Algorithms and Data Structures for Data Science
Recursion and Sorting

CS 277
Brad Solomon

September 29, 2021

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

mp_train due today!

Optional extra credit will be worth extra credit, currently 0 points

Can request a 24-hour extension policy

https://forms.gle/1KreTgcfJb1fXQeQ6

Request one-on-one office hours by emailing with a list of times

mp_sort released today

Due October 13th

A ‘toy’ version of a complete data science project

Informal Early Feedback released today

Form available on Moodle (along with lab surveys)

Feedback form open until October 13th

If >50% class participates, everyone gets 10 bonus points

Reminder: Sign up for exam 1!

Sign up at: https://cbtf.engr.illinois.edu/sched

Email DRES accomodations: http://cbtf@illinois.edu/

Exam Locations: 057 Grainger Library; L520 Digital Comp Lab

Programming questions on exam:

Submitted answer incorrect: 0%

Submitted at 2021-09-27 16:51:15 (CDT)

hide

Score: 0/2 (0%)

Test Results

✕ [0/1] Testing provided examples

Max points: 1

Earned points: 0

Message

```
'Peak count for [1]: ' is inaccurate
'Peak count for [1, 2, 3, 2]: ' is inaccurate
'Peak count for [2, 1, 0, 1]: ' looks good
'Peak count for [2, 2]: ' is inaccurate
```

✕ [0/1] Testing other examples...

Learning Objectives

Review sorting fundamentals

Introduce recursion

Analyze recursive strategies to list sorting

The Sorting Problem

Given a collection of objects, C, with comparable values, order the objects such that $\forall x \in C, x_i \leq x_{i+1}$

Input:

8	4	3	1	2	5	6	9	0	7

Output:

0	1	2	3	4	5	6	7	8	9

Optimal Sorting

Claim: Any deterministic comparison-based sorting algorithm must perform $O(n \log n)$ comparisons to sort n objects.

0	1	2
0	2	1

1	0	2
1	2	0

2	0	1
2	1	0

Selection vs InsertionSort

```
def selectionSort(inList):
    n = len(inList)

    for i in range(n):
        mindex = i
        for j in range(i+1, n):
            if inList[j] < inList[mindex]:
                mindex = j

        inList[i], inList[mindex] =
inList[mindex], inList[i]

```

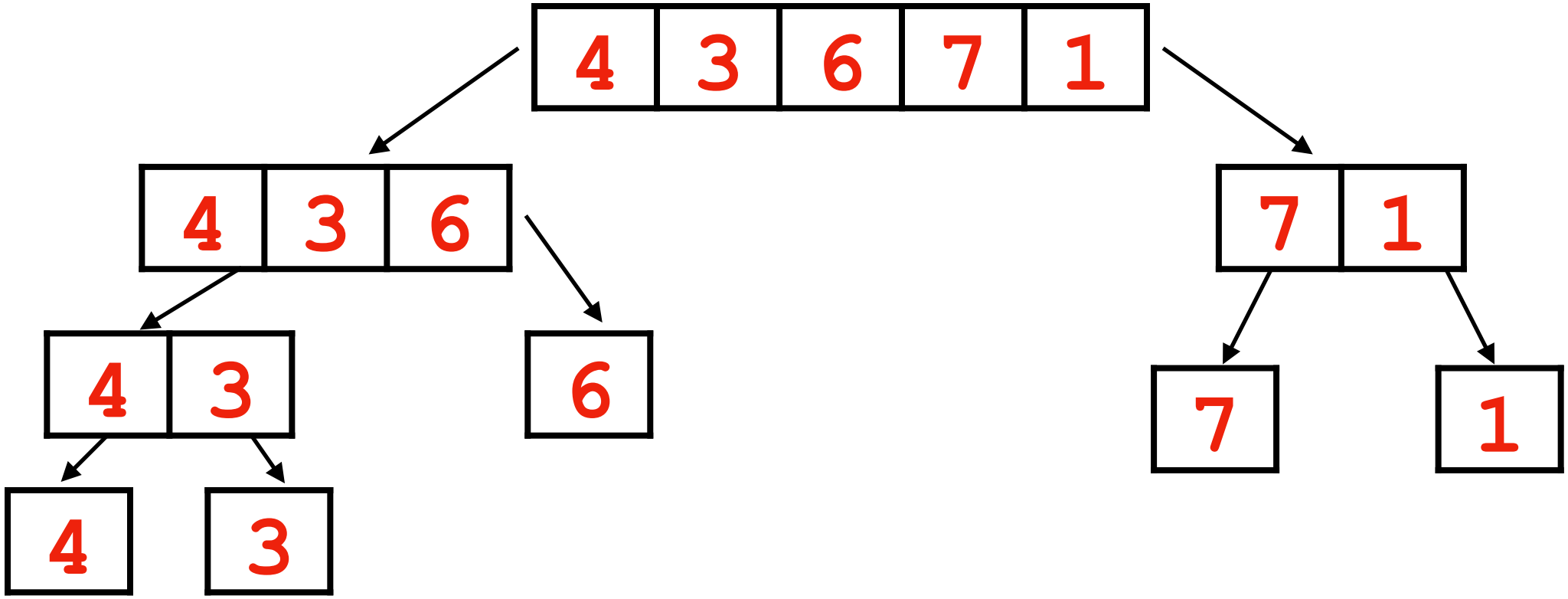
```
def insertionSort(inList):
    n = len(inList)

    for i in range(1, n):

        val = inList[i]

        j = i - 1
        while j >= 0 and val < inList[j]:
            inList[j+1]=inList[j]
            j -= 1

        inList[j+1]=val
```

Time:

Space:

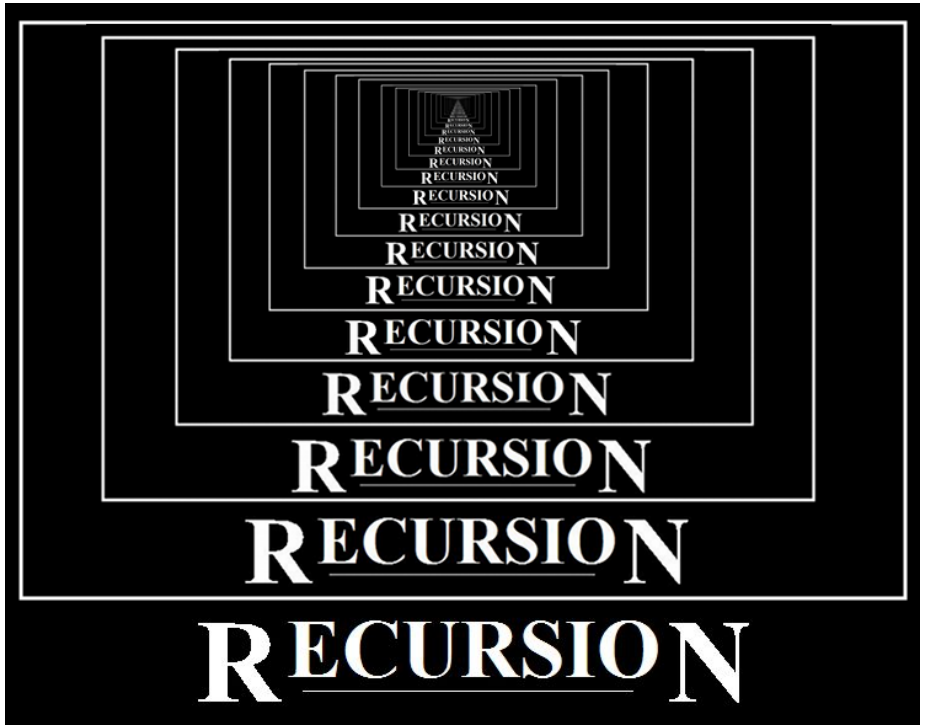
Divide and Conquer Algorithms

Recursively break a problem into sub-problems until the the problems become simple enough to solve directly



Recursion

The process by which a function calls itself directly or indirectly is called ***recursion.***



Recursive For Loop

```
for i in range(5):
    print(i)
```

```
def recursiveFor(n):

    if n == 0:
        print(n)
        return




    recursiveFor(n-1)

    print(n)

```

Recursion

Base Case: What is the smallest sub-problem? What is the trivial solution?

Recursive Step: How can I systematically reduce my problem to an easier one?

Combining: How can I build my solution from recursive pieces?

Recursive findMax

8	4	3	1	2	5	6	9	0	7

Base Case:

Recursive Step:

Combining:

Recursive Fibonacci

$$Fib(n) = Fib(n-1) + Fib(n-2), \quad n > 1$$

Base Case:

Recursive Step:

Combining:

Recursive Array Sorting

0	3	7	5	8	9	2	1	4	6

Base Case:

Recursive Step:

Combining:

Merging two sorted lists

Merging two sorted arrays can be done in $O(n + m)$ time

0	2	4	6	8

1	3	5	7	9

Merging two sorted lists

Merging two sorted arrays can be done in $O(n + m)$ time

0	2	4	6	8

1	3	4

Merging two sorted lists

Let $i =$ index for L_1, $j =$ index for L_2, $k =$ index for L_m

Case 1: Both lists contain unmerged values

Merging two sorted lists

Let $i =$ index for L_1, $j =$ index for L_2, $k =$ index for L_m

Case 2: All values in L_1 have been merged

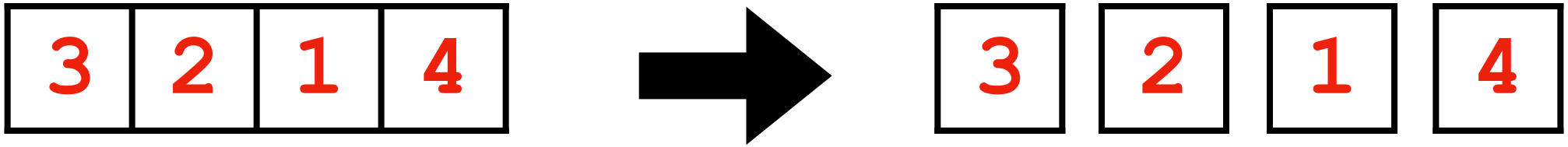
Merging two sorted lists

Let $i =$ index for L_1, $j =$ index for L_2, $k =$ index for L_m

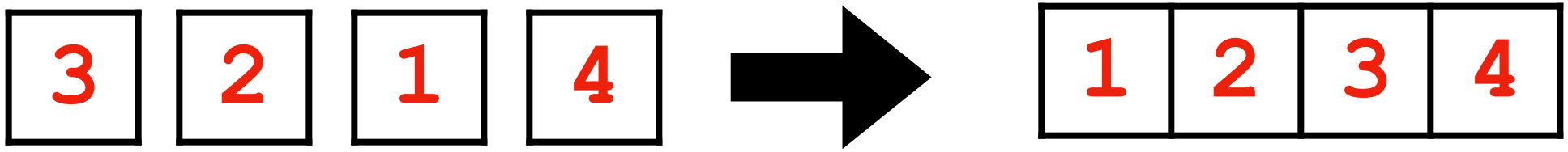
Case 3: All values in L_2 have been merged

Conceptual MergeSort

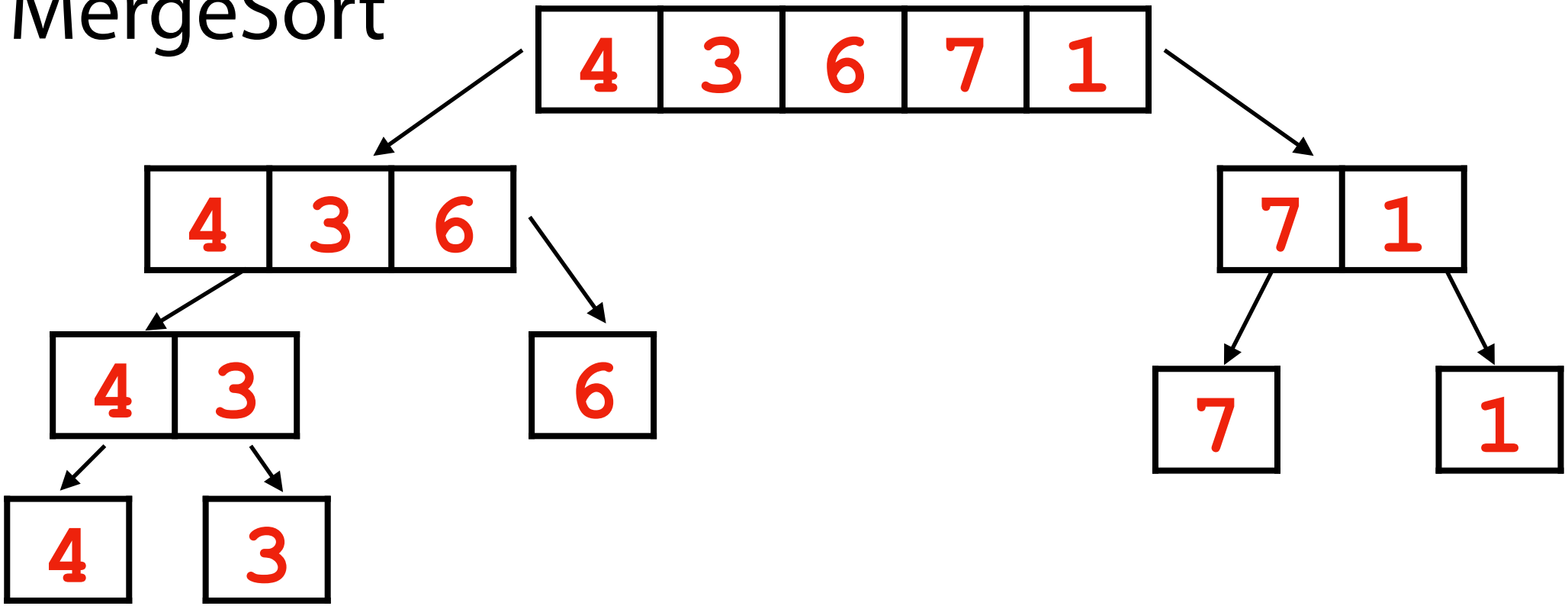
1) Input list recursively split to “sorted” base case



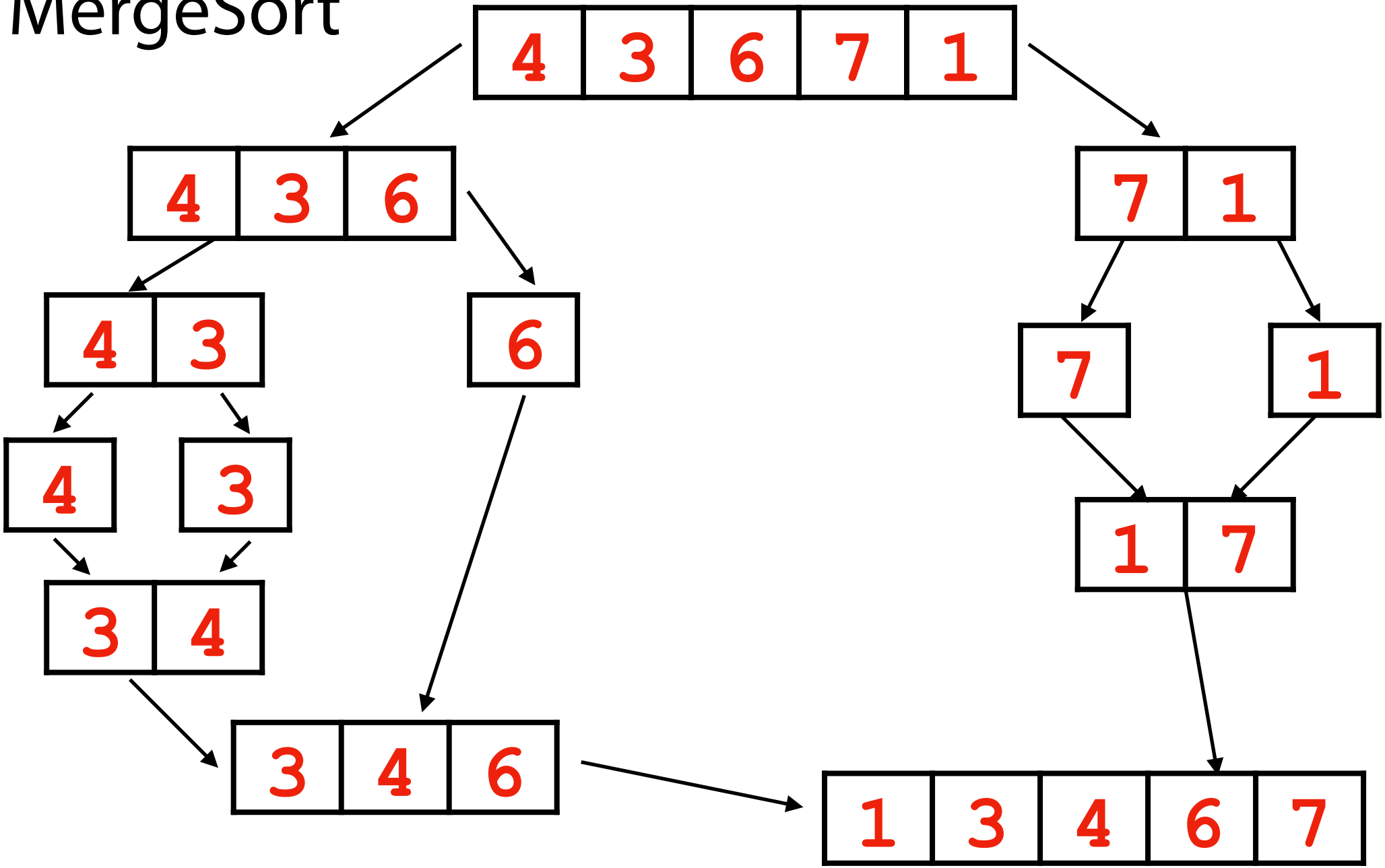
2) Sorted lists are merged back together



MergeSort

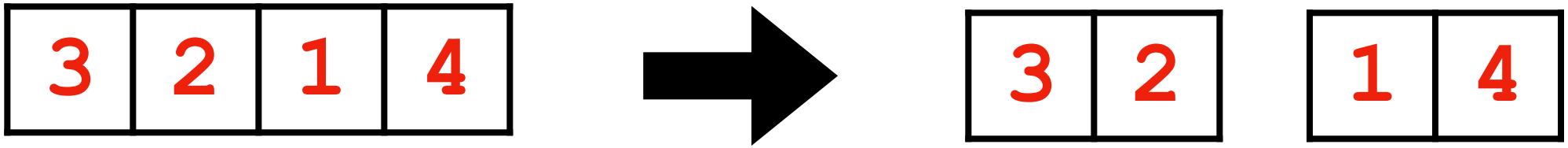


MergeSort



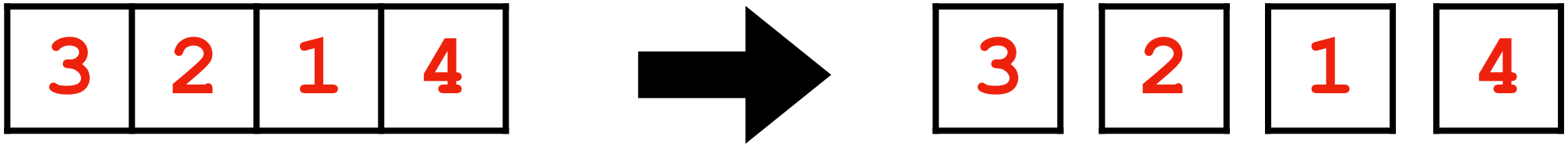
MergeSort Efficiency

1) Input list recursively split to "sorted" base case



MergeSort Efficiency

1) Input list recursively split to "sorted" base case

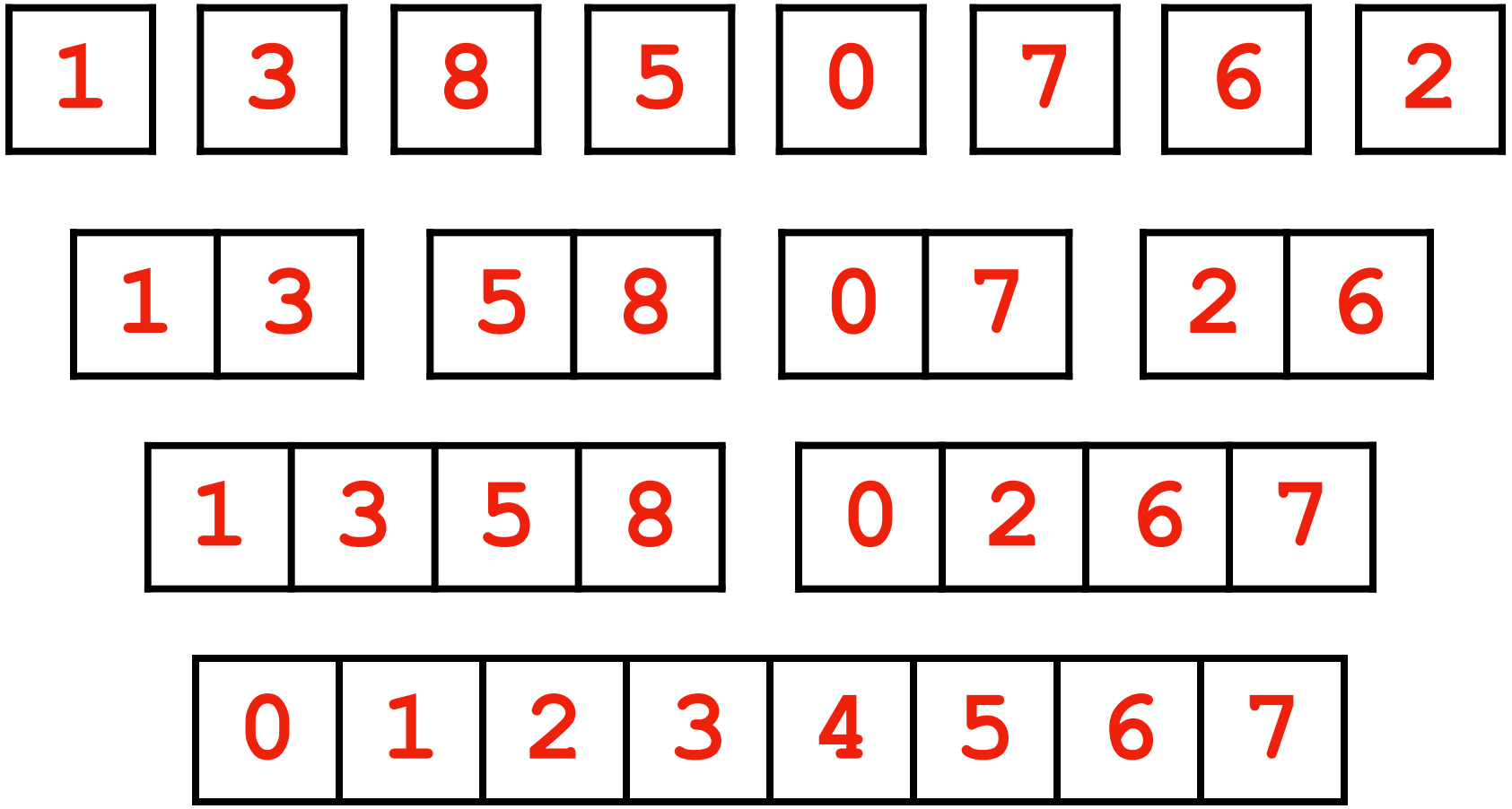


Time:

Space:

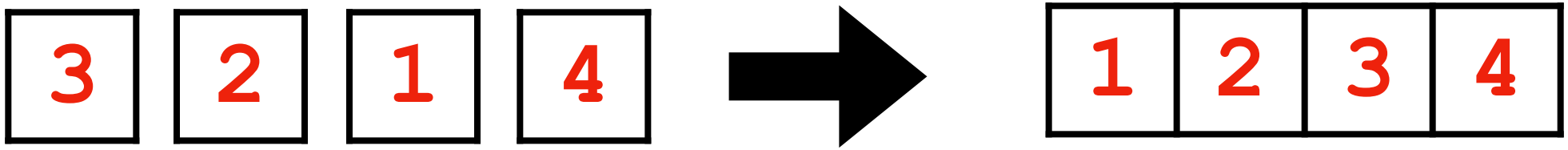
MergeSort Efficiency

2) Sorted lists are merged back together



MergeSort Efficiency

2) Sorted lists are merged back together



Time:

Space:

MergeSort

	Best Case	Worst Case
Time:		
Space:		