Sorting

2024 Winter APS 105: Computer Fundamentals
Jon Eyolfson

Lecture 32
1.1.0

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

Sorting is One of the Most Common Problems

Computers are very fast, and they don't care about the order of the data

However, programs almost always sort results to make it presentable

We'll discuss several sorting algorithms throughout the lecture

Sorting is One of the Most Common Problems

Computers are very fast, and they don't care about the order of the data

However, programs almost always sort results to make it presentable

We'll discuss several sorting algorithms throughout the lecture

An algorithm is just a set of steps to solve a problem

We May Need to Sort an Array

Given an array such as:

```
int array[] = {2, 5, 3, 1};
```

We want to write a function to sort the array in ascending order:

```
void sort(int array[], int arrayLength);
```

(lowest number first, followed by the next lowest, etc.)

After sorting, the elements of the array should be:

```
{1, 2, 3, 5}
```

Any Ideas How to Sort?

Remember, computers can only compare two elements at once

We can also swap two elements in an array (we wrote `swap` previously)

The First Idea Is Called Bubble Sort

We can compare the first two elements (index 0 and 1) in the array
  If the first element is larger than the second, we swap them

Now, compare the next two elements (index 1 and 2), swapping if needed

Repeat these steps until the end of the array
  The idea is that the larger values go towards the end, in order

The First Idea Is Called Bubble Sort

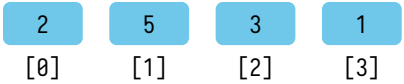
We can compare the first two elements (index 0 and 1) in the array
  If the first element is larger than the second, we swap them

Now, compare the next two elements (index 1 and 2), swapping if needed

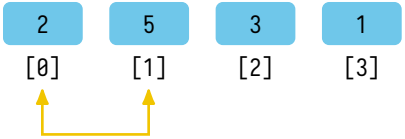
Repeat these steps until the end of the array
  The idea is that the larger values go towards the end, in order

Now the array is closer to being sorted, repeat the entire process again
  until the array is sorted

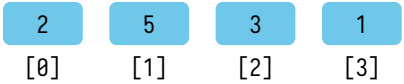
Sorting an Array Using Bubble Sort



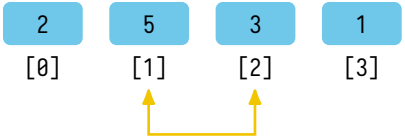
Sorting an Array Using Bubble Sort



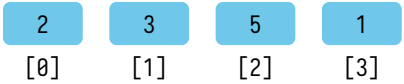
Sorting an Array Using Bubble Sort



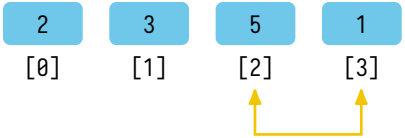
Sorting an Array Using Bubble Sort



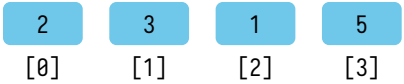
Sorting an Array Using Bubble Sort



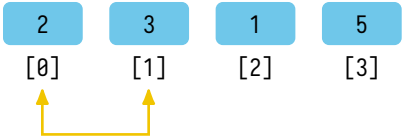
Sorting an Array Using Bubble Sort



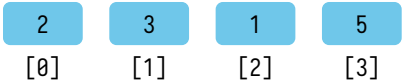
Sorting an Array Using Bubble Sort



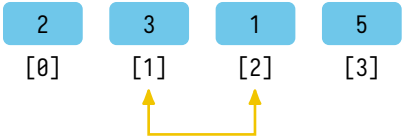
Sorting an Array Using Bubble Sort



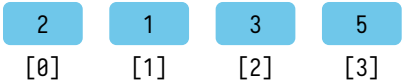
Sorting an Array Using Bubble Sort



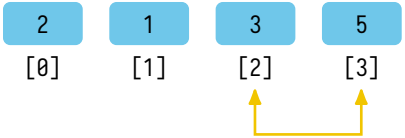
Sorting an Array Using Bubble Sort



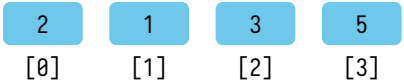
Sorting an Array Using Bubble Sort



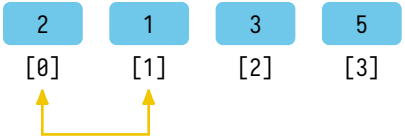
Sorting an Array Using Bubble Sort



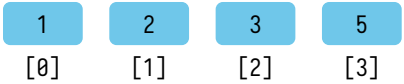
Sorting an Array Using Bubble Sort



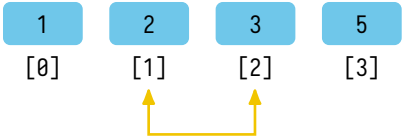
Sorting an Array Using Bubble Sort



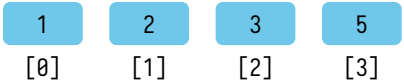
Sorting an Array Using Bubble Sort



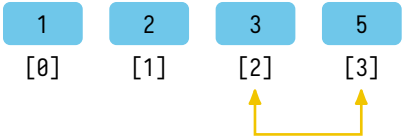
Sorting an Array Using Bubble Sort



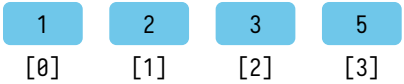
Sorting an Array Using Bubble Sort



Sorting an Array Using Bubble Sort



Sorting an Array Using Bubble Sort



Our Implementation of Bubble Sort

```
void bubbleSort(int array[], int arrayLength) {
    bool swapped;
    do {
        swapped = false;
        for (int i = 0; i < arrayLength - 1; ++i) {
            if (array[i] > array[i + 1]) {
                swap(&array[i], &array[i + 1]);
                swapped = true;
            }
        }

    } while (swapped);
}
```

Why Don’t We Just Find the Smallest Number?

We could find the smallest number, and put it at the beginning of the array

Find the next smallest element in the rest of the array, that’s the next element

Repeat this process until we’ve gone through all the elements

Why Don’t We Just Find the Smallest Number?

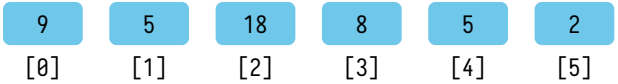
We could find the smallest number, and put it at the beginning of the array

Find the next smallest element in the rest of the array, that’s the next element

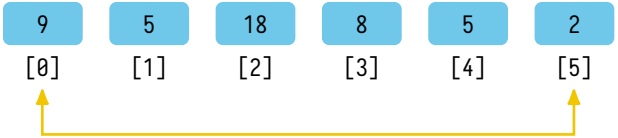
Repeat this process until we’ve gone through all the elements

This algorithm is called selection sort

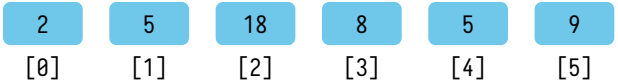
Sorting an Array Using Selection Sort



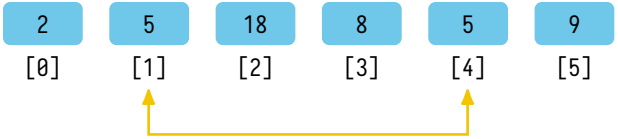
Sorting an Array Using Selection Sort



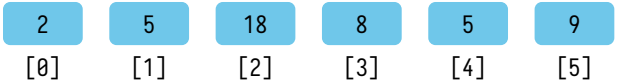
Sorting an Array Using Selection Sort



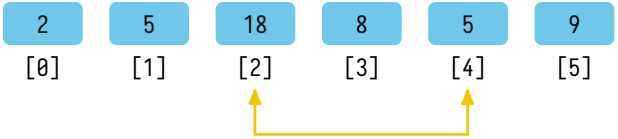
Sorting an Array Using Selection Sort



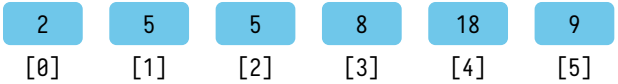
Sorting an Array Using Selection Sort



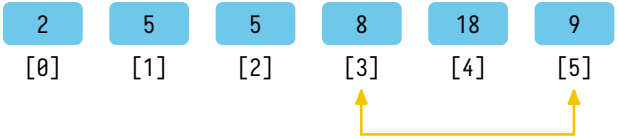
Sorting an Array Using Selection Sort



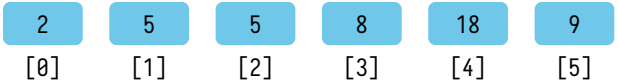
Sorting an Array Using Selection Sort



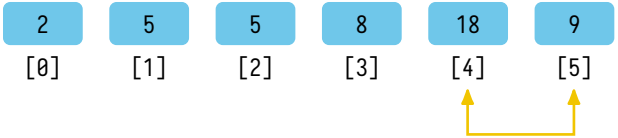
Sorting an Array Using Selection Sort



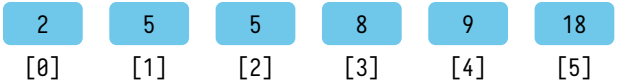
Sorting an Array Using Selection Sort



Sorting an Array Using Selection Sort



Sorting an Array Using Selection Sort



Our Implementation of Selection Sort

```
void selectionSort(int array[], int arrayLength) {
    for (int i = 0; i < arrayLength - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < arrayLength; ++j) {
            if (array[j] < array[minIndex]) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
            swap(&array[i], &array[minIndex]);
        }
    }
}
```

We Divided the Array into Two Parts, Sorted and Unsorted

Can we think of another method of sorting that uses the same idea?
How would you sort playing cards in your hand if you had to break it down?

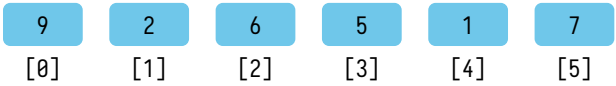
We Divided the Array into Two Parts, Sorted and Unsorted

Can we think of another method of sorting that uses the same idea?
How would you sort playing cards in your hand if you had to break it down?

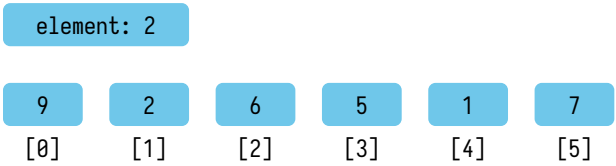
You could keep sorted cards on the left, and the unsorted cards on the right, take a card from the right, and put it in the correct position on the left

This algorithm is called insertion sort

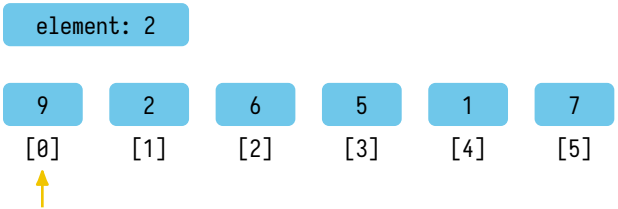
Sorting an Array Using Insertion Sort



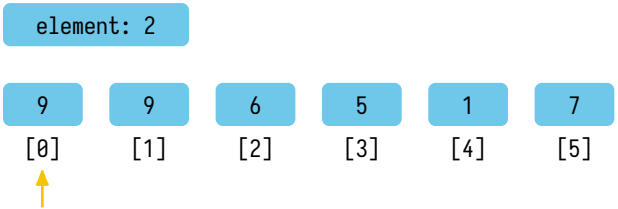
Sorting an Array Using Insertion Sort



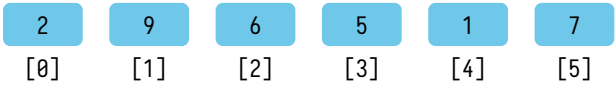
Sorting an Array Using Insertion Sort



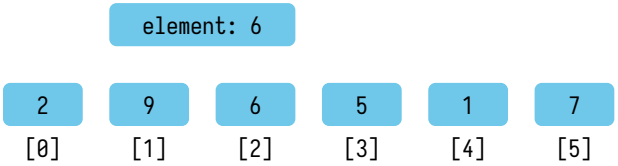
Sorting an Array Using Insertion Sort



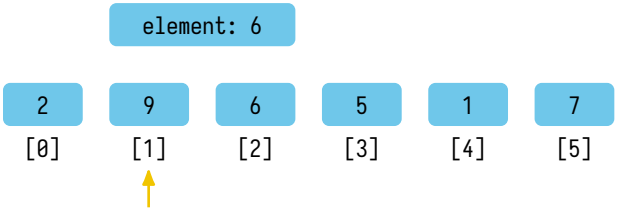
Sorting an Array Using Insertion Sort



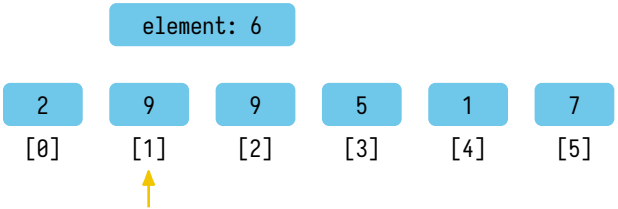
Sorting an Array Using Insertion Sort



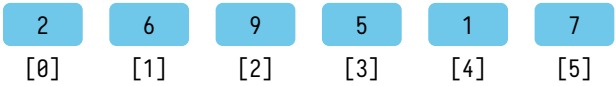
Sorting an Array Using Insertion Sort



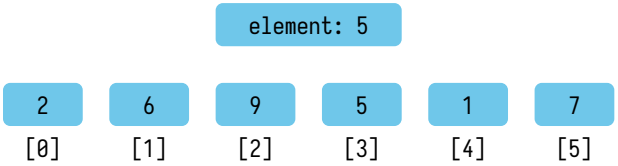
Sorting an Array Using Insertion Sort



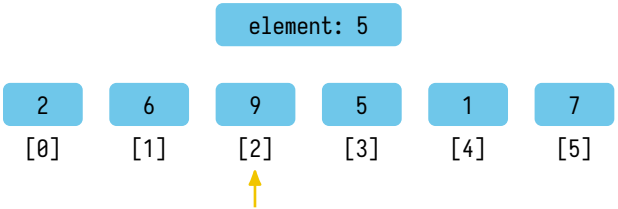
Sorting an Array Using Insertion Sort



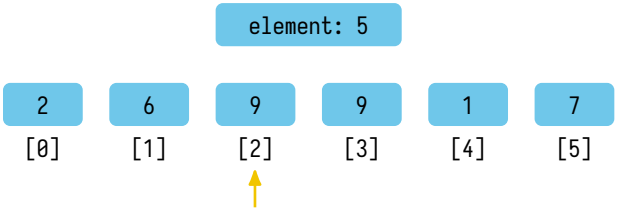
Sorting an Array Using Insertion Sort



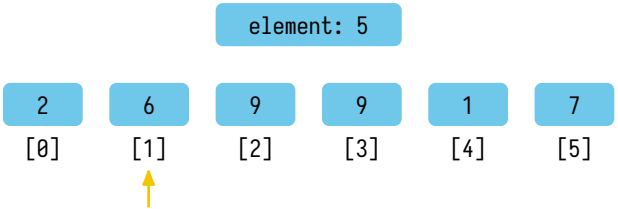
Sorting an Array Using Insertion Sort



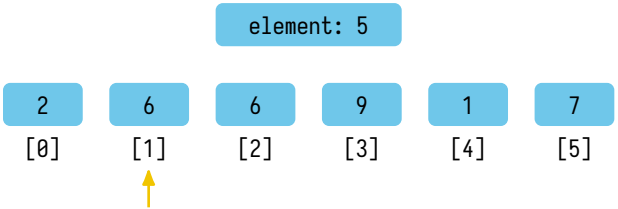
Sorting an Array Using Insertion Sort



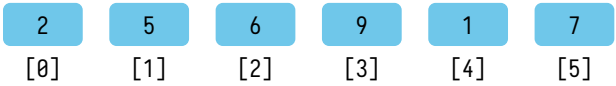
Sorting an Array Using Insertion Sort



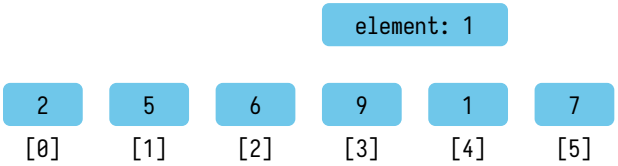
Sorting an Array Using Insertion Sort



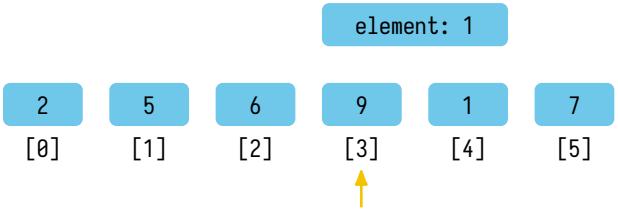
Sorting an Array Using Insertion Sort



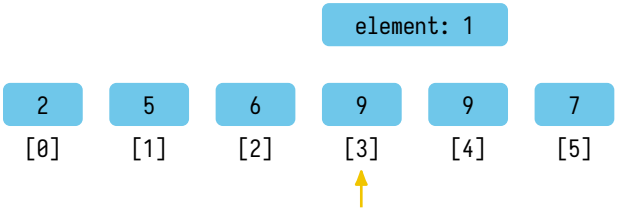
Sorting an Array Using Insertion Sort



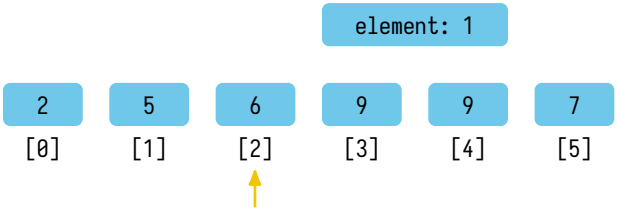
Sorting an Array Using Insertion Sort



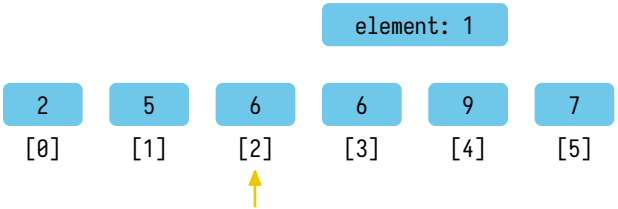
Sorting an Array Using Insertion Sort



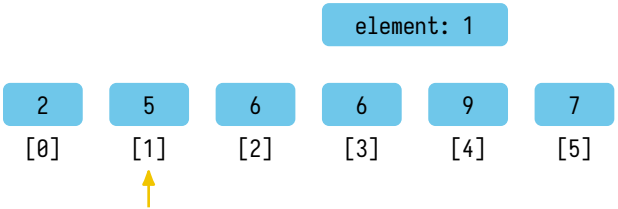
Sorting an Array Using Insertion Sort



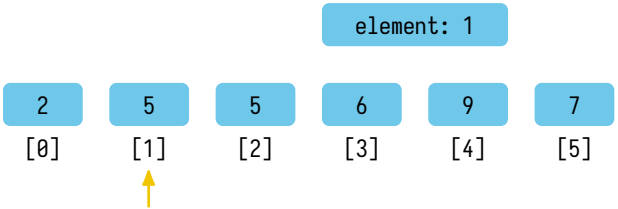
Sorting an Array Using Insertion Sort



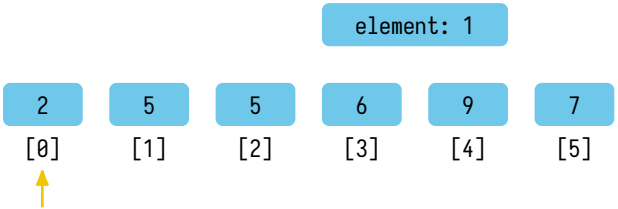
Sorting an Array Using Insertion Sort



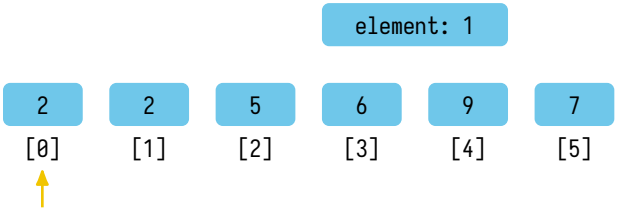
Sorting an Array Using Insertion Sort



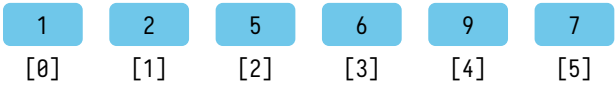
Sorting an Array Using Insertion Sort



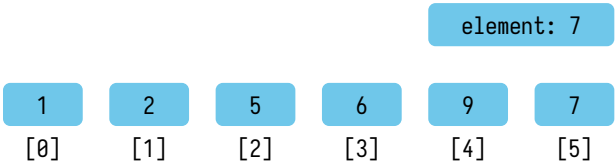
Sorting an Array Using Insertion Sort



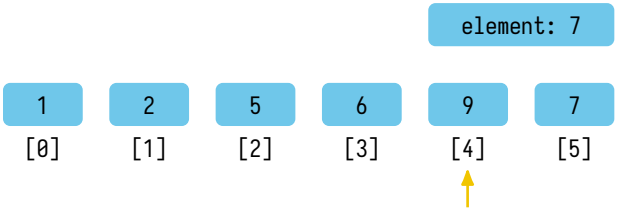
Sorting an Array Using Insertion Sort



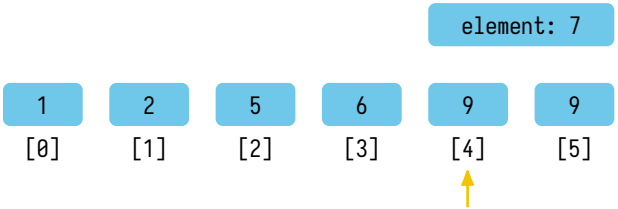
Sorting an Array Using Insertion Sort



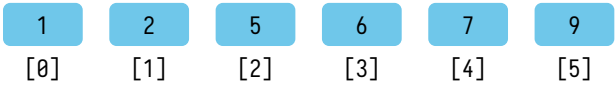
Sorting an Array Using Insertion Sort



Sorting an Array Using Insertion Sort



Sorting an Array Using Insertion Sort



Our Implementation of Insertion Sort

```
void insertionSort(int array[], int arrayLength) {
    for (int i = 1; i < arrayLength; ++i) {
        int element = array[i];
        int j = i;
        while (j > 0 && array[j - 1] > element) {
            array[j] = array[j - 1];
            --j;
        }
        if (j != i) {
          array[j] = element;
        }
    }
}
```

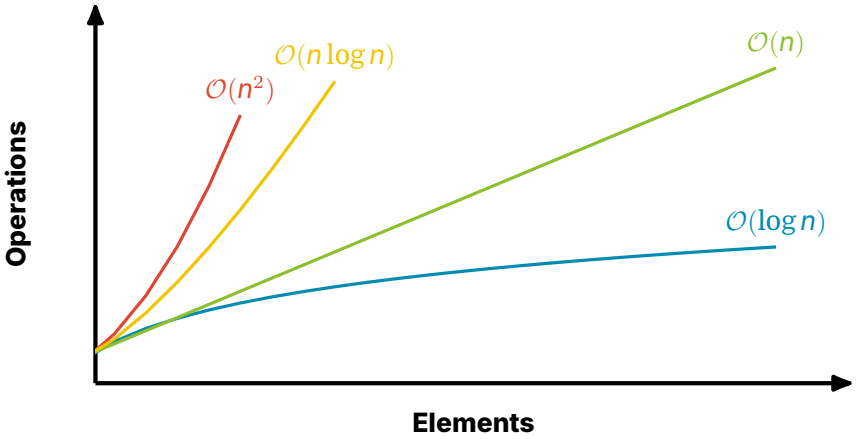
Computer Scientists Evaluate Runtime Using Big-$\mathcal{O}$ Notation

For performance, we care about how many operations an algorithm has to do

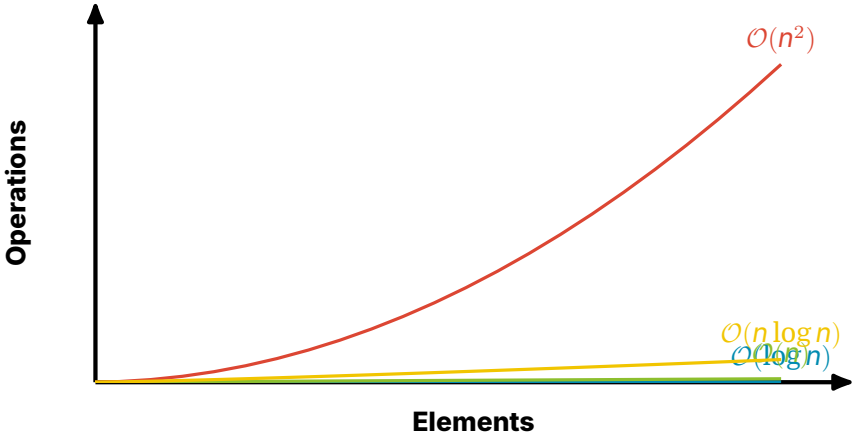
Big-$\mathcal{O}$ is the relationship is between the number of elements and operations, it tells you how number of operations scales with the number of elements (for large problems, we don't care about the constant values)

The sorting algorithms we implemented today are all $\mathcal{O}(n^2)$

For a Small Problem, The Differences Aren't That Large



However, a Larger Problem Presents Issues



Next Lecture We'll Explore Better Sorting Algorithms

However, today we implemented 3 sorting algorithms that get the job done
Bubble sort, Selection sort, and Insertion sort