Implement 5 Sorting Algorithms using JavaScript.

An introduction to some popular sorting techniques.

[Rajat M](https://medium.com/@rajat_m?source=post_page-----63c5a917e811--------------------------------)

[Rajat M](https://medium.com/@rajat_m?source=post_page-----63c5a917e811--------------------------------)

[Sep 20, 2020·7 min read](https://medium.com/@rajat_m/implement-5-sorting-algorithms-using-javascript-63c5a917e811?source=post_page-----63c5a917e811--------------------------------)

Most languages have a built-in method which serves the purpose of trying to sort a bunch of data. The common tendency amongst most developers, especially those who are just beginning their journey, might be to choose this path and avoid writing their own implementation. But, this can end up having unforeseen repercussions in terms of performance. Therefore, it is better to go with a sorting technique that is best suited for your current requirement.

The first 3 sorting algorithms that I cover in this article, have an average time complexity of O(n²). These are ones which are fairly popular, and are much more intuitive in their approach to sort data.

The other 2 have an average time complexity of O(n\*log n), and can be bit tricky to comprehend if you have no prior knowledge of recursion. So, I would suggest that you go through the following article to understand how recursion works.

[Recursion Demystified](https://www.freecodecamp.org/news/recursion-demystified-99a2105cb871/" \t "_blank)

[by Sachin Malhotra Recursion Demystified Source: https://thomaspark.co/wp/wp-content/uploads/2017/01/xkcd.png> In order…](https://www.freecodecamp.org/news/recursion-demystified-99a2105cb871/" \t "_blank)

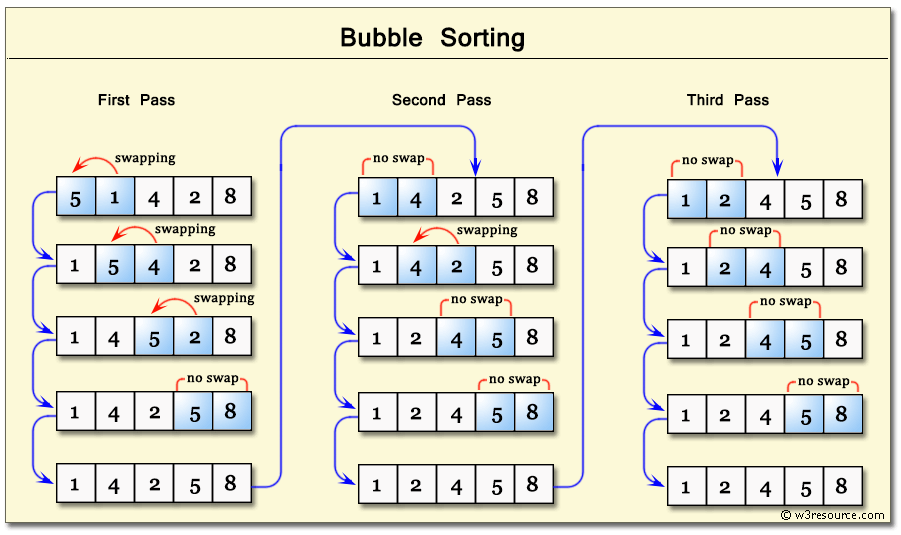
[www.freecodecamp.org](https://www.freecodecamp.org/news/recursion-demystified-99a2105cb871/" \t "_blank)

In the following sections, I shall give you a brief explanation as to how that particular algorithm goes about sorting data. Then, I give you some pseudocode in case you wish to go ahead and try to implement that algorithm on your own. Finally, I provide a gist for my implementation of the algorithm. I would suggest that you understand the pseudocode before diving into the gist, as that will help you grasp the use case for each algorithm better.

Let’s get started with Bubble Sort, shall we. The space complexity for the algorithm is O(1) and the average time complexity is O(n²).The pseudocode is as follows:

* Start iterating through the array, comparing 2 elements at a time.
* Swap them as required.
* At the end of first pass, the largest number has bubbled to the last index of the array, so ignore the last index in the next pass.
* Continue these passes until the array is sorted.

An illustration to understand how bubble sort works



An illustration to understand how bubble sort works

The code for the implementation in JS is as follows:

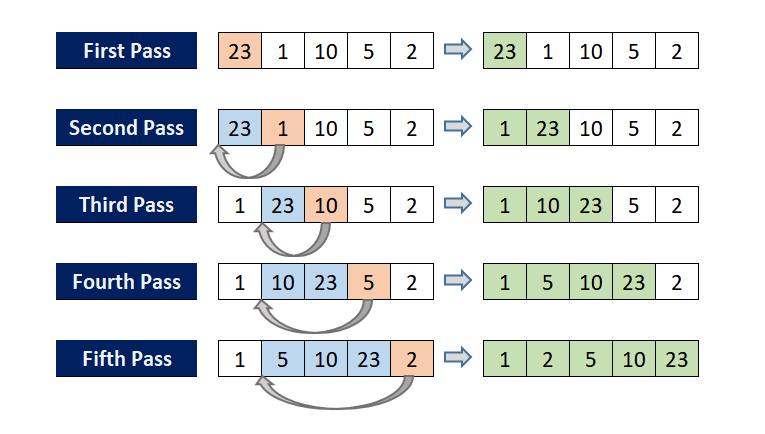
Note that the second implementation is slightly optimized to handle an array that is almost sorted.

The next sorting algorithm which has a time complexity of O(n²) is Insertion Sort, it also has a space complexity of O(1). This is most useful when there is a scenario wherein you are receiving a series of numbers in real time, and need them in a sorted array.

The main concept to understand when using this technique is that, there is a portion of the array that is always sorted and a section that remains unsorted.

* Start by comparing the 2nd element with the 1st element, swap if necessary.
* Iterate through the rest of the array. Then, for each element, iterate through the *sorted portion* of the array, and *insert* this element where it needs to be, by making comparisons.
* Keep doing this until all the elements have been inserted into their correct positions.

An illustration to understand how insertion sort works.



An illustration to understand how insertion sort works

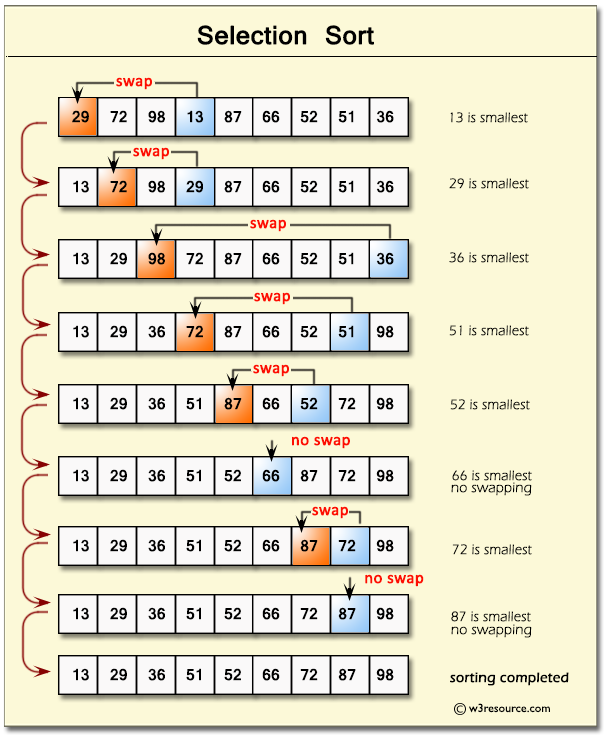
The code for the same is as shown below.

Selection Sort is the last sorting algorithm to have a time complexity of O(n²), included in this article. The space complexity is the same as the previous two techniques i.e, O(1). The pseudocode for this algorithm is as follows.

* Assume that the first element is the smallest. (Or largest, if sorting in descending order).
* Find the minimum value from the array and swap this with the first element of the array. This completes one pass, wherein the smallest element of the array is now at the 0th index.
* Repeat this procedure for the rest of the array elements, but for the next pass do not compare the element we just placed at the 0th index.

This is usually not that useful in most situations, but still helps a beginner grasp the concepts of implementing an algorithm to solve a problem.

An illustration to understand how selection sort works.



An illustration to understand how selection sort works.

My implementation is as follows. Note that the sorted array is built up from the 0th index.

You may have noticed that it is quite difficult to get a performant sorting algorithm using these techniques. Hence, in order to have an algorithm that is better than O(n²) in terms of time complexity, we have to use recursion.

The next 2 techniques can seem less intuitive at first go. So do read the pseudocode before you jump to the code, in order to make sense of the procedure followed!

Both of them have an average time complexity of O(n \* log n). Their space complexities varies depending on the technique.

Let’s take a look at how merge sort is able to use recursion to implement an algorithm with a better time complexity.

The main concept here is that an array with size 0 or 1 is inherently sorted. This means that if we are able to split our array into smaller subarrays of size 0 or 1, and merge them correctly, we have sorted our array!

So there are two things that we need to do before we can implement merge sort. We need to find a way to divide an array into halves continuously, until we end up with arrays of size 0 or 1. Then, we merge them in a way that results in a larger (but still sorted) array.

The pseudocode to continuously divide an array, and end up with a bunch of arrays of size 0 or 1, is as follows.

* We use recursion to do this. Use [slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice) to halve the array, and do this until the base case of arr.length ≤ 1 is reached.

Now, let’s tackle the issue of merging two arrays (of size≤1) such that we end up with a sorted array.

* Start by making an empty array.
* Compare the first elements of the 2 subarrays, and push the smaller of the two, to the the new array.
* Suppose 1st element of 1st array is smaller, then push that to the new array. Now compare the 2nd element of the first array to the 1st element of the 2nd array, and so on.
* If we have exhausted the array elements in any of the 2 subarrays, then just push the other subarray to the new array we had created.

See the image below to see how this technique needs to work.

Illustration for merge sort.

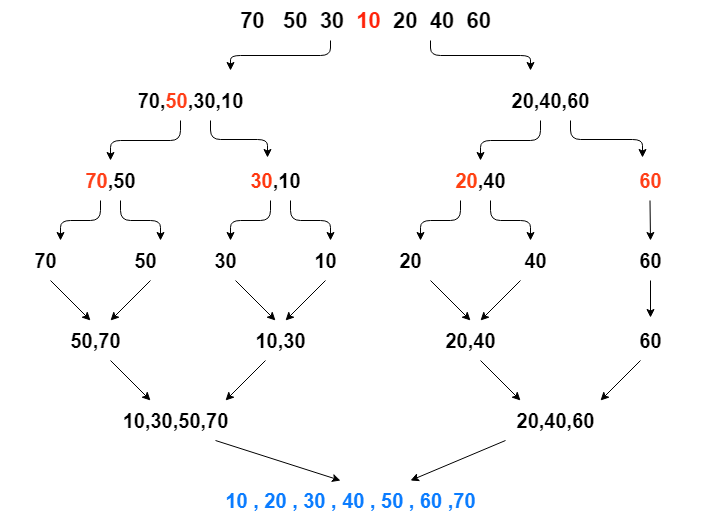


Illustration for merge sort. **Note:** The middle element is colored red.

The code for the merge sort algorithm is as follows. Note the use of helper function to implement the merging of 2 subarrays, and it is pretty evident that the space complexity for this algorithm is O(n).

Lastly, let us see how quick sort justifies its name and goes about sorting an array.

It works by choosing a pivot element, and making sure that all the elements to the left of the pivot element is less than the pivot(not necessarily sorted, they just need to be less than the pivot) and that all the elements to the right of the pivot are all greater than it.

The only 2 tasks we need to do in order to implement quick sort’s algorithm is to correctly identify the index for the pivot and place the pivot element at that index. Initially, we assume the pivot to any element in the array, in this example I shall consider the 0th element to be the initial pivot.

The pseudocode to correctly return the index for the pivot element is as follows. Note that this is also called the partition function.

* Choose a pivot, store its index in a variable, let’s say pivotIndex. Loop through the array, if the current element is less than than the pivot, then increment the pivotIndex, and swap the current element with the element present at the new pivotIndex
* After one iteration through the array, swap the pivot with the element present at the pivotIndex

Once you have a helper function to do the above task, we need to recursively place all the pivot elements in their correct positions. The pseudocode to so that is as follows.

* Assume left indicates the start of a subarray, and right indicates the last index of the subarray.
* Do the following only if the left pointer is at a lesser index than the right pointer:  
  — Start by calling the partition() on the entire array by defaulting the left and right pointers to the first and last element of the array respectively.  
  — Then store the return value in the pivotIndex  
  — Use this to recursively call quickSort() with the same array, but from leftup until (pivotIndex-1), for the left part of the array.  
  — For the right part of the array, call quickSort() again, with the same array, but from (pivotIndex + 1) upto right
* Once the base case becomes invalid, it means that left equals right , so we return the array.

The video shows a visualization of quick sort algorithm. The pivot elements are colored yellow.

Video to visualize quick sort algorithm

The code for implementing quick sort in JavaScript is as follows. Note that the space complexity is O(log n).

Now that you know how to implement these 5 sorting algorithms, the next step is to understand which technique works best for the situation you find yourself in. To see some regular use cases, you can checkout the following article.

[When to use each Sorting Algorithm - GeeksforGeeks](https://www.geeksforgeeks.org/when-to-use-each-sorting-algorithms/" \t "_blank)

[A sorting algorithm is an algorithm that makes arrange in a certain order. The fundamental task is to put the items in…](https://www.geeksforgeeks.org/when-to-use-each-sorting-algorithms/" \t "_blank)

[www.geeksforgeeks.org](https://www.geeksforgeeks.org/when-to-use-each-sorting-algorithms/" \t "_blank)

[Rajat M](https://medium.com/@rajat_m?source=post_sidebar--------------------------post_sidebar-----------)

Web developer who loves to solve problems using code.

Follow

8

1

8

1

* [JavaScript](https://medium.com/tag/javascript)
* [Sorting Algorithms](https://medium.com/tag/sorting-algorithms)
* [Tutorial](https://medium.com/tag/tutorial)
* [Programming](https://medium.com/tag/programming)
* [Beginner](https://medium.com/tag/beginner)

[More from Rajat M](https://medium.com/@rajat_m?source=follow_footer-------------------------------------)

Follow

Web developer who loves to solve problems using code.

[Sep 13, 2020](https://medium.com/@rajat_m/understand-life-cycle-methods-in-react-js-71a5464867cf?source=follow_footer---------0----------------------------)

[Understand Life Cycle Methods in React.js](https://medium.com/@rajat_m/understand-life-cycle-methods-in-react-js-71a5464867cf?source=follow_footer---------0----------------------------)

Learn how to effectively use React life cycle methods.

To build a project using React, the first step is to figure out the various Components that are required to bring your project to life! Once you are able to visualize the client-side of your project as a collection of Components, half the battle is won. So it is fair to say that Components form the crux of any React application.

But how does a collection of Components end up becoming a [single page application](https://developer.mozilla.org/en-US/docs/Glossary/SPA)? This is no different to the way *every single* website is rendered by your browser i.e, by creating a DOM. But in case of React…

[Read more · 5 min read](https://medium.com/@rajat_m/understand-life-cycle-methods-in-react-js-71a5464867cf?readmore=1&source=follow_footer---------0----------------------------)

67

[Sep 6, 2020](https://medium.com/@rajat_m/what-are-restful-routes-and-how-to-use-them-929129ae7bf6?source=follow_footer---------1----------------------------)

[What Are RESTful Routes and How to Use Them?](https://medium.com/@rajat_m/what-are-restful-routes-and-how-to-use-them-929129ae7bf6?source=follow_footer---------1----------------------------)

An introduction to RESTful routing.

Chances are that you have already heard about REST APIs even before you stumbled upon this article. But maybe you never fully understood what that meant, or as to why it is used so frequently. The crux of the matter is that, it is just a set of constraints in API design.

*But why would this be necessary? Can’t an API endpoint have any pattern a developer wishes to keep? After all, an API is just an interface for data transfer, it would work regardless of the pattern of the server requests, as long as it correctly matches the API…*

[Read more · 5 min read](https://medium.com/@rajat_m/what-are-restful-routes-and-how-to-use-them-929129ae7bf6?readmore=1&source=follow_footer---------1----------------------------)

2

[Aug 30, 2020](https://medium.com/@rajat_m/what-are-service-workers-and-how-to-use-them-e993c1f497e6?source=follow_footer---------2----------------------------)

[What Are Service Workers and How to Use Them?](https://medium.com/@rajat_m/what-are-service-workers-and-how-to-use-them-e993c1f497e6?source=follow_footer---------2----------------------------)

An introduction to service workers.

What is a web application? An app that can be accessed *only*through a browser? An app that cannot replicate the rich offline experience that a native application can provide? Or, an app that requires a constant internet connection to work properly?

In case you agree with any of these answers, this article will be a huge revelation for you! But as most of us already know, a modern web application can do so much more than just running a few scripts on the browser to display HTML.