Sort an Array of Objects in JavaScript

Challenge mode! Find out when you stand! Try sets, solve problems & win rewards! Go to Challenge

Learn via video course

JavaScript for Beginners
by Arnab Bhattacharya
0/0 · 0 Modules Left
Free Course

Certificate · Assessments
Start Learning

Overview

Sorting refers to the process of arranging the data into a particular order, either increasing or decreasing. We might encounter several different kinds of data in JavaScript, for example, array, list, map, etc.

Sorting an array of objects in javascript can be done using the array.sort() method in javascript and mandatorily passing the compare function, which will determine on what basis our array of objects will be sorted.

Scope

- In this article, we will discuss how to sort an array of objects in JavaScript. We will learn about what it is and its syntax, creation, the comparison function, etc.
- We will learn about sorting an array of objects by numbers and sorting an array of objects by strings and dates.
- We will also learn about the sorting stability, with relevant examples.

Introduction

In array of objects, a collection of array is an object. In array, We must be aware that an object is an unordered collection of key value pairs. Hence our array of objects contains key value pairs, that can be accessed by a [ ] dot operator.

In JavaScript, arrays come with a built-in function sort(), whose major goal is to sort any element based on alphabetical order.

The sort() method sorts the elements of the array in place and returns the sorted array, by default sorting it in ascending order.

Usually, we can sort an array of objects using the Array.sort function in javascript. The sorting is based on the value of the property that every object has. However, we need to provide the sort() method, a function that defines how our sorting should work, or else, using sorting mechanism.

Typically, the sorting is based on a value of a property every object has. We can use the Array.sort function, but we need to provide a function that defines our sorting mechanism.

For example, there are multiple blocks of different lengths and widths in the above image. We decide to sort them based on their widths and hence we get that result.

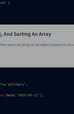
Had we sorted them based on their lengths, we would have got the below result :

Hence, while sorting an array of objects in JavaScript, it majorly depends upon the function we define for sorting our array.

Description

We can sort an array of objects in JavaScript by using our built-in Array.sort() method, however, we must pass a compare function to our built-in sort method. This function is needed for the comparison of the properties based on which we want to sort our array of objects.

The regular Array.sort() method does not work on an array of objects, for example,



In the above image, you can see that when we sort an array of strings using the Array.sort() method, it gets rarely sorted. But when we try to sort an array of objects.

For example, here we have an array of objects : Flower names and their prices, in that case, our normal Array.sort() method does not work. Hence, we need to pass a custom function to our sort method, based on which it will be able to sort our array of objects.

Syntax

The syntax for the sorting of an array of objects in JavaScript is given below :

```
// Array function
array.sort(a, b) => { /* ... */ }

// Compare function
sort(compareFunction)

// Inline compare function
sort(function compareFunction(a, b) { /* sort code here */ })
```

In the above syntax, you can see that we use sort our array of objects in JavaScript using the normal Array.sort() method. Also, here we mandatorily need to pass a function (in our sort() method) only in case of an array of objects compare function is mandatory, otherwise it is optional, which will determine our sorting mechanism. Let us understand each parameter taken by the sort function below.

Parameters

The Array.sort() function takes a comparing function as it's parameter. The compareFunction is an optional parameter generally, but when it comes to the array of objects, it becomes vital.

The compare function basically specifies a function that defines the sort order of our array of objects. The compare function can be passed in the sort method in the form of an arrow function, an inline compare function, or a normal function.

Return Value

The sort() method returns a sorted array of objects. Also, the sorting is done in place. By in-place sorting, we mean that there is no separate copy of the array made after sorting. Hence, it takes a constant amount of space for sorting the array of objects.

The compare function

To sort an array of objects in JavaScript, use the sort() method with a compare function. A compare function helps to enforce our logic on the sorting of the array of objects. They allow us to sort arrays of objects by strings, integers, dates, or any other custom property.

The compare function returns a negative, zero, or positive value, depending on the arguments passed to it.

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value by the compare function. Let us see, where a compare function returns a particular value :

```
function a(){return 1 - 2}
```

- If the compare function returns a negative value, then a is kept before a in the sorted result.
- If zero, the compare function returns a positive value, then a is kept before a in the sorted result.
- Else, if the compare function returns a zero, then the order in swaps done between a and p into the sorted result, and their order remains the same.

Inactually, it can be visualized as below :

```
if (a value > p value) {
  return 1;
}
if (a value < p value) {
  return -1;
}
else {
  return 0;
}
```

Creating, Displaying, And Sorting An Array

Let us create, display, and then sorts an array or an object based on its value. We will also see how to write the sorting function.

Code :

```
let products = [
  {
    "product_name": "The Witchers",
    "type": "book",
    "manufacturer": "Netflix",
    "manufactured": new Date('2019-05-12'),
    "price": 800
  },
  {
    "product_name": "Black Mesh",
    "type": "Shoes",
    "manufacturer": "Nike",
    "manufactured": new Date('2011-07-06'),
    "price": 1200
  },
  {
    "product_name": "Pickbags",
    "type": "bags",
    "manufacturer": "Nike",
    "manufactured": new Date('2020-09-22'),
    "price": 1200
  },
  {
    "product_name": "OnePlus 9",
    "type": "Mobile Phone",
    "manufacturer": "OnePlus",
    "manufactured": new Date('2011-03-27'),
    "price": 45000
  },
]
console.log("Original Products are:")
console.log(products)
console.log("Products sorted based on ascending order:")
  (a, p) => (a.price > p.price) ? 1 : (a.price < p.price) ? -1 : 0);
console.log("Products sorted based on descending order of their prices are:")
console.log(sortedProducts);
```

Output :

```
"Original Products are:"
Array [
Object { product_name: "The Witchers", type: "book", manufacturer: New Map (1 items) Ob
Object { product_name: "Black Mesh", type: "Shoes", manufacturer: Tue Mar 12 2019 05:30
Object { product_name: "Pickbags", type: "bags", manufacturer: Tue Mar 12 2019 05:30:00
Object { product_name: "OnePlus 9", type: "Mobile Phone", manufacturer: Tue Mar 12 20
]

"Products sorted based on descending order of their prices are:"
Array [
Object { product_name: "OnePlus 9", type: "Mobile Phone", manufacturer: Tue Mar 12 20
Object { product_name: "Black Mesh", type: "Shoes", manufacturer: Tue Jul 06 2011 05:
Object { product_name: "Pickbags", type: "bags", manufacturer: Tue Sep 22 2020 05:30:
Object { product_name: "The Witchers", type: "book", manufacturer: New Map (1 items) Ob
]
```

Explanation :

In the above example, we defined an array of objects that consisted of an array of products containing different products, their types, manufacture dates, and prices. To sort them, we wrote our custom function in the inbuilt sort method of javascript, as given below :

Code :

```
let products = [
  {
    "product_name": "The Witchers",
    "type": "book",
    "manufacturer": new Date('2019-05-12'),
    "price": 800
  },
  {
    "product_name": "Black Mesh",
    "type": "Shoes",
    "manufacturer": new Date('2011-07-06'),
    "price": 1200
  },
  {
    "product_name": "Pickbags",
    "type": "bags",
    "manufacturer": new Date('2020-09-22'),
    "price": 1200
  },
  {
    "product_name": "OnePlus 9",
    "type": "Mobile Phone",
    "manufacturer": new Date('2011-03-27'),
    "price": 45000
  },
]

// Our sorting function
products.sort(
  (a, p) =>
  (a.price > p.price) ? 1 : (a.price < p.price) ? -1 : 0);
```

Output :

```
Array [
Object { product_name: "OnePlus 9", type: "Mobile Phone", manufacturer: Tue Mar 12 20
Object { product_name: "Black Mesh", type: "Shoes", manufacturer: Tue Jul 06 2011 05:
Object { product_name: "Pickbags", type: "bags", manufacturer: Tue Sep 22 2020 05:30:
Object { product_name: "The Witchers", type: "book", manufacturer: New Map (1 items) Ob
]
```

Explanation :

In the above example, we defined an array of objects that consisted of an array of products containing different products, their types, manufacture dates, and prices. To sort them, we wrote our custom function in the inbuilt sort method of javascript, as given below :

Code :

```
let carmakeprice = products.sort((p, a) => (p.manufactured > a).manufactured);
console.log("Products sorted based on ascending order of their manufacture dates are:
console.log(carmakeprice);
```

Output :

```
"Products sorted based on ascending order of their manufacture dates are:"
Array [
Object { product_name: "The Witchers", type: "book", manufacturer: New Map (1 items) Ob
Object { product_name: "Pickbags", type: "bags", manufacturer: Tue Sep 22 2020 05:30:00
Object { product_name: "OnePlus 9", type: "Mobile Phone", manufacturer: Tue Mar 12 20
Object { product_name: "Black Mesh", type: "Shoes", manufacturer: Tue Jul 06 2011 05:
]
```

Explanation :

In the above example, we have tried to sort our array based on the manufacture dates, in ascending order. We have written our custom function, which compares two dates and returns a positive value, negative value, or a zero based on the dates. Likewise, we sort the complete array of objects based on this property.

Sort An Array Of Objects By Numbers

Let us create an array of objects and sort it based on the numbers. Below given is an example to depict the same.

Code :

```
// Example to demonstrate Sorting an array of objects by numbers
var students = [
  {
    "name": "Isabella Williams",
    "roll": 34,
  },
  {
    "name": "Sophia Martin",
    "roll": 45,
  },
  {
    "name": "Ava Benjamin",
    "roll": 17,
  },
  {
    "name": "Amelia Brown",
    "roll": 8,
  },
]
var sortedRolls = students.sort(rs, rb) => (rs.roll2 > rb.roll2) ? 1 : (rs.roll2 < rb.ro
console.log("Students sorted based on ascending order of their roll numbers are:")
console.log(sortedRolls);
```

Output :

```
"Students sorted based on ascending order of their roll numbers are:"
Array [
Object { name: "Amelia Brown", roll: 8 },
Object { name: "Ava Benjamin", roll: 17 },
Object { name: "Isabella Williams", roll: 34 },
Object { name: "Sophia Martin", roll: 45 }]
```

Explanation :

In the above example, we sort an array of objects based on the roll numbers. Here, we sort the array students here, based on the property "roll number". We create our custom function, which takes roll numbers as parameters and compares them.

Free Courses by Top Scaler Instructors
Sonu Gupta
Design
0/0
0 Modules Left
Ankur Gupta
Python for Beginners
0/0
0 Modules Left
Kumkum Thapar
Programming
0/0
0 Modules Left
Vishwas Kumar
DSA
0/0
0 Modules Left
Certificate · Assessments
Free Mock Tests · Mini Interviews and Resumes
View All

Let us look at an even simpler way to sort our array of students, based on their roll numbers :

Code:

```
// students = [
//
//   {
//     "name": "Isabella William",
//     "roll": 10,
//   },
//
//   {
//     "name": "Sophia Neal",
//     "roll": 45,
//   },
//
//   {
//     "name": "Ava Benjamin",
//     "roll": 15,
//   },
//
//   {
//     "name": "Amelia Brown",
//     "roll": 8,
//   },
//
//
//
//  // Our Sorting function
//  students.sort(lr, ll) => {
//    return ll.roll - lr.roll;
//
//  });
//
//  // printing out results
//  console.log("Students sorted based on ascending order of their roll numbers are:")
//  console.log(students);
```

Output :

```
Students sorted based on ascending order of their roll numbers are:
Array [
Object { name: "Amelia Brown", roll: 8 },
Object { name: "Ava Benjamin", roll: 15 },
Object { name: "Isabella William", roll: 10 },
Object { name: "Sophia Neal", roll: 45 }
]
```

Explanation:

In the above code, we are just calculating the differences between each roll. By doing this, we get 3 different values based on which we can perform sorting.

Say we are comparing two roll numbers a and b. If our comparison function returns a value that is less than 0, a will appear before b. If it is greater than 0, lr will appear before a. If it's exactly 0, then the original order will be maintained.

Sort An Array Of Objects By Strings

Let us now sort an array of objects based on the name of the students.

Below, the following code snippet shows how can we sort the names of the students by their first name, in an **ascending order** and vice-versa/reverse :

Code:

```
// Example to demonstrate Sorting an array of objects by a string
// students = [
//
//   {
//     "name": "Isabella William",
//     "roll": 10,
//   },
//
//   {
//     "name": "Sophia Neal",
//     "roll": 45,
//   },
//
//   {
//     "name": "Ava Benjamin",
//     "roll": 15,
//   },
//
//   {
//     "name": "Amelia Brown",
//     "roll": 8,
//   },
//
//
//  // sort by name
//  students.sort(function(l, ll) {
//    const name1 = l.name.toUpperCase(); // ignore upper and lowercase
//    const name2 = ll.name.toUpperCase(); // ignore upper and lowercase
//    if (name1 < name2) {
//      return -1;
//    }
//    if (name1 > name2) {
//      return 1;
//    }
//
//    // names must be equal
//    return 0;
//  });
//
//
//  console.log("Students sorted based on the descending order of their names are:")
//
//  console.log(students);
```

Output :

```
Students sorted based on the descending order of their names are:
Array [
Object { name: "Amelia Brown", roll: 8 },
Object { name: "Isabella William", roll: 10 },
Object { name: "Ava Benjamin", roll: 15 },
Object { name: "Sophia Neal", roll: 45 }
]
```

Explanation:

In the above example, we sorted the names of the students in **descending** order. For that, we wrote our inline custom compare function. In that function, we compared the names of the students. Bad thing, we converted them into uppercase to perform a **case-insensitive sort**.

Now, depending upon the value our search function returns, we will sort our array. As we have already seen, it can return **positive**, **negative**, and **zero** values.

Displaying in descending order :

To display the names in a **descending** order, we just reverse the order of our comparison. Let us look at the below code for a clear understanding.

Code:

```
let students = [
//
//   {
//     "name": "Isabella William",
//     "roll": 10,
//   },
//
//   {
//     "name": "Sophia Neal",
//     "roll": 45,
//   },
//
//   {
//     "name": "Ava Benjamin",
//     "roll": 15,
//   },
//
//   {
//     "name": "Amelia Brown",
//     "roll": 8,
//   },
//
//
//  students.sort(function(l, ll) {
//    const name1 = l.name.toUpperCase(); // ignore upper and lowercase
//    const name2 = ll.name.toUpperCase(); // ignore upper and lowercase
//
//  // sort in an ascending order
//    if (name1 < name2) {
//      return 1;
//    }
//    if (name1 > name2) {
//      return -1;
//    }
//
//    // names must be equal
//    return 0;
//  });
```

Output :

```
Array [
Object { name: "Amelia Brown", roll: 8},
Object { name: "Ava Benjamin", roll: 15},
Object { name: "Isabella William", roll: 10},
Object { name: "Sophia Neal", roll: 45}
]
```

Explanation:

By just reversing the order of our **comparison** operation, we can sort our data in descending order. Please refer to how the **sort** function works via different return values of the compare function.

Displaying data using a for/arch loop :

You might have noticed, that we are using the **"console.log"** statements to log our output. Let us also learn to display the data using for/arch loop into a particular format.

Code:

```
let students = [
//
//   {
//     "name": "Isabella William",
//     "roll": 10,
//   },
//
//   {
//     "name": "Sophia Neal",
//     "roll": 45,
//   },
//
//   {
//     "name": "Ava Benjamin",
//     "roll": 15,
//   },
//
//   {
//     "name": "Amelia Brown",
//     "roll": 8,
//   },
//
//
//  students.sort(function(l, ll) {
//    const name1 = l.name.toUpperCase(); // ignore upper and lowercase
//    const name2 = ll.name.toUpperCase(); // ignore upper and lowercase
//
//  // sort in an ascending order
//    if (name1 < name2) {
//      return 1;
//    }
//    if (name1 > name2) {
//      return -1;
//    }
//
//    // names must be equal
//    return 0;
//  });
//
//
//  console.log("Students sorted based on ascending order of their names are:")
//
//  // displaying student data using for/arch loop
//  students.forEach(ele => {
//    console.log( `${ele.name} ${ele.roll}` );
//
//  });
```

Output:

```
Students sorted based on ascending order of their names are:
Isabella William 10
Sophia Neal 45
Ava Benjamin 15
Amelia Brown 8
```

Explanation :

In the above example, we have used the for/arch loop to display the student name and roll number.

Sort An Array Of Objects By Dates

Suppose that we have an array of student objects as given in the code example below. We will try to sort it out based on their registration date. Let us learn how to sort an array of objects by date.

To sort an array of objects by date strings, all we need to do is, provide a **compare** function that **parses** the date string first, using the **new Date()** constructor and subtract them from each other.

The following code illustrates the main concept of sorting by date in an array of objects :

Code:

```
// Example to demonstrate Sorting an array of objects by dates
let students = [
//
//   {
//     "name": "Isabella William",
//     "roll": 10,
//     "registrationDate": "January 20, 2019"
//   },
//
//   {
//     "name": "Sophia Neal",
//     "roll": 45,
//     "registrationDate": "March 17, 2012"
//   },
//
//   {
//     "name": "Ava Benjamin",
//     "roll": 15,
//     "registrationDate": "April 10, 2017"
//   },
//
//   {
//     "name": "Amelia Brown",
//     "roll": 8,
//     "registrationDate": "July 15, 2017"
//   },
//
//
//  // sort by the registration date
//  students.sort(lr, ll => {
//    x = new Date(lr.registrationDate);
//    y = new Date(ll.registrationDate);
//    return x - y;
//  });
//
//
//  console.log("Students sorted based on ascending order of their registration dates are:")
//
//  // displaying the values
//  students.forEach(ele => {
//    console.log( `${ele.name} ${ele.roll} ${ele.registrationDate}` );
//
//  });
```

Output :

```
Students sorted based on ascending order of their registration dates are:
Sophia Neal 45 March 17, 2012
Amelia Brown 8 July 15, 2017
Isabella William 10 January 20, 2019
Sophia Neal 45 March 17, 2012
```

Explanation:

In the above example, we are **sorting** our array of objects in **ascending** order, based on the registration date values. To sort them, we first have with dummy dates and then we'll then by comparing two dates to our custom inline sort function.

For example, the below piece of code will be used for sorting our dates :

Code:

```
let students = [
//
//   {
//     "name": "Isabella William",
//     "roll": 10,
//     "registrationDate": "January 20, 2019"
//   },
//
//   {
//     "name": "Sophia Neal",
//     "roll": 45,
//     "registrationDate": "March 17, 2012"
//   },
//
//   {
//     "name": "Ava Benjamin",
//     "roll": 15,
//     "registrationDate": "April 10, 2017"
//   },
//
//   {
//     "name": "Amelia Brown",
//     "roll": 8,
//     "registrationDate": "July 15, 2017"
//   },
//
//
//  students.sort(lr, ll => {
//    x = new Date(lr.registrationDate);
//    y = new Date(ll.registrationDate);
//    return x - y;
//  });
```

Output :

```
Array [
Object { name: "Ava Benjamin", roll: 15, registrationDate: "April 10, 2017" },
Object { name: "Amelia Brown", roll: 8, registrationDate: "July 15, 2017" },
Object { name: "Isabella William", roll: 10, registrationDate: "January 20, 2019"},
Object { name: "Sophia Neal", roll: 45, registrationDate: "March 17, 2012" }
]
```

Here we are taking 2 strings (which are dates) and creating them into dates using the **new Date()** constructor. After that, we find out the difference between the two dates, and based on the return value, that is, either **positive**, **negative** or zero return value, we are sorting the dates.

Sorting Stability

In general, the **stable** sorting algorithms are the ones, that maintain the relative order or position of the elements. In other words, **stable sorting** algorithms maintain the position of two equal elements relative to one another.

Array.sort is stable according to the specifications, since ES 2019 or version 10. Before that, the sorting stability was not guaranteed.

For example, say we have an array of players, where each player has a name and a score.

Now, if we sort the players in the **descending** order of their scores, we will see that the players with the same scores will maintain their relative positions. For example, we sort that above player object in the code given below.

After that, we will display our data using the for/arch loop in javascript.

Code:

```
const player = [
//   { name: "Jack", score: 40 },
//   { name: "Tom", score: 55 },
//   { name: "Bob", score: 100 },
//   { name: "Edward",  score: 50 }
//
//
//  // sorting the players in ascending orderr of their score
//  player.sort((firstPlayer, secondPlayer) => firstPlayer.score - secondPlayer.score);
//
//  // displaying the values
//  player.forEach(ele => {
//    console.log( `${ele.name} ${ele.score}` );
//
//  });
```

Finally, we will get the below output :

Output:

```
Edward 50
Tom 55
Jack 40
Bob 100
```

Here, in the above output, the players Jim and Rob have the same scores, and the order of Jim coming before Rob is maintained. This is what stable sort guarantee.

Before version 10 in EcmaScript 2019, **sort stability** was not guaranteed, which means, we could also get some output as shown below :

Output :

```
Edward 50
Tom 55
Jack 40
Bob 100
```

Conclusion

- An **array of objects** is a collection of multiple objects into an array. The objects can contain **key-value** pair and have properties and values.
- We can sort the array of objects using the **sort()** method in javascript and then provide a comparison function that will ultimately determine the order of the objects.
- A **compare** function applies rules to sort arrays that are defined by our logic.
- Compare Function allows us to sort arrays of objects by **strings**, **integers**, **dates**, or any other custom property.
- We can sort an array of objects on the property string by comparing them and returning -1, 1, and 0, depending on the string comparison.
- To sort an array of objects by date, we can convert the joined dates from strings to date objects. And then sorting our array based on dates.
- Similarly, we can sort the numbers in an array of objects by calculating the difference between them and returning **positive**, **negative**, or zero values. Based on that, our order of sorts will be defined.
- **Array.sort()** is stable since version 10 or ES 2019.

Challenge Time!

Time to test your skills and win rewards!

Start Challenge

Note: Rewards will be credited after the next product update.

This article is written by

Deepa Pandey

Updated : 12 Jul 2022 ● 17 min read ● Published : 11 Jul 2022

Scroll to top

JavaScript

How would you rate this article?

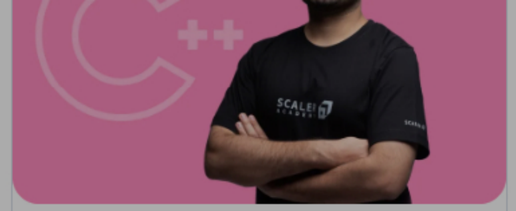
Free Courses by top Scaler instructors

Java for Beginners

A Course by **Tarun Luthra**

Learn programming fundamentals and core concepts of Java, the most wid…

12 Modules Certificate Included



C++ for Beginners

A Course by **Prateek Narang**

Learn logic building & basics of programming by learning C++, one o…

14 Modules Certificate Included

SQL v/s NoSQL

A Course by **Mohit Yadav**

Understand difference between SQL and NoSQL databases.

1 Modules Certificate Included

Quora System Design

A Course by **Tarun Malhotra**

Learn how to do system design for a QnA website like Quora.

1 Modules Certificate Included

View All

Got suggestions? **We would love to hear your feedback.**

SCALER Topics

A Free learning platform made with by SCALER

Download the App!

GET IT ON Google Play

Explore Scaler

Academy
Academy

Data Science & ML

Neovarsity

Explore Topics

Courses
Courses

Challenges

Contest

Reading Tracks

Articles

Events

Resources

About Us
About Us

Blog

Careers

Review

Get a **Free** personalized Career Roadmap from SCALER

Privacy Policy • Terms of Use • Contact Us

Privacy Policy • Terms of Use • Contact Us

Copyright 2022 InterviewBit Technologies Pvt. Ltd.
All Rights Reserved.

Get a **Free** personalized Career Roadmap from SCALER