Sorting

Saikrishna Arcot
M. Hudachek-Buswell

August 18, 2018

Sorting - Divide and Conquer Algorithms

- A divide and conquer algorithm divides a large problem into two or more smaller sub-problems of similar type.

Sorting - Divide and Conquer Algorithms

- A divide and conquer algorithm divides a large problem into two or more smaller sub-problems of similar type.
- The solutions to the sub-problems are combined to solve the larger original problem.

Sorting - Divide and Conquer Algorithms

- A divide and conquer algorithm divides a large problem into two or more smaller sub-problems of similar type.
- The solutions to the sub-problems are combined to solve the larger original problem.
- These algorithms can be done in-place or out-of-place.

Sorting - Divide and Conquer Algorithms

- A divide and conquer algorithm divides a large problem into two or more smaller sub-problems of similar type.
- The solutions to the sub-problems are combined to solve the larger original problem.
- These algorithms can be done in-place or out-of-place.
- Some of the algorithms are stable.

Sorting - Divide and Conquer Algorithms

- A divide and conquer algorithm divides a large problem into two or more smaller sub-problems of similar type.
- The solutions to the sub-problems are combined to solve the larger original problem.
- These algorithms can be done in-place or out-of-place.
- Some of the algorithms are stable.
- There can be improvement on time complexity from the $O(n^2)$ algorithms previously presented.

Sorting Divide and Conquer Algorithms

- We will cover these algortihms in this second set of slides

Sorting Divide and Conquer Algorithms

- We will cover these algortihms in this second set of slides
  - Merge sort

Sorting Divide and Conquer Algorithms

- We will cover these algortihms in this second set of slides
  - Merge sort
  - Quick sort

Sorting Divide and Conquer Algorithms

- We will cover these algortihms in this second set of slides
  - Merge sort
  - Quick sort
  - Radix sort

Sorting Divide and Conquer Algorithms

- We will cover these algortihms in this second set of slides
  - Merge sort
  - Quick sort
  - Radix sort
- All of the above algorithms except for the last one are known as **comparison sorts** because they directly compare two items; radix sort does not directly compare two items.

Sorting Divide and Conquer Algorithms

- We will cover these algortihms in this second set of slides
  - Merge sort
  - Quick sort
  - Radix sort
- All of the above algorithms except for the last one are known as **comparison sorts** because they directly compare two items; radix sort does not directly compare two items.
- These three sorting algorithms can improve on the performance $O(n^2)$.

Merge and Quick Sorts

- Unlike the previous sorts that have been covered, merge and quick sorts are recursive sorts rather than iterative sorts.

Merge and Quick Sorts

- Unlike the previous sorts that have been covered, merge and quick sorts are recursive sorts rather than iterative sorts.
- To be more specific, these two sorts are also known as divide-and-conquer sorts, which means that the array is divided into two (or more) parts, and those parts are then sorted. If needed, at the end, the sorted parts are merged together.

Merge and Quick Sorts

- Unlike the previous sorts that have been covered, merge and quick sorts are recursive sorts rather than iterative sorts.
- To be more specific, these two sorts are also known as divide-and-conquer sorts, which means that the array is divided into two (or more) parts, and those parts are then sorted. If needed, at the end, the sorted parts are merged together.
- One major advantage of this is that portions of the sorting algorithm can be done in parallel (i.e. in multiple threads).

Merge and Quick Sorts

- Unlike the previous sorts that have been covered, merge and quick sorts are recursive sorts rather than iterative sorts.
- To be more specific, these two sorts are also known as divide-and-conquer sorts, which means that the array is divided into two (or more) parts, and those parts are then sorted. If needed, at the end, the sorted parts are merged together.
- One major advantage of this is that portions of the sorting algorithm can be done in parallel (i.e. in multiple threads).
- In both cases, remember that an array of length 1 is always sorted.

Merge Sort

- In merge sort, divide the array into 2 equal parts; one part contains the elements from the left half of the array while the other part contains the elements from the right half of the array. (If there is an odd number of elements, the middle element will go to either the first part or the second part.)

Merge Sort

- In merge sort, divide the array into 2 equal parts; one part contains the elements from the left half of the array while the other part contains the elements from the right half of the array. (If there is an odd number of elements, the middle element will go to either the first part or the second part.)
- Then, perform merge sort on each half of the array. After this is done, each part should be sorted.

Merge Sort

- Finally, merge the two parts together. To do this, have a marker on the first item in each part. Take the smaller of the two items and add that into the larger (merged) array, and move that marker forward. Repeat until all of the items have been added into the merged array.

Merge Sort

- Finally, merge the two parts together. To do this, have a marker on the first item in each part. Take the smaller of the two items and add that into the larger (merged) array, and move that marker forward. Repeat until all of the items have been added into the merged array.
- Note that while merging the two parts together, if all of the items in one of the parts have been added into the larger array, you can directly copy over the remaining items from the other part into the larger array.

Merge Sort

86	6	31	72	38	43	93

Merge Sort

86	6	31	72	38	43	93

86	6	31

72	38	43	93

Merge Sort

86	6	31	72	38	43	93

86	6	31

72	38	43	93

86

6	31

Merge Sort

| 86 | 6 | 31 | 72 | 38 | 43 | 93 |

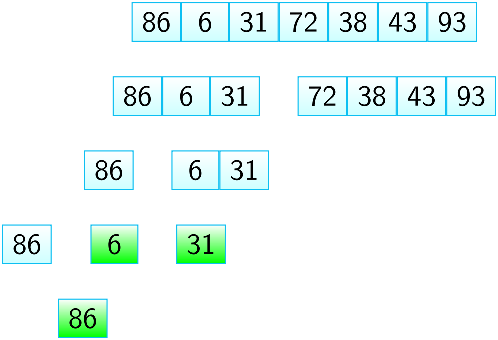
| 86 | 6 | 31 | | 72 | 38 | 43 | 93 |

| 86 | | 6 | 31 |

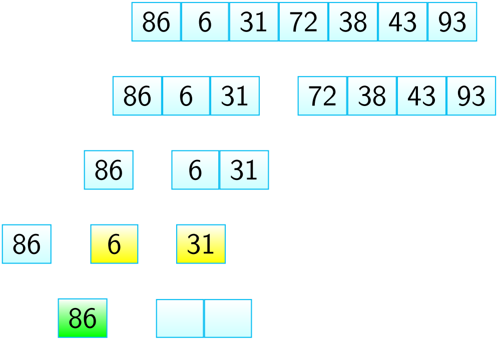
| 86 |

| 86 |

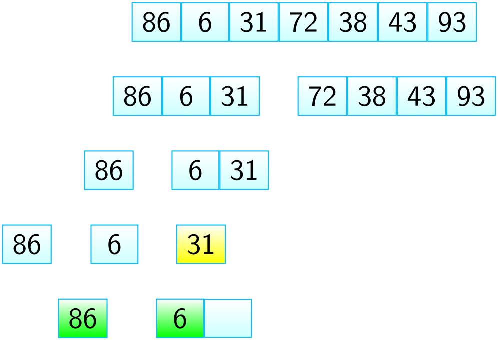
Merge Sort



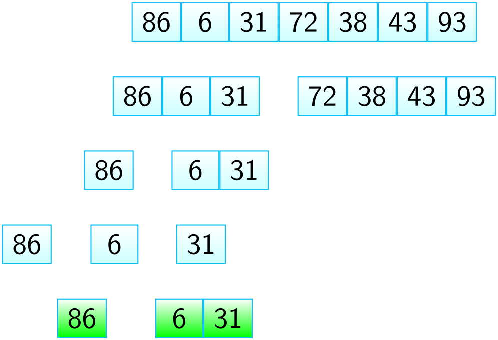
Merge Sort



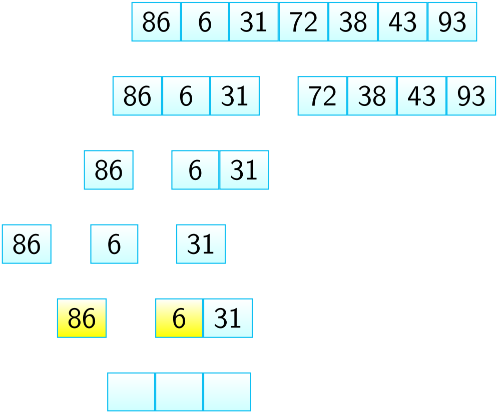
Merge Sort



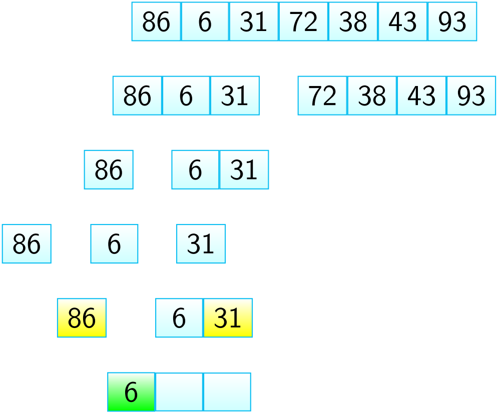
Merge Sort



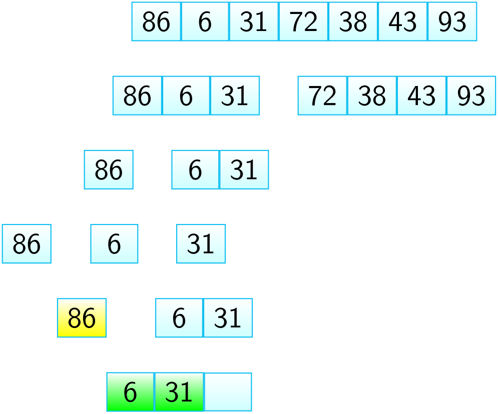
Merge Sort



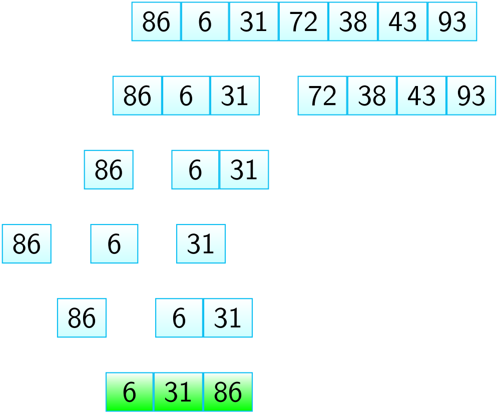
Merge Sort



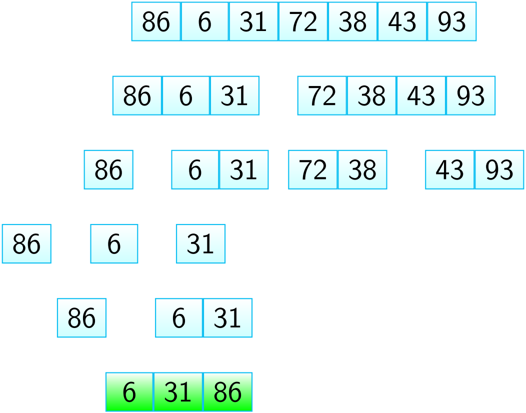
Merge Sort



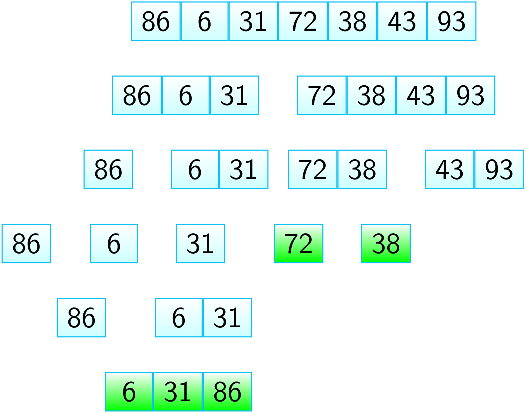
Merge Sort



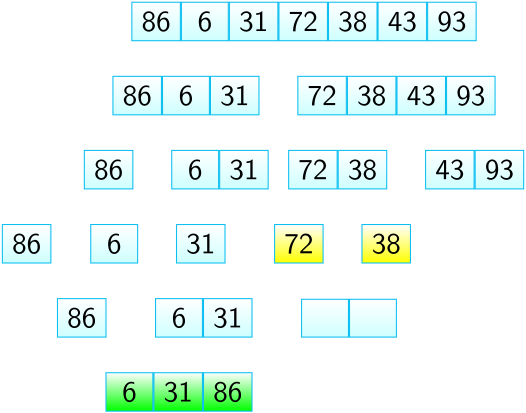
Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
86
6 31
6 31 86

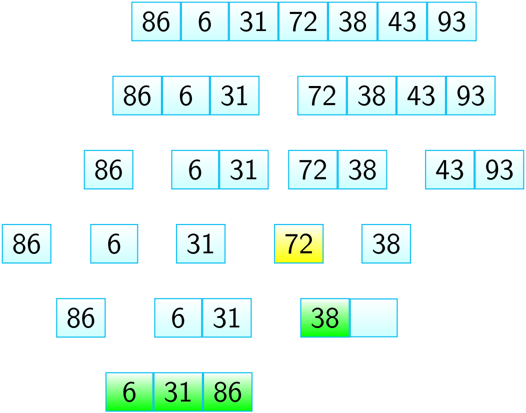
Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
86
6 31
6 31 86

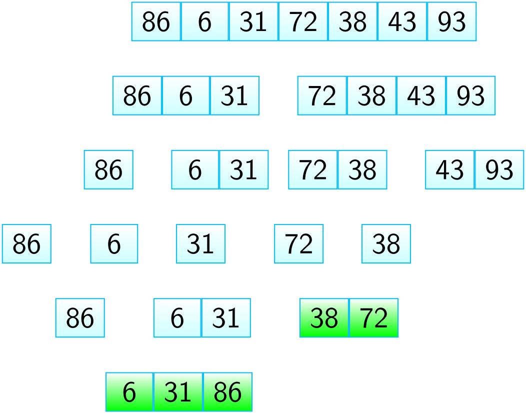
Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
86
6 31
6 31 86

Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
86
6 31
38
6 31 86

Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
86
6 31
38 72
6 31 86

Merge Sort

86	6	31	72	38	43	93

86	6	31

72	38	43	93

86

6 31

72 38

43 93

86 6 31 72 38 43 93

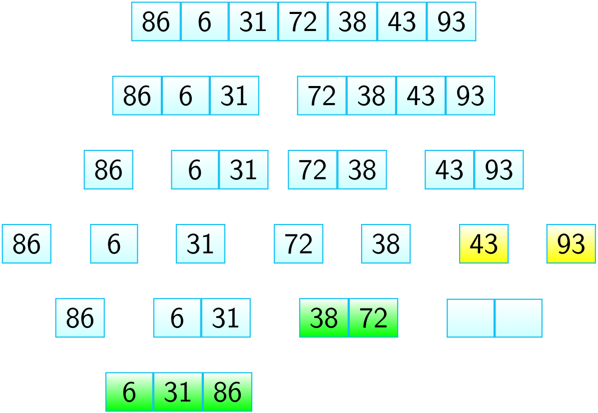
86

6 31

38 72

6	31	86

Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
43
93
86
6 31
38 72
6 31 86

Merge Sort

| 86 | 6 | 31 | 72 | 38 | 43 | 93 |

| 86 | 6 | 31 | | 72 | 38 | 43 | 93 |

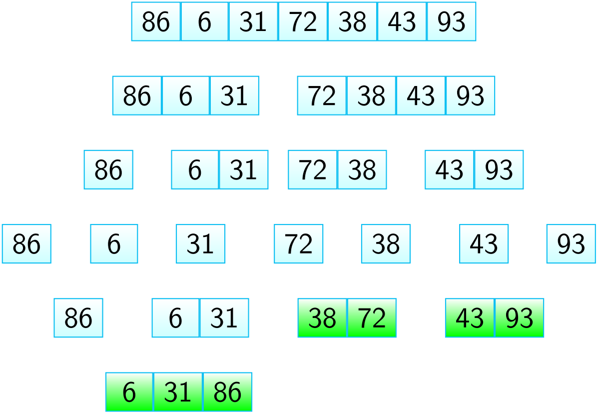
| 86 | | 6 | 31 | | 72 | 38 | | 43 | 93 |

| 86 | | 6 | | 31 | | 72 | | 38 | | 43 | | 93 |

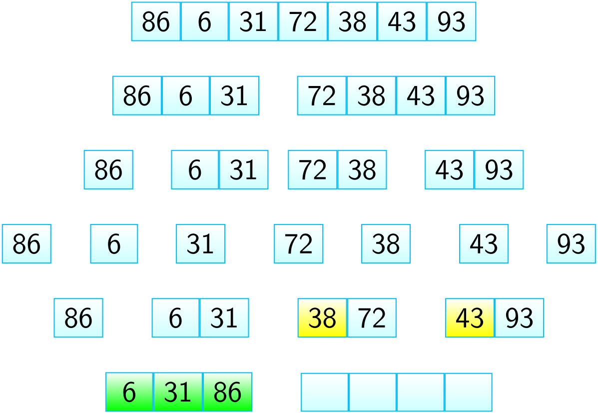
| 86 | | 6 | 31 | | 38 | 72 | | 43 | |

| 6 | 31 | 86 |

Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
43
93
86
6 31
38 72
43 93
6 31 86

Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
43
93
86
6 31
38 72
43 93
6 31 86

Merge Sort

86	6	31	72	38	43	93

86 6 31 | 72 38 43 93

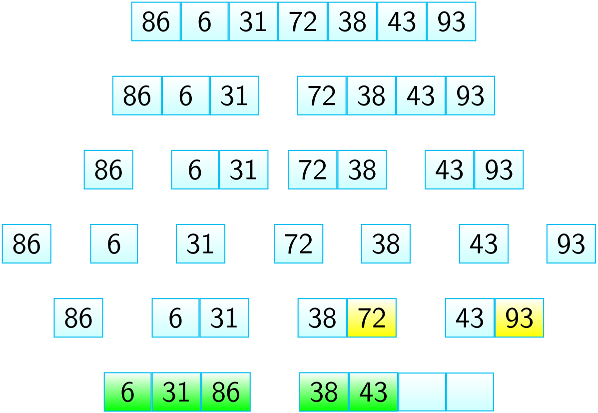
86 | 6 31 | 72 38 | 43 93

86 | 6 | 31 | 72 | 38 | 43 | 93

86 | 6 31 | 38 72 | 43 93

6 31 86 | 38

Merge Sort


86 6 31 72 38 43 93
86 6 31 72 38 43 93
86 6 31 72 38 43 93
86 6 31 72 38 43 93
86 6 31 38 72 43 93
6 31 86 38 43

Merge Sort

| 86 | 6 | 31 | 72 | 38 | 43 | 93 |

| 86 | 6 | 31 | | 72 | 38 | 43 | 93 |

| 86 | | 6 | 31 | | 72 | 38 | | 43 | 93 |

| 86 | | 6 | | 31 | | 72 | | 38 | | 43 | | 93 |

| 86 | | 6 | 31 | | 38 | 72 | | 43 | 93 |

| 6 | 31 | 86 | | 38 | 43 | 72 | |

Merge Sort

86 6 31 72 38 43 93

86 6 31   72 38 43 93

86   6 31   72 38   43 93

86   6   31   72   38   43   93

86   6 31   38 72   43 93

6 31 86   38 43 72 93

Merge Sort

86	6	31	72	38	43	93

86	6	31

72	38	43	93

86   6 31   72 38   43 93

86   6   31   72   38   43   93

86   6 31   38 72   43 93

6	31	86

38	43	72	93

Merge Sort

86	6	31	72	38	43	93

86 6 31   72 38 43 93

86   6 31   72 38   43 93

86   6   31   72   38   43   93

86   6 31   38 72   43 93

6 31 86   38 43 72 93

6						

Merge Sort

86	6	31	72	38	43	93

86 6 31   72 38 43 93

86   6 31   72 38   43 93

86   6   31   72   38   43   93

86   6 31   38 72   43 93

6 31 86   38 43 72 93

6	31					

Merge Sort

86 | 6 | 31 | 72 | 38 | 43 | 93

86 | 6 | 31   72 | 38 | 43 | 93

86   6 | 31   72 | 38   43 | 93

86   6   31   72   38   43   93

86   6 | 31   38 | 72   43 | 93

6 | 31 | 86   38 | 43 | 72 | 93

6 | 31 | 38 | | | |

Merge Sort

86	6	31	72	38	43	93

86 6 31   72 38 43 93

86   6 31   72 38   43 93

86   6   31   72   38   43   93

86   6 31   38 72   43 93

6 31 86   38 43 72 93

6	31	38	43			

Merge Sort

86 6 31 72 38 43 93

86 6 31 | 72 38 43 93

86 | 6 31 | 72 38 | 43 93

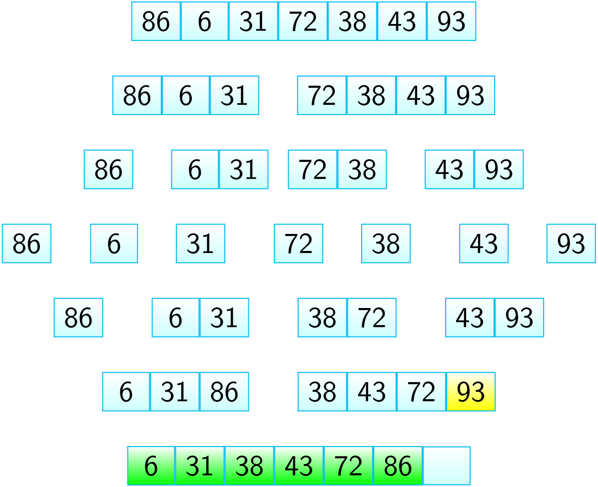
86 | 6 | 31 | 72 | 38 | 43 | 93

86 | 6 31 | 38 72 | 43 93

6 31 86 | 38 43 72 93

6 31 38 43 72

Merge Sort


86 6 31 72 38 43 93
86 6 31
72 38 43 93
86
6 31
72 38
43 93
86
6
31
72
38
43
93
86
6 31
38 72
43 93
6 31 86
38 43 72 93
6 31 38 43 72 86

Merge Sort

86	6	31	72	38	43	93

86	6	31		72	38	43	93

86   6 31   72 38   43 93

86   6   31   72   38   43   93

86   6 31   38 72   43 93

6 31 86   38 43 72 93

6	31	38	43	72	86	93

Merge Sort

procedure $\textsc{MergeSort}(array)$
  $length \leftarrow$ length of $array$
  $midIndex \leftarrow length/2$
  $leftArray \leftarrow array[0..midIndex - 1]$
  $rightArray \leftarrow array[midIndex..length - 1]$
  $\textsc{MergeSort}(leftArray)$
  $\textsc{MergeSort}(rightArray)$
  $leftIndex \leftarrow 0$
  $rightIndex \leftarrow 0$
  $currentIndex \leftarrow 0$

Merge Sort

while $leftIndex < midIndex$ and $rightIndex < length - midIndex$ **do**
  if $leftArray[leftIndex] <= rightArray[rightIndex]$ **then**
    $array[currentIndex] \leftarrow leftArray[leftIndex]$
    $leftIndex \leftarrow leftIndex + 1$
  else
    $array[currentIndex] \leftarrow rightArray[rightIndex]$
    $rightIndex \leftarrow rightIndex + 1$
  end if
  $currentIndex \leftarrow currentIndex + 1$
end while

Merge Sort

while $leftIndex < midIndex$ **do**
  $array[currentIndex] \leftarrow leftArray[leftIndex]$
  $leftIndex \leftarrow leftIndex + 1$
  $currentIndex \leftarrow currentIndex + 1$
end while
while $rightIndex < length - midIndex$ **do**
  $array[currentIndex] \leftarrow rightArray[rightIndex]$
  $rightIndex \leftarrow rightIndex + 1$
  $currentIndex \leftarrow currentIndex + 1$
end while
end procedure

Merge Sort Performance

- In the worst case (when the array is sorted in reverse order, for example), merge sort will run in $O(n \log n)$ time.

Merge Sort Performance

- In the worst case (when the array is sorted in reverse order, for example), merge sort will run in $O(n \log n)$ time.
- The best case for merge sort is when, for example, you are merging two arrays, and all of the items in one array are smaller than all of the items in the other array. In this case, you will make m comparisons, where m is the length of the smaller array. However, even in this case, the big-O of merge sort is $O(n \log n)$.

Merge Sort Performance

- In the worst case (when the array is sorted in reverse order, for example), merge sort will run in $O(n \log n)$ time.
- The best case for merge sort is when, for example, you are merging two arrays, and all of the items in one array are smaller than all of the items in the other array. In this case, you will make m comparisons, where m is the length of the smaller array. However, even in this case, the big-O of merge sort is $O(n \log n)$.
- Merge sort is out-of-place, but it is stable.

Quick Sort

- In quick sort, choose an item at random to be the pivot.

Quick Sort

- In quick sort, choose an item at random to be the pivot.
- Swap the pivot with the first item.

Quick Sort

- In quick sort, choose an item at random to be the pivot.
- Swap the pivot with the first item.
- Have a left “marker” that starts with the second item (the item after the pivot), and a right “marker” that starts at the last item.

Quick Sort

- In quick sort, choose an item at random to be the pivot.
- Swap the pivot with the first item.
- Have a left "marker" that starts with the second item (the item after the pivot), and a right "marker" that starts at the last item.
- If the item pointed to by the left marker is smaller than the pivot, move the marker one item to the right. Repeat this step until the marker points to an item that is larger than the pivot or goes *beyond* the right marker (they cross over). (If the item and the pivot are equal, then either can be done.)

Quick Sort

- If the item pointed to by the right marker is larger than the pivot, move the marker one item to the left. Repeat this step until the marker points to an item that is smaller than the pivot or goes *beyond* the left marker. (If the item and the pivot are equal, then either can be done.)

Quick Sort

- If the item pointed to by the right marker is larger than the pivot, move the marker one item to the left. Repeat this step until the marker points to an item that is smaller than the pivot or goes *beyond* the left marker. (If the item and the pivot are equal, then either can be done.)
- After the markers cross over, swap the pivot with the right marker (note that the right marker is now **to the left of the left marker**).

Quick Sort

- If the item pointed to by the right marker is larger than the pivot, move the marker one item to the left. Repeat this step until the marker points to an item that is smaller than the pivot or goes *beyond* the left marker. (If the item and the pivot are equal, then either can be done.)
- After the markers cross over, swap the pivot with the right marker (note that the right marker is now **to the left of the left marker**).
- The pivot is now in the right place within the final sorted array. All items to the left of the pivot (if there are any) are smaller than the pivot, and all items to the right of the pivot (if there are any) are larger than the pivot. Perform quicksort on the smaller items and on the larger items.

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

86	6	31	72	38	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

38	6	31	72	86	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

	L						
38	6	31	72	86	43	93	69
							R

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

		L					
38	6	31	72	86	43	93	69
							R

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

L

38	6	31	72	86	43	93	69

R

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

			L				
38	6	31	72	86	43	93	69
						R	

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

			L				
38	6	31	72	86	43	93	69
					R		

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

			L				
38	6	31	72	86	43	93	69
				R			

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

			L				
38	6	31	72	86	43	93	69
			R				

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

			L				
38	6	31	72	86	43	93	69
		R					

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

31	6

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

31	6

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6

38

72	86	43	93	69

L

31	6

R

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

L

31	6

R

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6		38		72	86	43	93	69

6	31

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6		38		72	86	43	93	69

6		31

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

6	31	38

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6		38		72	86	43	93	69

6		31		38		72	86	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)

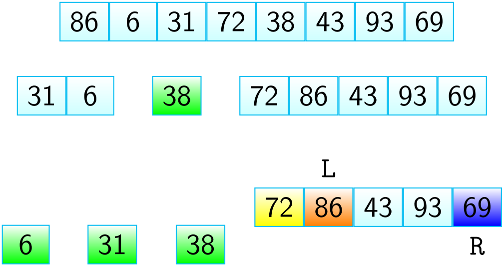
86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

6	31	38	72	86	43	93	69

Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6		38		72	86	43	93	69

L

72	69	43	93	86

6		31		38

R

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31	6	38	72	86	43	93	69

L

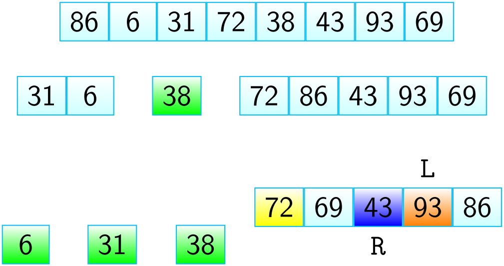
72	69	43	93	86

R

6	31	38

Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31 6   38   72 86 43 93 69

6   31   38   43 69 72 93 86

Quick Sort

(Randomly-selected pivots are in yellow.)

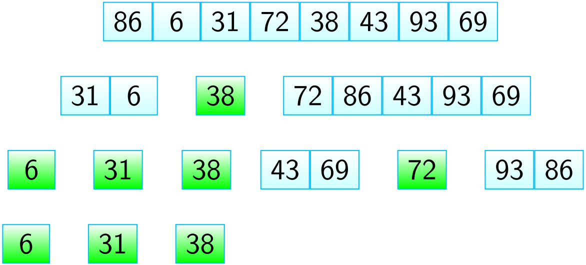
86	6	31	72	38	43	93	69

31 6   38   72 86 43 93 69

6   31   38   43 69   72   93 86

Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

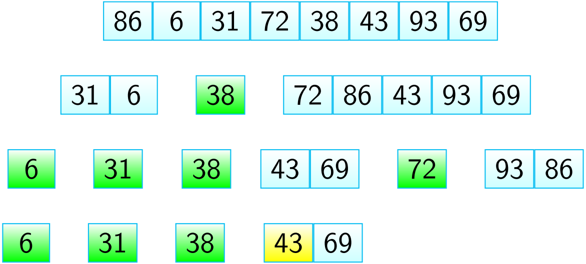
31 6   38   72 86 43 93 69

6   31   38   43 69   72   93 86

6   31   38   43 69

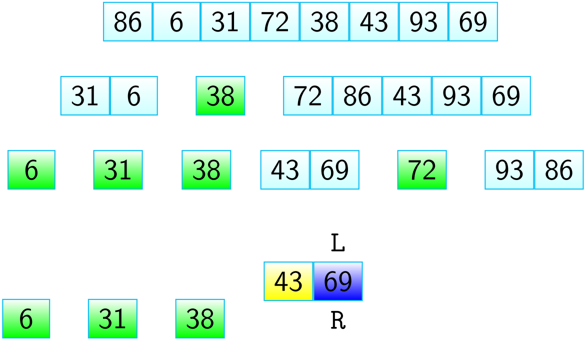
Quick Sort

(Randomly-selected pivots are in yellow.)



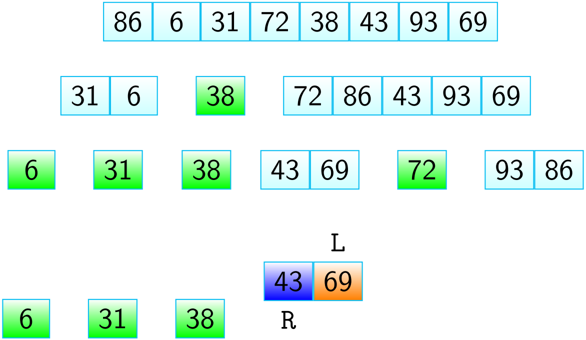
Quick Sort

(Randomly-selected pivots are in yellow.)



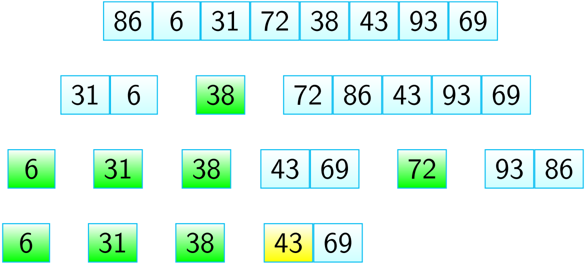
Quick Sort

(Randomly-selected pivots are in yellow.)



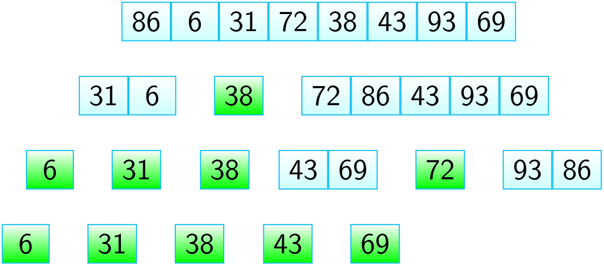
Quick Sort

(Randomly-selected pivots are in yellow.)



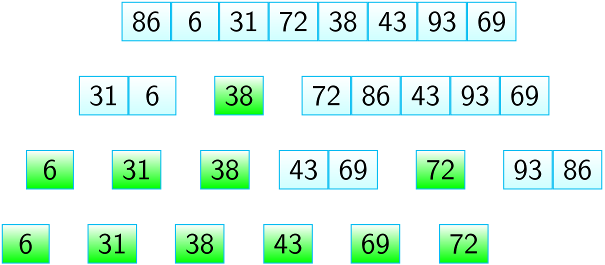
Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

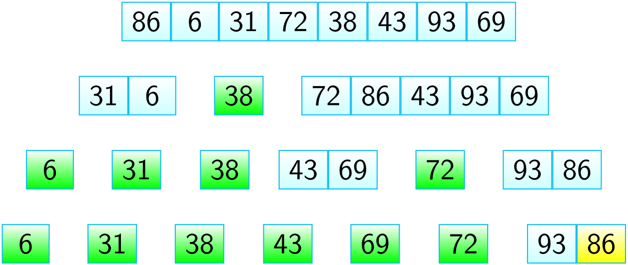
31 6   38   72 86 43 93 69

6   31   38   43 69   72   93 86

6   31   38   43   69   72   93 86

Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

(Randomly-selected pivots are in yellow.)

86 6 31 72 38 43 93 69

31 6 38 72 86 43 93 69

6 31 38 43 69 72 93 86

L

86 93

R

6 31 38 43 69 72

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

31 6   38   72 86 43 93 69

6   31   38   43 69   72   93 86

L

6   31   38   43   69   72   86 93

R

Quick Sort

(Randomly-selected pivots are in yellow.)

86 6 31 72 38 43 93 69

31 6   38   72 86 43 93 69

6   31   38   43 69   72   93 86

6   31   38   43   69   72   86 93

Quick Sort

(Randomly-selected pivots are in yellow.)

86	6	31	72	38	43	93	69

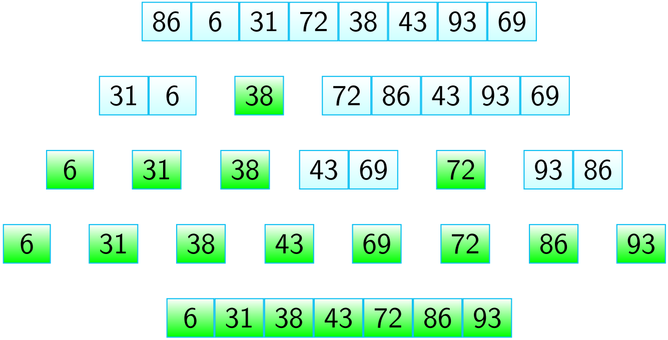
31	6		38		72	86	43	93	69

6		31		38		43	69		72		93	86

6		31		38		43		69		72		86		93

Quick Sort

(Randomly-selected pivots are in yellow.)



Quick Sort

procedure QUICKSORT(*array*)
    QUICKSORT(*array*, 0, length of *array*)
end procedure
procedure QUICKSORT(*array*, *left*, *right*)
    pivotIndex ← randomly-selected index within bounds of region being sorted
    pivot ← *array*[*pivotIndex*]
    Swap *array*[*left*] and *array*[*pivotIndex*]
    leftIndex ← *left* + 1
    rightIndex ← *right* − 1

Quick Sort

while $leftIndex <= rightIndex$ **do**
    while $leftIndex <= rightIndex$ and $array[leftIndex] <= pivot$ **do**
        $leftIndex \leftarrow leftIndex + 1$
    end while
    while $leftIndex <= rightIndex$ and $array[rightIndex] >= pivot$ **do**
        $rightIndex \leftarrow rightIndex - 1$
    end while

Quick Sort

if *leftIndex* $<=$ *rightIndex* **then**
    Swap *array*[*leftIndex*] and *array*[*rightIndex*]
    $leftIndex \leftarrow leftIndex + 1$
    $rightIndex \leftarrow rightIndex - 1$
end if
end while
Swap *pivot* and *array*[*rightIndex*]
QuickSort(*array*, *left*, *rightIndex*)
QuickSort(*array*, *rightIndex* + 1, *right*)
end procedure

Quick Sort Performance

- In the best case, the pivot that is chosen each time will divide the array in half. This results in $O(n \log n)$ performance.

Quick Sort Performance

- In the best case, the pivot that is chosen each time will divide the array in half. This results in $O(n \log n)$ performance.
- In the worst case, the pivot that is chosen each time will be either the largest or smallest value. This means that all of the other values will be smaller or larger than the pivot. This effectively turns quicksort into something like selection sort, and results in $O(n^2)$ performance.

Quick Sort Performance

- In the best case, the pivot that is chosen each time will divide the array in half. This results in $O(n \log n)$ performance.
- In the worst case, the pivot that is chosen each time will be either the largest or smallest value. This means that all of the other values will be smaller or larger than the pivot. This effectively turns quicksort into something like selection sort, and results in $O(n^2)$ performance.
- This quick sort is in-place, but it is **not** stable. Quick sort can also be done such that it is stable, but it must be out-of-place.

Radix Sort

- Radix sort is different than all of the previous sorting algorithms in that no comparisons are actually done.

Radix Sort

- Radix sort is different than all of the previous sorting algorithms in that no comparisons are actually done.
- However, radix sort can only be done on numbers of some base (base 10, base 8, base 16, base 256 (this lets you perform radix sort on ASCII letters), etc.)

Radix Sort

- Radix sort is different than all of the previous sorting algorithms in that no comparisons are actually done.
- However, radix sort can only be done on numbers of some base (base 10, base 8, base 16, base 256 (this lets you perform radix sort on ASCII letters), etc.)
- There are two variants of radix sort:

Radix Sort

- Radix sort is different than all of the previous sorting algorithms in that no comparisons are actually done.
- However, radix sort can only be done on numbers of some base (base 10, base 8, base 16, base 256 (this lets you perform radix sort on ASCII letters), etc.)
- There are two variants of radix sort:
    - One variant starts by looking at the least significant digit and works upwards. This is called Least Significant Digit (LSD) Radix Sort.

Radix Sort

- Radix sort is different than all of the previous sorting algorithms in that no comparisons are actually done.
- However, radix sort can only be done on numbers of some base (base 10, base 8, base 16, base 256 (this lets you perform radix sort on ASCII letters), etc.)
- There are two variants of radix sort:
    - One variant starts by looking at the least significant digit and works upwards. This is called Least Significant Digit (LSD) Radix Sort.
    - The other variant starts by looking at the most significant digit and works downwards. This is called Most Significant Digit (MSD) Radix Sort.

LSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. Treat each bucket as a queue.

LSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. Treat each bucket as a queue.
- For each number, take the first digit (least significant digit), and add the number into the appropriate bucket. (For example, if the number is 27, then the first digit is 7, and add the number into bucket 7.)

LSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. Treat each bucket as a queue.
- For each number, take the first digit (least significant digit), and add the number into the appropriate bucket. (For example, if the number is 27, then the first digit is 7, and add the number into bucket 7.)
- After all of the numbers have been added, remove all of the numbers one at a time, starting from bucket 0.

LSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. Treat each bucket as a queue.
- For each number, take the first digit (least significant digit), and add the number into the appropriate bucket. (For example, if the number is 27, then the first digit is 7, and add the number into bucket 7.)
- After all of the numbers have been added, remove all of the numbers one at a time, starting from bucket 0.
- Repeat this process for each digit in the *longest* (not necessarily the *largest*) number. (In other words, if the longest number has 4 digits, then you would repeat this process 3 more times.)

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
						86			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
						86			
						6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31					86			
						6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72				86			
						6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72				86		38	
						6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	
						6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31							

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31							

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72						

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72						

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43					

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93				

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93				

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93				

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93				

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86			

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6		

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6		

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6		

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
	31	72	43			86		38	69
			93			6			

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
			31						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
			31				72		

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
			31	43			72		

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
			31	43			72		93

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
			31	43			72	86	93

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
6			31	43			72	86	93

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
6			31	43			72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6							

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6							

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6							

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6							

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31						

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38					

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38					

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43				

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43				

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43				

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69			

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69			

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69	72		

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69	72		

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69	72	86	

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69	72	86	

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69	72	86	93

0	1	2	3	4	5	6	7	8	9

LSD Radix Sort

86	6	31	72	38	43	93	69

31	72	43	93	86	6	38	69

6	31	38	43	69	72	86	93

0	1	2	3	4	5	6	7	8	9

LSD Radix Sort

procedure LSDRadixSort(*array*)
    buckets ← list of 10 lists
    iterations ← length of longest number
    length ← length of array
    for $i \leftarrow 1, iterations$ **do**
        for $j \leftarrow 0, length - 1$ **do**
            $bucket \leftarrow i^{th}$ digit of $array[j]$
            add $array[j]$ to the end of $buckets[bucket]$
        end for
        $index \leftarrow 0$

LSD Radix Sort

for $bucket \leftarrow 0, 10$ **do**
    while $buckets[bucket]$ isn't empty **do**
        $array[index] \leftarrow$ remove first item from $buckets[bucket]$
        $index \leftarrow index + 1$
    end while
end for
end for
end procedure

LSD Radix Sort Performance

- Unlike the previous sorting algorithms, the efficiency of radix sort depends on the number of items in the array (n) *and* on the length of the longest number (k).

LSD Radix Sort Performance

- Unlike the previous sorting algorithms, the efficiency of radix sort depends on the number of items in the array (n) *and* on the length of the longest number (k).
- In the best, worst, and average case, LSD radix sort runs in $O(kn)$ time.

LSD Radix Sort Performance

- Unlike the previous sorting algorithms, the efficiency of radix sort depends on the number of items in the array (n) *and* on the length of the longest number (k).
- In the best, worst, and average case, LSD radix sort runs in $O(kn)$ time.
- LSD radix sort is stable, but not in-place.

MSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. (Note that the buckets here aren’t exactly queues.)

MSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. (Note that the buckets here aren’t exactly queues.)
- Find the length of the longest number. This will be the digit you start with.

MSD Radix Sort

- (Assuming radix sort is done in base 10) Create 10 “buckets”, and label them from 0 to 9. (Note that the buckets here aren't exactly queues.)
- Find the length of the longest number. This will be the digit you start with.
- For each number, take the digit in the position you found in the previous step, and add the number into the appropriate bucket. (For example, if the longest number has 4 digits, then you would get the 4th digit (most significant digit) of each number.)

MSD Radix Sort

- After all of the numbers have been added into the buckets, for each bucket, if there are two or more numbers in the bucket, run MSD radix sort again, but use the next smaller/lower digit. After this is done, the numbers in each bucket should be sorted in ascending order.

MSD Radix Sort

- After all of the numbers have been added into the buckets, for each bucket, if there are two or more numbers in the bucket, run MSD radix sort again, but use the next smaller/lower digit. After this is done, the numbers in each bucket should be sorted in ascending order.
- Starting with the first bucket, remove the first number of each bucket until the bucket is empty. The numbers should now be sorted.

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
								86	

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6								86	

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31					86	

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31				72	86	

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31				72	86	
			38						

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43			72	86	
			38						

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43			72	86	93
			38						

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	38

0	1	2	3	4	5	6	7	8	9

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	38

0	1	2	3	4	5	6	7	8	9

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	38

0	1	2	3	4	5	6	7	8	9
	31								

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

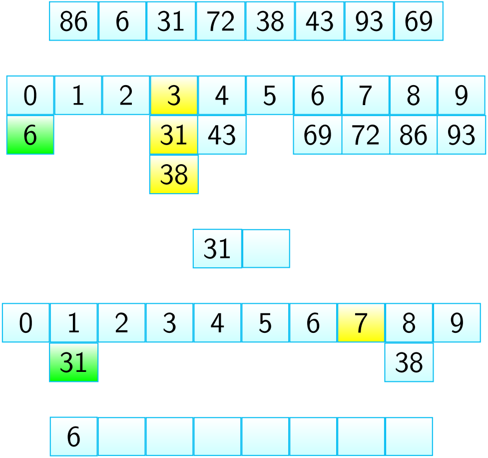
0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort



MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	38

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	38

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

31	38

0	1	2	3	4	5	6	7	8	9
	31							38	

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6							

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31						

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38					

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38					

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43				

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43				

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43				

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69			

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69			

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69	72		

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69	72		

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69	72	86	

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69	72	86	

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69	72	86	93

MSD Radix Sort

86	6	31	72	38	43	93	69

0	1	2	3	4	5	6	7	8	9
6			31	43		69	72	86	93
			38						

6	31	38	43	69	72	86	93

MSD Radix Sort

procedure $\textsc{MSDRadixSort}(array)$
    $maxLength \leftarrow$ length of longest number
    $\textsc{MSDRadixSort}(array, maxLength)$
end procedure
procedure $\textsc{MSDRadixSort}(array, i)$
    $buckets \leftarrow$ list of 10 lists
    $length \leftarrow$ length of array
    for $j \leftarrow 0, length - 1$ **do**
        $bucket \leftarrow i^{th}$ digit of $array[j]$
        add $array[j]$ to $buckets[bucket]$
    end for

MSD Radix Sort

$index \leftarrow 0$
for $bucket \leftarrow 0, 9$ **do**
  if number of items in $buckets[bucket] > 1$ and $i > 1$ **then**
    $\textsc{MSDRadixSort}(buckets[bucket], i - 1)$
  end if
  while $buckets[bucket]$ isn't empty **do**
    $array[index] \leftarrow$ remove first item from $buckets[bucket]$
    $index \leftarrow index + 1$
  end while
end for
end procedure

MSD Radix Sort Performance

- In the best, worst, and average case, MSD radix sort runs in $O(kn)$ time, where k is the length of the longest number.

MSD Radix Sort Performance

- In the best, worst, and average case, MSD radix sort runs in $O(kn)$ time, where k is the length of the longest number.
- However, note that, when “long” numbers are used, and the most significant digits are different, then MSD radix sort will likely take fewer iterations to finish than LSD radix sort, possibly just one (if there are few numbers).

MSD Radix Sort Performance

- In the best, worst, and average case, MSD radix sort runs in $O(kn)$ time, where k is the length of the longest number.
- However, note that, when "long" numbers are used, and the most significant digits are different, then MSD radix sort will likely take fewer iterations to finish than LSD radix sort, possibly just one (if there are few numbers).
- MSD radix sort is stable only if a memory buffer is used, which is usually done in implementations for this class.