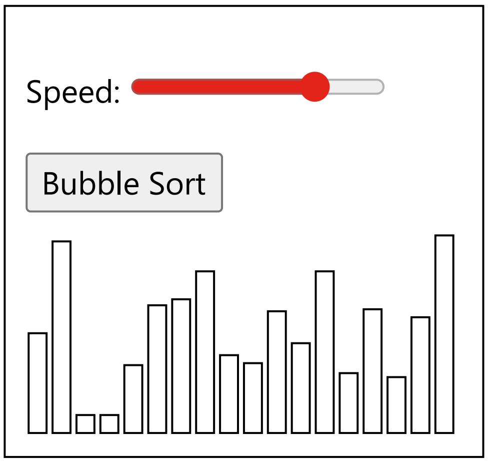
DSA Bubble Sort

Bubble Sort

Bubble Sort is an algorithm that sorts an array from the lowest value to the highest value.



Run the simulation to see how it looks like when the Bubble Sort algorithm sorts an array of values. Each value in the array is represented by a column.

The word 'Bubble' comes from how this algorithm works, it makes the highest values 'bubble up'.

1. Go through the array, one value at a time.
2. For each value, compare the value with the next value.
3. If the value is higher than the next one, swap the values so that the highest value comes last.
4. Go through the array as many times as there are values in the array.

Continue reading to fully understand the Bubble Sort algorithm and how to implement it yourself.

Manual Run Through

Before we implement the Bubble Sort algorithm in a programming language, let's manually run through a short array only one time, just to get the idea.

Step 1: We start with an unsorted array.

```
[7, 12, 9, 11, 3]
```

Step 2: We look at the two first values. Does the lowest value come first? Yes, so we don't need to swap them.

```
[7, 12, 9, 11, 3]
```

Step 3: Take one step forward and look at values 12 and 9. Does the lowest value come first? No.

```
[7, 12, 9, 11, 3]
```

Step 4: So we need to swap them so that 9 comes first.

```
[7, 9, 12, 11, 3]
```

Step 6: We must swap so that 11 comes before 12.

```
[7, 9, 11, 12, 3]
```

Step 7: Looking at 12 and 3, do we need to swap them? Yes.

```
[7, 9, 11, 12, 3]
```

Step 8: Swapping 12 and 3 so that 3 comes first.

```
[7, 9, 11, 3, 12]
```

Run the simulation below to see the 8 steps above animated:

Run Through Once

[7, 12, 9, 11, 3]

Manual Run Through: What Happened?

We must understand what happened in this first run through to fully understand the algorithm, so that we can implement the algorithm in a programming language.

Can you see what happened to the highest value 12? It has bubbled up to the end of the array, where it belongs. But the rest of the array remains unsorted.

So the Bubble Sort algorithm must run through the array again, and again, and again, each time the next highest value bubbles up to its correct position. The sorting continues until the lowest value 3 is left at the start of the array. This means that we need to run through the array 4 times, to sort the array of 5 values.

Bubble Sort

[7, 12, 9, 11, 3]

We will now use what we have learned to implement the Bubble Sort algorithm in a programming language.

Bubble Sort Implementation

To implement the Bubble Sort algorithm in a programming language, we need:

1. An array with values to sort.
2. An inner loop that goes through the array and swaps values if the first value is higher than the next value. This loop must loop through one less value each time it runs.
3. An outer loop that controls how many times the inner loop must run. For an array with n values, this outer loop must run n-1 times.

The resulting code looks like this:

Example

```
my_array = [64, 34, 25, 12, 22, 11, 90, 5]

n = len(my_array)
for i in range(n-1):
    for j in range(n-i-1):
        if my_array[j] > my_array[j+1]:
            my_array[j], my_array[j+1] = my_array[j+1], my_array[j]

print("Sorted array:", my_array)
```

Try it Yourself »

The Bubble Sort algorithm can be improved a little bit more.

Imagine that the array is almost sorted already, with the lowest numbers at the start, like this for example:

```
my_array = [7, 3, 9, 12, 11]
```

In this case, the array will be sorted after the first run, but the Bubble Sort algorithm will continue to run, without swapping elements, and that is not necessary.

If the algorithm goes through the array one time without swapping any values, the array must be finished sorted, and we can stop the algorithm, like this:

Example

```
my_array = [7, 3, 9, 12, 11]

n = len(my_array)
for i in range(n-1):

    for j in range(n-i-1):
        if my_array[j] > my_array[j+1]:
            my_array[j], my_array[j+1] = my_array[j+1], my_array[j]




print("Sorted array:", my_array)
```

Try it Yourself »

For a more thorough and detailed explanation of Bubble Sort time complexity, visit this page.

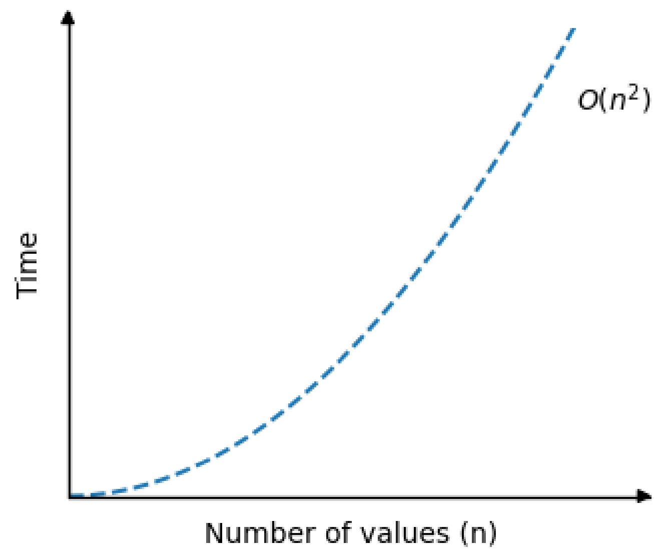
The Bubble Sort algorithm loops through every value in the array, comparing it to the value next to it. So for an array of n values, there must be n such comparisons in one loop.

And after one loop, the array is looped through again and again n times.

This means there are $n \cdot n$ comparisons done in total, so the time complexity for Bubble Sort is:

$$\underline{\underline{O(n^2)}}$$

The graph describing the Bubble Sort time complexity looks like this:



As you can see, the run time increases really fast when the size of the array is increased.

Luckily there are sorting algorithms that are faster than this, like Quicksort, that we will look at later.

You can simulate Bubble Sort below, where the red and dashed line is the theoretical time complexity $O(n^2)$. You can choose a number of values n, and run an actual Bubble Sort implementation where the operations are counted and the count is marked as a blue cross in the plot below. How does theory compare with practice?

Set values: 300

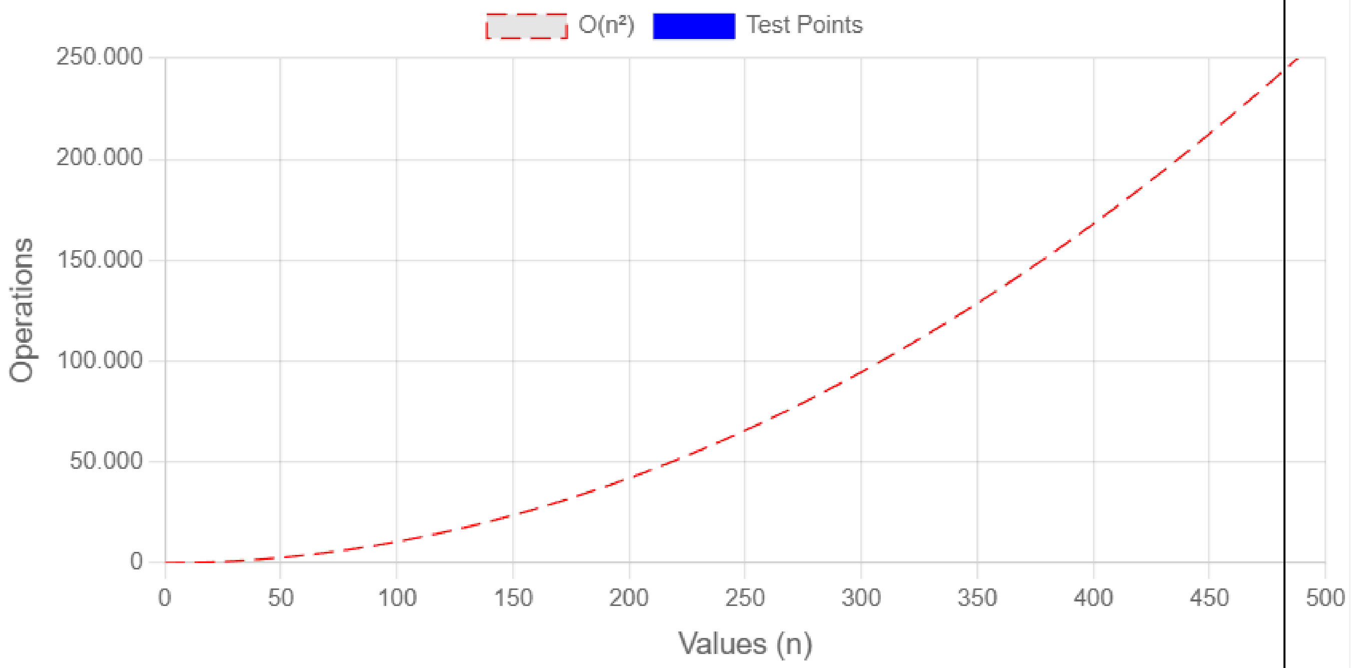
○ Worst Case

○ Best Case

Operations: 0

○ 10 Random

Run Clear



DSA Exercises

Test Yourself With Exercises

Exercise:

Using Bubble Sort on this array:

```
[7,14,11,8,9]
```

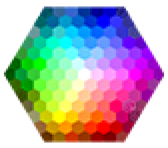
To sort the values from left to right in an increasing (ascending) order.

[, , , ,]

Submit Answer »

Start the Exercise

❮ Previous Sign in to track progress Next ❯



Tutorials References Exercises Sign In

SS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C

FORUM ABOUT ACADEMY