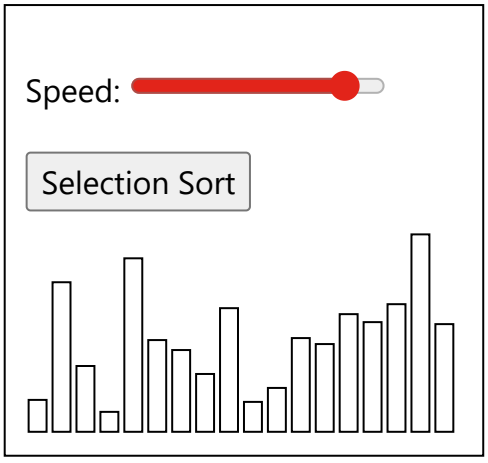
DSA Selection Sort

‹ Previous Next ›

Selection Sort

The Selection Sort algorithm finds the lowest value in an array and moves it to the front of the array.



The algorithm looks through the array again and again, moving the next lowest values to the front, until the array is sorted.

1. Go through the array to find the lowest value.
2. Move the lowest value to the front of the unsorted part of the array.
3. Go through the array again as many times as there are values in the array.

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

Manual Run Through

Before we implement the Selection 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: Go through the array, one value at a time. Which value is the lowest? 3, right?

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

Step 3: Move the lowest value 3 to the front of the array.

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

Step 4: Look through the rest of the values, starting with 7. 7 is the lowest value, and already at the front of the array, so we don't need to move it.

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

Step 5: Look through the rest of the array: 12, 9 and 11. 9 is the lowest value.

Step 6: Move 9 to the front.

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

Step 7: Looking at 12 and 11, 11 is the lowest.

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

Step 8: Move it to the front.

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

Finally, the array is sorted.

Run the simulation below to see the steps above animated:

Selection Sort

[7, 12, 9, 11, 3]

Manual Run Through: What Happened?

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

Can you see what happened to the lowest value 3? In step 3, it has been moved to the start of the array, where it belongs, but at that step the rest of the array remains unsorted.

So the Selection Sort algorithm must run through the array again and again, each time the next lowest value is moved in front of the unsorted part of the array, to its correct position. The sorting continues until the highest value 12 is left at the end of the array. This means that we need to run through the array 4 times, to sort the array of 5 values.

programming language.

Selection Sort Implementation

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

1. An array with values to sort.
2. An inner loop that goes through the array, finds the lowest value, and moves it to the front of the array. 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:

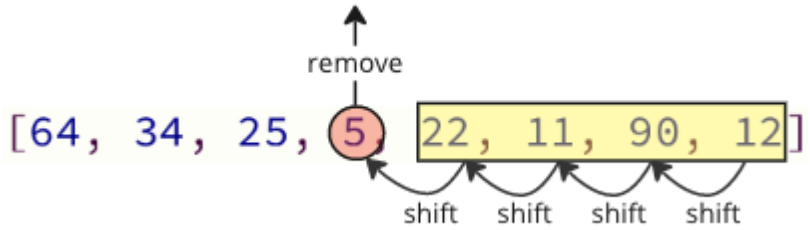
Example

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

n = len(my_array)
for i in range(n-1):
    min_index = i
    for j in range(i+1, n):
        if my_array[j] < my_array[min_index]:
            min_index = j
    min_value = my_array.pop(min_index)
    my_array.insert(i, min_value)

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

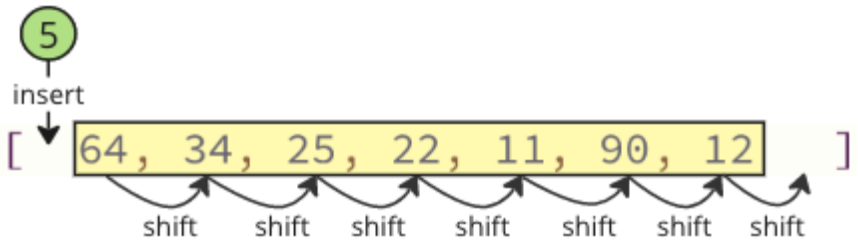
Try it Yourself »

Selection Sort Shifting Problem

Each time the next lowest value array element is removed, all following elements must be shifted one place down to make up for the removal.



These shifting operation takes a lot of time, and we are not even done yet! After the lowest value (5) is found and removed, it is inserted at the start of the array, causing all following values to shift one position up to make space for the new value, like the image below shows.

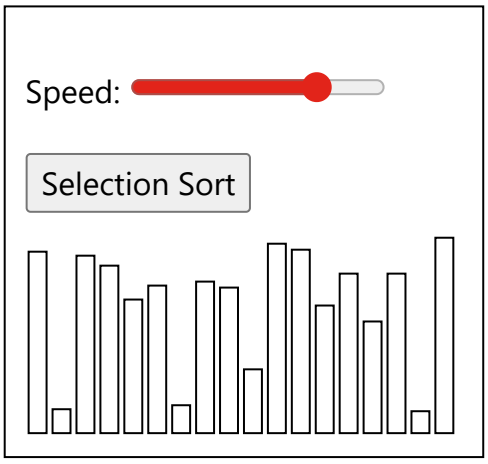


Note: You will not see these shifting operations happening in the code if you are using a high level programming language such as Python or Java, but the shifting operations are still happening in the background. Such shifting operations require extra time for the computer to do, which can be a problem.

Solution: Swap Values!

Instead of all the shifting, swap the lowest value (5) with the first value (64) like below.

We can swap values like the image above shows because the lowest value ends up in the correct position, and it does not matter where we put the other value we are swapping with,



Here is an implementation of the improved Selection Sort, using swapping:

Example

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

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


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

Try it Yourself »

Selection Sort Time Complexity

For a general explanation of what time complexity is, visit this page.

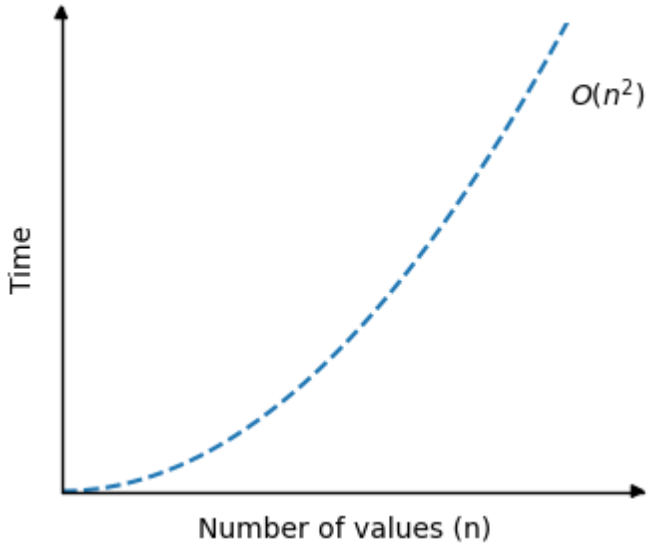
On average, about $\frac{n}{2}$ elements are compared to find the lowest value in each loop.

And Selection Sort must run the loop to find the lowest value approximately n times.

We get time complexity:

$$O(\frac{n}{2} \cdot n) = \underline{\underline{O(n^2)}}$$

The time complexity for the Selection Sort algorithm can be displayed in a graph like this:



As you can see, the run time is the same as for Bubble Sort: The run time increases really fast when the size of the array is increased.

Run the simulation below for different sized arrays.

The red dashed line represents the theoretical time complexity $O(n^2)$.

Blue crosses appear when you run the simulation. The blue crosses show how many operations are needed to sort an array of a certain size.

Random

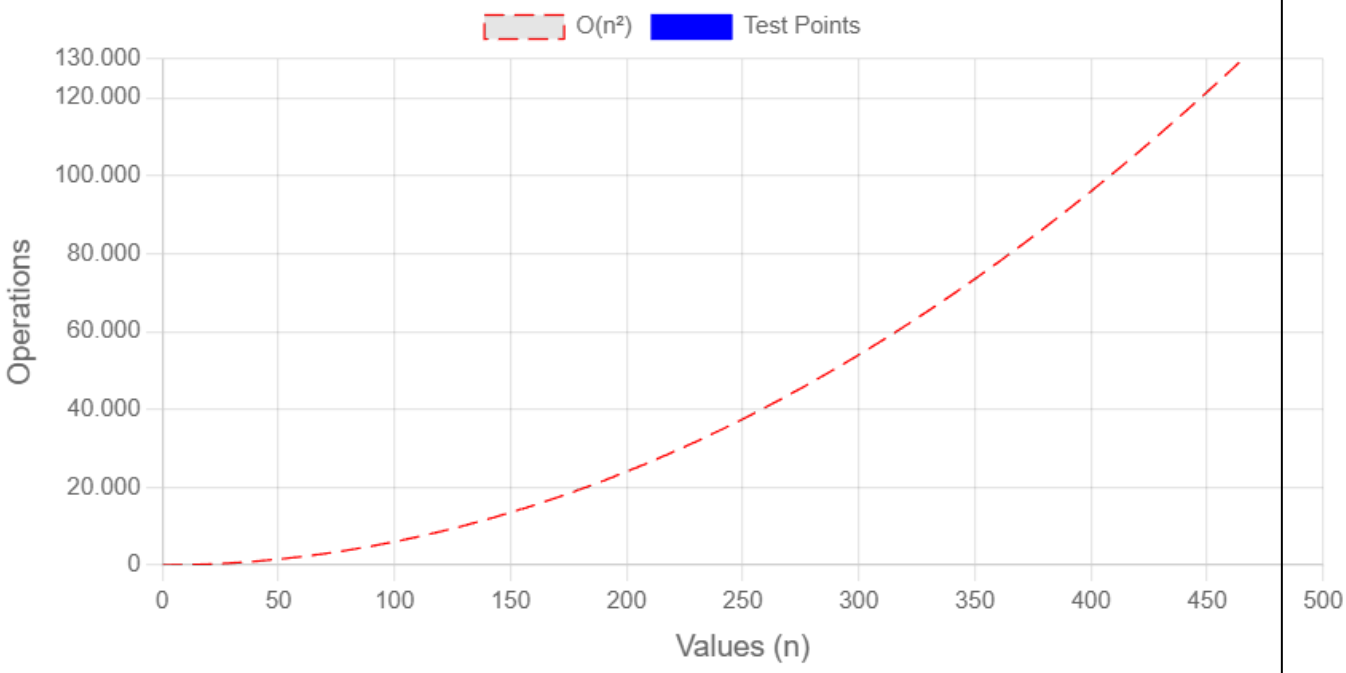
Worst Case

Best Case

Operations: 0

10 Random

Run Clear



The most significant difference from Bubble sort that we can notice in this simulation is that best and worst case is actually almost the same for Selection Sort ($O(n^2)$), but for Bubble Sort the best case runtime is only $O(n)$.

The difference in best and worst case for Selection Sort is mainly the number of swaps. In the best case scenario Selection Sort does not have to swap any of the values because the array is already sorted. And in the worst case scenario, where the array already sorted, but in the wrong order, so Selection Sort must do as many swaps as there are values in array.

DSA Exercises

Exercise:

Using Selection Sort on this array:

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

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

What is the value of the LAST element after the first run through?

Submit Answer »

Start the Exercise

COLOR PICKER

PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
AngularJS Reference
jQuery Reference

Top Examples

HTML Examples
CSS Examples
JavaScript Examples
How To Examples
SQL Examples
Python Examples
W3.CSS Examples
Bootstrap Examples
PHP Examples
Java Examples
XML Examples
jQuery Examples

Get Certified

HTML Certificate
CSS Certificate
JavaScript Certificate
Front End Certificate
SQL Certificate
Python Certificate
PHP Certificate
jQuery Certificate
Java Certificate
C++ Certificate
C# Certificate
XML Certificate