Merge Sort

Basic Algorithm Design Techniques

- Divide and conquer
- Dynamic Programming
- Greedy

- Common Theme: To solve a large, complicated problem, break it into many smaller sub-problems.

Divide and Conquer

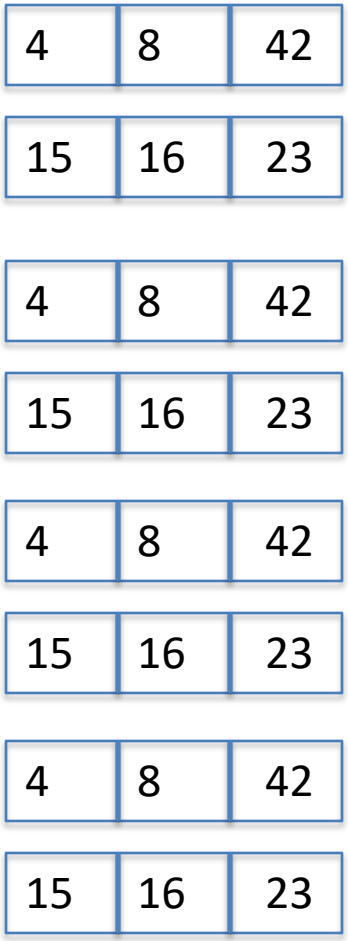
- **Idea:** Break the problem into several unrelated sub-problems, then combine the solutions to solve the original problem.

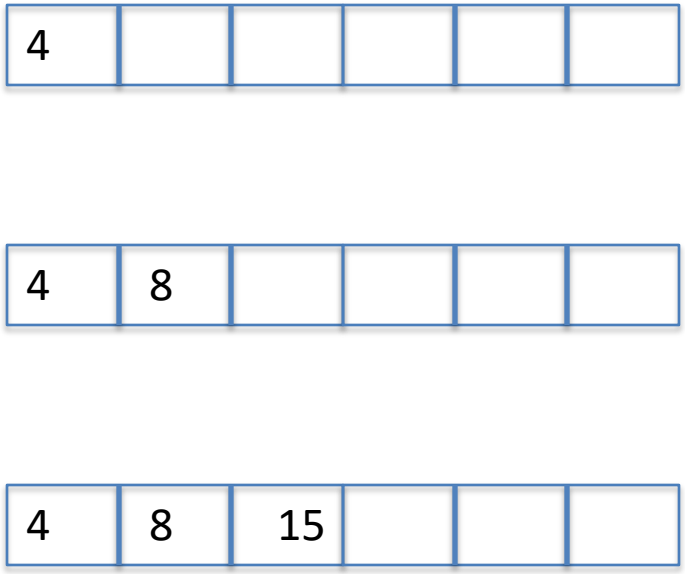
- Recipe:
- 1. (divide) Divide the problem into sub-problems
- 2. (conquer) Solve the sub-problems recursively.
- 3. (merge) Combine the solutions.

MergeSort

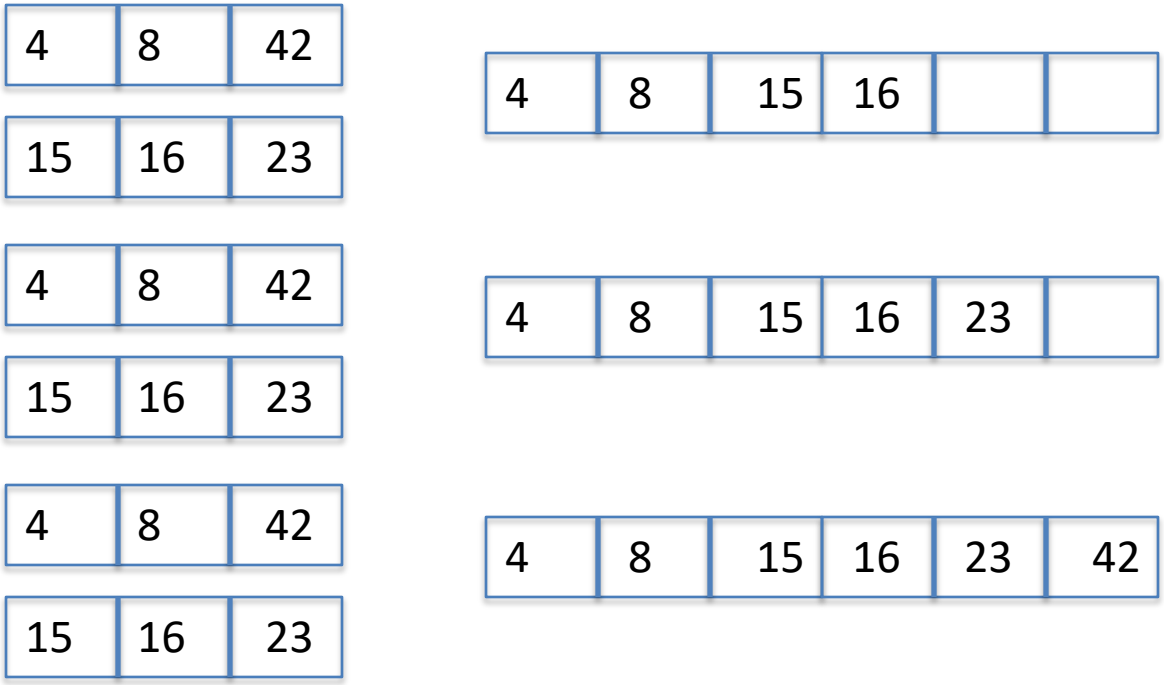
- Suppose you have 2 sorted arrays
- How do you merge them into a sorted array?
  - Just look at the first element in both arrays
  - Put the smaller element into the new array.
  - Then look at the next element in the array that the smaller element came from and compare with the current element in the other array
  - Repeat…

Merging





Merging

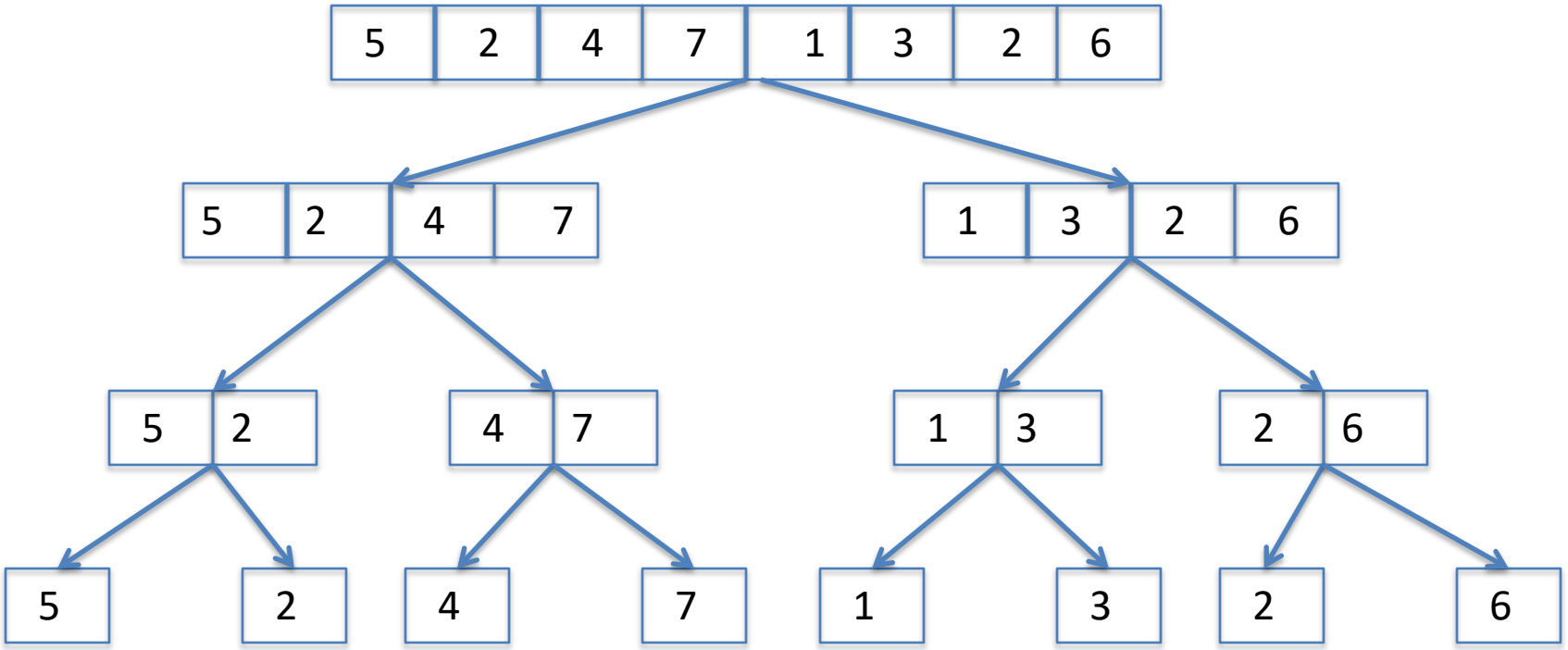


The complexity of merge sort is O(n)

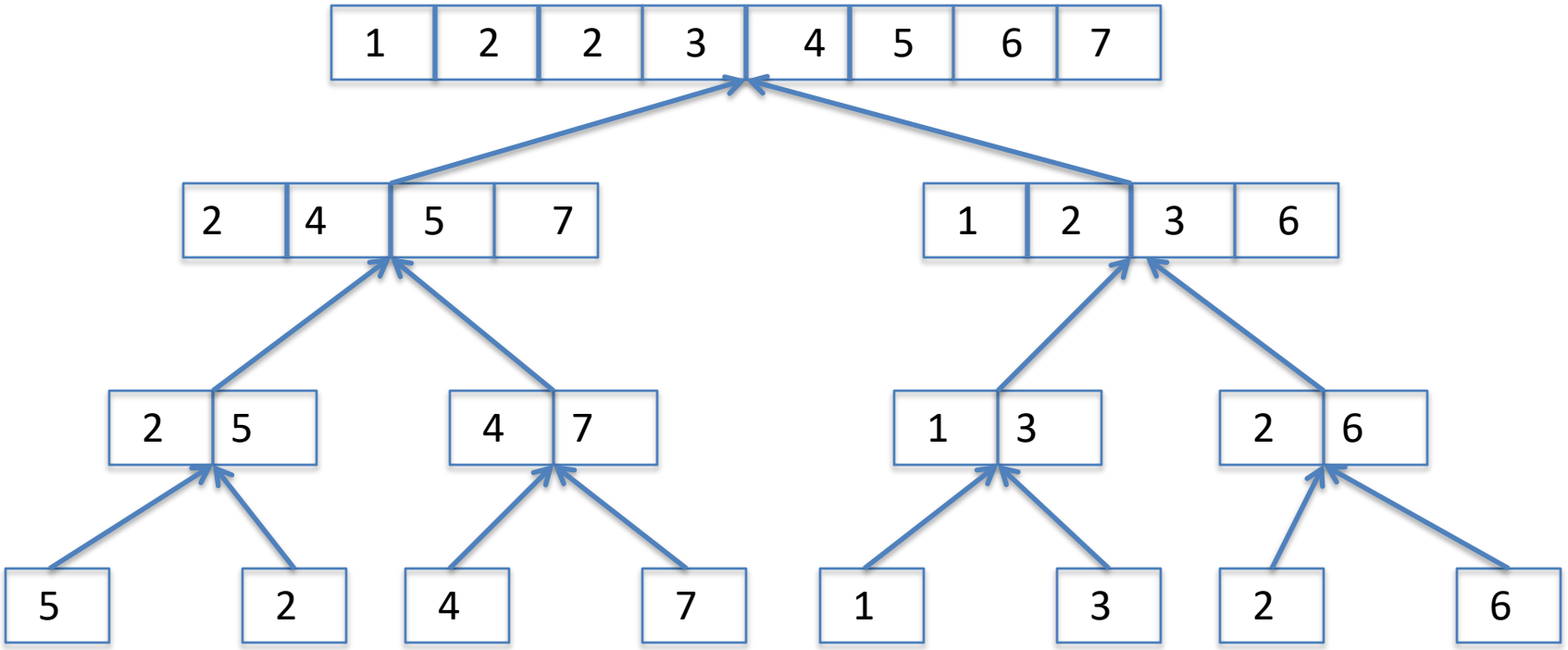
MergeSort

- But wait a second…you aren't given two already sorted arrays
- Recursion to the rescue!
  - Keep dividing the array in half (until the subarray size is 1)
  - A subarray of size 1 is sorted
  - Merge the sorted halves together

MergeSort Illustrated



MergeSort Illustrated



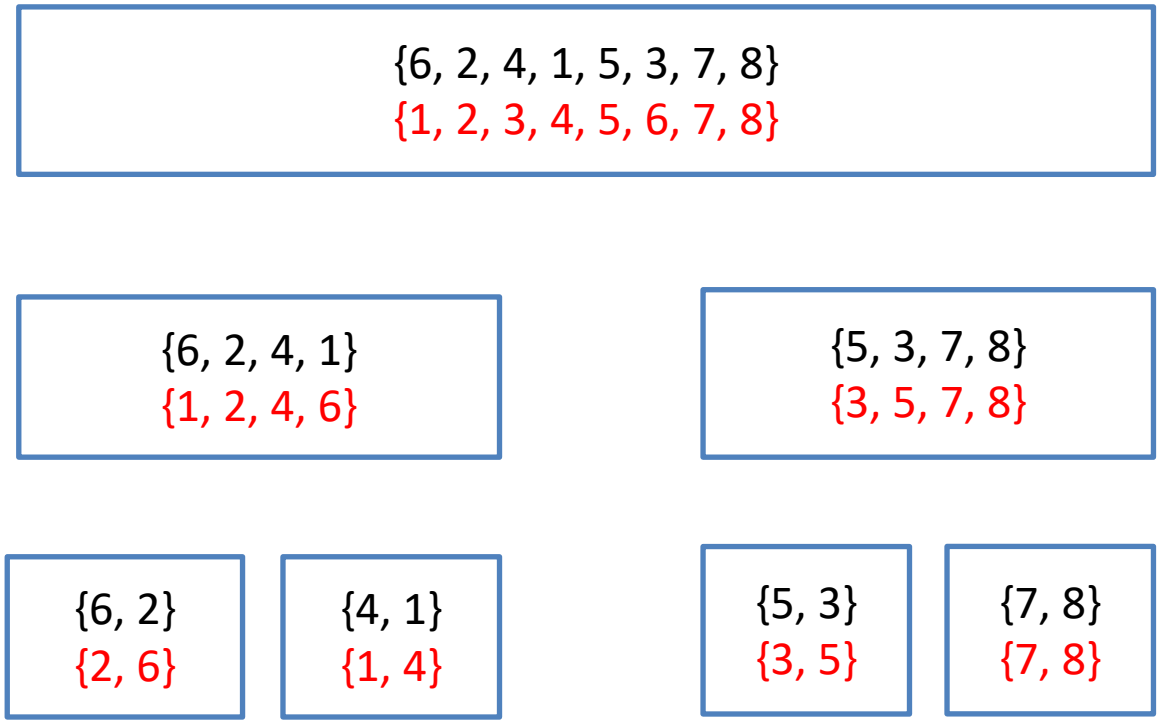
Example 1 Merge Sort

- Goal: Sort an array of numbers in ascending order.
- Input: Array a[] = {6, 2, 4, 1, 5, 3, 7, 8}

- Algorithm:

```
MergeSort(a[])
1. IF Length(a) < 2 THEN RETURN a.                    (Base Case)
2. Partition a[] evenly into two arrays b[], c[].     (Divide)
3. b[] = MergeSort(b[])
4. c[] = MergeSort(c[])                               (Conquer)
5. RETURN Merge(b[], c[]).                            (Merge)
```

Merge Sort: Example:



Merge

```
Merge(b[], c[])
1. a[] = empty
2. i = 0, j = 0
3. WHILE i <= Length(b[]) OR j <= Length(c[])
4.       IF b[i] < c[j] THEN
5.             a.append(b[i]); i = i+1
6.       ELSE
7.             a.append(c[j]); j = j+1
8. RETURN a[]
```