

Quick Sort

IIITS

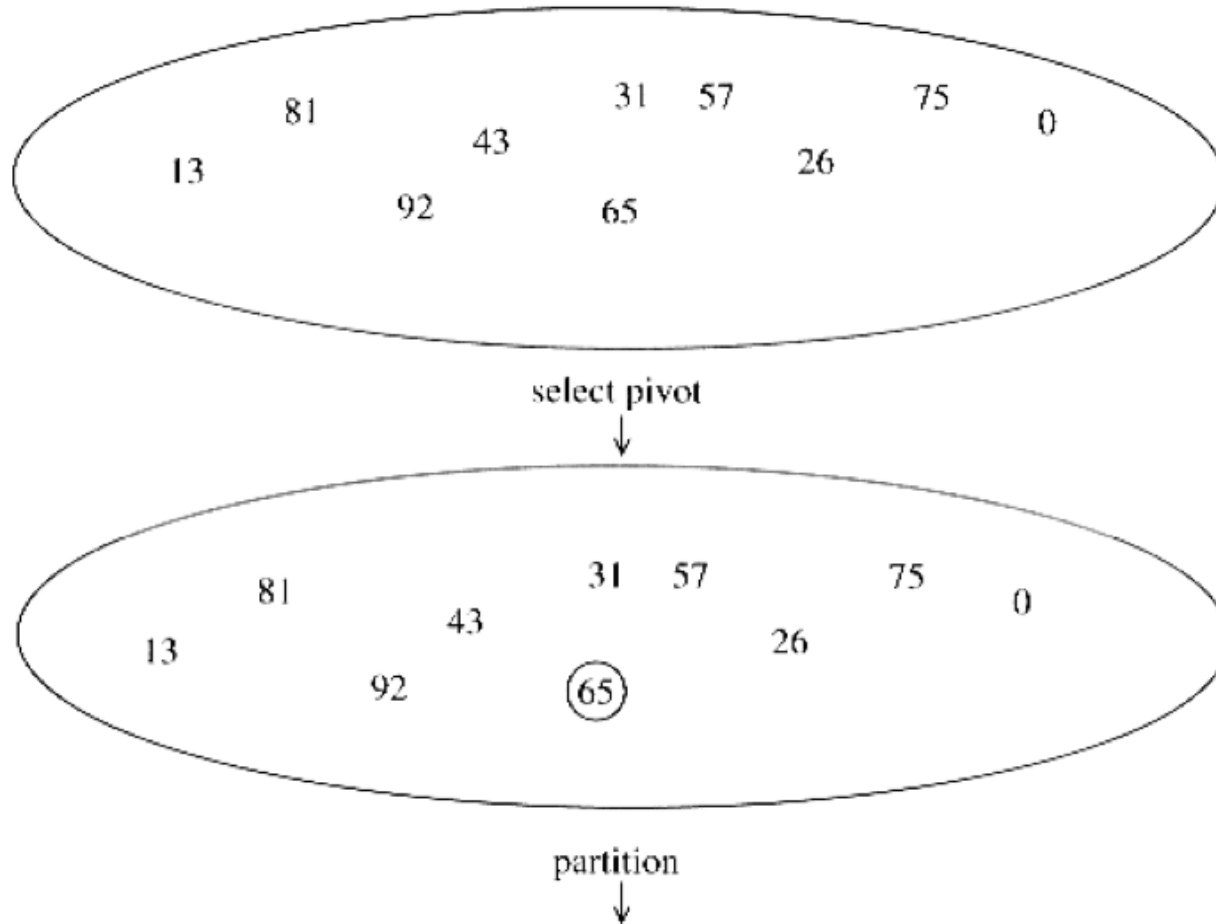
Quick Sort

- **Fastest** known sorting algorithm in practice
- Average case: $O(N \log N)$
- Worst case: $O(N^2)$
 - But the worst case can be made exponentially unlikely.
- A divide-and-conquer recursive algorithm.

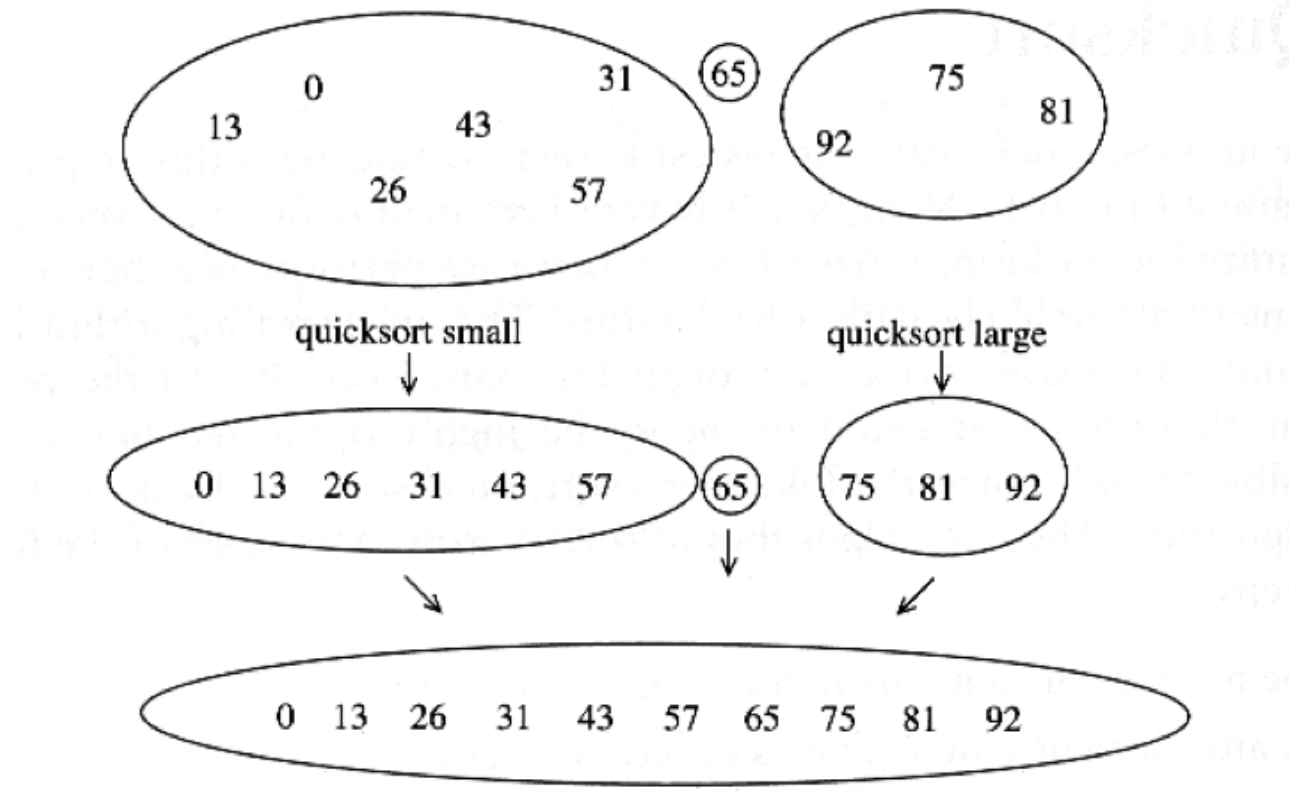
Quick Sort: Main Idea

1. If the number of elements in S is 0 or 1, then return (base case).
2. Pick any element v in S (called the pivot).
3. Partition the elements in S except v into two disjoint groups:
 1. $S_1 = \{x \in S - \{v\} \mid x \leq v\}$
 2. $S_2 = \{x \in S - \{v\} \mid x \geq v\}$
4. Return $\{\text{QuickSort}(S_1) + v + \text{QuickSort}(S_2)\}$

Quick Sort: Example



Example of Quick Sort...



Quick Sort - Example

54	26	93	17	77	31	44	55	20
----	----	----	----	----	----	----	----	----

Quick Sort - Example

54	26	93	17	77	31	44	55	20
----	----	----	----	----	----	----	----	----



pivot

Quick Sort - Example

Partitioning begins by locating two position markers—let's call them **leftmark** and **rightmark**—at the beginning and end of the remaining items in the list.

The goal of the partition process is to move items that are on the wrong side with respect to the pivot value while also converging on the split point.

54	26	93	17	77	31	44	55	20
----	----	----	----	----	----	----	----	----

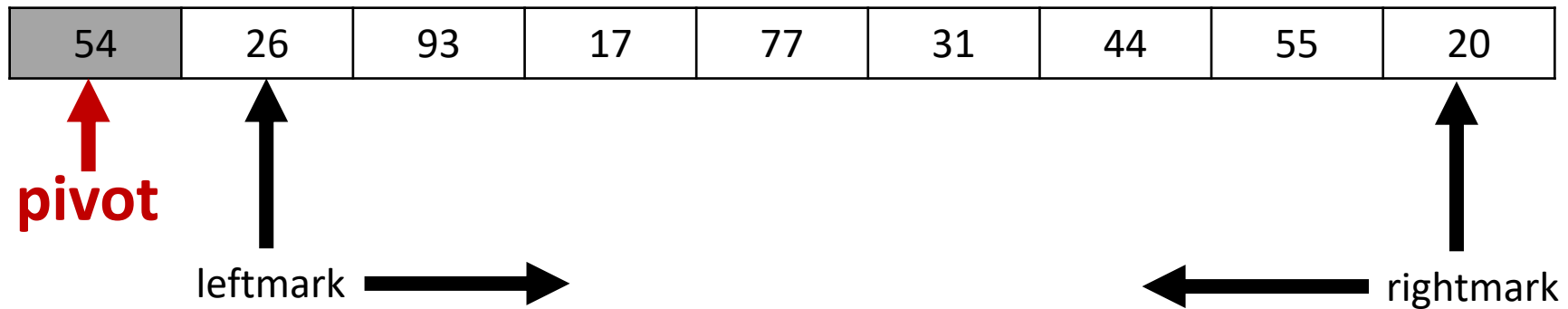


pivot

Quick Sort - Example

Partitioning begins by locating two position markers—let's call them **leftmark** and **rightmark**—at the beginning and end of the remaining items in the list.

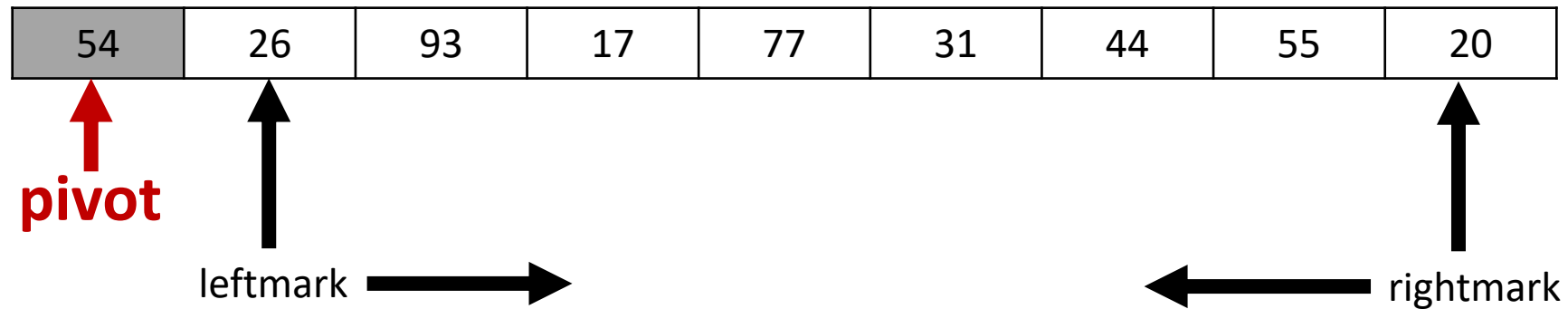
The goal of the partition process is to move items that are on the wrong side with respect to the pivot value while also converging on the split point.



Quick Sort - Example

Partitioning begins by locating two position markers—let's call them **leftmark** and **rightmark**—at the beginning and end of the remaining items in the list.

The goal of the partition process is to move items that are on the wrong side with respect to the pivot value while also converging on the split point.



leftmark and the rightmark will converge on split point

Quick Sort - Example


Is $26 < 54$


YES

Move to right

54	26	93	17	77	31	44	55	20
----	----	----	----	----	----	----	----	----


pivot


leftmark


rightmark

Quick Sort - Example


Is $93 < 54$


NO

Stop

54	26	93	17	77	31	44	55	20
----	----	----	----	----	----	----	----	----


pivot


leftmark


rightmark

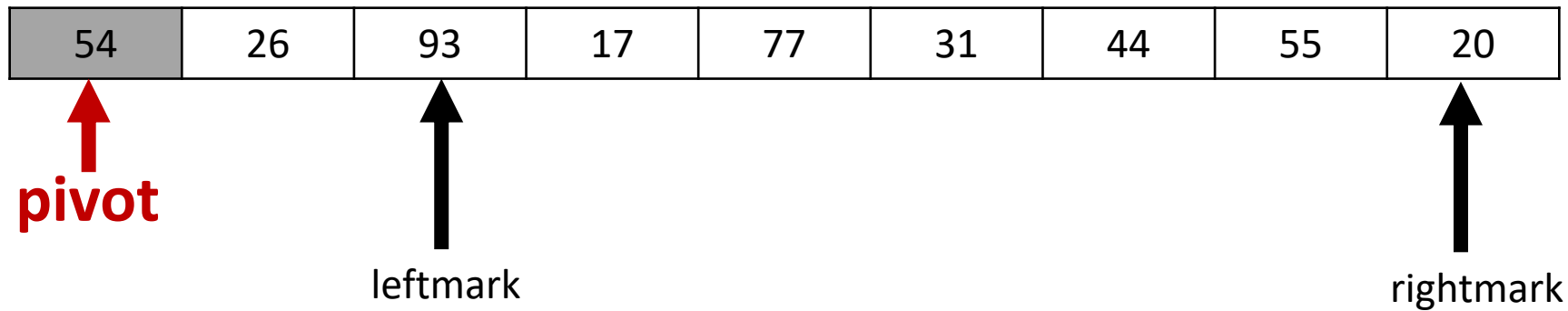
Quick Sort - Example

Now consider the rightmark

Is $20 > 54$

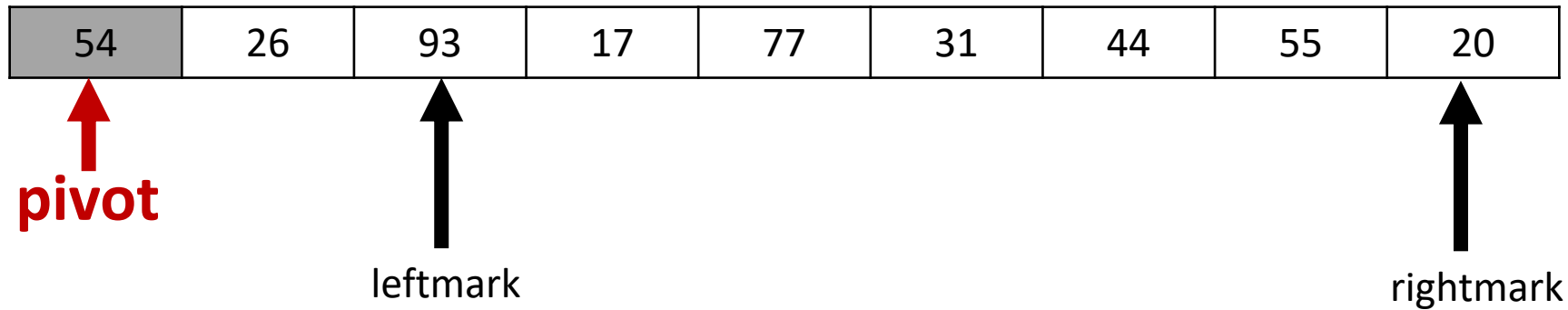
NO

Stop



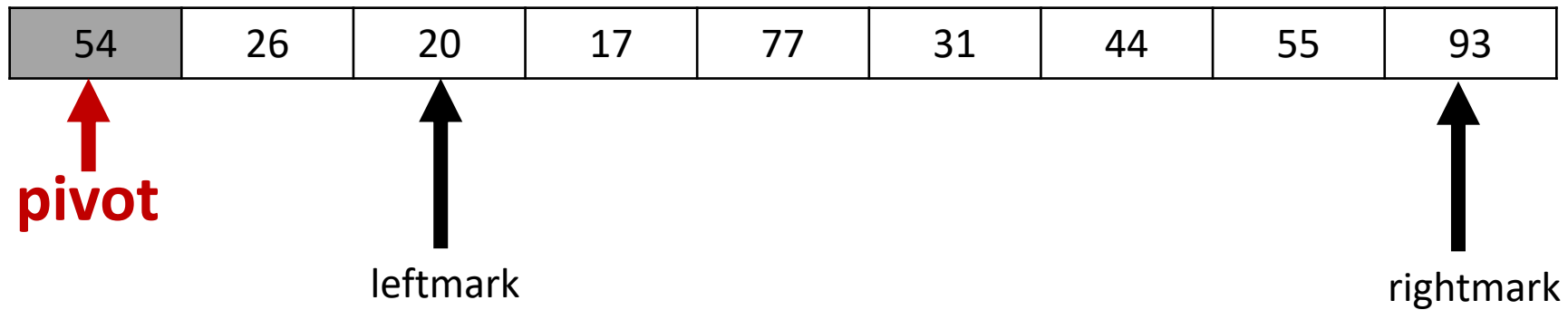
Quick Sort - Example

Exchange 20 and 93



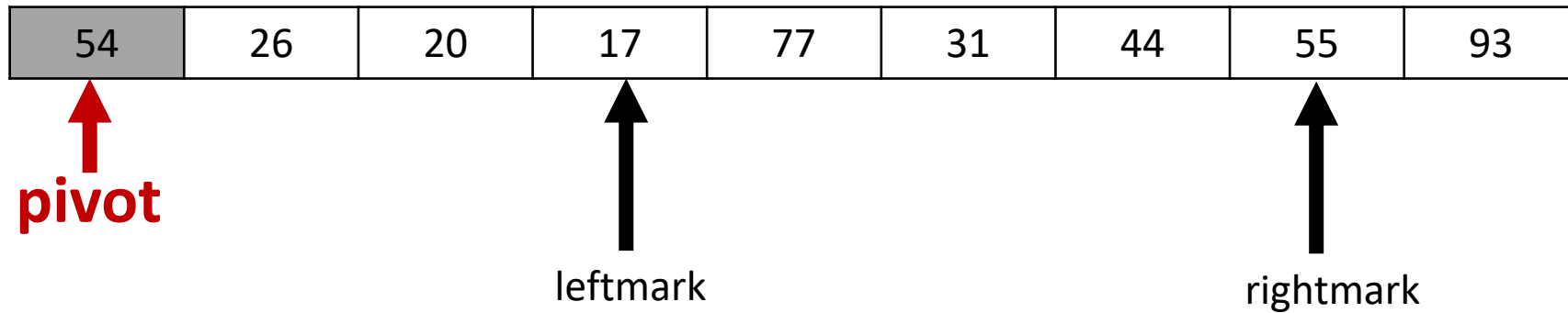
Quick Sort - Example

Exchange 20 and 93



Quick Sort - Example

Move leftmark and rightmark



Quick Sort - Example


Is $17 < 54$


YES

Move to right

54	26	20	17	77	31	44	55	93
----	----	----	----	----	----	----	----	----


pivot


leftmark


rightmark

Quick Sort - Example


Is $77 < 54$


NO

stop

54	26	20	17	77	31	44	55	93
----	----	----	----	----	----	----	----	----


pivot


leftmark


rightmark

Quick Sort - Example


Is $55 > 54$


YES

Move to left

54	26	20	17	77	31	44	55	93
----	----	----	----	----	----	----	----	----


pivot


leftmark


rightmark

Quick Sort - Example


Is $44 > 54$


NO

stop

54	26	20	17	77	31	44	55	93
----	----	----	----	----	----	----	----	----

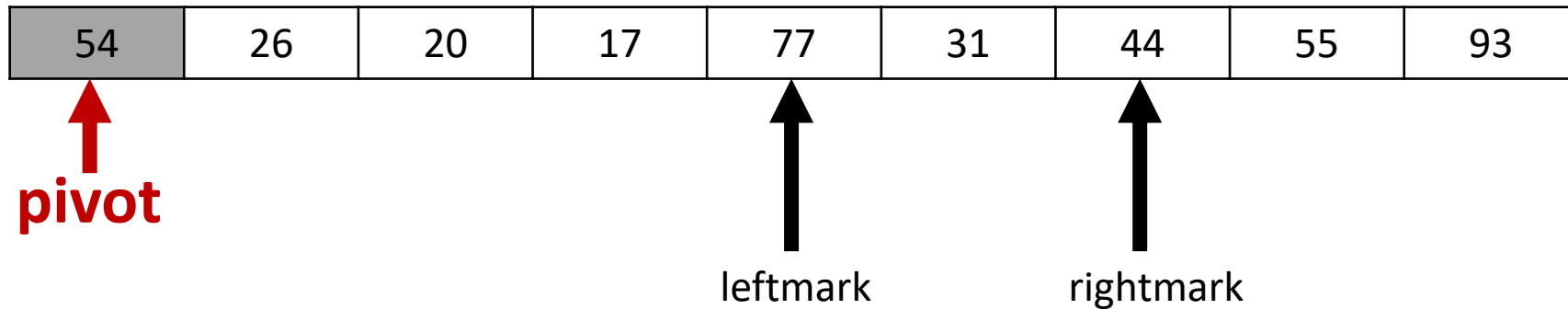

pivot


leftmark


rightmark

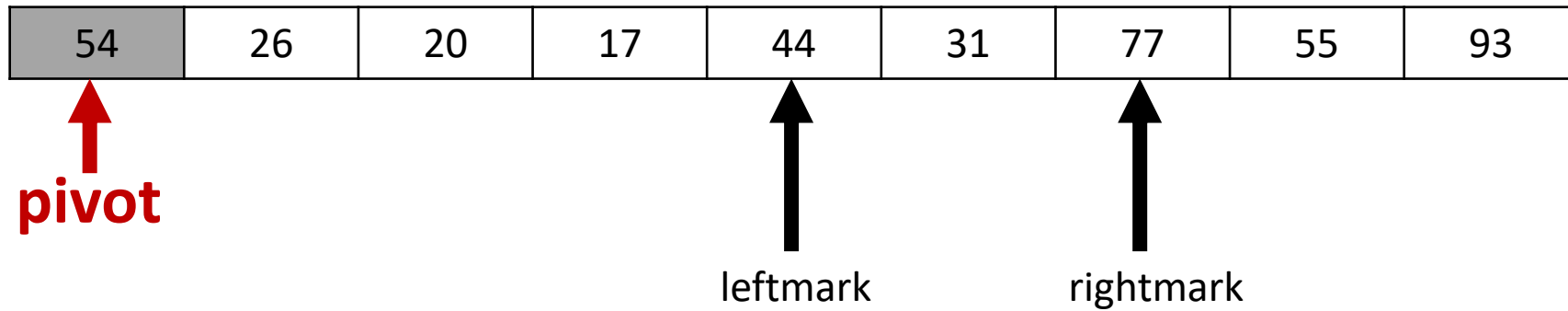
Quick Sort - Example

Exchange 77 and 44



Quick Sort - Example

Exchange 77 and 44



Quick Sort - Example

Change the leftmark

Change the rightmark

54	26	20	17	44	31	77	55	93
----	----	----	----	----	----	----	----	----


pivot


leftmark
rightmark

Quick Sort - Example


If $31 < 54$

YES

Move to right

54	26	20	17	44	31	77	55	93
----	----	----	----	----	----	----	----	----


pivot


leftmark
rightmark

When leftmark == rightmark – split point found


exchange 54 and 31

Quick Sort - Example

leftmark and rightmark crossing each other

54	26	20	17	44	31	77	55	93
----	----	----	----	----	----	----	----	----


pivot


leftmark
rightmark

When leftmark == rightmark – split point found

exchange 54 and 31

Quick Sort - Example

31	26	20	17	44	54	77	55	93
----	----	----	----	----	----	----	----	----


pivot

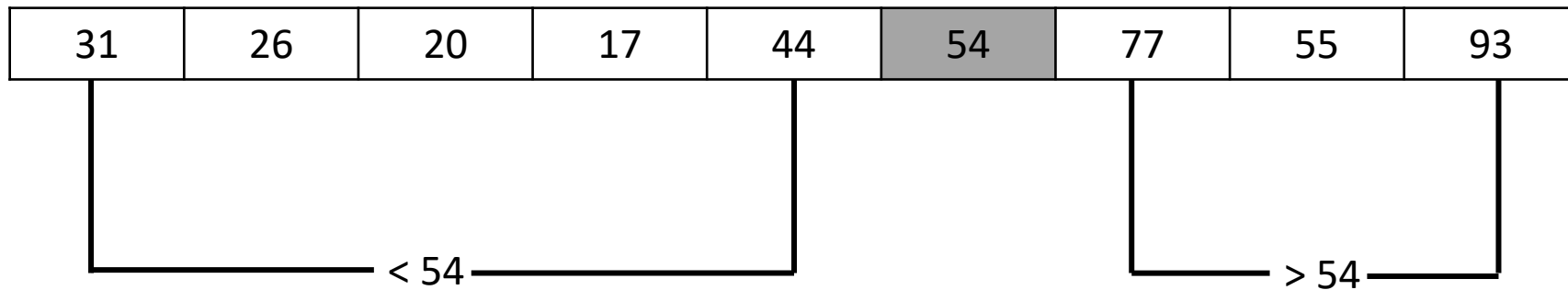

leftmark
rightmark

When leftmark == rightmark – split point found

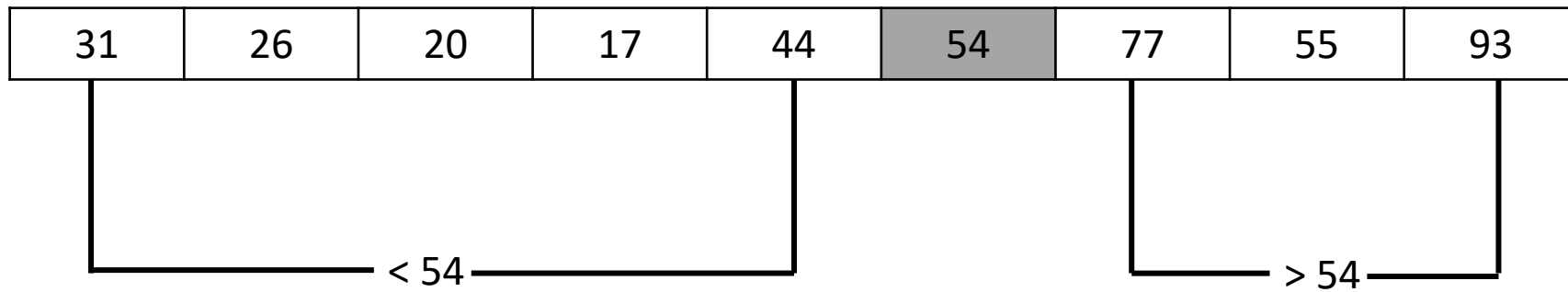
exchange 54 and 31

Quick Sort - Example

54 is in place



Quick Sort - Example



31	26	20	17	44
----	----	----	----	----

quicksort lefthalf

77	55	93
----	----	----

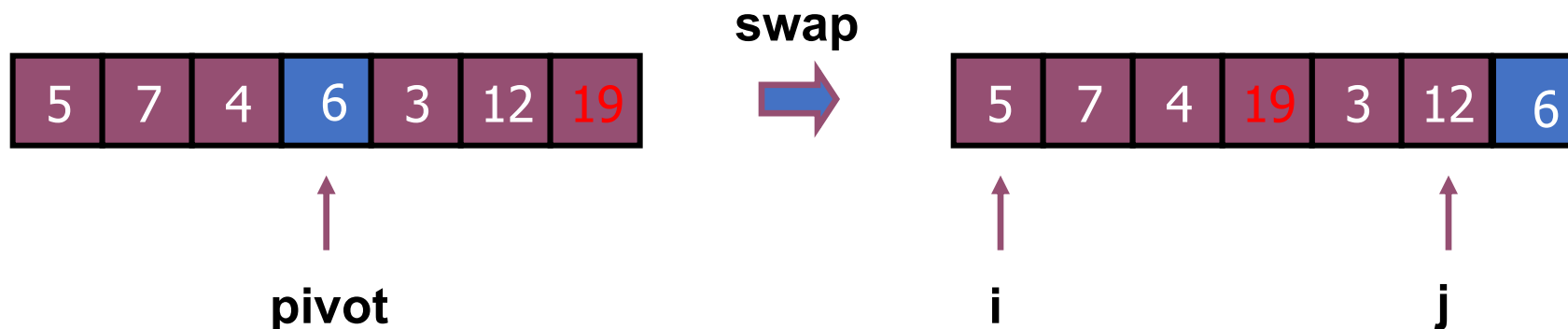
quicksort righthalf

Issues To Consider

- How to partition?
 - Several methods exist.
 - The one we consider is known to give good results and to be easy and efficient.
 - We discuss the partition strategy first.
- How to pick the pivot?
 - Many methods

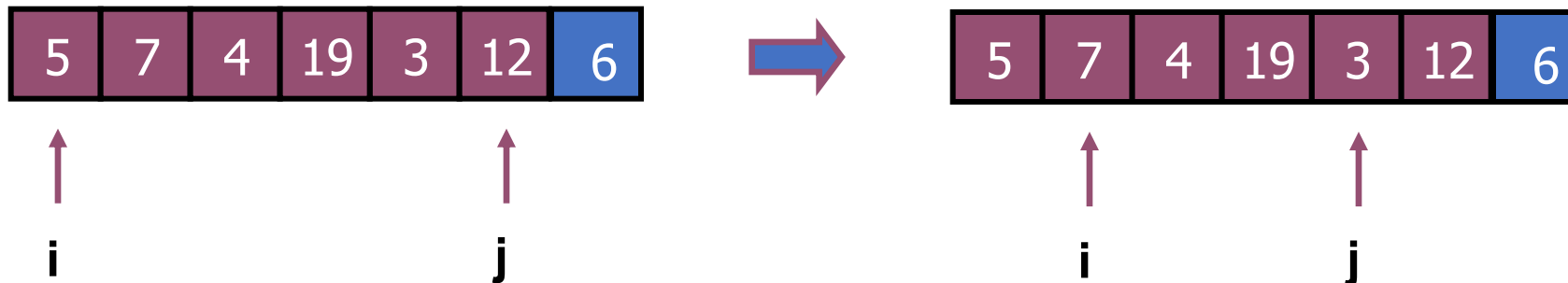
Partitioning Strategy

- For now, assume that $\text{pivot} = A[(\text{left} + \text{right})/2]$.
- We want to partition array $A[\text{left} .. \text{right}]$.
- First, get the pivot element out of the way by swapping it with the last element (swap pivot and $A[\text{right}]$).
- Let i start at the first element and j start at the next-to-last element ($i = \text{left}$, $j = \text{right} - 1$)



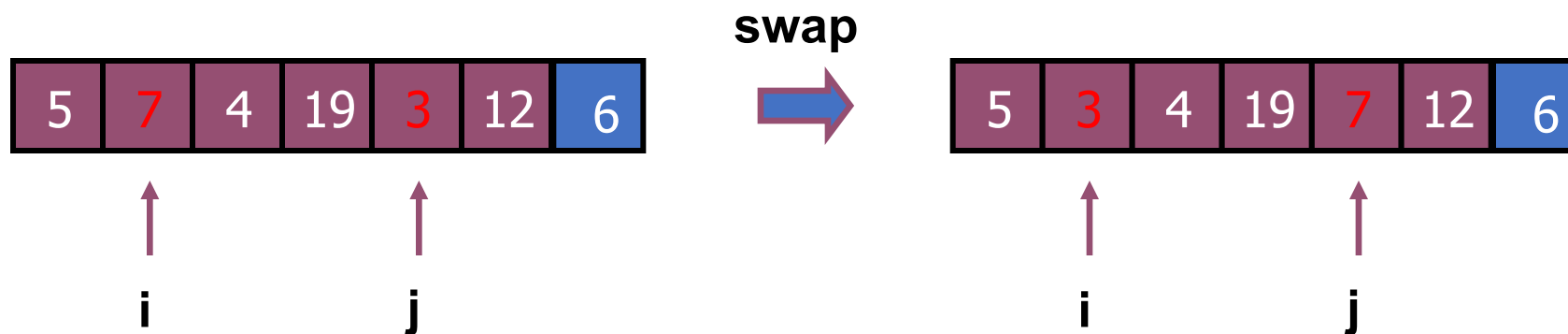
Partitioning Strategy

- Want to have
 - $A[k] \leq \text{pivot}$, for $k < i$
 - $A[k] \geq \text{pivot}$, for $k > j$
- When $i < j$
 - Move i right, skipping over elements smaller than the pivot
 - Move j left, skipping over elements greater than the pivot
 - When both i and j have stopped
 - $A[i] \geq \text{pivot}$
 - $A[j] \leq \text{pivot} \Rightarrow A[i]$ and $A[j]$ should now be swapped



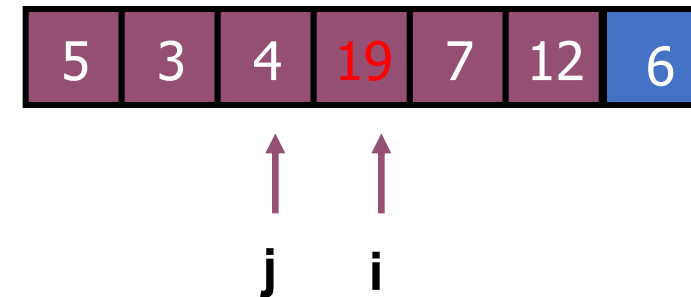
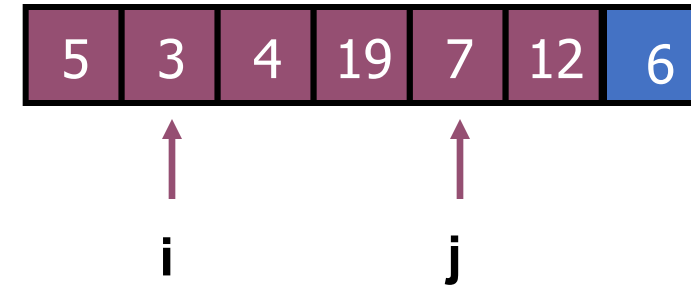
Partitioning Strategy (2)

- When i and j have stopped and i is to the left of j (thus legal)
 - Swap $A[i]$ and $A[j]$
 - The large element is pushed to the right and the small element is pushed to the left
 - After swapping
 - $A[i] \leq \text{pivot}$
 - $A[j] \geq \text{pivot}$
 - Repeat the process until i and j cross

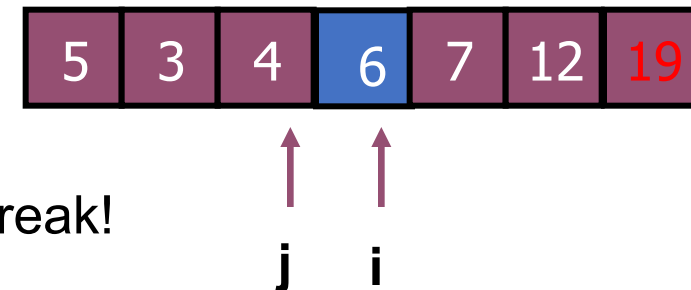


Partitioning Strategy (3)

- When i and j have crossed
 - swap $A[i]$ and pivot
- Result:
 - $A[k] \leq \text{pivot}$, for $k < i$
 - $A[k] \geq \text{pivot}$, for $k > i$



swap $A[i]$ and pivot



Break!

Picking the Pivot

- There are several ways to pick a pivot.
- Objective: Choose a pivot so that we will get 2 partitions of (almost) equal size.

Picking the Pivot (2)

- Use the first element as pivot
 - if the input is random, ok.
 - if the input is presorted (or in reverse order)
 - all the elements go into S_2 (or S_1).
 - this happens consistently throughout the recursive calls.
 - results in $O(N^2)$ behavior (we analyze this case later).
- Choose the pivot randomly
 - generally safe,
 - but random number generation can be expensive and does not reduce the running time of the algorithm.

Picking the Pivot (3)

- Use the median of the array (ideal pivot)
 - The $\lceil N/2 \rceil$ *th* largest element
 - Partitioning always cuts the array into roughly half
 - An **optimal** quick sort ($O(N \log N)$)
 - However, hard to find the exact median
- Median-of-three partitioning
 - eliminates the bad case for sorted input.
 - reduces the number of comparisons by 14%.

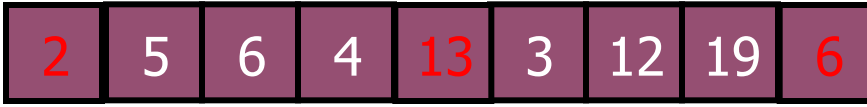
Median of Three Method

- Compare just three elements: the leftmost, rightmost and center
 - Swap these elements if necessary so that
 - A[left] = Smallest
 - A[right] = Largest
 - A[center] = Median of three
 - Pick A[center] as the pivot.
 - Swap A[center] and A[right - 1] so that the pivot is at the second last position (why?)

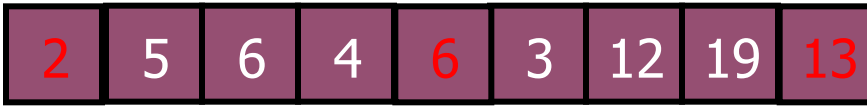
```
int center = ( left + right ) / 2;
if( a[ center ] < a[ left ] )
    swap( a[ left ], a[ center ] );
if( a[ right ] < a[ left ] )
    swap( a[ left ], a[ right ] );
if( a[ right ] < a[ center ] )
    swap( a[ center ], a[ right ] );

// Place pivot at position right - 1
swap( a[ center ], a[ right - 1 ] );
```

Median of Three: Example



$A[\text{left}] = 2$, $A[\text{center}] = 13$,
 $A[\text{right}] = 6$



Swap $A[\text{center}]$ and $A[\text{right}]$



Choose $A[\text{center}]$ as **pivot**

↑
pivot



Swap pivot and $A[\text{right} - 1]$

↑
pivot

We only need to partition $A[\text{left} + 1, \dots, \text{right} - 2]$.

Quick Sort: Pseudo-code

```
if( left + 10 <= right )  
{
```

```
    Comparable pivot = median3( a, left, right );
```

Choose pivot

```
        // Begin partitioning
```

```
        int i = left, j = right - 1;  
        for( ; ; )  
        {  
            while( a[ ++i ] < pivot ) { }  
            while( pivot < a[ --j ] ) { }  
            if( i < j )  
                swap( a[ i ], a[ j ] );  
            else  
                break;  
        }
```

Partitioning

```
        swap( a[ i ], a[ right - 1 ] ); // Restore pivot
```

```
        quicksort( a, left, i - 1 );    // Sort small elements  
        quicksort( a, i + 1, right );  // Sort large elements
```

Recursion

```
    }  
else // Do an insertion sort on the subarray  
    insertionSort( a, left, right );
```

For small arrays

Partitioning Part

- The partitioning code we just saw works only if pivot is picked as **median-of-three**.
 - $A[\text{left}] \leq \text{pivot}$ and $A[\text{right}] \geq \text{pivot}$
 - Need to partition only $A[\text{left} + 1, \dots, \text{right} - 2]$
- j will not run past the beginning
 - because $A[\text{left}] \leq \text{pivot}$
- i will not run past the end
 - because $A[\text{right}-1] = \text{pivot}$

```
int i = left, j = right - 1;
for( ; ; )
{
    while( a[ ++i ] < pivot ) { }
    while( pivot < a[ --j ] ) { }
    if( i < j )
        swap( a[ i ], a[ j ] );
    else
        break;
}
```



```
/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low   --> Starting index,
high  --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```
/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
   array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];    // pivot
    int i = (low - 1);    // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

Small Arrays - Nuances

- For very small arrays, quick sort does not perform as well as insertion sort
- Do not use quick sort recursively for small arrays
 - Use a sorting algorithm that is efficient for small arrays, such as insertion sort.
- When using quick sort recursively, switch to insertion sort when the sub-arrays have between 5 to 20 elements (10 is usually good).
 - saves about 15% in the running time.
 - avoids taking the median of three when the sub-array has only 1 or 2 elements.

Assignment

- Assume the pivot is chosen as the middle element of an array: $\text{pivot} = a[(\text{left} + \text{right}) / 2]$ - Median of Three . Rewrite the partitioning code and the whole quick sort algorithm.

Analysis

Assumptions:

- A random pivot (no median-of-three partitioning)
 - No cutoff for small arrays (to make it simple)
1. If the number of elements in S is 0 or 1, then return (base case).
 2. Pick an element v in S (called the pivot).
 3. Partition the elements in S except v into two disjoint groups:
 1. $S_1 = \{x \in S - \{v\} \mid x \leq v\}$
 2. $S_2 = \{x \in S - \{v\} \mid x \geq v\}$
 4. Return $\{\text{QuickSort}(S_1) + v + \text{QuickSort}(S_2)\}$

Worst-Case Scenario

- What will be the worst case?
 - The pivot is the smallest element, all the time
 - Partition is always unbalanced

$$T(N) = T(N - 1) + cN$$

$$T(N - 1) = T(N - 2) + c(N - 1)$$

$$T(N - 2) = T(N - 3) + c(N - 2)$$

$$\vdots$$

$$T(2) = T(1) + c(2)$$

$$T(N) = T(1) + c \sum_{i=2}^N i = O(N^2)$$

Best-Case Scenario

- What will be the best case?
 - Partition is perfectly balanced.
 - Pivot is always in the middle (median of the array).
- $T(N) = T(N/2) + T(N/2) + cN = 2T(N/2) + cN$
- This recurrence is similar to the merge sort recurrence.
- The result is $O(N\log N)$.

Average-Case Analysis

- Assume that each of the sizes for S_1 is equally likely \Rightarrow has probability $1/N$.
- This assumption is valid for the pivoting and partitioning strategy just discussed (but may not be for some others),
- On average, the running time is $O(N \log N)$.
- Proof: pp 272–273, Data Structures and Algorithm Analysis by M. A. Weiss, 2nd edition

Merge Sort

- Yet another divide and Conquer sort

Divide and Conquer

- **Divide and Conquer cuts the problem in half each time, but uses the result of both halves:**
 - cut the problem in half until the problem is trivial
 - solve for both halves
 - combine the solutions

Merge Sort

- **Divide** the unsorted collection **into two**
- **Until the sub-arrays only contain one element**
- **Then merge the sub-problem solutions together**

Algorithm

Mergesort(Passed an array)

if array size > 1

Divide array in half

Call Mergesort on first half.

Call Mergesort on second half.

Merge two halves.

Merge(Passed two arrays)

Compare leading element in each array

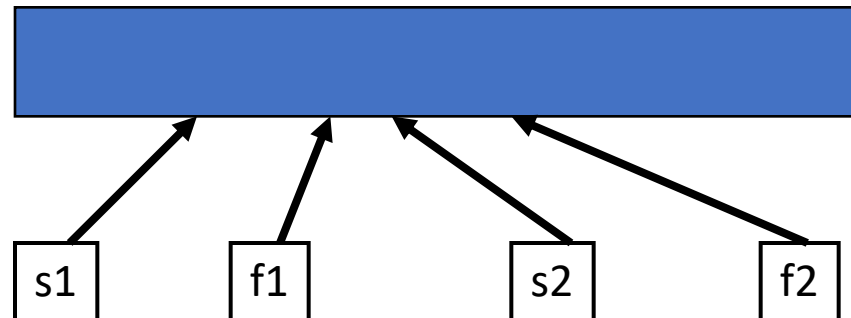
Select lower and place in new array.

(If one input array is empty then place
remainder of other array in output array)

More TRUTH in CS

- We don't really pass in two arrays!
- We pass in one array with indicator variables which tell us where one set of data starts and finishes and where the other set of data starts and finishes.

- Honest.



98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98	23
----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98	23
----	----

23

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98	23
----	----

23	98
----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98	23
----	----

45	14
----	----

23	98
----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98	23
----	----

45	14
----	----

23	98
----	----

14

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

14	45
----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

14	45
----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

14	45
----	----

14

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

14	45
----	----

14	23
----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

14	45
----	----

14	23	45
----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

98

23

45

14

23	98
----	----

14	45
----	----

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

23	98
----	----

14	45
----	----

14	23	45	98
----	----	----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

23	98
----	----

14	45
----	----

14	23	45	98
----	----	----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

23	98
----	----

14	45
----	----

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

23	98
----	----

14	45
----	----

6

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

23	98
----	----

14	45
----	----

6	67
---	----

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

14	23	45	98
----	----	----	----

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33
---	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42
---	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14
---	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23
---	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33
---	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42
---	----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45
---	----	----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45	67
---	----	----	----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45	67	98
---	----	----	----	----	----	----	----

Merge

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

98	23	45	14
----	----	----	----

6	67	33	42
---	----	----	----

98	23
----	----

45	14
----	----

6	67
---	----

33	42
----	----

98

23

45

14

6

67

33

42

23	98
----	----

14	45
----	----

6	67
---	----

33	42
----	----

14	23	45	98
----	----	----	----

6	33	42	67
---	----	----	----

6	14	23	33	42	45	67	98
---	----	----	----	----	----	----	----

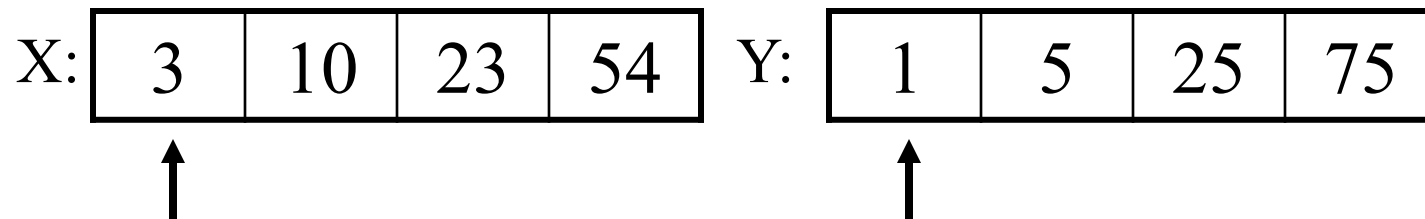
98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----



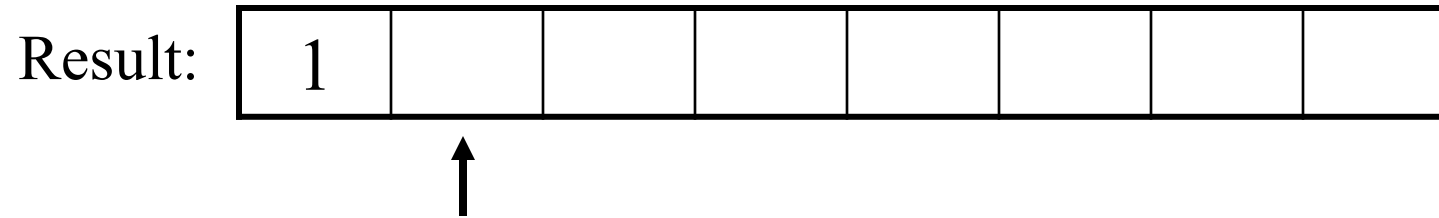
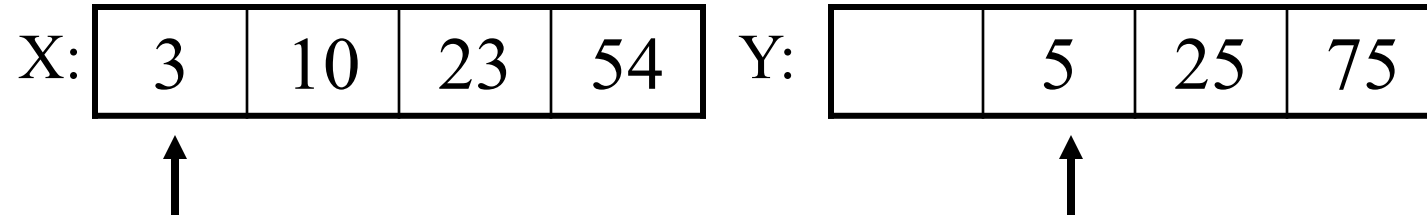
6	14	23	33	42	45	67	98
---	----	----	----	----	----	----	----

The Merging

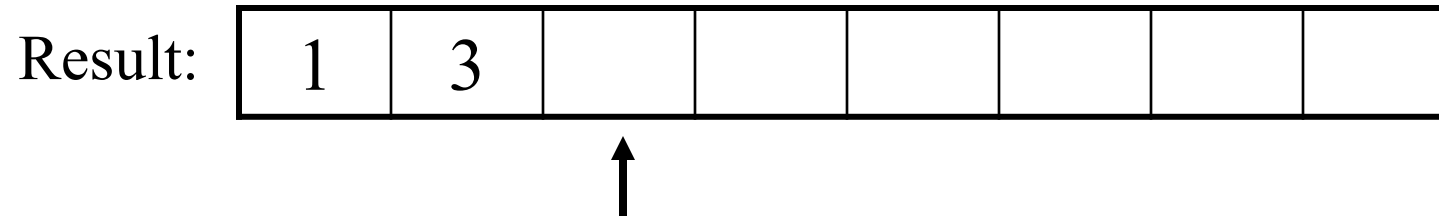
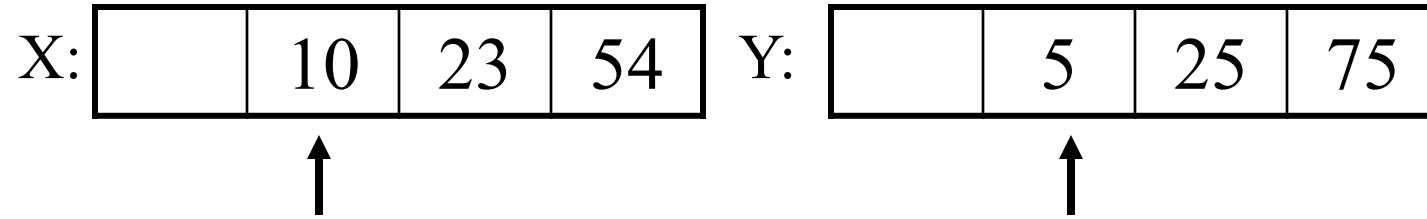
Merging



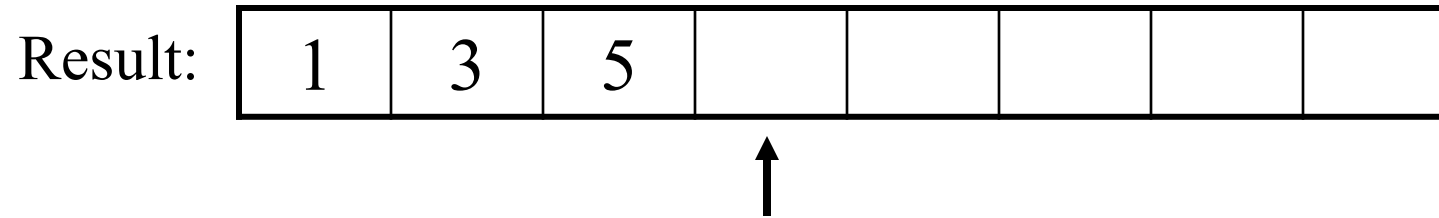
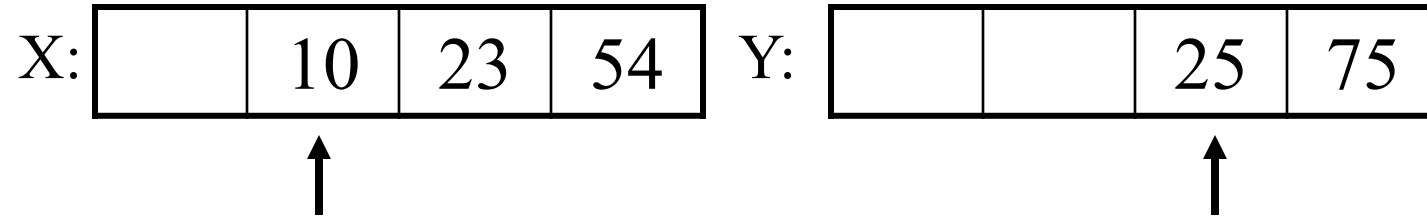
Merging (cont.)



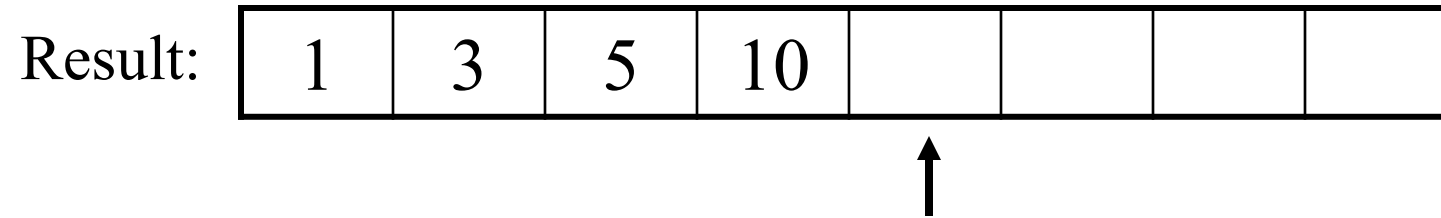
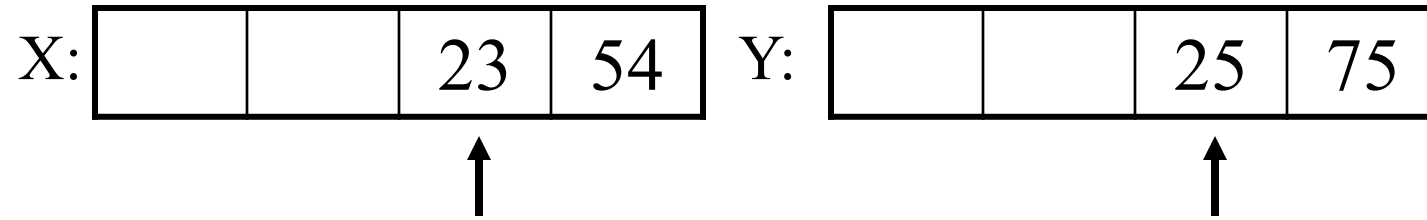
Merging (cont.)



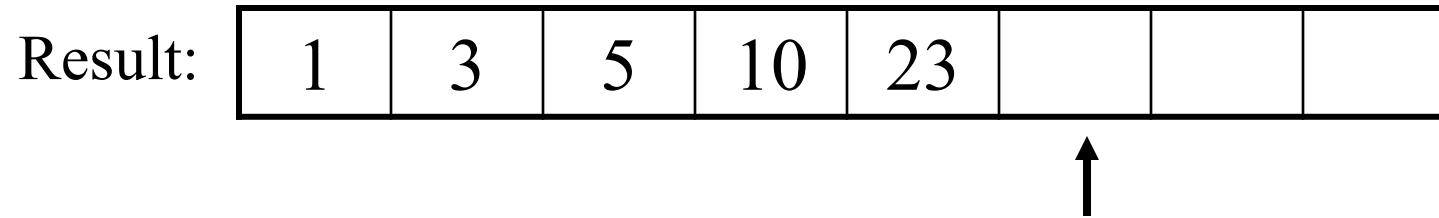
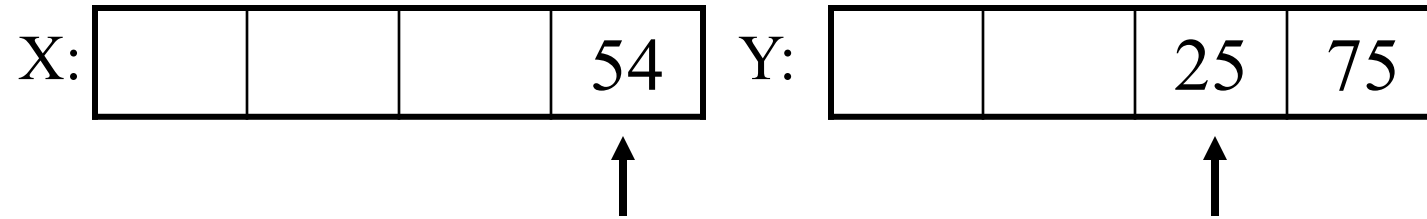
Merging (cont.)



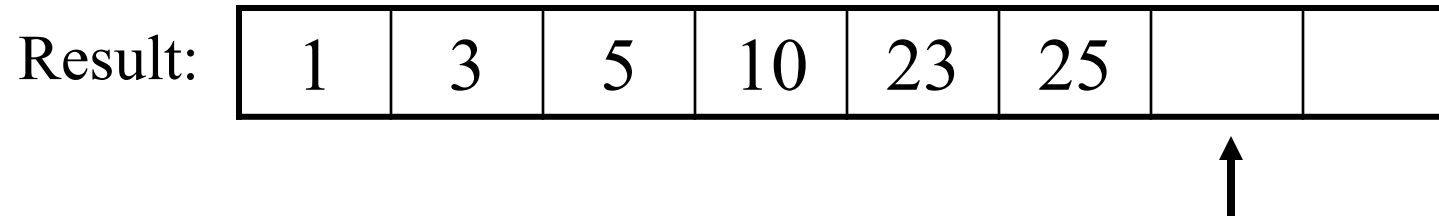
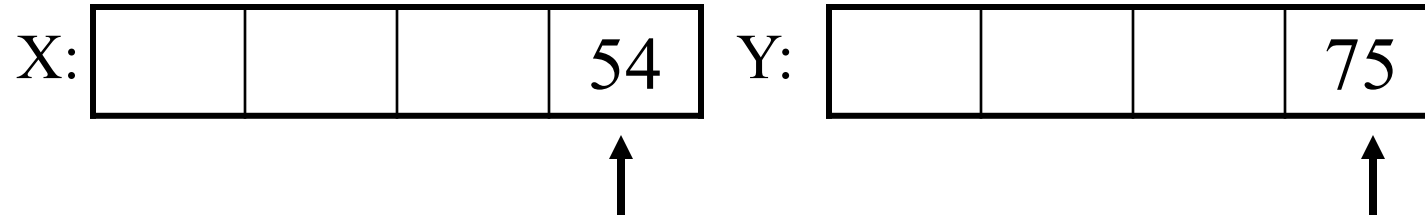
Merging (cont.)



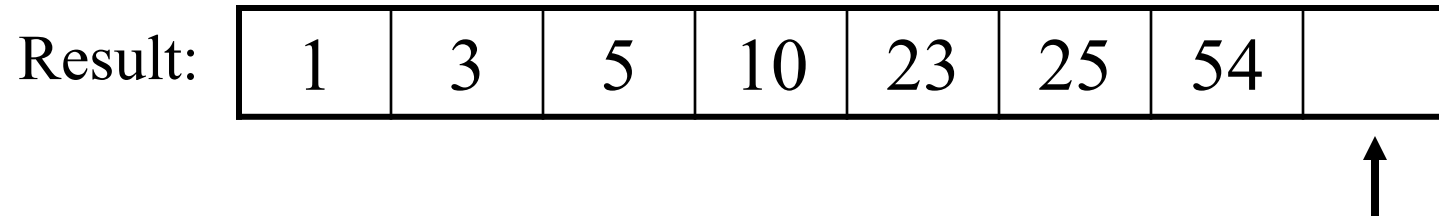
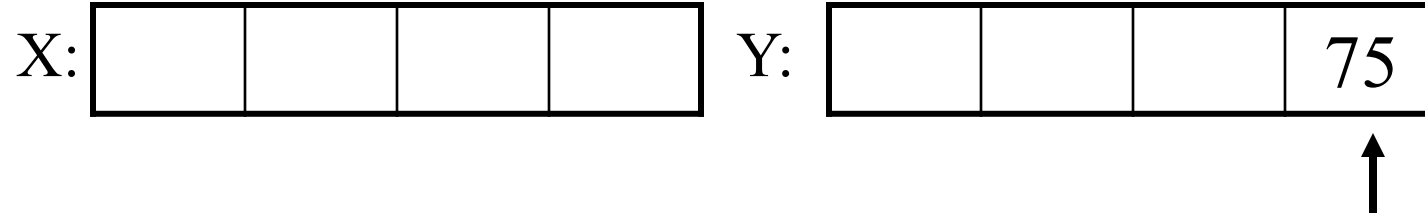
Merging (cont.)



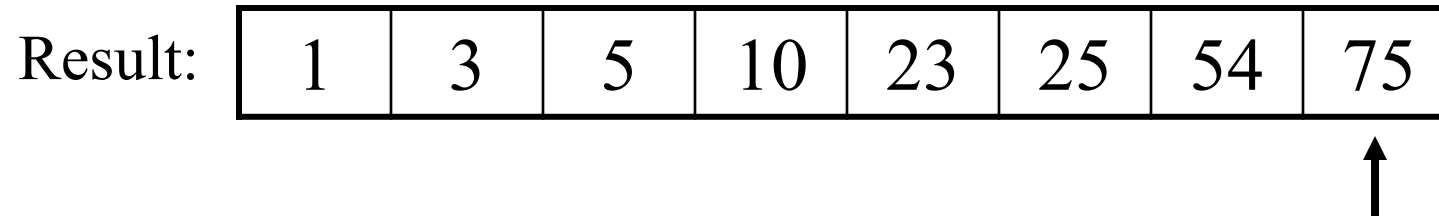
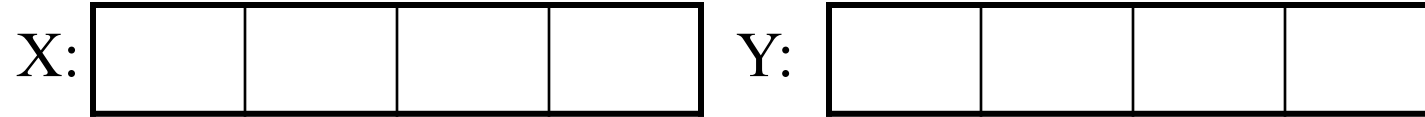
Merging (cont.)



Merging (cont.)



Merging (cont.)




```
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

```
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
```

```
/* Merge the temp arrays back into arr[l..r]*/  
i = 0; // Initial index of first subarray  
j = 0; // Initial index of second subarray  
k = 1; // Initial index of merged subarray  
while (i < n1 && j < n2)  
{  
    if (L[i] <= R[j])  
    {  
        arr[k] = L[i];  
        i++;  
    }  
    else  
    {  
        arr[k] = R[j];  
        j++;  
    }  
    k++;  
}
```

```
/* Copy the remaining elements of L[], if there  
   are any */
```

```
while (i < n1)
```

```
{
```

```
    arr[k] = L[i];
```

```
    i++;
```

```
    k++;
```

```
}
```

```
/* Copy the remaining elements of R[], if there  
   are any */
```

```
while (j < n2)
```

```
{
```

```
    arr[k] = R[j];
```

```
    j++;
```

```
    k++;
```

```
}
```

```
}
```

Merge Sort Complexity

- Merge sort always partitions the array equally.
- Thus, the recursive depth is always $O(\log n)$
- The amount of work done at each level for merging is $O(n)$
- Intuitively, the complexity should be $O(n \log n)$
- We have,
 - $T(n) = 2T(n/2) + \Theta(n)$ for $n > 1$, $T(1) = 0 \Rightarrow \Theta(n \log n)$
- The amount of extra memory used is $O(n)$

Merge Sort - Limitations

- Its not a in-place sorting algorithm!
- **What do you mean by “in-place”?** A standard merge sort requires you to allocate a buffer equal in length to the input being sorted.
 - For example, if you wish to sort an array with 1000 elements, it is necessary to allocate an additional scratch array with 1000 elements as a space for merging elements before copying them back to the input array.

Can you design a in-place Merge Sort?

Merge Sort – In-Place Variation

- The idea is to begin from last element of ar2[] and search it in ar1[].
- If there is a greater element in ar1[], then we move last element of ar1[] to ar2[].
- To keep ar1[] and ar2[] sorted, we need to place last element of ar2[] at correct place in ar1[].
- We can use **Insertion Sort** type of insertion for this.

Input:

ar1[]	1	5	9	10	15	20
ar2[]	2	3	8	13		



Ist iteration:

ar1[]	1	5	9	10	13	15
ar2[]	2	3	8	20		



IIInd Iteration:

ar1[]	1	5	8	9	10	13
ar2[]	2	3	15	20		



IIIrd Iteration:

ar1[]	1	3	5	8	9	10
ar2[]	2	13	15	20		



IVth Iteration:

ar1[]	1	2	3	5	8	9
ar2[]	10	13	15	20		

Merge Sort – In-Place Variation - Assignment

- Obviously write the normal Merge Sort.
- Write the above example in code!
- 2 variations
 - Linear search Insertion – **complexity?**
 - Binary search Insertion – **complexity?**
 - **Any other method?**

Bottom line: memory is cheap, use the scratch array!