Week 14:
Sorting Algorithms

CS 220: Software Design II — D. Mathias

Sorting

- Sorted data is crucial to human usability of data
  - e.g., phone book, entering grades, dates
- Sorted data is **also** crucial to computational efficiency in accessing data
  - i.e., how can a computer most efficiently find data?
- So, how do we sort data?

Sorting Algorithms

- There are dozens of sorting algorithms[1]
- Sorting algorithms can be evaluated in many ways
  - run time
  - memory usage
  - general approach
    - e.g., exchanging, sorting
  - parallelizability
    - i.e., can it be performed in parallel?

1: https://en.wikipedia.org/wiki/Sorting_algorithm

Sorting Algorithms We'll Explore

- selection sort (typically iterative)
- insertion sort (typically iterative)
- merge sort (typically recursive)
- quicksort (typically recursive)

Selection Sort

- Considered one of the classic sorting algorithms
- Very simple, but very inefficient (this tradeoff often occurs)

- Thumbnail sketch:
  - scan through the array multiple times
  - each time find the smallest “remaining” element
  - move that element to correct position

Selection Sort

- Array is divided into two parts: sorted (left part) and unsorted (right part)
  - initially, everything is unsorted
- Scan through the unsorted part for the smallest element
- Swap the smallest element with the leftmost unsorted value
- Length of sorted part increases by one, length of unsorted part decreases by one
- Repeat

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{8 \quad 3 \quad 2 \quad 5 \quad 9 \quad 7}$

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{8}\quad 3\quad 2\quad 5\quad 9\quad 7$

`smallestIndex = 0`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$8\quad \boxed{3}\quad 2\quad 5\quad 9\quad 7$

`smallestIndex = 1`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$8\quad 3\quad \boxed{2}\quad 5\quad 9\quad 7$

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$8\quad 3\quad 2\quad \boxed{5}\quad 9\quad 7$

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$8 \quad 3 \quad 2 \quad 5 \quad \boxed{9} \quad 7$

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$8 \quad 3 \quad 2 \quad 5 \quad 9 \quad \boxed{7}$

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{8} \quad 3 \quad \boxed{2} \quad 5 \quad 9 \quad 7$

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2} \quad 3 \quad \boxed{8} \quad 5 \quad 9 \quad 7$

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2}$ 3 8 5 9 7

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2}$ $\boxed{3 \quad 8 \quad 5 \quad 9 \quad 7}$

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2}$ $\boxed{3}$ 8 5 9 7

`smallestIndex = 1`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2}$ 3 $\boxed{8}$ 5 9 7

`smallestIndex = 1`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

```
2   3   8   5   9   7

smallestIndex = 1
```

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

```
2   3   8   5   9   7

smallestIndex = 1
```

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

```
2   3   8   5   9   7

smallestIndex = 1
```

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

```
2   3   8   5   9   7

smallestIndex = 1
```

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

`smallestIndex = 2`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2 \quad 3 \quad 5} \quad 8 \quad 9 \quad 7$

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2 \quad 3 \quad 5} \; \boxed{8 \quad 9 \quad 7}$

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2 \quad 3 \quad 5} \; \boxed{8} \quad 9 \quad 7$

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2 \quad 3 \quad 5} \quad 8 \; \boxed{9} \; 7$

`smallestIndex = 3`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 9 8

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 9 8

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	7	9	8

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	7	9	8

`smallestIndex = 4`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	7	9	8

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	7	9	8

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 8 9

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 8 9

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 8 9

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 8 9

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2\quad 3\quad 5\quad 7\quad 8}\ \boxed{9}$

`smallestIndex = 5`

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

$\boxed{2\quad 3\quad 5\quad 7\quad 8\quad 9}$

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 8 9

Selection Sort: Code

```
private static void selectionSort(int arr[]) {
    // Gradually move boundary of unsorted portion
    for (int i = 0; i < arr.length-1; i++) {

        // Find the index of the smallest unsorted item
        int smallestIndex = i;
        for (int j = i+1; j < arr.length; j++) {
            if (arr[j] < arr[smallestIndex]) {
                smallestIndex = j;
            }
        }

        // Swap
        int temp = arr[smallestIndex];
        arr[smallestIndex] = arr[i];
        arr[i] = temp;
    }
}
```

Insertion Sort

- Considered one of the classic sorting algorithms
- Very simple, but very inefficient

- Thumbnail sketch:
  - places next unsorted element into sorted part of array by…
  - …searching for correct position within the sorted part
  - that position may not be the element's final position

Insertion Sort

- Array is divided into two parts: sorted (left part) and unsorted (right part)
  - initially, first element is sorted, everything else is unsorted
- Look at the leftmost unsorted value
- Move it down the sorted list until it is in the correct place
- Length of sorted part increases by one, length of unsorted part decreases by one
- Repeat

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

8 | 3 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 8 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 8 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 8 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 8 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 8 2 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 2 8 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 | 9 | 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 | 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 | 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 8 9 | 7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	8	9	7

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	8	7	9

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	7	8	9

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2	3	5	7	8	9

Insertion Sort: Code

```
private static void insertionSort(int arr[]) {

    // Gradually look at each unsorted number
    for (int i = 0; i < arr.length; i++) {
        // Current value to sort
        int value = arr[i];
        int j = i-1;

        // Shift elements until value is in place
        while (j >= 0 && arr[j] > value) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = value;
    }
}
```

Algorithm Analysis

First, consider what is the best and worst case scenarios for sorting an array

Then, fill out the chart below with the run time (i.e., big O):

	selection sort	insertion sort
best case scenario		
worst case scenario		

Selection Sort & Insertion Sort

```
private static void selectionSort(int arr[]) {
    // Gradually move boundary of unsorted portion
    for (int i = 0; i < arr.length-1; i++) {

        // Find the index of the smallest unsorted item
        int smallestIndex = i;
        for (int j = i+1; j < arr.length; j++) {
            if (arr[j] == arr[smallestIndex]) {
                smallestIndex = j;
            }
        }

        // Swap
        int temp = arr[smallestIndex];
        arr[smallestIndex] = arr[i];
        arr[i] = temp;
    }
}
```

```
private static void insertionSort(int arr[]) {

    // Gradually look at each unsorted number
    for (int i = 0; i < arr.length; i++) {
        // Current value to sort
        int value = arr[i];
        int j = i-1;

        // Shift elements until value is in place
        while (j >= 0 && arr[j] > value) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = value;
    }
}
```

Algorithm Analysis

First, consider what are the best and worst case scenarios for each algorithm

Then, fill out the chart below with the run time (i.e., big O):

	selection sort	insertion sort
best case scenario		
worst case scenario		

Algorithm Analysis

First, consider what are the best and worst case scenarios for each algorithm

Then, fill out the chart below with the run time (i.e., big O):

	selection sort	**insertion sort**
best case scenario	$O(n^2)$	$O(n)$ (already sorted)
worst case scenario	$O(n^2)$	$O(n^2)$ (reverse order)

Merge Sort

- Considered one of the classic sorting algorithms
- More complex than selection/insertion sort
  - …but more efficient!

- Thumbnail sketch:
  - break the array up into individual elements
  - sorts pairs of elements, then pairs of pairs, etc…until you have one unified array

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6 3 8

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6 3 8

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6 | 3 8

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6 | 3 8 | 2 5

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6 | 3 8 | 2 5

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 4 8 3 2 5 9 7

4 6 | 3 8 | 2 5

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6	4	8	3	2	5	9	7
4	6	3	8	2	5	7	9

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6	4	8	3	2	5	9	7
4	6	3	8	2	5	7	9

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6	4	8	3	2	5	9	7
4	6	3	8	2	5	7	9

Merge Sort

Array is divided into its smallest unit

i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6	4	8	3	2	5	9	7
4	6	3	8	2	5	7	9

N.B.: the individual subarrays are already sorted, so we just need to compare the first element in each subarray

Merge Sort

Array is divided into its smallest unit

i.e., a single element

6 4 8 3 2 5 9 7

Sort and merge each paired subarray of elements

4 6 | 3 8 | 2 5 | 7 9

Repeat sort/merge until there is only one array

3

Merge Sort

Array is divided into its smallest unit

i.e., a single element

6 4 8 3 2 5 9 7

Sort and merge each paired subarray of elements

4 6 | 3 8 | 2 5 | 7 9

Repeat sort/merge until there is only one array

3 4

Merge Sort

Array is divided into its smallest unit

i.e., a single element

6 4 8 3 2 5 9 7

Sort and merge each paired subarray of elements

4 6 | 3 8 | 2 5 | 7 9

Repeat sort/merge until there is only one array

3 4 6

Merge Sort

Array is divided into its smallest unit

i.e., a single element

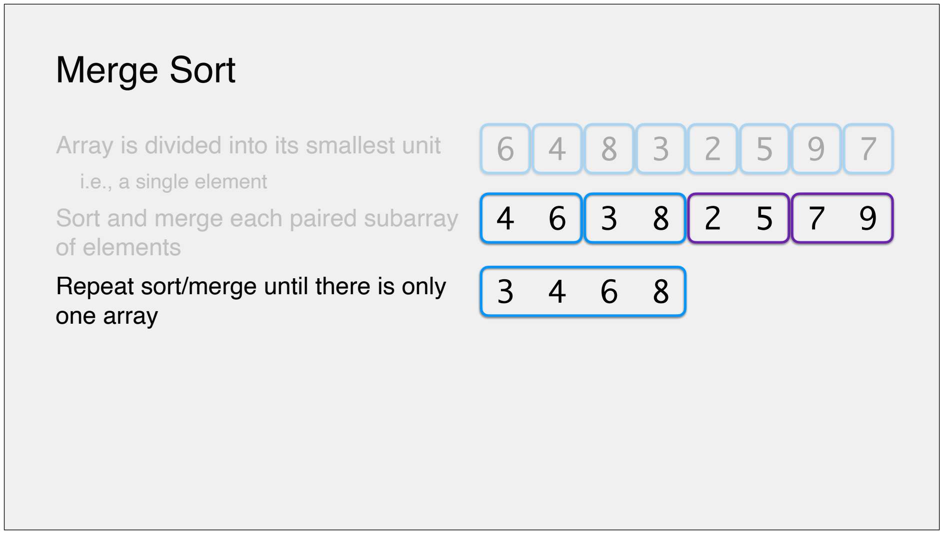
6 4 8 3 2 5 9 7

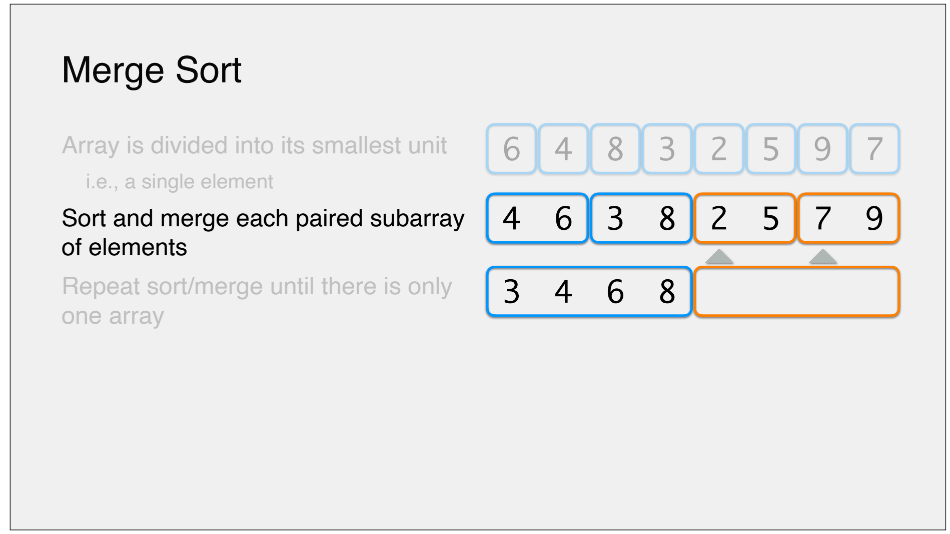
Sort and merge each paired subarray of elements

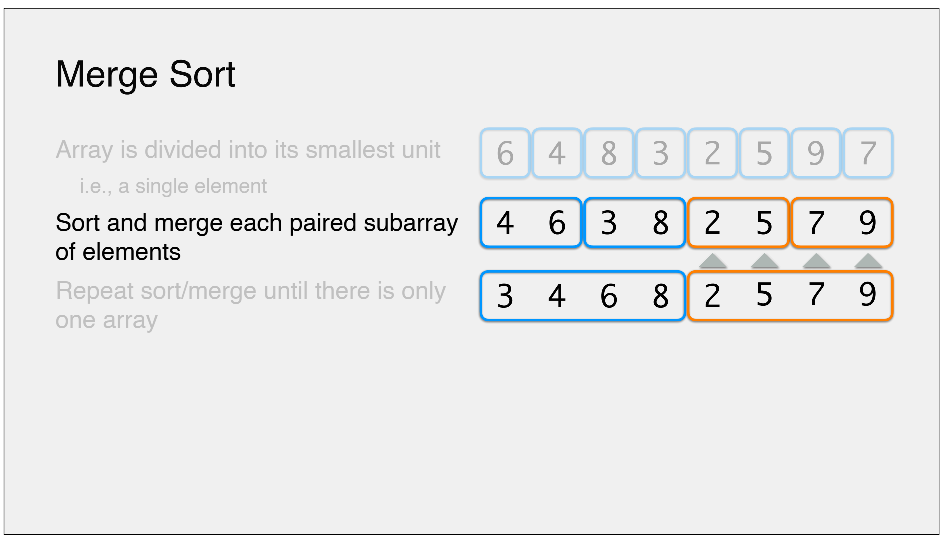
4 6 | 3 8 | 2 5 | 7 9

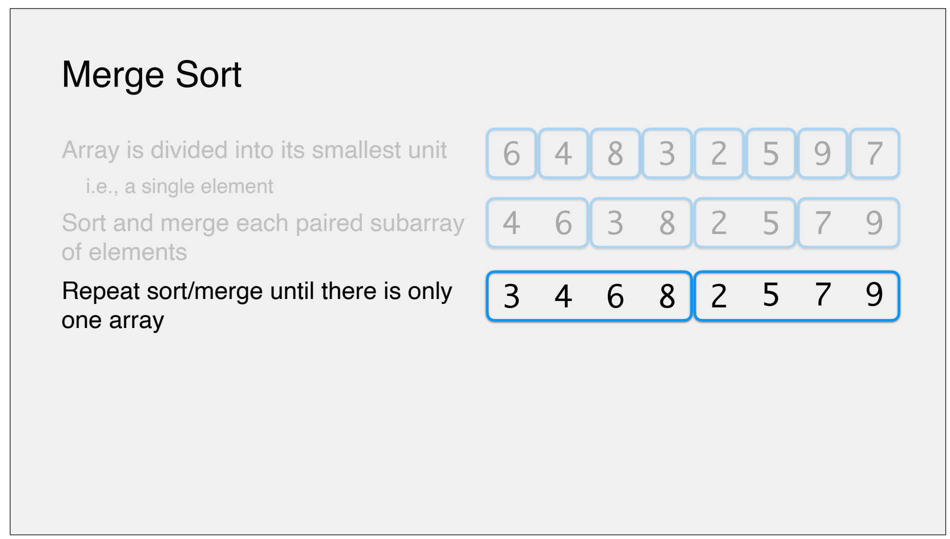
Repeat sort/merge until there is only one array

3 4 6 8


Merge Sort
Array is divided into its smallest unit
i.e., a single element
Sort and merge each paired subarray of elements
Repeat sort/merge until there is only one array
6 4 8 3 2 5 9 7
4 6 3 8 2 5 7 9
3 4 6 8


Merge Sort
Array is divided into its smallest unit
i.e., a single element
Sort and merge each paired subarray of elements
Repeat sort/merge until there is only one array
6 4 8 3 2 5 9 7
4 6 3 8 2 5 7 9
3 4 6 8


Merge Sort
Array is divided into its smallest unit
i.e., a single element
Sort and merge each paired subarray of elements
Repeat sort/merge until there is only one array
6 4 8 3 2 5 9 7
4 6 3 8 2 5 7 9
3 4 6 8 2 5 7 9


Merge Sort
Array is divided into its smallest unit
i.e., a single element
Sort and merge each paired subarray of elements
Repeat sort/merge until there is only one array
6 4 8 3 2 5 9 7
4 6 3 8 2 5 7 9
3 4 6 8 2 5 7 9

Merge Sort

Array is divided into its smallest unit

- i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 | 4 | 8 | 3 | 2 | 5 | 9 | 7

4 6 | 3 8 | 2 5 | 7 9

3 4 6 8 | 2 5 7 9

Merge Sort

Array is divided into its smallest unit

- i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 | 4 | 8 | 3 | 2 | 5 | 9 | 7

4 6 | 3 8 | 2 5 | 7 9

3 4 6 8 | 2 5 7 9

2 3 4 5 6 7 8 9

Merge Sort

Array is divided into its smallest unit

- i.e., a single element

Sort and merge each paired subarray of elements

Repeat sort/merge until there is only one array

6 | 4 | 8 | 3 | 2 | 5 | 9 | 7

4 6 | 3 8 | 2 5 | 7 9

3 4 6 8 | 2 5 7 9

2 3 4 5 6 7 8 9

Merge Sort Analysis

Called a *divide and conquer* algorithm

At each level, we look at *n* elements

Calculating the run time requires also calculating the number of levels

O(n log(n))

- both best and worst case

log(n) levels

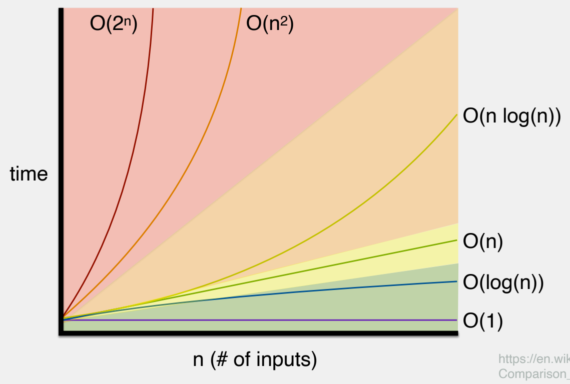
n 6 | 4 | 8 | 3 | 2 | 5 | 9 | 7

n 4 6 | 3 8 | 2 5 | 7 9

n 3 4 6 8 | 2 5 7 9

n 2 3 4 5 6 7 8 9

Big O Notation



O(n log(n)) isn't great…

…but it's the best we can do for a general sorting algorithm

and we've proven that

https://en.wikipedia.org/wiki/Comparison_sort#Number_of_comparisons_required_to_sort_a_list

Downsides to Merge Sort

Always performs O(n log(n))

even if the array is already sorted!

Takes up more memory

insertion and selection sort are *in-place* sorts

i.e., they swap items around in the same array

merge sort requires additional arrays to move from each level

Quicksort

- Considered one of the classic sorting algorithms
- Similar to merge sort in terms of complexity, run time
  - another divide-and-conquer algorithm

- Thumbnail sketch:
  - repeatedly subdivide elements by comparing to a single element called the pivot
  - use recursion to sort the subdivisions

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

7 4 8 3 2 5 9 6

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

7 4 8 3 2 5 9 $\boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

$\boxed{7}$ 4 8 3 2 5 9 $\boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

7 $\boxed{4}$ 8 3 2 5 9 $\boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

$\boxed{4}$ $\boxed{7}$ 8 3 2 5 9 $\boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

$\boxed{4}\quad 7\quad \boxed{8}\quad 3\quad 2\quad 5\quad 9\quad \boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

$\boxed{4}\quad 7\quad 8\quad \boxed{3}\quad 2\quad 5\quad 9\quad \boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

$\boxed{4\quad 3}\quad 8\quad \boxed{7}\quad 2\quad 5\quad 9\quad \boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

$\boxed{4\quad 3}\quad 8\quad 7\quad \boxed{2}\quad 5\quad 9\quad \boxed{6}$

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 7 8 5 9 6

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 7 8 5 9 6

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 8 7 9 6

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 8 7 9 6

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 8 7 9 6

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

4 3 2 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

- we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 9 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 | 7 | 9 | 8

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 | 7 | 8 | 9

Quicksort

Choose pivot

we'll choose the last element

Subdivide in relation to the pivot

Move pivot

Sorts subdivisions, repeat until no more divisions can be made

2 3 4 5 6 7 8 9

What does **best case** and **worst case** mean with quicksort?

Algorithm Analysis

First, consider what is the best and worst case scenarios for sorting an array

not the same scenarios for the two different sorts!

Then, fill out the chart below with the run time (i.e., big O):

	merge sort
best case scenario	
worst case scenario	
	quicksort
best case scenario	
worst case scenario	

Algorithm Analysis

First, consider what is the best and worst case scenarios for sorting an array

not the same scenarios for the two different sorts!

Then, fill out the chart below with the run time (i.e., big O):

	merge sort
array already sorted (best case scenario)	
array sorted backwards (worst case scenario)	
	quicksort
pivots are all range midpoints (best case scenario)	
pivots are min/max in range (worst case scenario)	

Algorithm Analysis

First, consider what is the best and worst case scenarios for sorting an array

not the same scenarios for the two different sorts!

Then, fill out the chart below with the run time (i.e., big O):

	merge sort
array already sorted (best case scenario)	O(n log(n))
array sorted backwards (worst case scenario)	O(n log(n))
	quicksort
pivots are all range midpoints (best case scenario)	O(n log(n))
pivots are min/max in range (worst case scenario)	$O(n^2)$

Expanding Our Sorting Efforts

What about linked lists? Singly vs doubly linked?

What about objects? How do we define equality/inequality?

Expanding Our Sorting Efforts

What about linked lists? Singly vs doubly linked?

- can use any sort that only requires accessing our values in a sequential order
  - i.e., insertion sort, selection sort, merge sort
- quicksort requires random access, and has worse performance

What about objects? How do we define equality/inequality?

Expanding Our Sorting Efforts

What about linked lists? Singly vs doubly linked?

- can use any sort that only requires accessing our values in a sequential order
  - i.e., insertion sort, selection sort, merge sort
- quicksort requires random access, and has worse performance

What about objects? How do we define equality/inequality?

- `equals(Object o)`, `compareTo(Object o)`

Merge Sort vs Quicksort, Array vs Linked List

Consider what we know about the strengths and weaknesses of access and insertion in arrays and linked lists. How do those strengths and weaknesses play out in these two sorting algorithms?

	merge sort	**quicksort**
arrays		
linked lists		