
until we get sequence with only 1 number



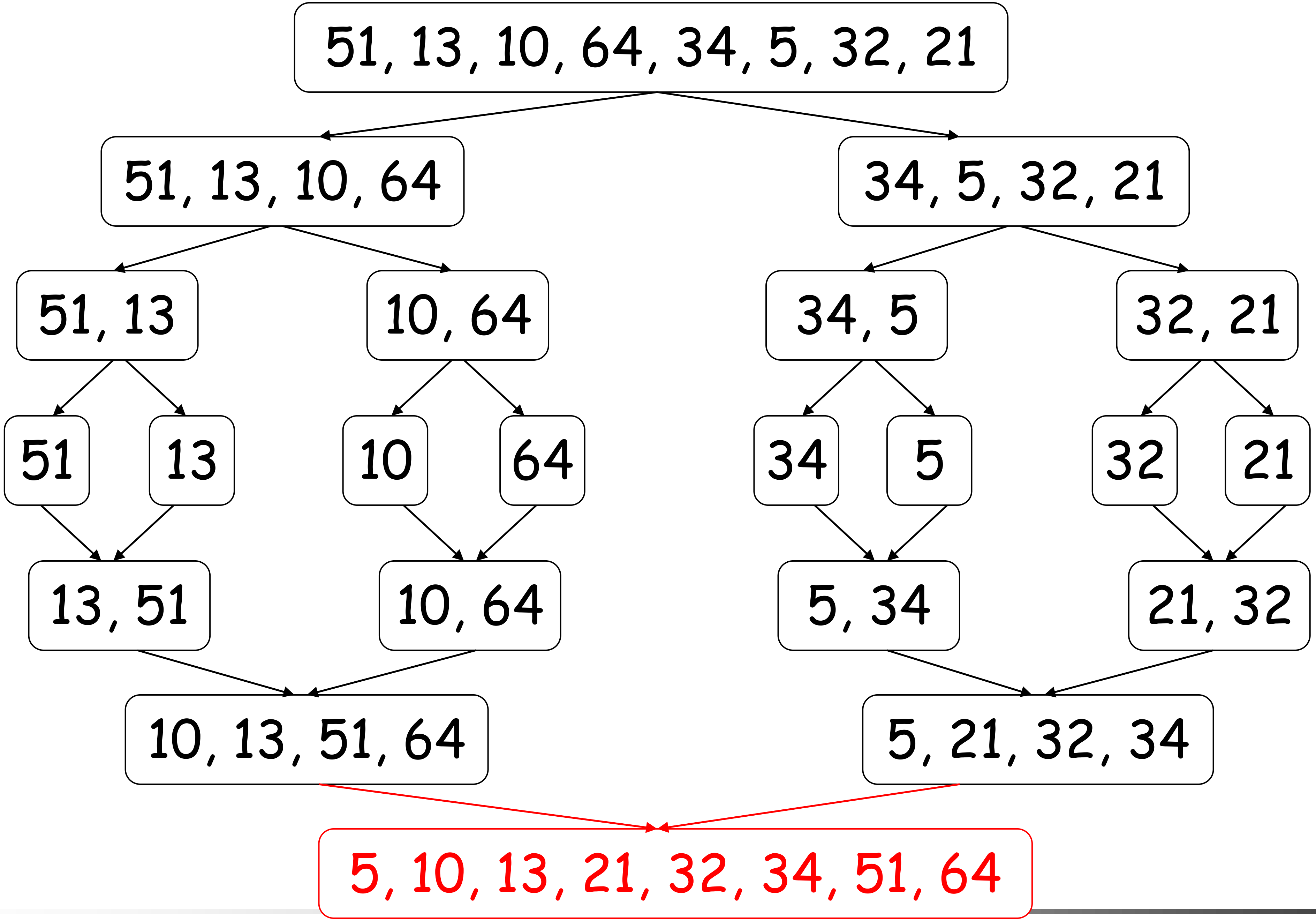
merge pairs of single number into a sequence of 2 sorted numbers



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

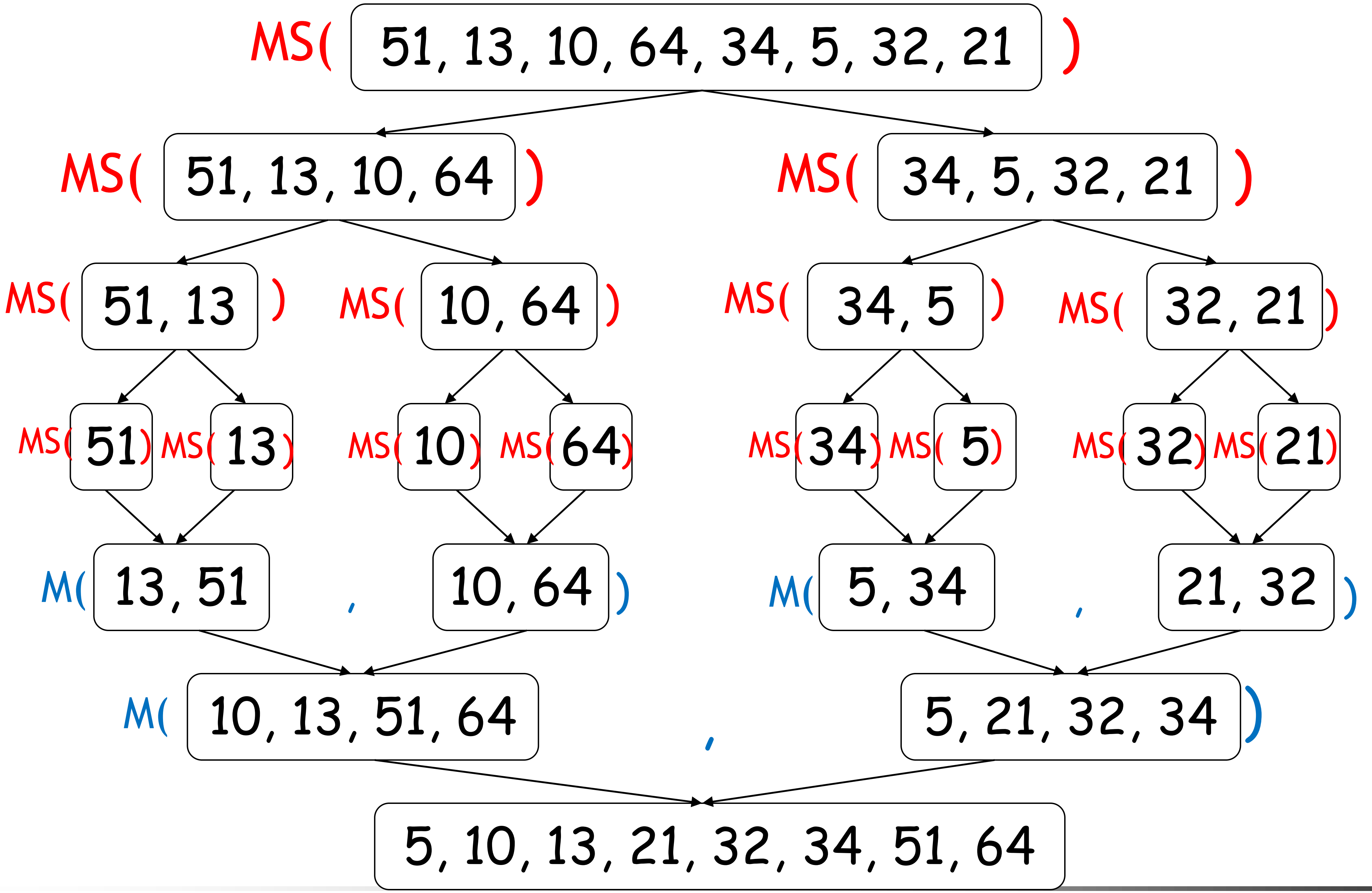
5, 21, 32, 34

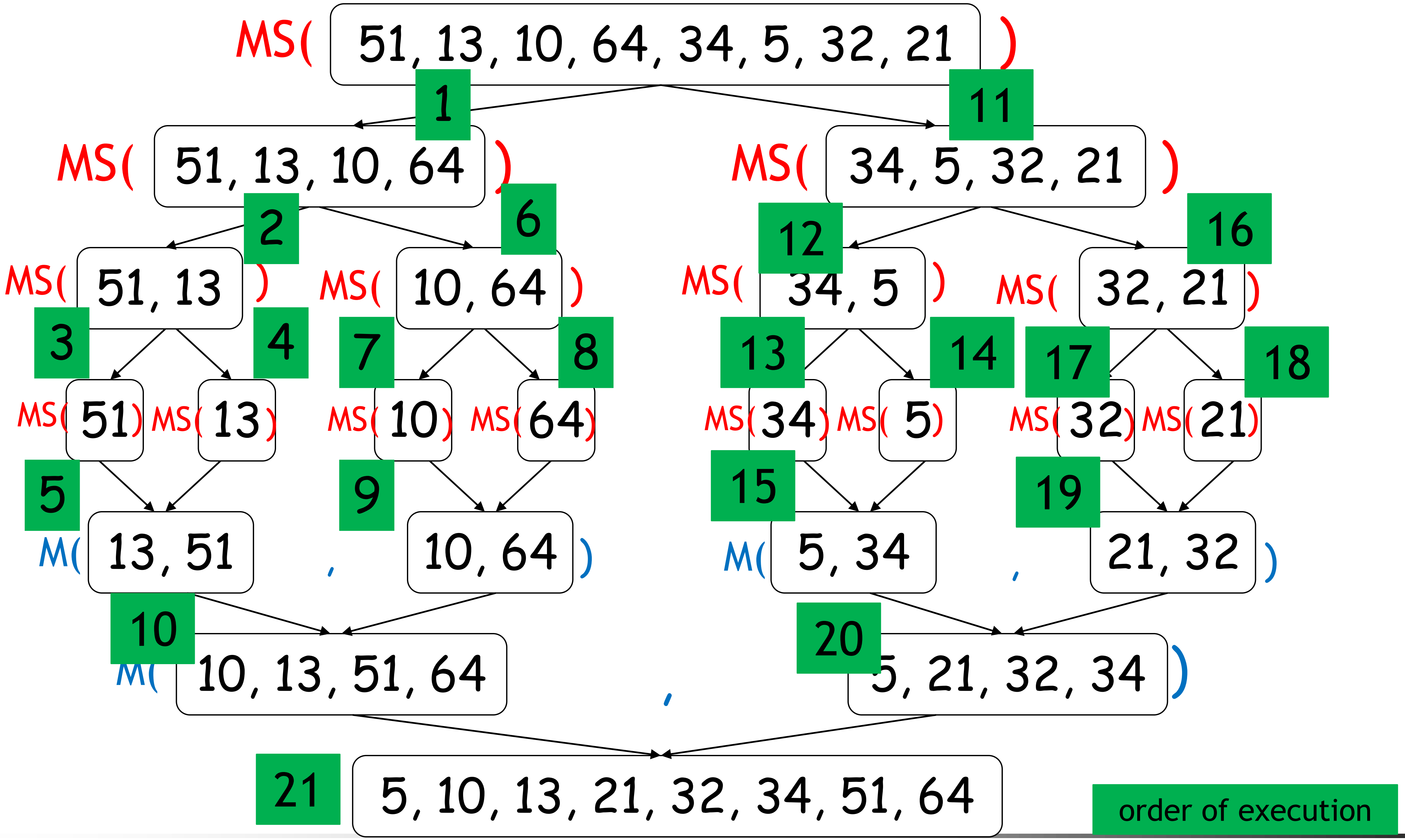
Result: 5, 10, 13, 21, 32, 34, 51, 64

Then we obtain the final sorted sequence

Pseudo code

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



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

Pseudo code

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

p=4

B: 10, 13, 51, 64

q=4

C: 5, 21, 32, 34

	i	j	k	A[ ]
Before loop	1	1	1	empty
End of 1st iteration	1	2	2	5
End of 2nd iteration	2	2	3	5, 10
End of 3rd	3	2	4	5, 10, 13
End of 4th	3	3	5	5, 10, 13, 21
End of 5th	3	4	6	5, 10, 13, 21, 32
End of 6th	3	5	7	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.

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

We call this formula a recurrence.

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

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

Time complexity

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

- ***Make a guess:*** $T(n) \le 2\, n \log n$ (We prove by MI)

For the base case when n=2,
L.H.S = $T(2) = 2\times T(1) + 2 = 4$,
R.H.S = $2 \times 2 \log 2 = 4$
L.H.S $\le$ R.H.S

Time complexity

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

- **Make a guess:** $T(n) \leq 2\ n \log n$ (We prove by MI)

  Assume true for all n'<n [assume $T(\frac{n}{2}) \leq 2 \times (\frac{n}{2}) \times \log(\frac{n}{2})$]

  $T(n) = 2 \times T(\frac{n}{2}) + n$

  $\leq 2 \times \mathbf{(2 \times (\frac{n}{2}) \times \log(\frac{n}{2}))} + n$

  $= 2\ n\ (\log n - 1) + n$

  $= 2\ n \log n - 2n + n$

  $\leq 2\ n \log n$

Example

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

- **Guess: $T(n) \le 2 \log n$**

Example

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

- **Guess: $T(n) \leq 2n - 1$**

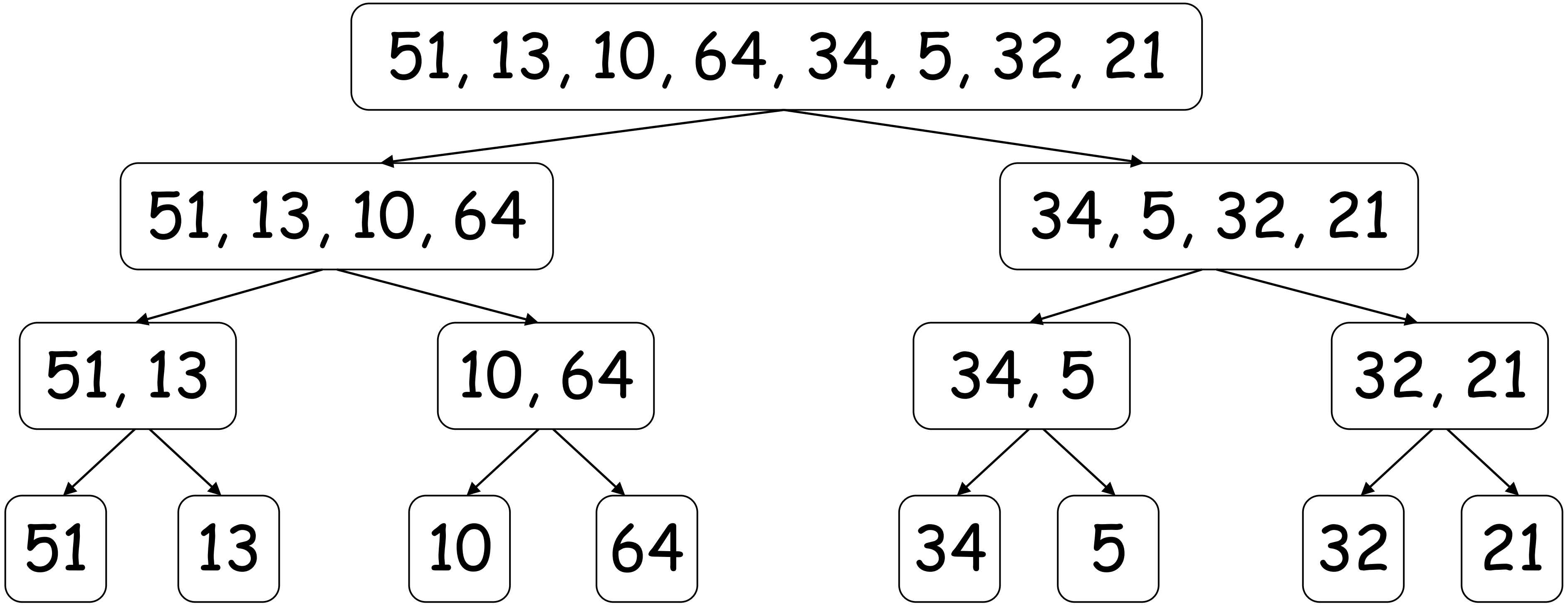
Summary

- Depending on the recurrence, we can guess the order of growth

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

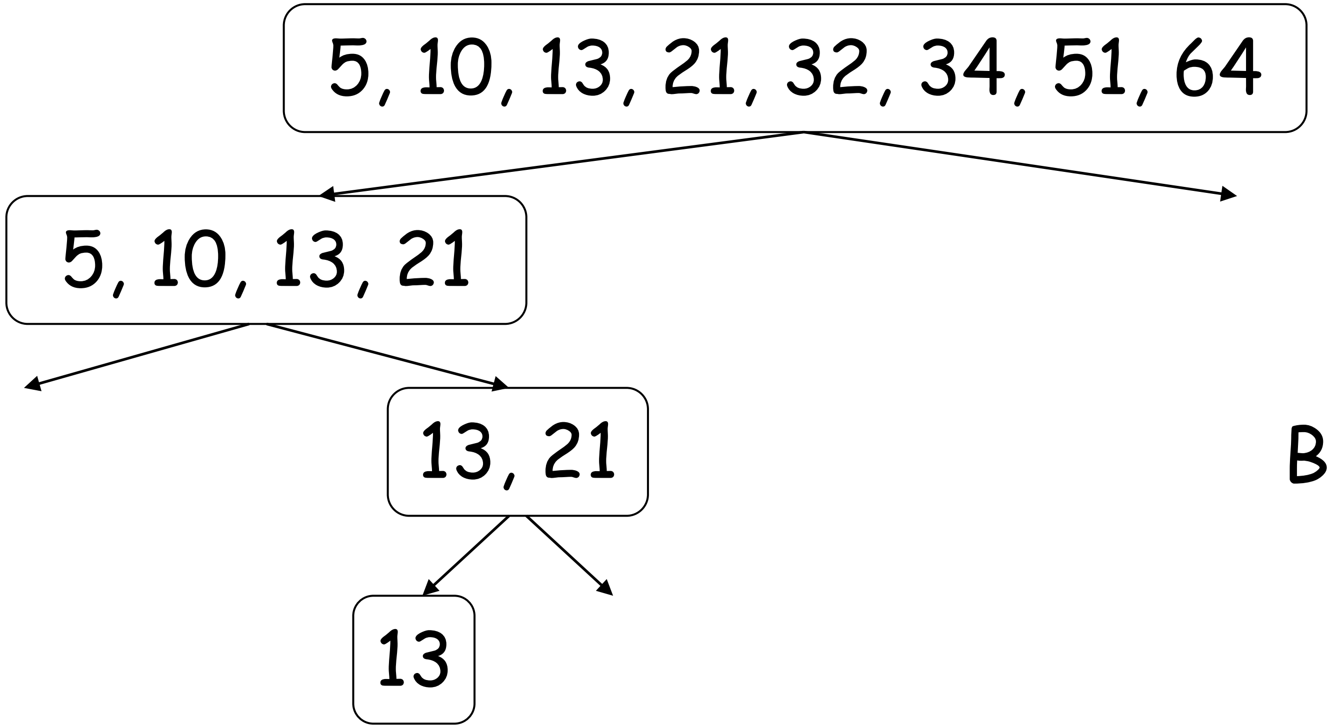
Recursion-tree method

Merge Sort



How many levels?
How many nodes? (rectangles)
How much time is needed for each level/node?

Recursion-tree method

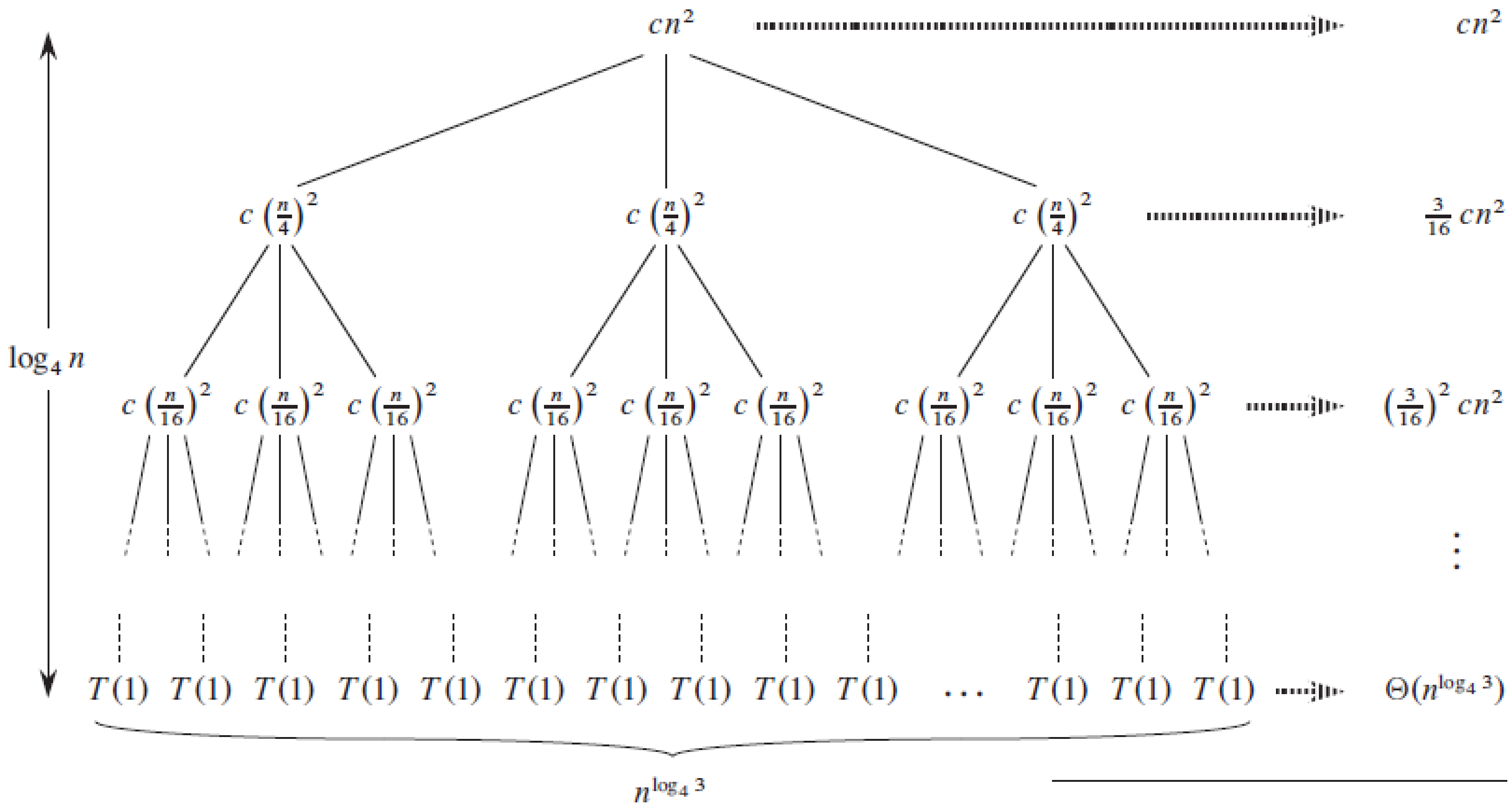


Binary Search

How many levels?
How many nodes? (rectangles)
How much time is needed for each level/node?

Recursion-tree method

- $T(n) = 3T(\frac{n}{4}) + cn^2$



The master method

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Exercises

$T(n) = aT(n/b) + f(n)$

$\Theta(n^{\log_b a})$? $\Theta(f(n))$

- $T(n) = 9T(n/3) + n$ — $\Theta(n^2)$
- $T(n) = T(2n/3) + 1$ — $\Theta(logn)$
- $T(n) = 3T(n/4) + nlogn$ — $\Theta(nlogn)$
- $T(n) = T(n/2) + \Theta(1)$ — $\Theta(logn)$
- $T(n) = 2T(n/2) + \Theta(1)$ — $\Theta(n)$
- $T(n) = 2T(n/2) + \Theta(n)$ — $\Theta(nlogn)$
- $T(n) = 8T(n/2) + \Theta(n^2)$ — $\Theta(n^3)$
- $T(n) = 7T(n/2) + \Theta(n^2)$ — $\Theta(n^{log7})$
- ▲ $T(n) = 2T(n/2) + nlogn$ — $\Theta(nlog^2n)$

Learning outcomes

- Understand how divide and conquer works

- See examples of divide and conquer methods

- solving recurrence
  - Substitution method: Guess a bound + Mathematic induction
  - Recursion-tree method: covert the recurrence into a tree
  - Master method