# How does sort function work in JavaScript, along with compare function

Asked 8 years, 3 months ago Active 3 years, 1 month ago Viewed 56k times

58

46

As already asked: how does sort function work in JavaScript, along with the `compare` function? If I have an array, and I do `array.sort(compare)` now it was written in the book that if the `compare` function returns `a-b` (two indices of the array) then it works based on the fact that whether the result is greater than 0, less than 0 or equal to 0. But, how exactly does it work? I could not work it out.

javascript

edited Aug 19 '15 at 20:05
Zentaurus
**540** 1 8 24

asked Jul 4 '11 at 6:17
Kraken
**9,392** 27 81 134

What exactly do you need to know? I'm pretty sure the sort algorithm used is implementation-specific. – ThiefMaster Jul 4 '11 at 6:25

what does compare function has to do with the functioning of sorting, wont it just compare the two variables and give me back the result for these two, how does the whole array get sorted? – Kraken Jul 4 '11 at 6:29

possible duplicate of [How does Javascript's sort() work?](https://stackoverflow.com) – Bergi Jun 21 at 21:35

## 5 Answers

134

The "compare" function must take two arguments, often referred to as **a** and **b**. Then you make the compare function return 0, greater than 0, or less than 0, based on these values, **a** and **b**.

1. Return greater than 0 if **a** is greater than **b**
2. Return 0 if **a** equals **b**
3. Return less than 0 if **a** is less than **b**

type, or complex data structures.

Then, when you call sort(), with your custom compare function, the compare function is called on pairs in your to-be-sorted list, to determine the proper ordering.

Lets walk through a simple example... Suppose you're only sorting some numbers, so we have a very simple compare function:

```
function compare(a,b) {
    return a - b;
}
```

Simply subtracting b from a will always return greater than zero if a is larger than b, 0 if they are equal, or less than zero if a is less than b. So it meets the requirements for a compare function.

Now lets suppose this is our list of numbers to sort:

```
var numbers = [1,5,3.14];
```

When you call `numbers.sort(compare)` , internally it will actually execute:

```
compare(1,5);     // Returns -4, a is less than b
compare(1,3.14);  // Return -2.14, a is less than b
compare(5,3.14);  // returns 1.86, a is greater than b
```

If you've ever done manual sorting or alphabetizing, you've done precisely the same thing, probably without realizing it. Even though you may have dozens or hundreds of items to compare, you're constantly comparing only two numbers (or author's last names, or whatever) at a time. Going through or short list of three numbers again, you'd start by comparing the first two numbers:

1. Is 1 greater than or less than 5? Less than, so put these two numbers in our list: 1,5
2. Is 3.14 greater than or less than 1? Greater than, so it goes after 1 in the new list
3. Is 3.14 greater than or less than 5 in our new list? Less than, so it goes before 5. Our new list is now [1,3.14,5]

Because you can provide your own compare() function, it is possible to sort arbitrarily complex data, not just numbers.

edited Mar 17 '15 at 14:54 answered Jul 4 '11 at 6:38

How about time complexity in the above case ? – ThisIzKp Jun 27 '14 at 4:46

Every single function has a time complexity. Im pretty sure that javascript .sort() uses the quicksort (n log n ), but he asks, by calling the secondary function, compare(a,b), how much will add. I would say since the comparison is linear, the function still keeps an asymptotic behaviour. If someone can confirme this one, would be great! – David Sánchez May 30 '15 at 15:24

So, should we declare the compare function ourselves before we sort anything or otherwise how does sort() method know which compare function we are using? – Fadeoc Khaos Jun 8 '15 at 15:47

@FadeocKhaos when you do number.sort(a, b) (as in above example), you are passing compare function as a parameter to sort(). sort() in JavaScript arrays accepts comparison fn as an optional parameter. Since comparison fn in this case only acts as reference, it is acting as an anonymous function. This is a much broader topic in itself but I hope this helps. I can give you detailed info on this if posted as a separate question. Thanks. – rock Aug 20 '15 at 17:42

@ranjith Thank u, it helps. – Fadeoc Khaos Aug 22 '15 at 6:13

29

By default, the array `sort()` method sorts alphabetically ascending. If you want to sort in some other order, because your array contains numbers or objects then you can pass a function in to the `sort()` .

The function you pass in takes two parameters, often called a and b, and returns: a negative number if the first argument should be sorted before the second (a < b) 0 if the arguments are equal (a==b) a positive number if the first argument should be sorted after the second (a > b)

Now, **here is the key bit**: the function you pass as a parameter to `sort()` will be called repeatedly by `sort()` as it processes the whole array. `sort()` doesn't know or care about the datatype of the things in the array: each time it needs to know "Does item A come before item B?" it just calls your function. You don't need to worry about what type of sort algorithm is used internally by `sort()` , indeed one browser might use a different algorithm to another, but that's OK because you just have to provide a way for it to compare any two items from your array.

Your function could have an `if / else if / else` structure to decide what result to return, but for numbers simply returning (a-b) will achieve this for you because the result of the subtraction will be -ve, 0 or +ve and correctly put the numbers in ascending order. Returning (b-a) would put them descending:

```
var sortedArray = myArray.sort(function(a,b){
                                    return (a-b);
                                });
```

```
{ id : 1,
  name : "Fred",
  address : "12 Smith St",
  phone : "0262626262" }
```

Then you could sort an array of such objects by their 'id' attribute as follows:

```
var sortedArray = myArray.sort(function(a,b){
                                  return (a.id - b.id);
                              });
```

Or you could sort an array of such objects by their 'name' attribute (alphabetical) as follows:

```
var sortedArray = myArray.sort(function(a,b){
                                   if (a.name < b.name)
                                      return -1;
                                   else if (a.name == b.name)
                                      return 0;
                                   else
                                      return 1;
                               });
```

Note that in my final example I've put the full `if / else if / else` structure I mentioned earlier.

For the example where you are sorting objects with multiple properties you could expand that further to include a secondary sort, that is, (in my example) if the name properties are equal you could then return a comparison of, say, the phone property.

edited Jul 4 '11 at 6:48

answered Jul 4 '11 at 6:36
nnnnnn
**127k** 21 156 213

I like this answer better because it includes an example with string comparisons +1. – Klik Feb 23 '13 at 0:56

2 I like the description 'sort() will be called repeatedly' +1 – Gangadhar Jannu May 17 '16 at 13:29

1 So, when sorting numbers, I could return -1, 0, 1 OR I could return `b - a` OR `a - b`. The result will be the same, is it? No significant difference or anything special about `-1` and `1` in this context? – CodeFinity Jan 5 '18 at 12:05

1 @VicWebsoft Yes, the result will be the same. – nnnnnn Jan 5 '18 at 23:45

5

This method uses the syntax and parameters of the order of Array.sort (the compareFunction the sortOptions), whose parameters are defined as follows:

compareFunction - a comparison function used to determine the sort order of the array elements. This parameter is optional. The comparison function should be used to compare the two parameters. A and B of a given element, the result of compareFunction can have a negative value, 0 or a positive value:

If the return value is negative, it means that A appears before B in the sorted sequence. If the return value is 0, then A and B have the same sort order. If the return value is positive, it means that A appears after B in the sorted sequence.

answered Dec 5 '12 at 9:49
gamgam
**51** 1 3

---

1

> sort method alone treat numbers as strings so if the array of strings you don't need the compare function. but if the array of numbers you need the compare function to alter the build behavior of the sort method.

**ex1 :strings**

```
var animals = ["Horse", "Cat", "Tiger", "Lion"];
animals.sort();
```

**ex2 : numbers**

```
var marks = [70, 90, 60, 80 ];
marks.sort(function(a, b){return a > b}); //ascending , a < b descending .
```

answered Aug 24 '16 at 22:29
Eiko No Shance
**11** 2

0

the sort result array `M` .

I don't know the exact sort algorithm , and different browsers even return different results if `c` is neither `(a-b)` nor `(b-a)` (say, if `c` is `"b-2"` , `"a+b"` or some other expressions).

But according to `ECMA-262` , the sort result should be like this:

a , b can be any two of the indexes. this means we actually passed an ordered pair to the compare function. `eg: (0,1),(1,4), or even (2,0) , (2,1) .`

The ECMAScript Language Specification says the result should have this property: `(a,b)` is an ordered pair passed to the compare function.

- if `c` (what the function returns) is less than zero, then `M(a)< M(b)` must be satisfied.

And the specification talks nothing about what would happen if c is zero or bigger than zero.

I am not sure whether this is right. At least this can easily explain why when `c` is `"a-b"` , the entries are sorted to be numerically and ascending, and why when `c` is `"b-a"` ,the entries are sorted to the opposite.

Are the js engines of the browsers are not really designed strictly according to `ECMA-262 or am I totally wrong?

Reference：

The Fifth Edition of ECMA-262 (Check page 129-130)

edited Oct 26 '12 at 12:51
amesh
**989** 3 15 45

answered Jan 27 '12 at 10:47
alexyangfox
**105** 7