CS151 Intro to Data Structures

Merge Sort
Quick Sort

Announcements

HW7 and Lab9 due Friday 4/18

Lab9 Manual checkoff

Outline

Warmup: data structure design question

Sorting algorithms:

1. MergeSort
2. QuickSort

Which data structure would you use?

You are designing a system to manage user accounts for a large-scale web application. Each user has a unique username, and the system needs to support the following operations efficiently:

- **Add a user**: Insert a new user with a unique username and associated profile data.
- **Remove a user**: Delete a user and their profile from the system by username.
- **Get user data**: Retrieve a user's profile information by their username.
- **Check if a user exists**: Determine if a user with a specific username is registered in the system.

MergeSort

What sorting algorithms have we seen thus far?

1. Selection sort
    a. How does it work?
    b. Runtime complexity
2. Heap sort
    a. How does it work?
    b. Runtime complexity?

Divide and Conquer algorithm

1. **Divide**: recursively break down the problem into sub-problems
2. **Conquer:** recursively solve the sub-problems
3. **Combine:** combine the solutions to the sub-problems until they are a solution to the entire problem

Binary search is a divide and conquer algorithm

Usually involves recursion

Merge Sort

1. **Divide**: Divide the unsorted list into lists with only one element

2. **Conquer**: merge them back together in a sorted manner

3. **Combine:** merge the sorted sequences

Merge Sort

https://youtu.be/4VqmGXwpLqc?si=WpYuXYLtJOuhvd77&t=24

Merge Sort

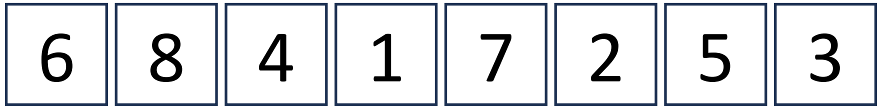
Sort a sequence of numbers A, $|A| = n$

Base: $|A| = 1$, then it’s already sorted

General

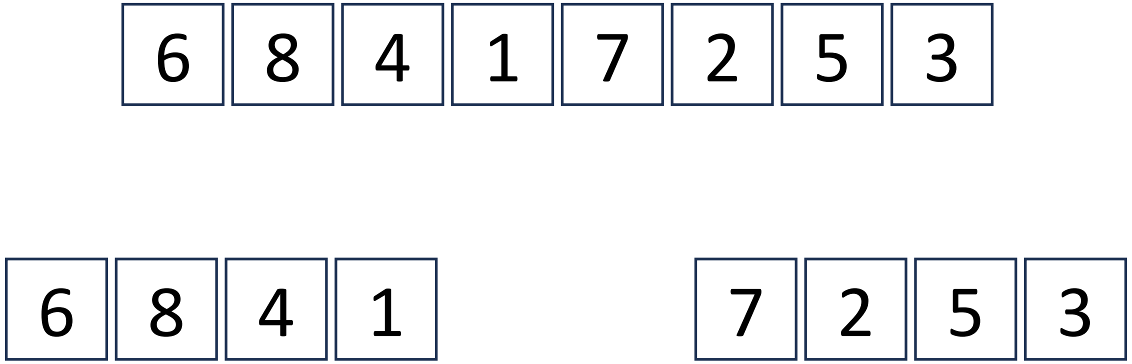
- divide: split A into two halves, each of size $\frac{n}{2}$ ($\left\lfloor\frac{n}{2}\right\rfloor$ and $\left\lceil\frac{n}{2}\right\rceil$)
- conquer: sort each half (by calling mergeSort recursively)
- combine: merge the two sorted halves into a single sorted list

Example



6	8	4	1	7	2	5	3

Example

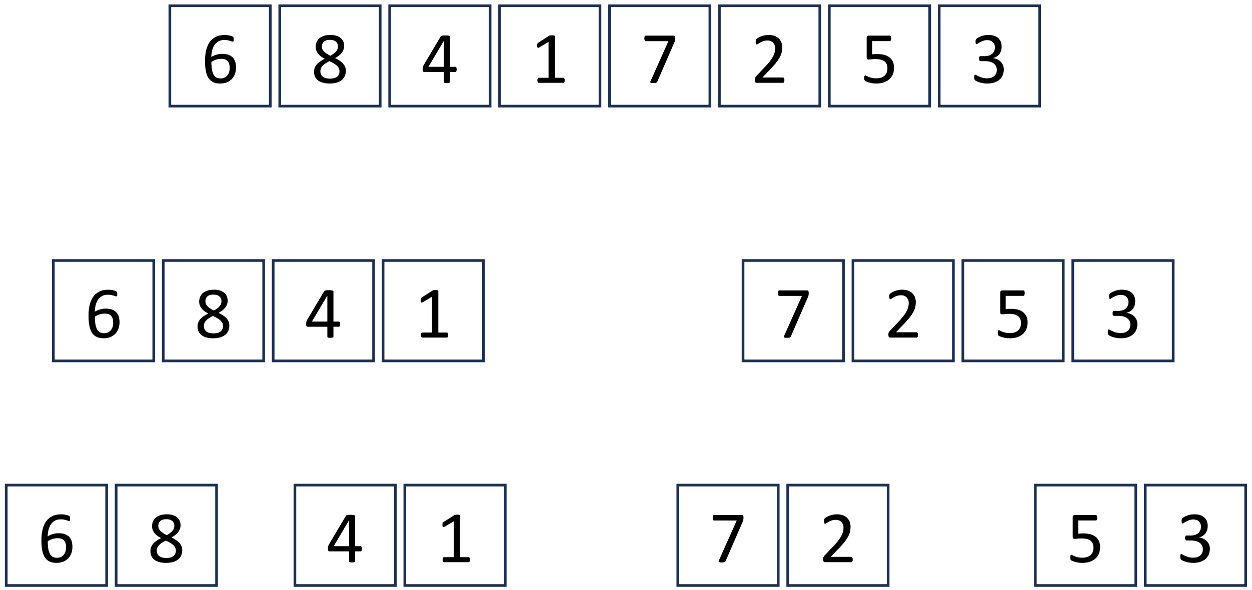


6	8	4	1	7	2	5	3

6	8	4	1

7	2	5	3

Example

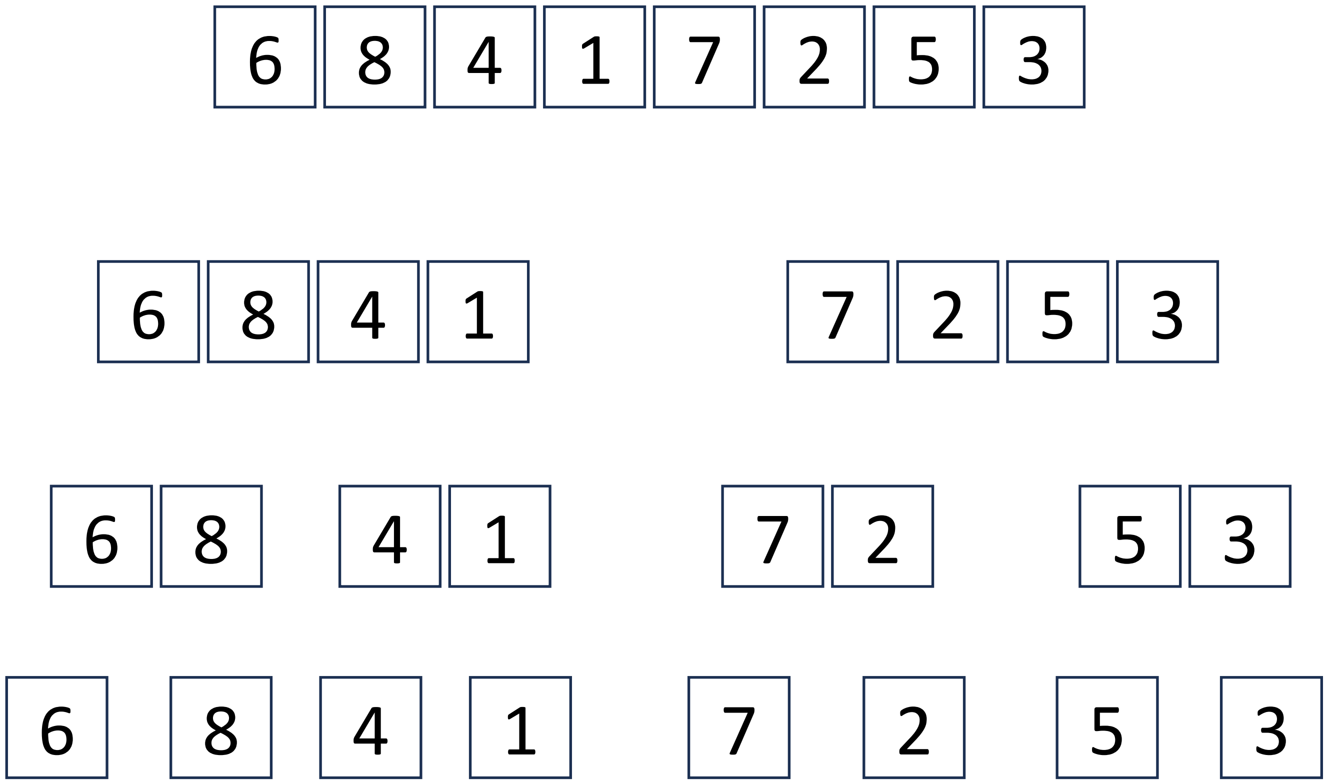


6 8 4 1 7 2 5 3

6 8 4 1   7 2 5 3

6 8   4 1   7 2   5 3

Example



6 8 4 1 7 2 5 3

6 8 4 1   7 2 5 3

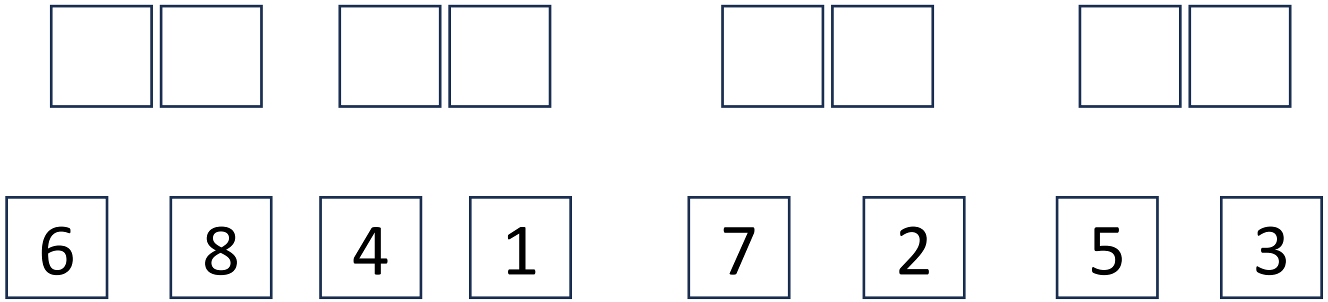
6 8   4 1   7 2   5 3

6   8   4   1   7   2   5   3

Example

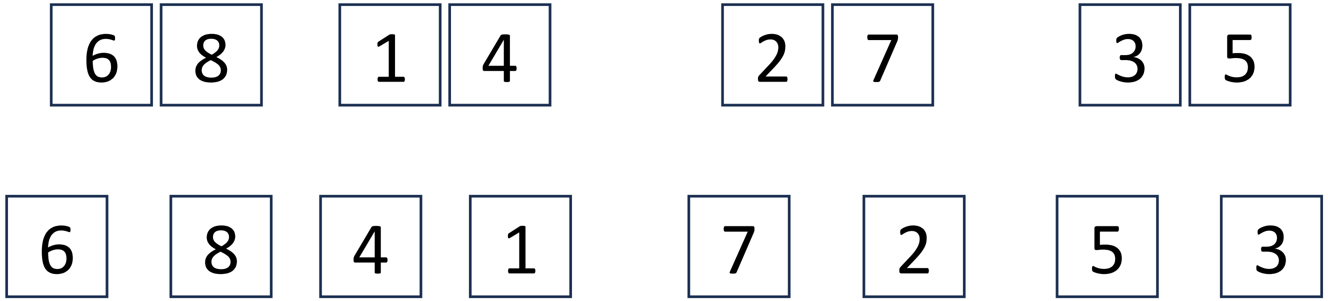
6 8 4 1 7 2 5 3

Example



6 8 4 1 7 2 5 3

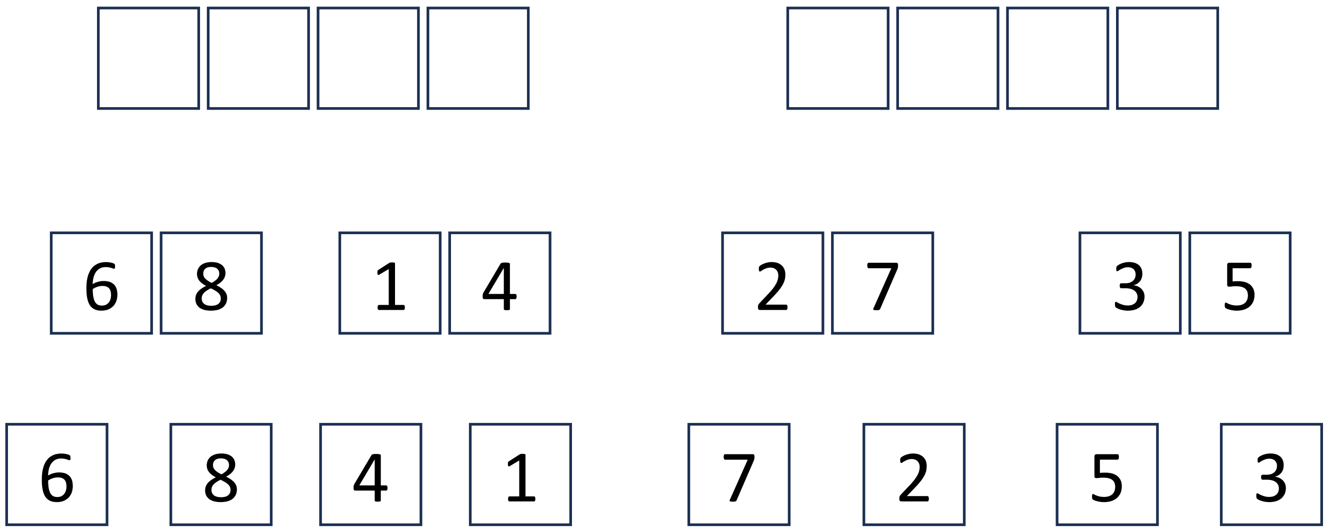
Example



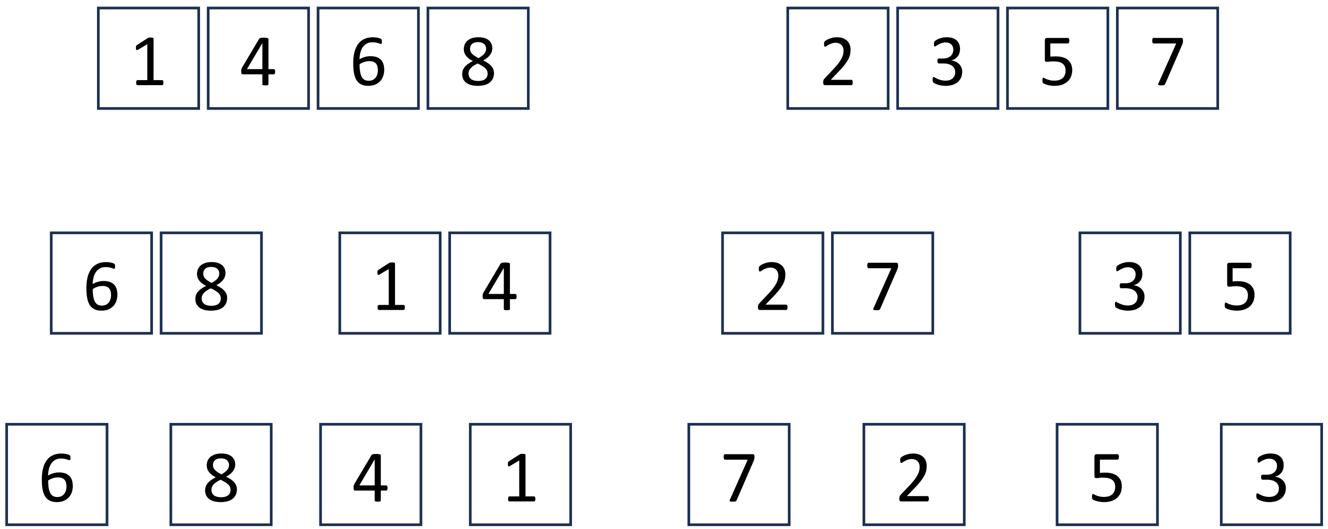
6 8 1 4 2 7 3 5

6 8 4 1 7 2 5 3

Example



Example

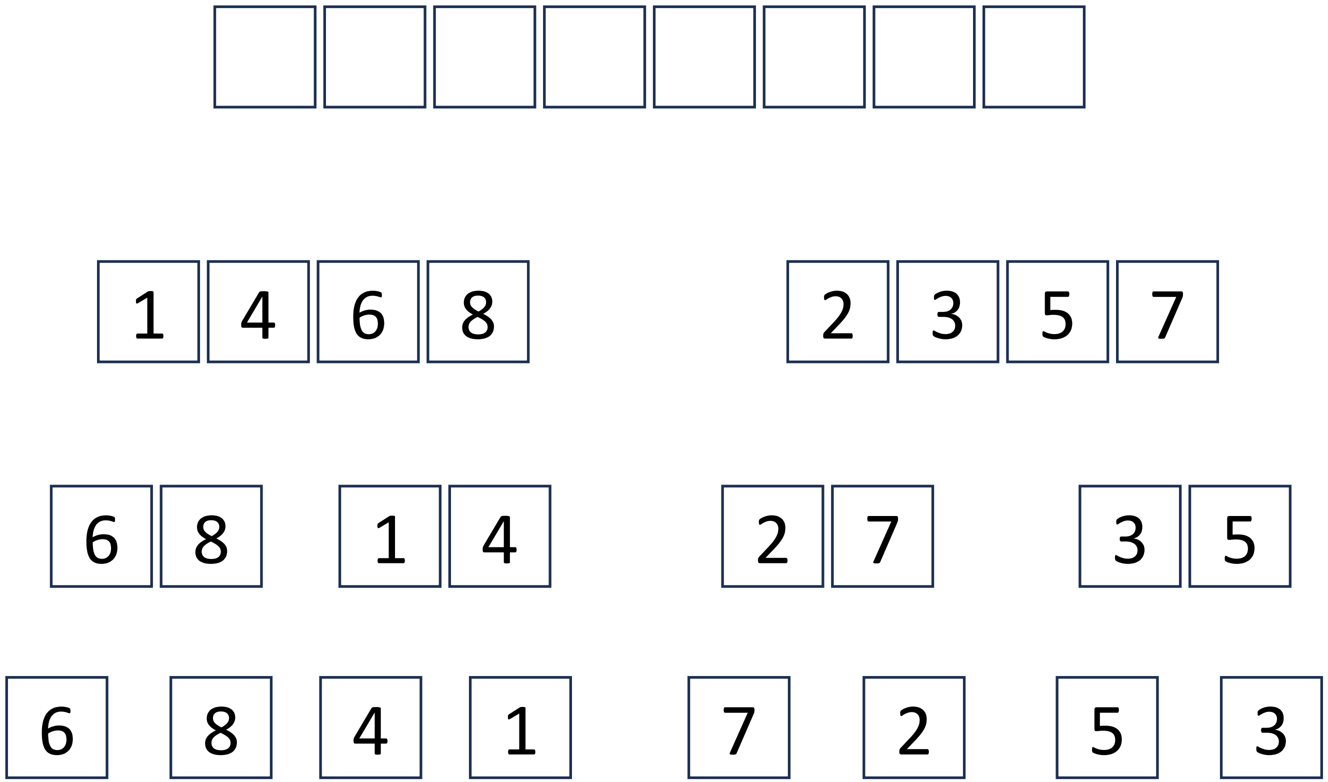


1 4 6 8   2 3 5 7

6 8   1 4   2 7   3 5

6   8   4   1   7   2   5   3

Example

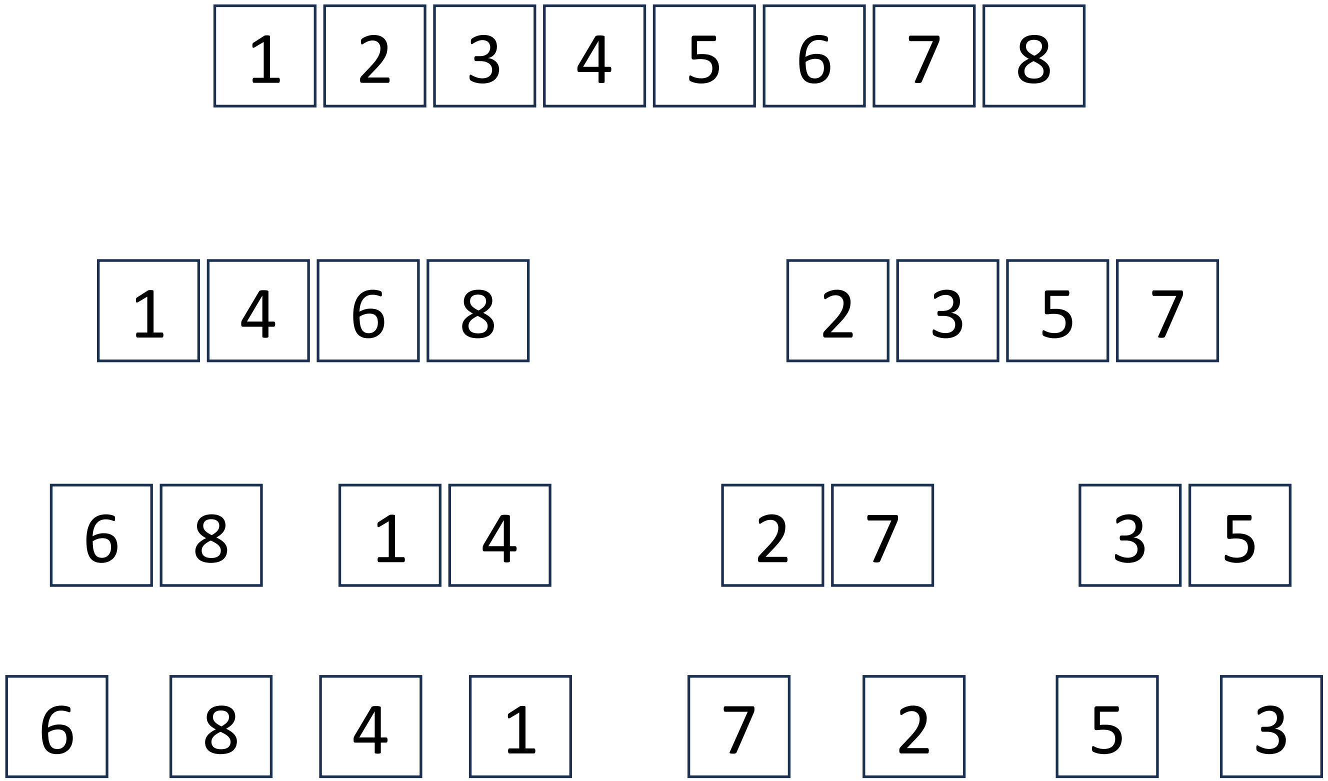


1 4 6 8   2 3 5 7

6 8   1 4   2 7   3 5

6   8   4   1   7   2   5   3

Example



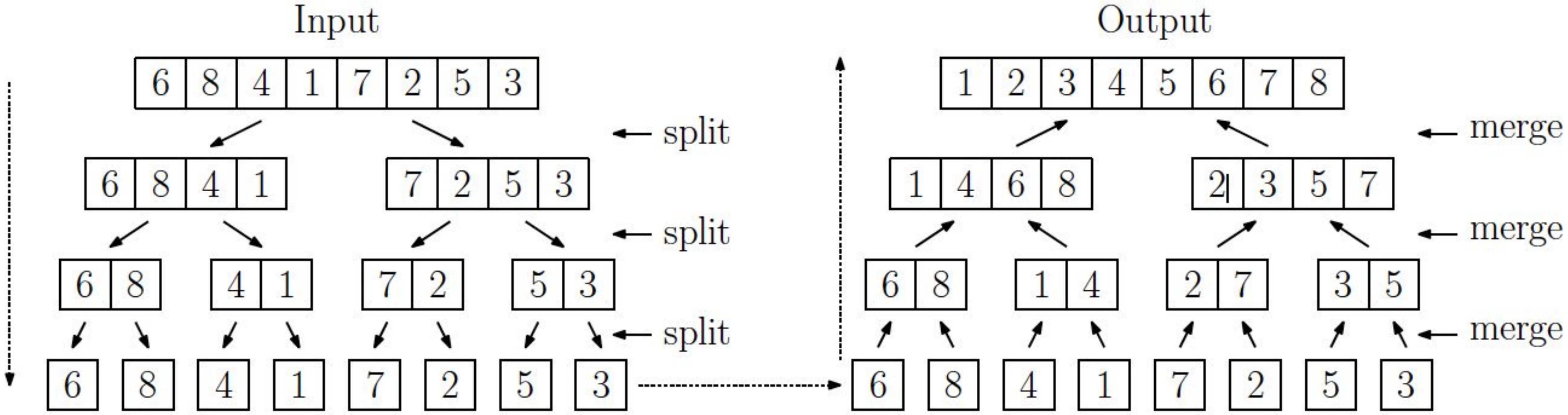
1 2 3 4 5 6 7 8

1 4 6 8   2 3 5 7

6 8   1 4   2 7   3 5

6   8   4   1   7   2   5   3

Example - summary



Merge - how do we sort two sorted lists?

```
Algorithm merge(A, B)
  S = []

  while(!A.isEmpty() and !B.isEmpty())
   if A[0] < B[0]
      S.add(A.removeFirst())
   else
      S.add(B.removeFirst())

   while (!A.isEmpty())
       S.add(A.removeFirst())
   while (!B.isEmpty())
      S.add(B.removeFirst())
   return S
```

runtime complexity?

O(n)

where n is A.length + B.length

Merge Sort Implementation

Runtime of MergeSort

Runtime of merging two sorted two lists A, B where |A| + |B| = n :

O(n)

How many times do we merge two sorted lists?

log n times

So total runtime is:

O(n * log(n))

Quicksort

Quicksort

- Divide and conquer
- **Divide:** select a *pivot* and create three sequences:
  a. L: stores elements less than the pivot
  b. E: stores elements equal to the pivot
  c. G: stores elements greater than the pivot
- **Conquer:** recursively sort L and G
- **Combine:** L + E + G is a sorted list

Quick Sort

Sort [2, 6, 5, 3, 8, 7, 1, 0]

1. choose a pivot
2. swap pivot to the end of the array
3. Find two items:
    a. left which is larger than our pivot
    b. right which is smaller than our pivot

1. swap left and right
2. repeat 3 and 4 until right < left
3. swap left and pivot
4. Sort L E and R recursively

Quick Sort - Choosing a pivot

What if we chose our pivot to be the smallest element?

We want a pivot that divides our list as evenly as possible.

Median-of-three: look at the first, middle, and last elems in the array, and pick the middle element.

Quicksort runtime complexity

Bad pivot:

O(n^2)

Good pivot:

O(nlogn)

Summary of Sorting Algorithms

Algorithm	Time
selection-sort	
heap-sort	
merge-sort	
quick-sort	

Summary

- Quicksort and Mergesort are **recursive O(nlogn)** sorting algorithms

- In quicksort, good pivots are important in achieving O(nlogn) runtime complexity

- HashTable + Quicksort homework due Friday!