

Data Structures and Algorithms

Prepared by Prof. Shereen Taie

IT251
Information Technology Department

Lecture 4





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY

Data Structures and Algorithms

Lecture 4

IT251

Information Technology Department

Prepared by
Prof. Shereen Taie

www.midocean.ae

Contents:

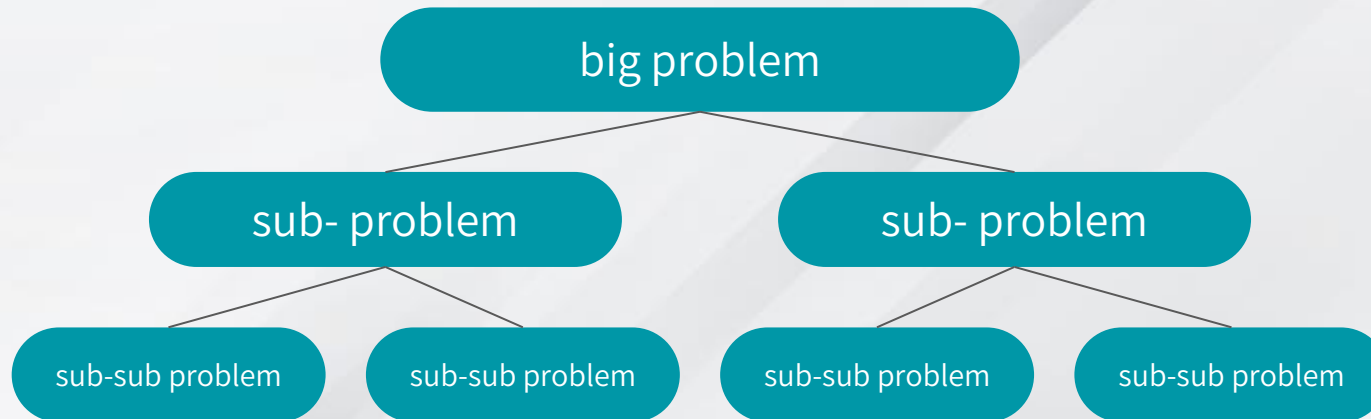
- Divide And Conquer
 - Divide-and-Conquer Examples
- Mergesort
 - Mergesort: Pseudocode
 - Mergesort: Recursive Calls
 - Mergesort: Merge Steps
 - Mergesort: Is It Fast?
 - MERGESORT: $O(n \log n)$ PROOF
 - Mergesort Recursion Tree
- Recurrence Relations
 - Recurrence Relations Examples

DIVIDE AND CONQUER (Merge Sort - Recurrence Relations)

DIVIDE AND CONQUER

- **An algorithm design paradigm:**

1. break up a problem into smaller subproblems
2. solve those subproblems *recursively*
3. combine the results of those subproblems to get the overall answer



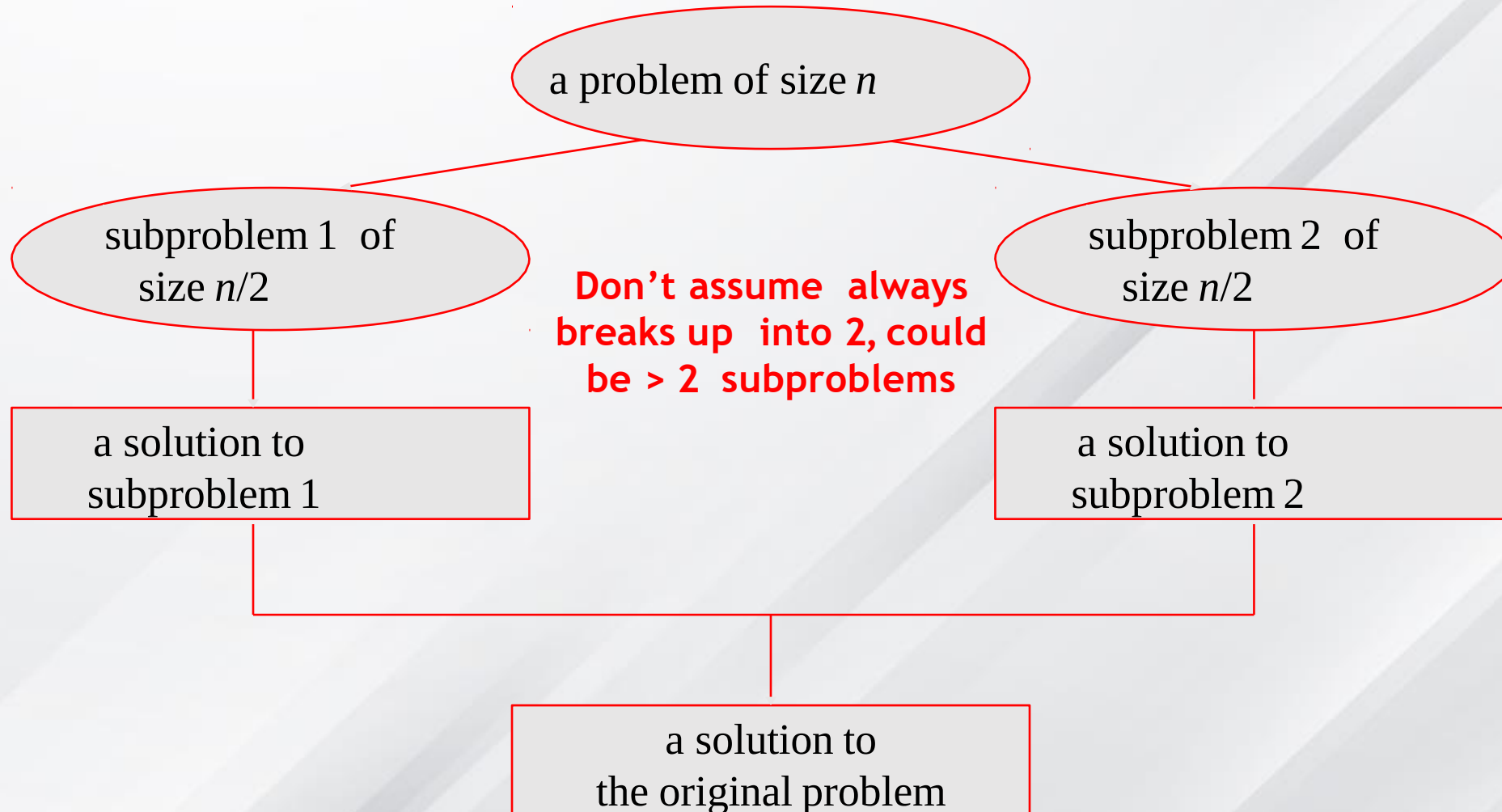
Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary tree traversals
- Binary search
- Multiplication of large integers
- Matrix multiplication: Strassen's algorithm

Divide and conquer

- Three steps are involved:
 - ✓ Divide the problem into a number of sub-problems
 - ✓ Conquer the sub-problems (solve them)
 - ✓ Combine the sub-problem solutions to get the solution to the original problem
- Note: often the “conquer” step is done recursively.

Divide-and-conquer technique



Divide and Conquer

Based on dividing problem into sub-problems

Approach

- Divide problem into smaller sub-problems
 1. Sub-problems must be of same type
 2. Sub-problems do not need to overlap
- Conquer by solving sub-problems recursively.
- If the sub-problems are small enough, solve them in brute force fashion
- Combine the solutions of sub-problems into a solution of the original problem.
- Usually contains two or more recursive calls.

Divide-and-conquer Algorithms

- For Divide-and-Conquer algorithms the running time is mainly affected by 3 criteria:

The number of sub-problems

The ratio of initial problem size to sub-problem size.

The number of steps required to:

divide the initial problem

combine sub-solutions.

Analyzing Divide-and Conquer Algorithms

The master theorem applies to recursive algorithms of the form:

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

Where,

n is the size of the input,

a is the number of recursive calls,

b is the factor by which the input is divided, and

f(n) is the non-recursive work done at each level (time to divide the problem of size n, time to combine the sub-problem solutions).

Chapter
02

Divide-and Conquer Example

Merge Sort



Merge sort:

- Merge sort is a sorting technique based on divide and conquer technique.
- It has a worst-case time complexity of $O(n \log n)$.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Merge sort:

- Sorting Problem:

- Sort a sequence of n elements into non decreasing order (For example, 1, 3, 3, 6, 8, 9 **are in non-decreasing order**).

- Divide:

- Divide the n -element sequence 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 sequence

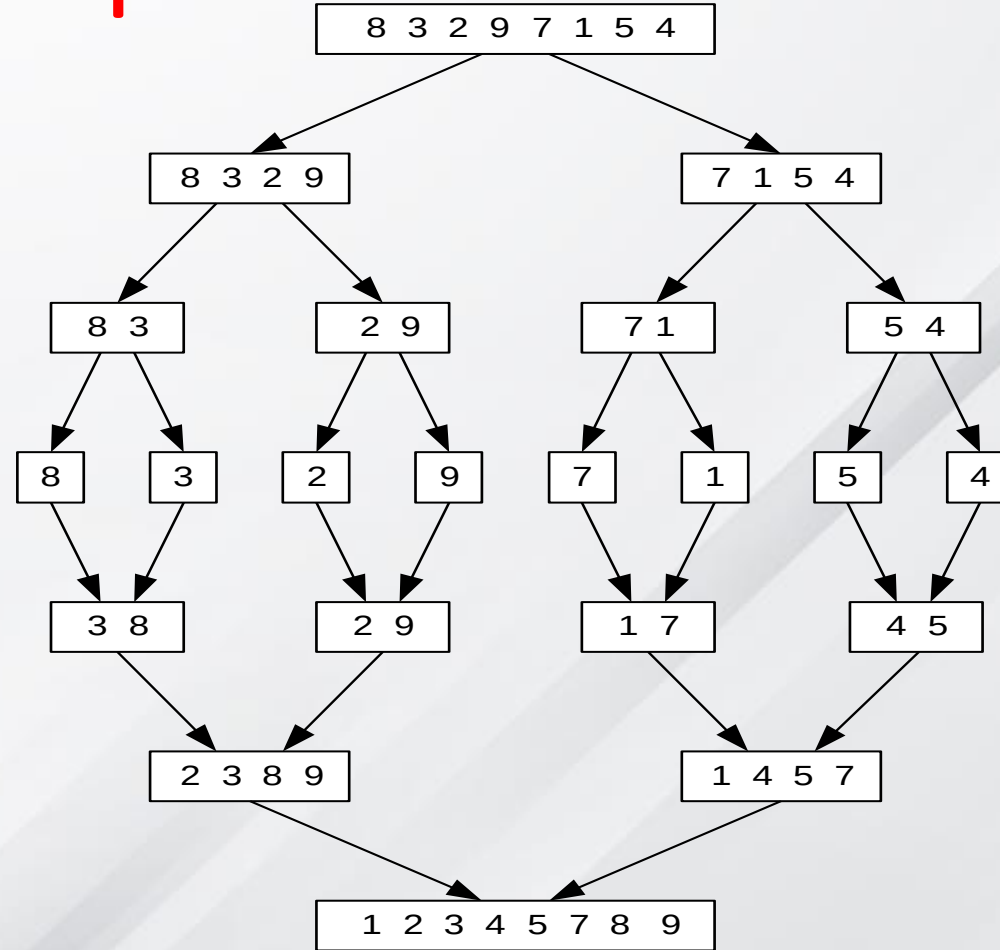
Chapter
02

MERGESORT

Algorithm, Example



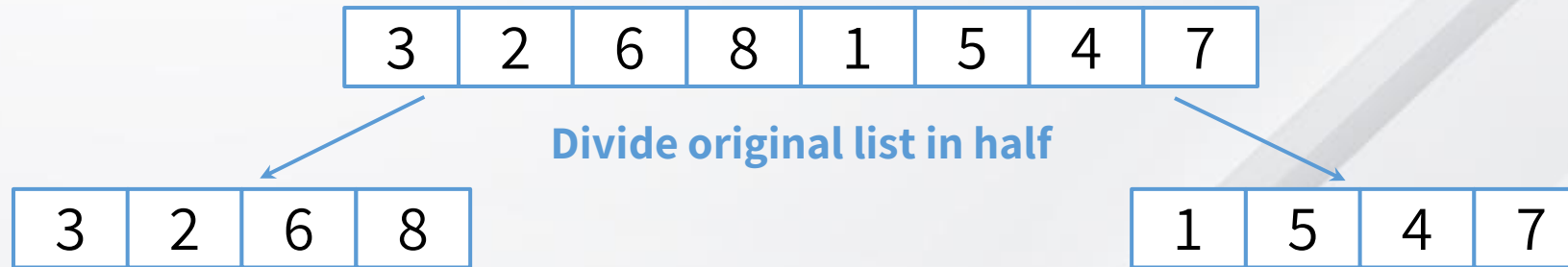
Mergesort Example:



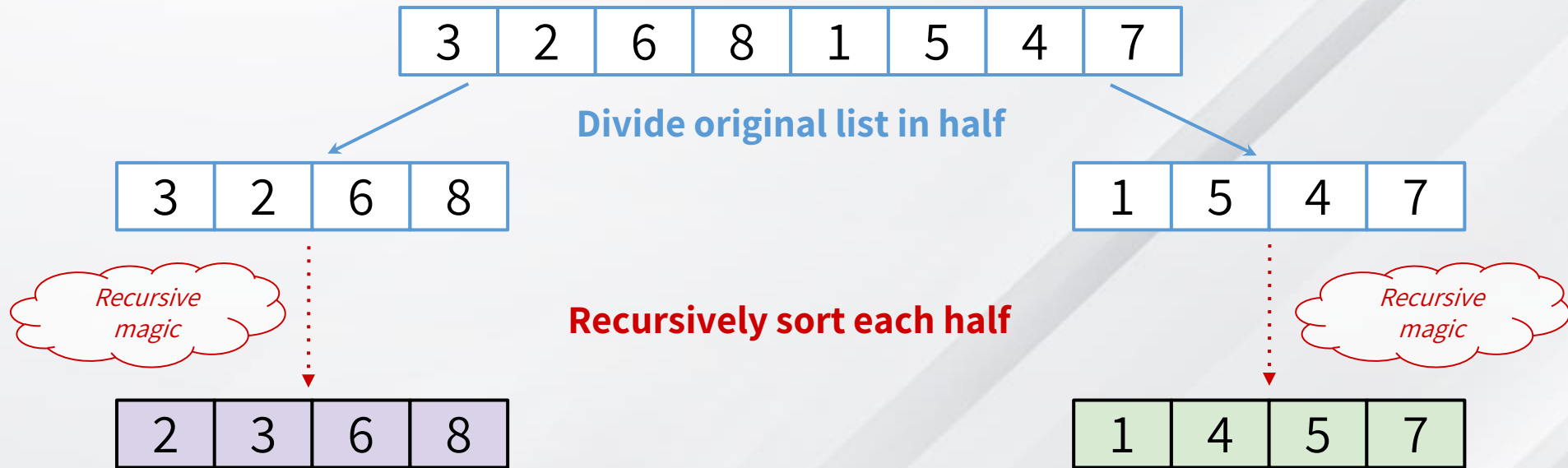
Mergesort:

3	2	6	8	1	5	4	7
---	---	---	---	---	---	---	---

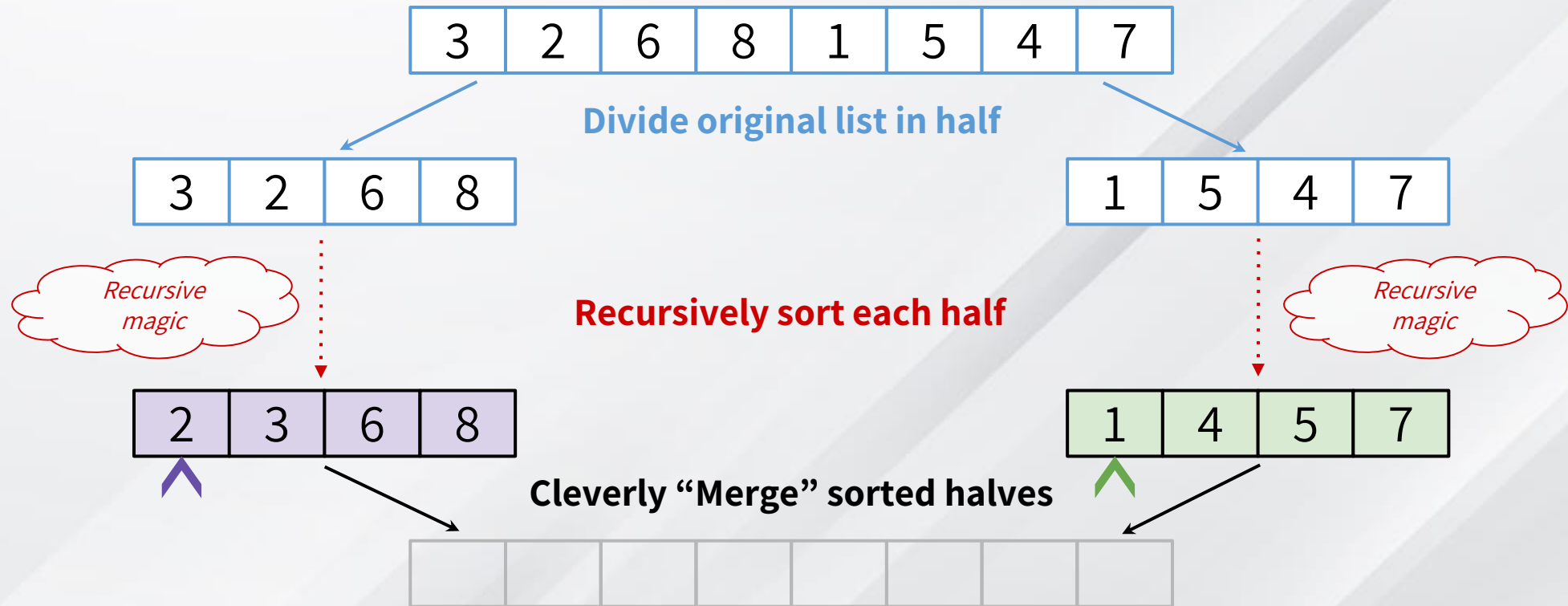
Mergesort:



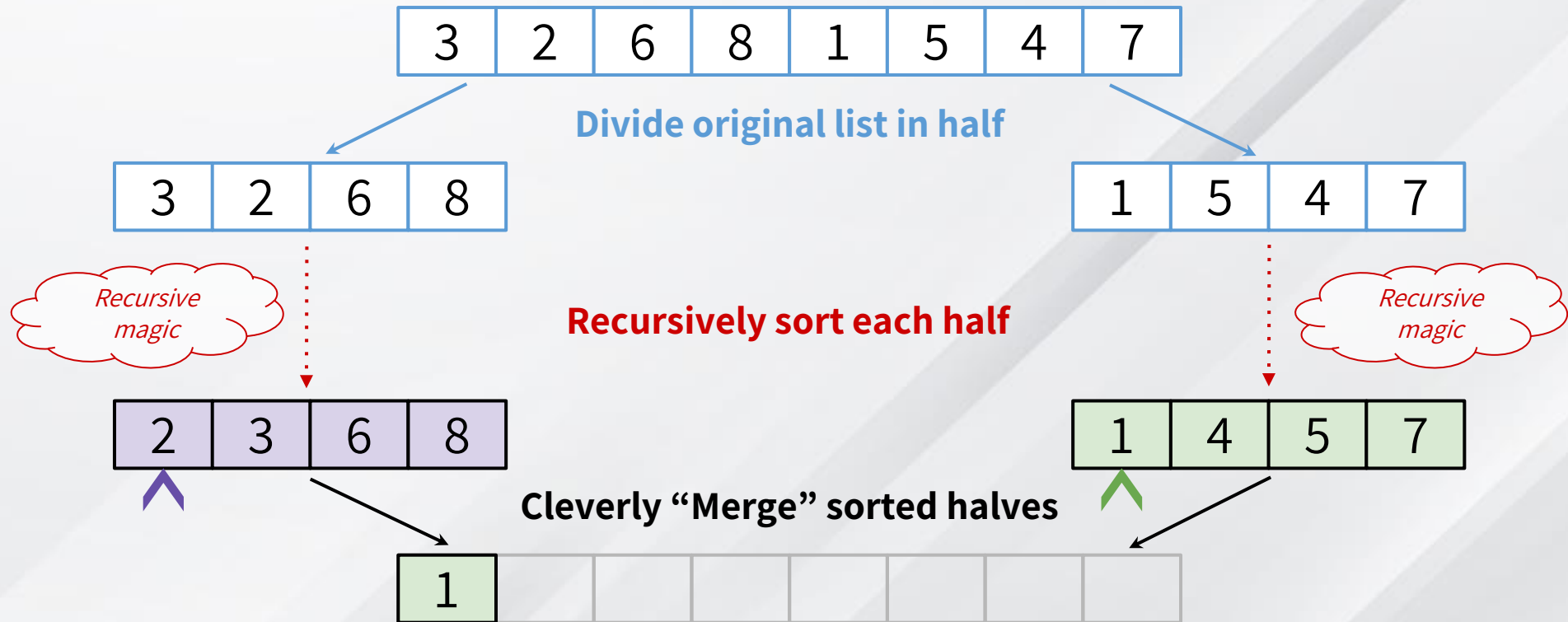
Mergesort:



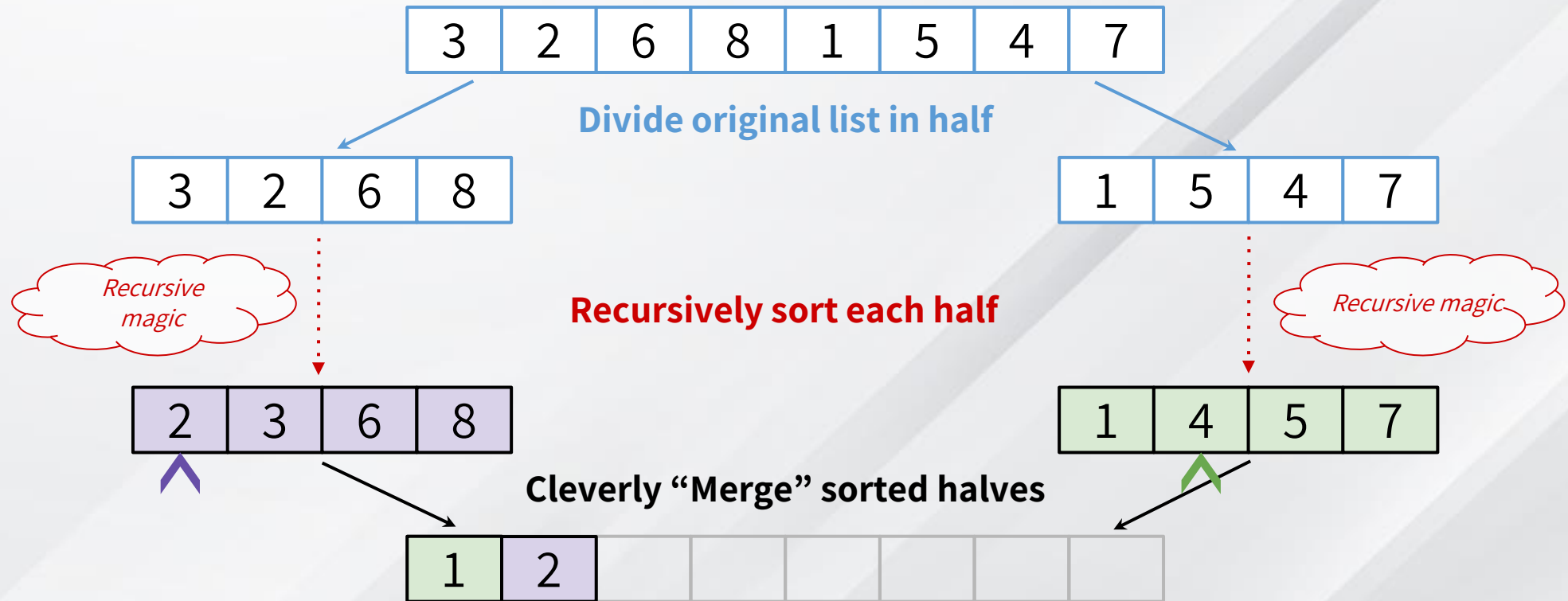
Mergesort:



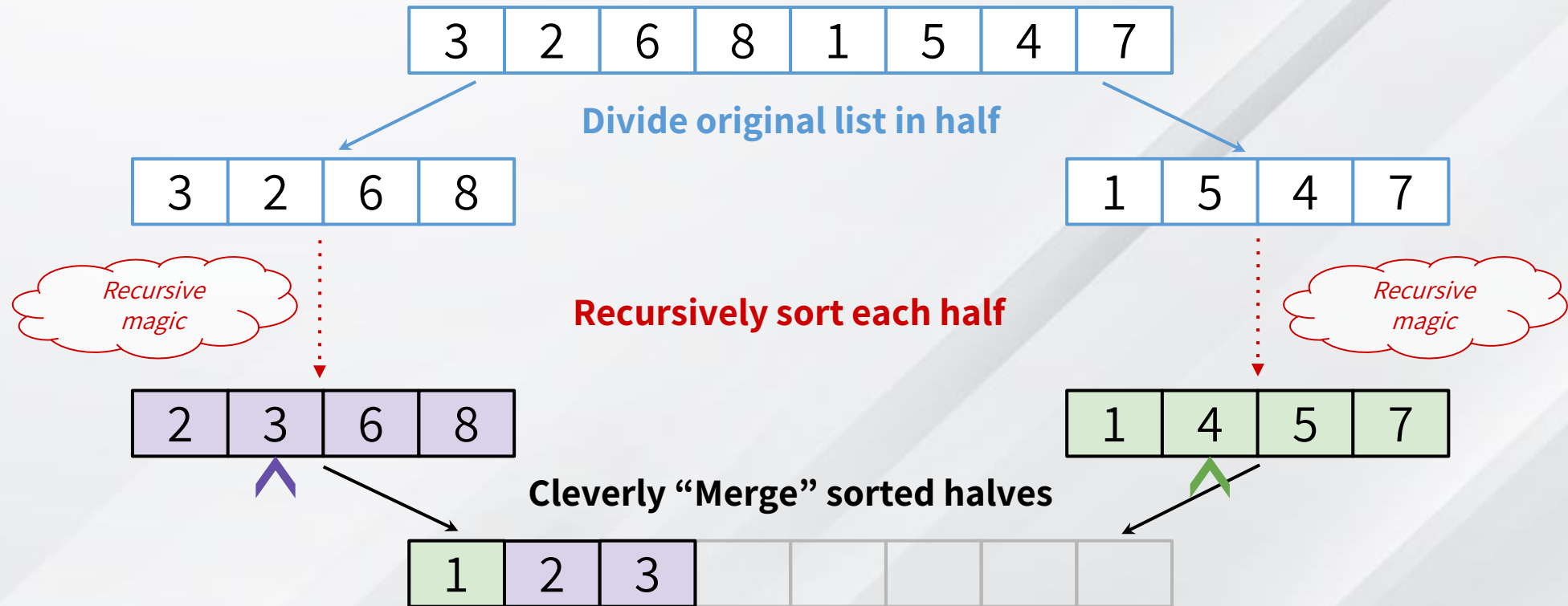
Mergesort:



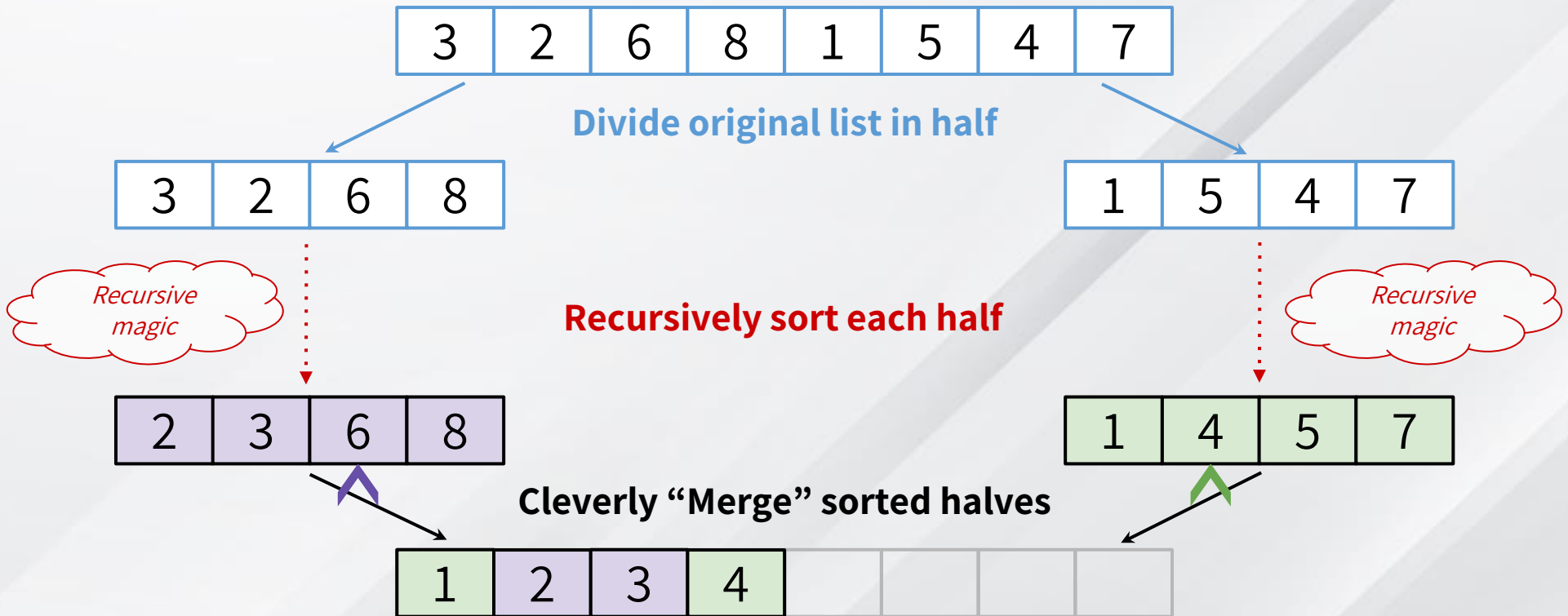
Mergesort:



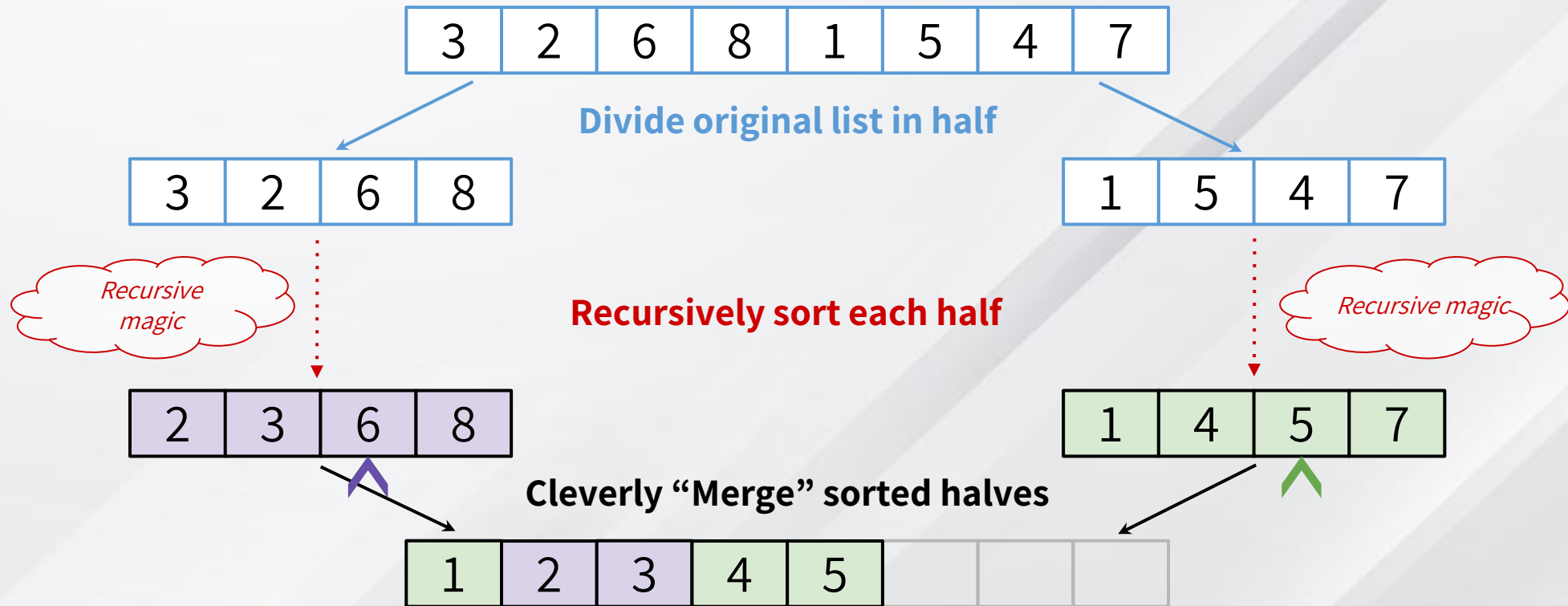
Mergesort:



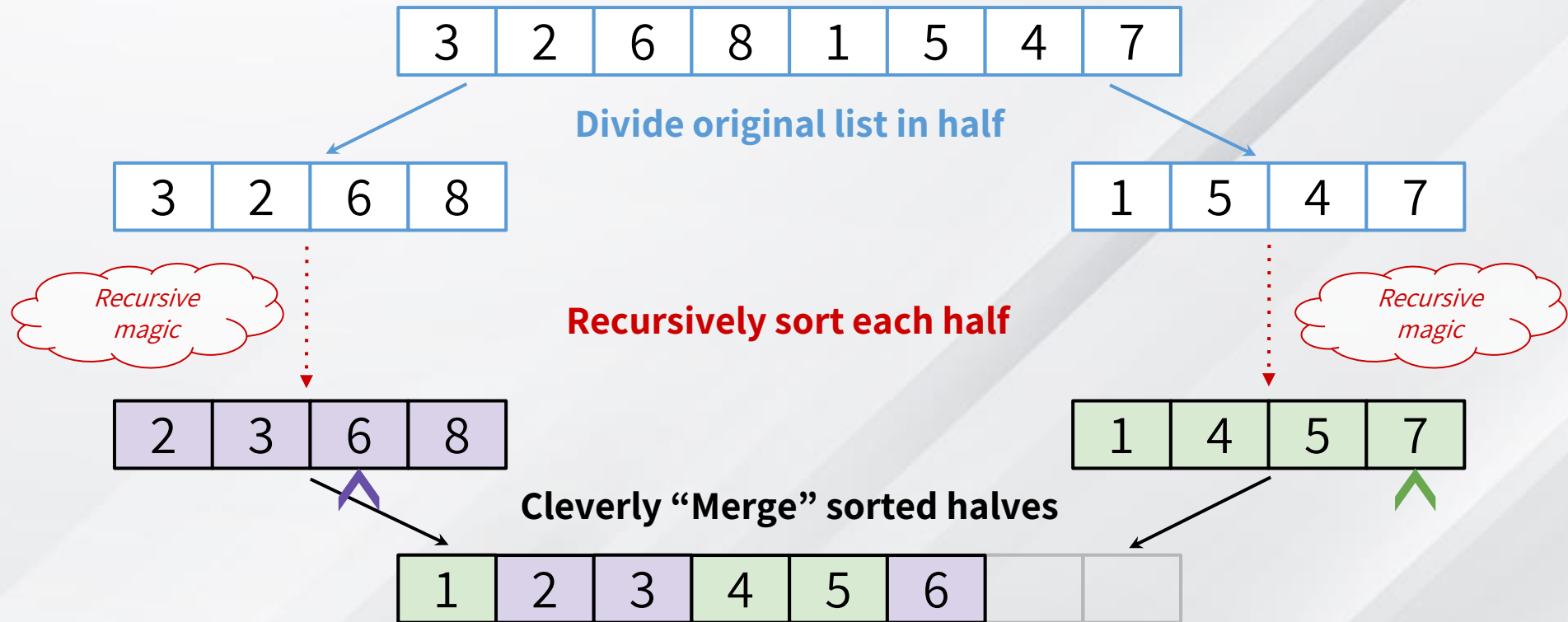
Mergesort



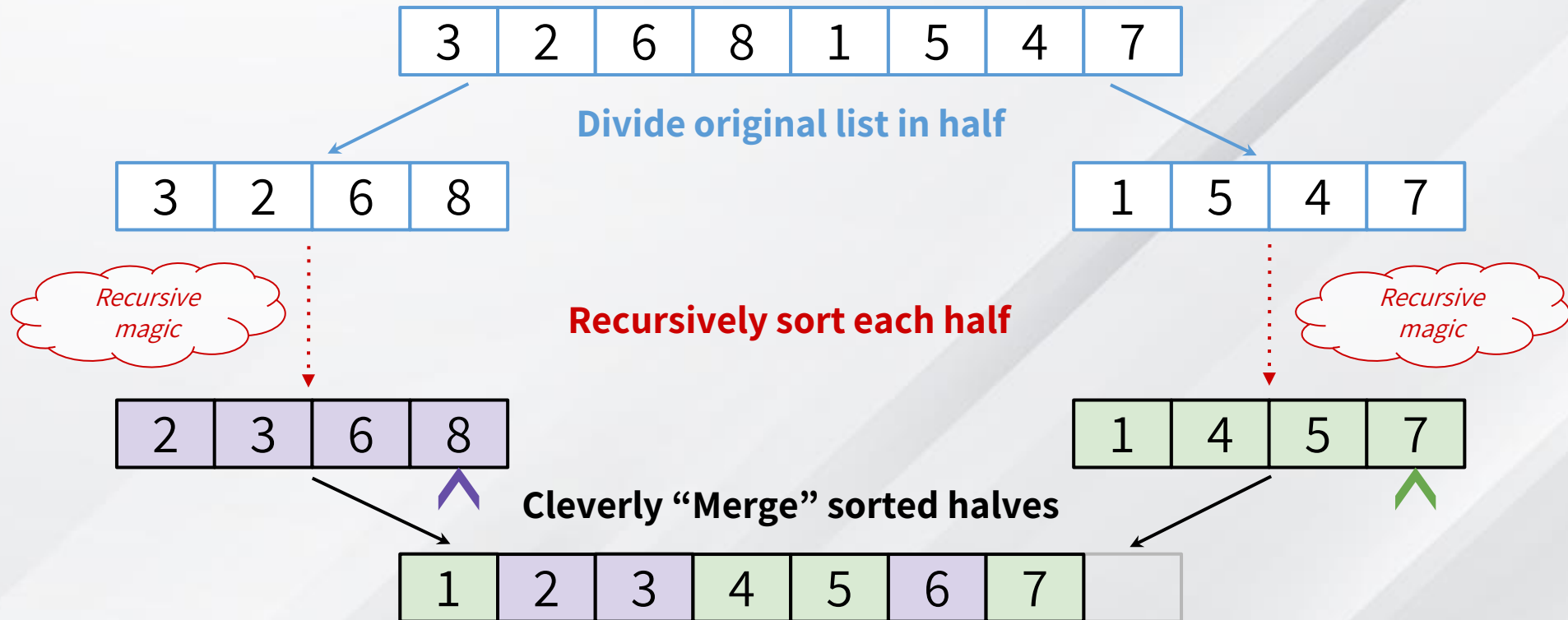
Mergesort



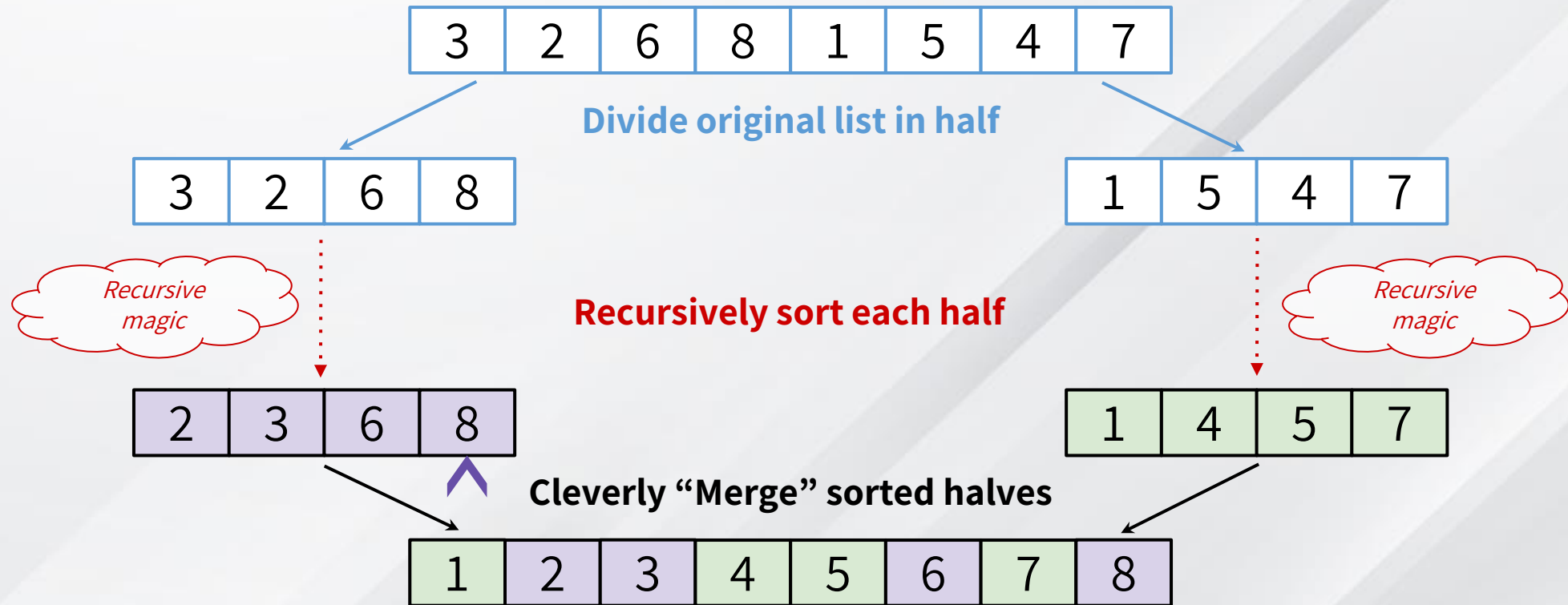
Mergesort



Mergesort



Mergesort



Mergesort: Pseudocode

Intuition: Divide and Conquer. If you sort your left and right halves, it's easier to “Merge” them into a sorted list.

MERGESORT(A):

Mergesort: Pseudocode

Intuition: Divide and Conquer. If you sort your left and right halves, it's easier to “Merge” them into a sorted list.

```
MERGESORT(A):  
  n = len(A)  
  if n <= 1:  
    return A
```

Mergesort: Pseudocode

Intuition: Divide and Conquer. If you sort your left and right halves, it's easier to “Merge” them into a sorted list.

```
MERGESORT(A):  
    n = len(A)  
    if n <= 1:  
        return A  
    L = MERGESORT(A[0:n/2])
```

Mergesort: Pseudocode

Intuition: Divide and Conquer. If you sort your left and right halves, it's easier to “Merge” them into a sorted list.

```
MERGESORT(A):  
    n = len(A)  
    if n <= 1:  
        return A  
    L = MERGESORT(A[0:n/2])  
    R = MERGESORT(A[n/2:n])
```

Mergesort: Pseudocode

Intuition: Divide and Conquer. If you sort your left and right halves, it's easier to “Merge” them into a sorted list.

```
MERGESORT(A):  
  n = len(A)  
  if n <= 1:           If A has length 1,  
                        It is already sorted!  
    return A  
  L = MERGESORT(A[0:n/2])  
  R = MERGESORT(A[n/2:n])  
  
  return MERGE(L,R)
```

For today, let's assume that n is a power of 2.

Sort the left half

Sort the right half

Merge the two halves

Mergesort: Pseudocode

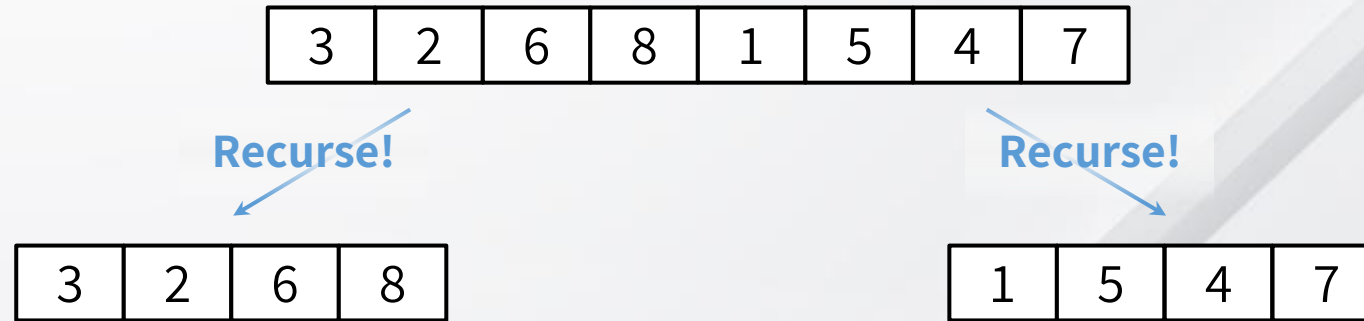
```
MERGESORT(A):  
    n = len(A)  
    if n <= 1:  
        return A  
    L = MERGESORT(A[0:n/2])  
    R = MERGESORT(A[n/2:n])  
    return MERGE(L,R)
```

```
MERGE(L,R):  
    result = length n array  
    i = 0, j = 0  
    for k in [0,...,n-1]:  
        if L[i] < R[j]:  
            result[k] = L[i]  
            i += 1  
        else:  
            result[k] = R[j]  
            j += 1  
    return result
```

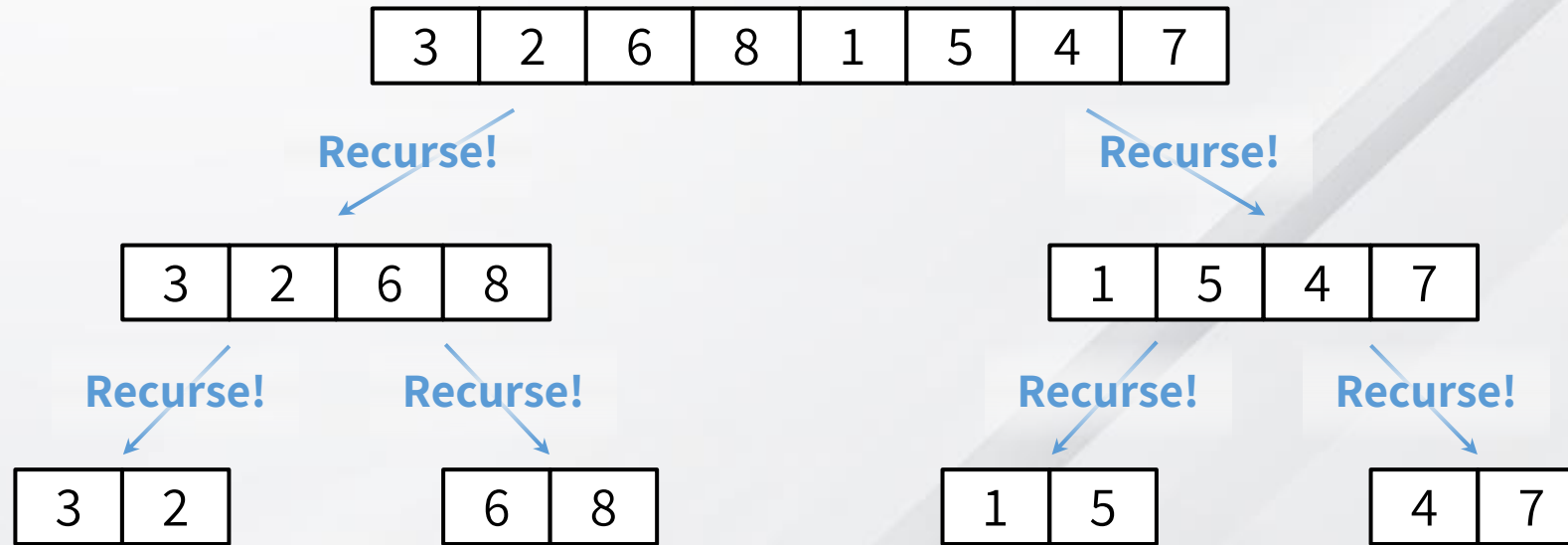
Mergesort: Recursive Calls

3	2	6	8	1	5	4	7
---	---	---	---	---	---	---	---

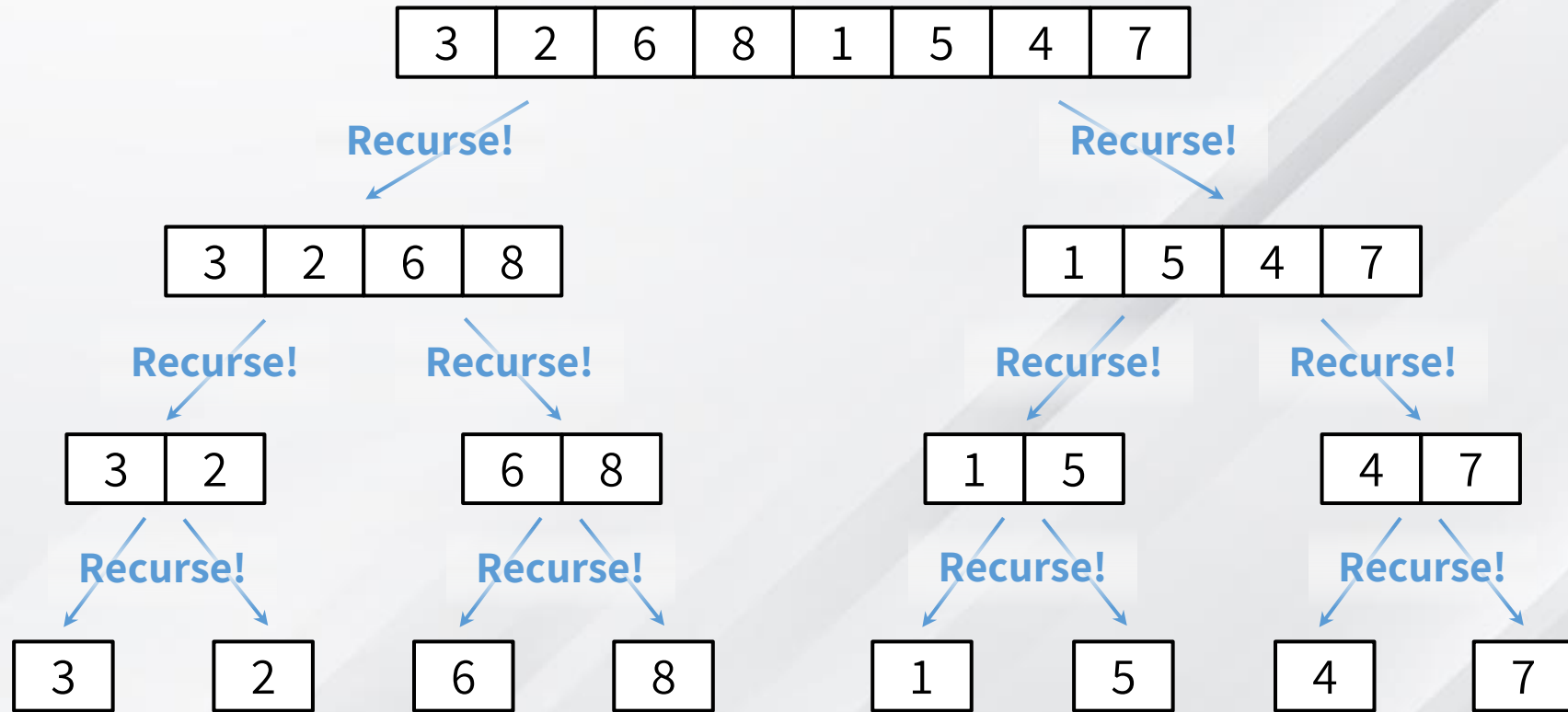
Mergesort: Recursive Calls



Mergesort: Recursive Calls



Mergesort: Recursive Calls



**This is where
we hit our
base case!**

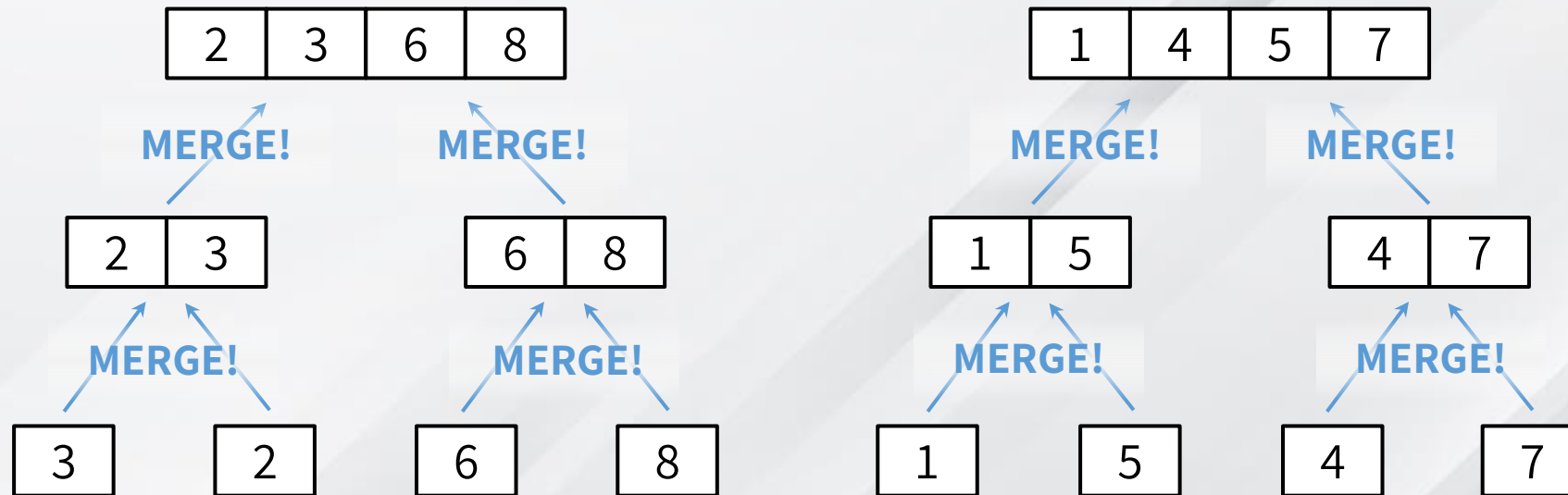
Mergesort: Recursive Calls

3 2 6 8 1 5 4 7

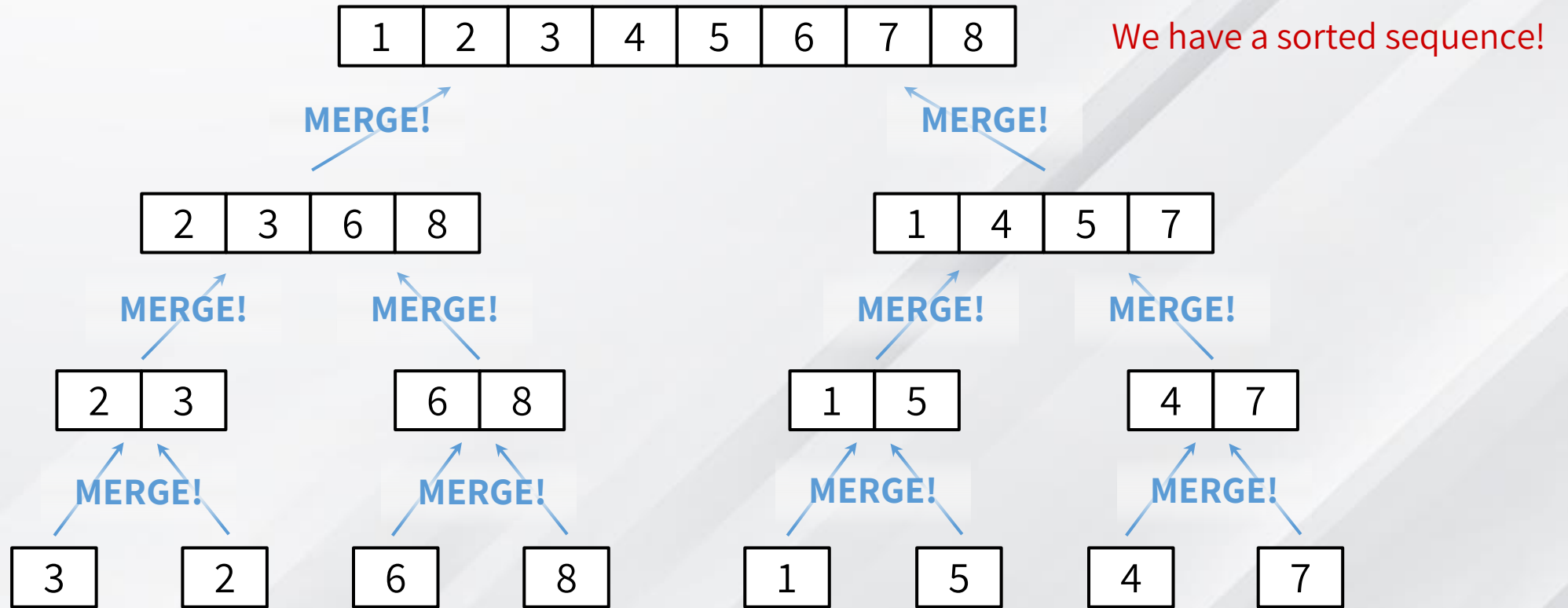
Mergesort: Merge Steps



Mergesort: Merge Steps



Mergesort: Merge Steps



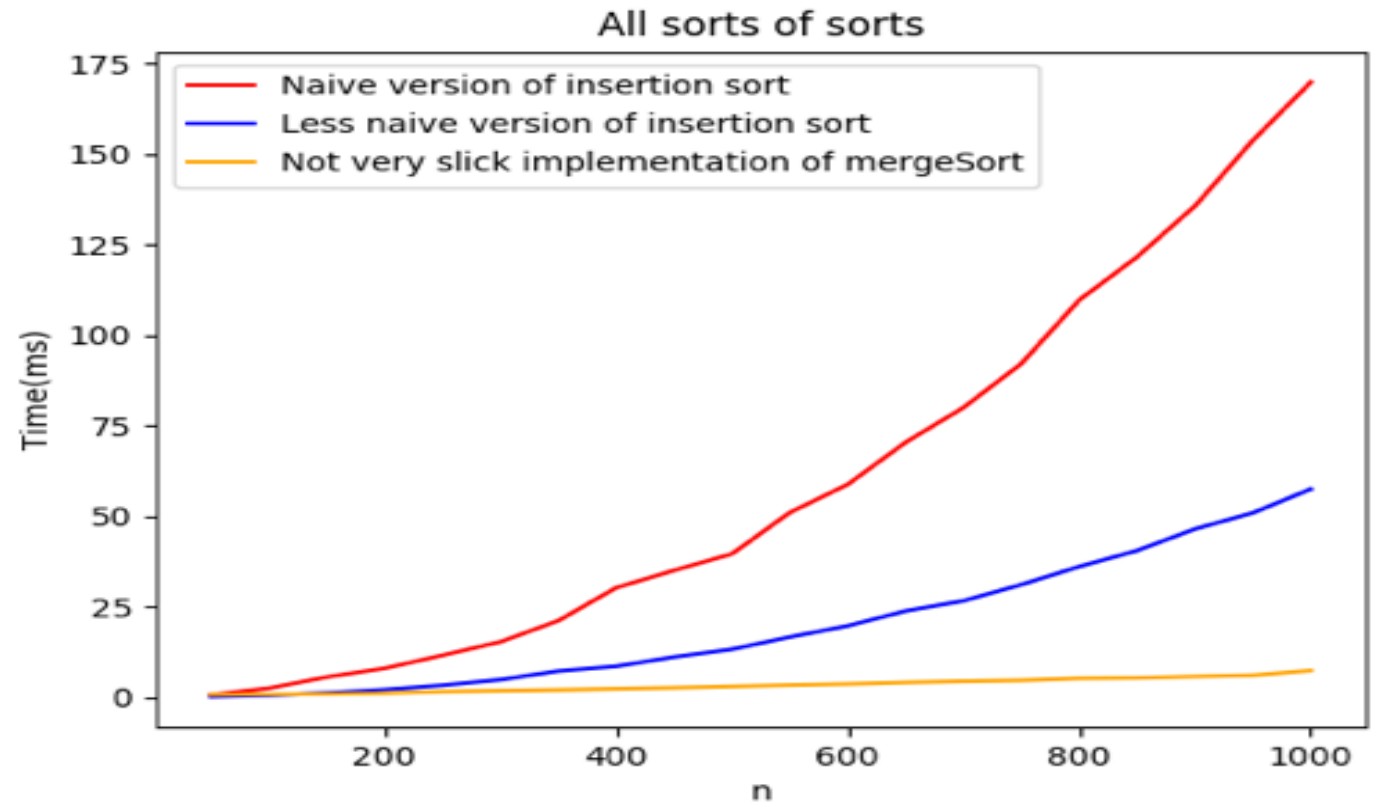
Two questions

1. Does this work?
2. Is it fast?

Empirically:

1. Seems to work.
2. Seems fast.

Python says...



Mergesort: Is It Fast?

```
MERGESORT(A):  
    n = len(A)  
    if n <= 1:  
        return A  
    L = MERGESORT(A[0:n/2])  
    R = MERGESORT(A[n/2:n])  
    return MERGE(L,R)
```

CLAIM: MergeSort runs in time **$O(n \log n)$**



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY



Thank You

