Sort array of objects by string property value

Asked 10 years ago Active 2 months ago Viewed 1.3m times

2383

563

I have an array of JavaScript objects:

```
var objs = [
    { first_nom: 'Lazslo', last_nom: 'Jamf'     },
    { first_nom: 'Pig',    last_nom: 'Bodine'   },
    { first_nom: 'Pirate', last_nom: 'Prentice' }
];
```

How can I sort them by the value of `last_nom` in JavaScript?

I know about `sort(a,b)`, but that only seems to work on strings and numbers. Do I need to add a `toString()` method to my objects?

javascript arrays sorting

edited Oct 8 '18 at 16:01
Alexander Abakumov
5,574 5 50 79

asked Jul 15 '09 at 3:17
Tyrone Slothrop
12.1k 3 13 7

3 This script allows you to do just that unless you want to write your own comparison function or sorter: thomasfrank.se/sorting_things.html – Baversjo Jul 15 '09 at 3:31

41 Answers

1 2 next

3435

It's easy enough to write your own comparison function:

```
function compare( a, b ) {
  if ( a.last_nom < b.last_nom ){
    return -1;
```

```
  }
  if ( a.last_nom > b.last_nom ){
    return 1;
  }
  return 0;
}

objs.sort( compare );
```

Or inline (c/o Marco Demaio):

```
objs.sort((a,b) => (a.last_nom > b.last_nom) ? 1 : ((b.last_nom > a.last_nom) ? -1 :
0));
```

edited Apr 21 at 14:30

KostasX
383 4 13

answered Jul 15 '09 at 3:35

Wogan
43.1k 3 28 31

390 Or inline: objs.sort(function(a,b) {return (a.last_nom > b.last_nom) ? 1 : ((b.last_nom > a.last_nom) ? -1 : 0);}); – Marco Demaio Feb 24 '10 at 18:29

33 Official docs: developer.mozilla.org/en/JavaScript/Reference/Global_Objects/… – mikemaccana May 18 '12 at 9:11

148 `return a.last_nom.localeCompare(b.last_nom)` will work, too. – Cerbrus Feb 14 '13 at 10:37

134 for those looking for a sort where the field is numeric, the compare function body: `return a.value - b.value;` (ASC) – Andre Figueiredo Jan 8 '14 at 12:06

15 @Cerbrus `localeCompare` is important when using accented characters in foreign languages, and more elegant as well. – Marcos Lima Jun 1 '16 at 16:38

771

You can also create a dynamic sort function that sorts objects by their value that you pass:

```
function dynamicSort(property) {
    var sortOrder = 1;
    if(property[0] === "-") {
        sortOrder = -1;
        property = property.substr(1);
    }
    return function (a,b) {
        /* next line works with strings and numbers,
```

```
         * and you may want to customize it to your needs
         */
        var result = (a[property] < b[property]) ? -1 : (a[property] > b[property]) ? 1
: 0;
        return result * sortOrder;
    }
}
```

So you can have an array of objects like this:

```
var People = [
    {Name: "Name", Surname: "Surname"},
    {Name:"AAA", Surname:"ZZZ"},
    {Name: "Name", Surname: "AAA"}
];
```

...and it will work when you do:

```
People.sort(dynamicSort("Name"));
People.sort(dynamicSort("Surname"));
People.sort(dynamicSort("-Surname"));
```

Actually this already answers the question. Below part is written because many people contacted me, complaining that it doesn't work with multiple parameters.

Multiple Parameters

You can use the function below to generate sort functions with multiple sort parameters.

```
function dynamicSortMultiple() {
    /*
     * save the arguments object as it will be overwritten
     * note that arguments object is an array-like object
     * consisting of the names of the properties to sort by
     */
    var props = arguments;
    return function (obj1, obj2) {
        var i = 0, result = 0, numberOfProperties = props.length;
        /* try getting a different result from 0 (equal)
         * as long as we have extra properties to compare
         */
        while(result === 0 && i < numberOfProperties) {
```

```
            result = dynamicSort(props[i])(obj1, obj2);
            i++;
        }
        return result;
    }
}
```

Which would enable you to do something like this:

```
People.sort(dynamicSortMultiple("Name", "-Surname"));
```

Subclassing Array

For the lucky among us who can use ES6, which allows extending the native objects:

```
class MyArray extends Array {
    sortBy(...args) {
        return this.sort(dynamicSortMultiple.apply(null, args));
    }
}
```

That would enable this:

```
MyArray.from(People).sortBy("Name", "-Surname");
```

edited May 18 at 15:09

answered Jan 21 '11 at 15:03

Ege Özcan
10.4k 2 26 50

Please note that property names in JavaScript can be any string and if you have properties starting with a "-" (extremely unlikely and *probably* not a good idea), you'll need to modify the dynamicSort function to use something else as a reverse sort indicator. – Ege Özcan Jan 10 '13 at 15:18

I've noticed that the `dynamicSort()` in the above example will place capital letters ahead of lowercase letters. For example, if I have the values `APd`, `Aklin`, and `Abe` - the results in an ASC sort should be `Abe`, `Aklin`, `APd`. But with your example, the results are `APd`, `Abe`, `Aklin`. Anyway to correct this behavior? – Lloyd Banks Jul 26 '17 at 22:06

5 @LloydBanks if you are using this strictly for strings, then you can use `var result = a[property].localeCompare(b[property]);` instead of `var result = (a[property] < b[property]) ? -1 : (a[property] > b[property]) ? 1 : 0;` . – Ege Özcan Aug 6 '17 at 8:55

1 @EgeÖzcan It should either put items to the top or bottom, here's implementation I've ended up using pastebin.com/g4dt23jU – dzh Jan 20 '18 at 0:37

1 This is great solution but have one problem if you compare numbers. Please add this before check: `if( !isNaN(a[property]) ) a[property] = Number(a[property]); if( !isNaN(b[property]) ) b[property] = Number(b[property]);` – Ivijan Stefan Stipić Apr 3 '18 at 5:48

290

In ES6/ES2015 or later you can do this way:

```
objs.sort((a, b) => a.last_nom.localeCompare(b.last_nom));
```

answered Jan 29 '16 at 19:44

Vlad Bezden
35.1k 15 153 121

22 this has been available since JS 1.1, the fat arrow piece to this is the ES6/2015 piece. But still very useful, and best answer in my opinion – Jon Harding Feb 22 '16 at 20:30

11 @PratikKelwalkar: if you need reverse it just switch a and b comparison: objs.sort((a, b) => b.last_nom.localeCompare(a.last_nom)); – Vlad Bezden May 26 '16 at 18:24

is it possible to use a index as well to address the field for sorting: instead of `last_nom` use just the number in the array: `1` ? – and-bri May 29 '17 at 18:15

4 @VladBezden thanks for your answer! This solution is the first with very small programmatic effort and correct sorting results and a string array like: ["Name1", "Name10", "Name2", "something else", "Name11"]. I got sorting to work correctly with `objs.sort((a, b) => a.last_nom.localeCompare(b.last_nom, undefined, {numberic: true}));` – scipper Jan 9 '18 at 8:49

To do this with numbers descending: .sort((a, b) => b.numberProperty - a.numberProperty). Ascending: .sort((a, b) => a.numberProperty - b.numberProperty) – Sebastian Patten May 23 at 5:05

178

underscore.js

use underscore, its small and awesome...

> sortBy_.sortBy(list, iterator, [context]) Returns a sorted copy of list, ranked in ascending order by the results of running each value through iterator. Iterator may also be the string name of the property to sort by (eg. length).

```
var objs = [
  { first_nom: 'Lazslo',last_nom: 'Jamf' },
```

```
  { first_nom: 'Pig', last_nom: 'Bodine'  },
  { first_nom: 'Pirate', last_nom: 'Prentice' }
];

var sortedObjs = _.sortBy( objs, 'first_nom' );
```

edited Mar 10 '16 at 19:34

Bill Sambrone
2,842 2 36 62

answered May 10 '12 at 21:24

David Morrow
4,712 4 24 24

18 David, could you edit the answer to say, `var sortedObjs = _.sortBy( objs, 'first_nom' ); . objs` will **not** be sorted itself as a result of this. The function will **return** a sorted array. That would make it more explicit. – Jess Jan 9 '14 at 4:01

9 To reverse sort: `var reverseSortedObjs = _.sortBy( objs, 'first_nom' ).reverse();` – Erdal G. Jan 31 '16 at 10:43

1 you need to load the javascript libary "underscore": `<script src="http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.3/underscore-min.js"> </script>` – and-bri May 29 '17 at 18:28

5 Also available in `Lodash` for the ones who prefer that one – WoJ Apr 17 '18 at 10:49

1 In lodash this would be the same: `var sortedObjs = _.sortBy( objs, 'first_nom' );` or if you want it in a different order: `var sortedObjs = _.orderBy( objs, ['first_nom'],['dsc'] );` – Travis Heeter Nov 15 '18 at 19:10

167

Don't get why people make it so complicated:

```
objs.sort(function(a, b){
  return a.last_nom > b.last_nom;
});
```

For stricter engines:

```
objs.sort(function(a, b){
  return a.last_nom == b.last_nom ? 0 : +(a.last_nom > b.last_nom) || -1;
});
```

Swap the operator to have it sorted by reverse alphabetical order.

edited Apr 22 '14 at 19:25

answered Jan 24 '14 at 19:35

p3lim


1,871 1 9 9

14 This is actually not correct as the function used in sort should return -1, 0, or 1 but the above function returns a boolean. The sort works fine in chrome but fails in PhantomJS for example. See code.google.com/p/phantomjs/issues/detail?id=1090 – schup Apr 22 '14 at 15:05

Some engines account for stupidity, this way was kind of exploiting that. I've updated my reply with a proper version. – p3lim Apr 22 '14 at 19:23

17 I'd suggest editing to take out the first version. Its more succinct so looks more attractive, but it doesn't work, at least not reliably. If someone tries it in one browser and it works, they may not even realise they have a problem (especially if they haven't read the comments). The second version works properly so there is really no need for the first. – Kate Mar 11 '16 at 15:01

2 @Simon I really appreciated having the "slightly wrong" version first, since the stricter implementation takes a few seconds to parse and understand and would be much harder without it. – Aaron Sherman Jun 24 '16 at 16:50

1 @Lion789 just do this: `if(a.count == b.count) return a.name > b.name; else return a.count > b.count;` – p3lim May 7 '17 at 17:16

61

If you have duplicate last names you might sort those by first name-

```
obj.sort(function(a,b){
  if(a.last_nom< b.last_nom) return -1;
  if(a.last_nom >b.last_nom) return 1;
  if(a.first_nom< b.first_nom) return -1;
  if(a.first_nom >b.first_nom) return 1;
  return 0;
});
```

edited Apr 29 '14 at 18:35

BadFeelingAboutThis
14.2k 2 30 38

answered Jul 15 '09 at 4:03

kennebec
82.5k 25 89 122

38 That `a< b  a >b` formatting is interesting. – Dodekeract Jan 26 '16 at 1:51

@BadFeelingAboutThis what does returning either -1 or 1 mean? I understand that -1 literally means that A is less than B just by the syntax, but why use a 1 or -1? I see everyone is using those numbers as return values, but why? Thanks. – Chris22 Aug 22 '18 at 6:15

1 @Chris22 a negative number returned means that `b` should come after `a` in the array. If a positive number is returned, it means `a` should come after `b`. If `0` is returned, it means they are considered equal. You can always read the documentation: developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – BadFeelingAboutThis Aug 22 '18 at 16:54

@BadFeelingAboutThis thanks, for the explanation and the link. Believe it or not, I googled various snippets of code using the `1, 0, -1` before I asked this here. I just wasn't finding the info I needed. – Chris22 Aug 22 '18 at 17:07

42

Simple and quick solution to this problem using prototype inheritance:

```
Array.prototype.sortBy = function(p) {
  return this.slice(0).sort(function(a,b) {
    return (a[p] > b[p]) ? 1 : (a[p] < b[p]) ? -1 : 0;
  });
}
```

Example / Usage

```
objs = [{age:44,name:'vinay'},{age:24,name:'deepak'},{age:74,name:'suresh'}];

objs.sortBy('age');
// Returns
// [{"age":24,"name":"deepak"},{"age":44,"name":"vinay"},{"age":74,"name":"suresh"}]

objs.sortBy('name');
// Returns
// [{"age":24,"name":"deepak"},{"age":74,"name":"suresh"},{"age":44,"name":"vinay"}]
```

Update: No longer modifies original array.

edited May 15 '15 at 21:45

Web_Designer
37k 77 186 241

answered Jul 10 '12 at 11:54

Vinay Aggarwal
1,033 1 8 17

5 It dosn't just return another array. but actually sorts the original one!. – Vinay Aggarwal Jul 21 '12 at 5:43

If you want to make sure you are using a natural sort with numbers (i.e., 0,1,2,10,11 etc...) use parseInt with the Radix set. developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… so: return (parseInt(a[p],10) > parseInt(b[p],10)) ? 1 : (parseInt(a[p],10) < parseInt(b[p],10)) ? -1 : 0; – Paul May 11 '15 at 19:14

@codehuntr Thanks for correcting it. but i guess instead of making sort function to do this sensitization, it's better if we make a separate function to fix data types. Because sort function can't not tell which property will contain what kind of data. :) – Vinay Aggarwal May 21 '15 at 16:55

Very nice. Reverse the arrows to get asc/desc effect. – Abdul Sadik Yalcin Nov 15 '17 at 15:08

3 never, ever suggest a solution that modifies the prototype of a core object – pbanka Oct 27 '18 at 0:17

30

As of 2018 there is a much shorter and elegant solution. Just use. Array.prototype.sort().

Example:

```
var items = [
  { name: 'Edward', value: 21 },
  { name: 'Sharpe', value: 37 },
  { name: 'And', value: 45 },
  { name: 'The', value: -12 },
  { name: 'Magnetic', value: 13 },
  { name: 'Zeros', value: 37 }
];

// sort by value
items.sort(function (a, b) {
  return a.value - b.value;
});
```

answered Jun 4 '18 at 15:24

0leg

4,185 6 40 62

5 In the question strings were used for comparison as opposed to numbers. Your answer works great for sorting by numbers, but isn't so good for comparisons by string. – smcstewart Jun 18 '18 at 10:02

The `a.value - b.value` used to compare the object's attributes (**numbers** in this case) can be adopted for the various times of data. For example, regex can be used to compare each pair of the neighboring **strings**. – 0leg Mar 26 at 9:10

1 @0leg I'd like to see an example here of using regex to compare strings in this way. – Bob Stein Mar 28 at 11:30

28

Instead of using a custom comparison function, you could also create an object type with custom `toString()` method (which is invoked by the default comparison function):

```
function Person(firstName, lastName) {
    this.firtName = firstName;
    this.lastName = lastName;
}

Person.prototype.toString = function() {
    return this.lastName + ', ' + this.firstName;
```

```
}

var persons = [ new Person('Lazslo', 'Jamf'), ...]
persons.sort();
```

answered Jul 15 '09 at 7:21

Christoph

134k 33 163 226

24

You can use

Easiest Way: *Lodash*

(https://lodash.com/docs/4.17.10#orderBy)

This method is like _.sortBy except that it allows specifying the sort orders of the iteratees to sort by. If orders is unspecified, all values are sorted in ascending order. Otherwise, specify an order of "desc" for descending or "asc" for ascending sort order of corresponding values.

Arguments

collection (Array|Object): The collection to iterate over. [iteratees=[_.identity]] (Array[]|Function[]|Object[]|string[]): The iteratees to sort by. [orders] (string[]): The sort orders of iteratees.

Returns

(Array): Returns the new sorted array.

```
var _ = require('lodash');
var homes = [
    {"h_id":"3",
     "city":"Dallas",
     "state":"TX",
     "zip":"75201",
     "price":"162500"},
    {"h_id":"4",
     "city":"Bevery Hills",
     "state":"CA",
     "zip":"90210",
```

```
 "price":"319250"},
{"h_id":"6",
 "city":"Dallas",
 "state":"TX",
 "zip":"75000",
 "price":"556699"},
{"h_id":"5",
 "city":"New York",
 "state":"NY",
 "zip":"00010",
 "price":"962500"}
];

_.orderBy(homes, ['city', 'state', 'zip'], ['asc', 'desc', 'asc']);
```

answered Aug 23 '18 at 14:39

Harshal

2,895 1 6 30

22

[Lodash.js](#) (superset of [Underscore.js](#))

It's good not to add a framework for every simple piece of logic, but relying on a well tested utility frameworks, speed up development and reduce the amount of bugs written is no shame.

Lodash produces very clean code and promotes a more *functional programming* style, which results in less bugs. In one glimpse it becomes clear what the intent if the code is.

OP's issue can simply be solved as:

```
const sortedObjs = _.sortBy(objs, 'last_nom');
```

More info? E.g. we have following nested object:

```
const users = [
  { 'user': {'name':'fred', 'age': 48}},
  { 'user': {'name':'barney', 'age': 36 }},
  { 'user': {'name':'wilma'}},
  { 'user': {'name':'betty', 'age': 32}}
];
```

We now can use the _.property shorthand `user.age` to specify the path to the property that should be matched. We will sort the user objects by the nested age property. Yes, it allows for nested property matching!

```
const sortedObjs = _.sortBy(users, ['user.age']);
```

Want it reversed? No problem. Use _.reverse.

```
const sortedObjs = _.reverse(_.sortBy(users, ['user.age']));
```

Want to combine both using Chaining instead?

```
const sortedObjs = _.chain(users).sortBy('user.age').reverse().value();
```

edited Jan 23 '18 at 22:42
Seraf
374 6 19

answered Aug 30 '17 at 14:15
Nico Van Belle
2,763 3 16 29

21

There are many good answers here, but I would like to point out that they can be extended very simply to achieve a lot more complex sorting. The only thing you have to do is to use the OR operator to chain comparision functions like this:

```
objs.sort((a,b)=> fn1(a,b) || fn2(a,b) || fn3(a,b) )
```

Where `fn1`, `fn2`, ... are the sort functions which return [-1,0,1]. This results in "sorting by fn1", "sorting by fn2" which is pretty much equal to ORDER BY in SQL.

This solution is based on the behaviour of `||` operator which evaluates to the first evaluated expression which can be converted to true.

The simplest form has only one inlined function like this:

```
// ORDER BY last_nom
objs.sort((a,b)=> a.last_nom.localeCompare(b.last_nom) )
```

Having two steps with `last_nom`, `first_nom` sort order would look like this:

```
// ORDER_BY last_nom, first_nom
objs.sort((a,b)=> a.last_nom.localeCompare(b.last_nom) ||
                  a.first_nom.localeCompare(b.first_nom)  )
```

A generic comparision function could be something like this:

```
// ORDER BY <n>
let cmp = (a,b,n)=>a[n].localeCompare(b[n])
```

This function could be extended to support numeric fields, case sensitity, arbitary datatypes etc.

You can them use it with chaining them by sort priority:

```
// ORDER_BY last_nom, first_nom
objs.sort((a,b)=> cmp(a,b, "last_nom") || cmp(a,b, "first_nom") )
// ORDER_BY last_nom, first_nom DESC
objs.sort((a,b)=> cmp(a,b, "last_nom") || -cmp(a,b, "first_nom") )
// ORDER_BY last_nom DESC, first_nom DESC
objs.sort((a,b)=> -cmp(a,b, "last_nom") || -cmp(a,b, "first_nom") )
```

The point here is that pure JavaScript with functional approach can take you a long way without external libraries or complex code. It is also very effective, since no string parsing have to be done

answered May 5 '16 at 11:36

Tero Tolonen

3,094 2 16 26

20

Example Usage:

```
objs.sort(sortBy('last_nom'));
```

Script:

```
/**
 * @description
 * Returns a function which will sort an
 * array of objects by the given key.
```

```
 *
 * @param  {String}  key
 * @param  {Boolean} reverse
 * @return {Function}
 */
const sortBy = (key, reverse) => {

  // Move smaller items towards the front
  // or back of the array depending on if
  // we want to sort the array in reverse
  // order or not.
  const moveSmaller = reverse ? 1 : -1;

  // Move larger items towards the front
  // or back of the array depending on if
  // we want to sort the array in reverse
  // order or not.
  const moveLarger = reverse ? -1 : 1;

  /**
   * @param  {*} a
   * @param  {*} b
   * @return {Number}
   */
  return (a, b) => {
    if (a[key] < b[key]) {
      return moveSmaller;
    }
    if (a[key] > b[key]) {
      return moveLarger;
    }
    return 0;
  };
};
```

edited Mar 21 at 17:18

answered Apr 30 '14 at 10:02

Jamie Mason

2,626 1 20 35

thank you for breaking this down, I am trying to understand why digits `1, 0, -1` are used for sort ordering. Even with your explanation above, which looks very good-- I'm still not quite understanding it. I always think of `-1` as when using array length property, i.e.: `arr.length = -1` means that the item isn't found. I'm probably mixing things up here, but could you help me understand why digits `1, 0, -1` are used to determine order? Thanks. – Chris22 Aug 22 '18 at 6:23

1 This isn't *entirely* accurate but it might help to think of it like this: the function passed to array.sort is called once for each item in the array, as the argument named "a". The return value of each function call is how the index (current position number) of item "a" should be altered compared to the

next item "b". The index dictates the order of the array (0, 1, 2 etc) So if "a" is at index 5 and you return -1 then 5 + -1 == 4 (move it nearer front) 5 + 0 == 5 (keep it where it is) etc. It walks the array comparing 2 neighbours each time until it reaches the end, leaving a sorted array. – Jamie Mason Aug 23 '18 at 9:09

thank you for taking the time to explain this further. So using your explanation and the MDN Array.prototype.sort, I'll tell you what I'm thinking of this: in comparison to `a` and `b`, if `a` is greater than `b` add 1 to the index of `a` and place it behind `b`, if `a` is less than `b`, subtract 1 from `a` and place it in front of `b`. If `a` and `b` are the same, add 0 to `a` and leave it where it is. – Chris22 Aug 23 '18 at 16:28

That sounds good to me @Chris22, thanks. – Jamie Mason Aug 24 '18 at 14:13

18

I have a piece of code that works for me:

```
arr.sort((a, b) => a.name > b.name)
```

UPDATE: Not working always, so it is not correct :(

edited Dec 7 '17 at 22:51

answered Oct 20 '17 at 12:10

Damjan Pavlica

11.8k 5 35 54

It works, but the result is unstable for some reason – TitanFighter Feb 3 '18 at 19:55

use '-' instead of '>'. That will work. – AO17 Feb 15 '18 at 16:43

@AO17 no it won't. You can't subtract strings. – Patrick Roberts Jul 18 '18 at 9:23

10 This should do it: `arr.sort((a, b) => a.name < b.name ? -1 : (a.name > b.name ? 1 : 0))` – Jean-François Beauchamp Aug 29 '18 at 15:33

1 @Jean-FrançoisBeauchamp, your solution works perfectly fine and much better. – Ahsan Jan 22 at 13:49

17

I haven't seen this particular approach suggested, so here's a terse comparison method I like to use that works for both `string` and `number`:

```
const objs = [
  { first_nom: 'Lazslo', last_nom: 'Jamf'     },
  { first_nom: 'Pig',    last_nom: 'Bodine'   },
  { first_nom: 'Pirate', last_nom: 'Prentice' }
```

```
];

const sortBy = fn => (a, b) => {
  const fa = fn(a)
  const fb = fn(b)
  return -(fa < fb) || +(fa > fb)
}
const getLastName = o => o.last_nom
const sortByLastName = sortBy(getLastName)

objs.sort(sortByLastName)
console.log(objs.map(getLastName))
```

Run code snippet Expand snippet

Here's an explanation of `sortBy()`:

`sortBy()` accepts a `fn` that selects what value from an object to use as comparison, and returns a function that can be passed directly to `Array.prototype.sort()`. In this example, we're using `o.last_nom` as the value for comparison, so whenever we receive two objects through `Array.prototype.sort()` such as

```
{ first_nom: 'Lazslo', last_nom: 'Jamf' }
```

and

```
{ first_nom: 'Pig', last_nom: 'Bodine' }
```

we use

```
(a, b) => {
  const fa = fn(a)
  const fb = fn(b)
  return -(fa < fb) || +(fa > fb)
}
```

to compare them.

Remembering that `fn = o => o.last_nom`, we can expand the compare function to the equivalent

```
(a, b) => {
  const fa = a.last_nom
  const fb = b.last_nom
  return -(fa < fb) || +(fa > fb)
}
```

The logical OR `||` operator has a short-circuiting functionality that's very useful here. Because of how it works, the body of the function above means

```
if (fa < fb) return -1
return +(fa > fb)
```

So if `fa < fb` we return `-1`, otherwise if `fa > fb` then we return `+1`, but if `fa == fb`, then `fa < fb` and `fa > fb` are `false`, so it returns `+0`.

As an added bonus, here's the equivalent in ECMAScript 5 without arrow functions, which is unfortunately more verbose:

```
var objs = [
  { first_nom: 'Lazslo', last_nom: 'Jamf'     },
  { first_nom: 'Pig',    last_nom: 'Bodine'   },
  { first_nom: 'Pirate', last_nom: 'Prentice' }
];

var sortBy = function (fn) {
  return function (a, b) {
    var fa = fn(a)
    var fb = fn(b)
    return -(fa < fb) || +(fa > fb)
  }
}

var getLastName = function (o) { return o.last_nom }
var sortByLastName = sortBy(getLastName)

objs.sort(sortByLastName)
console.log(objs.map(getLastName))
```

Run code snippet Expand snippet

edited Apr 30 at 7:35 answered Jul 18 '18 at 9:53

Patrick Roberts
24k 3 41 82

15

I know this question is too old, but I didn't see any implementation similar to mine.
This version is based on the Schwartzian transform idiom.

```
function sortByAttribute(array, ...attrs) {
  // generate an array of predicate-objects contains
  // property getter, and descending indicator
  let predicates = attrs.map(pred => {
    let descending = pred.charAt(0) === '-' ? -1 : 1;
    pred = pred.replace(/^-/, '');
    return {
      getter: o => o[pred],
      descend: descending
    };
  });
  // schwartzian transform idiom implementation. aka: "decorate-sort-undecorate"
  return array.map(item => {
    return {
      src: item,
      compareValues: predicates.map(predicate => predicate.getter(item))
    };
  })
  .sort((o1, o2) => {
    let i = -1, result = 0;
    while (++i < predicates.length) {
      if (o1.compareValues[i] < o2.compareValues[i]) result = -1;
      if (o1.compareValues[i] > o2.compareValues[i]) result = 1;
      if (result *= predicates[i].descend) break;
    }
    return result;
  })
  .map(item => item.src);
}
```

Here's an example how to use it:

```
let games = [
  { name: 'Pako',              rating: 4.21 },
  { name: 'Hill Climb Racing', rating: 3.88 },
  { name: 'Angry Birds Space', rating: 3.88 },
  { name: 'Badland',           rating: 4.33 }
```

```
];

// sort by one attribute
console.log(sortByAttribute(games, 'name'));
// sort by mupltiple attributes
console.log(sortByAttribute(games, '-rating', 'name'));
```

edited Nov 6 '16 at 13:26

answered Nov 6 '16 at 13:18

a8m

7,652 3 32 37

14

Sorting (more) Complex Arrays of Objects

Since you probably encounter more complex data structures like this array, I would expand the solution.

TL;DR

> Are more pluggable version based on **@ege-Özcan**'s very lovely answer.

Problem

I encountered the below and couldn't change it. I also did not want to flatten the object temporarily. Nor did I want to use underscore / lodash, mainly for performance reasons and the fun to implement it myself.

```
var People = [
   {Name: {name: "Name", surname: "Surname"}, Middlename: "JJ"},
   {Name: {name: "AAA", surname: "ZZZ"}, Middlename:"Abrams"},
   {Name: {name: "Name", surname: "AAA"}, Middlename: "Wars"}
];
```

Goal

The goal is to sort it primarily by `People.Name.name` and secondarily by `People.Name.surname`

Obstacles

Now, in the base solution uses bracket notation to compute the properties to sort for dynamically. Here, though, we would have to construct the bracket notation dynamically also, since you would expect some like `People['Name.name']` would work - which doesn't.

Simply doing `People['Name']['name']`, on the other hand, is static and only allows you to go down the *n*-th level.

Solution

The main addition here will be to walk down the object tree and determine the value of the last leaf, you have to specify, as well as any intermediary leaf.

```
var People = [
   {Name: {name: "Name", surname: "Surname"}, Middlename: "JJ"},
   {Name: {name: "AAA", surname: "ZZZ"}, Middlename:"Abrams"},
   {Name: {name: "Name", surname: "AAA"}, Middlename: "Wars"}
];

People.sort(dynamicMultiSort(['Name','name'], ['Name', '-surname']));
// Results in...
// [ { Name: { name: 'AAA', surname: 'ZZZ' }, Middlename: 'Abrams' },
//   { Name: { name: 'Name', surname: 'Surname' }, Middlename: 'JJ' },
//   { Name: { name: 'Name', surname: 'AAA' }, Middlename: 'Wars' } ]

// same logic as above, but strong deviation for dynamic properties
function dynamicSort(properties) {
  var sortOrder = 1;
  // determine sort order by checking sign of last element of array
  if(properties[properties.length - 1][0] === "-") {
    sortOrder = -1;
    // Chop off sign
    properties[properties.length - 1] = properties[properties.length - 1].substr(1);
  }
  return function (a,b) {
    propertyOfA = recurseObjProp(a, properties)
    propertyOfB = recurseObjProp(b, properties)
    var result = (propertyOfA < propertyOfB) ? -1 : (propertyOfA > propertyOfB) ? 1 : 0;
    return result * sortOrder;
  };
}

/**
 * Takes an object and recurses down the tree to a target leaf and returns it value
 * @param  {Object} root - Object to be traversed.
 * @param  {Array} leafs - Array of downwards traversal. To access the value: {parent:{
child: 'value'}} -> ['parent','child']
 * @param  {Number} index - Must not be set, since it is implicit.
 * @return {String|Number}       The property, which is to be compared by sort.
```

```
 */
function recurseObjProp(root, leafs, index) {
  index ? index : index = 0
  var upper = root
  // walk down one level
  lower = upper[leafs[index]]
  // Check if last leaf has been hit by having gone one step too far.
  // If so, return result from last step.
  if (!lower) {
    return upper
  }
  // Else: recurse!
  index++
  // HINT: Bug was here, for not explicitly returning function
  // https://stackoverflow.com/a/17528613/3580261
  return recurseObjProp(lower, leafs, index)
}

/**
 * Multi-sort your array by a set of properties
 * @param {...Array} Arrays to access values in the form of: {parent:{ child: 'value'}}
-> ['parent','child']
 * @return {Number} Number - number for sort algorithm
 */
function dynamicMultiSort() {
  var args = Array.prototype.slice.call(arguments); // slight deviation to base

  return function (a, b) {
    var i = 0, result = 0, numberOfProperties = args.length;
    // REVIEW: slightly verbose; maybe no way around because of `.sort`-'s nature
    // Consider: `.forEach()`
    while(result === 0 && i < numberOfProperties) {
      result = dynamicSort(args[i])(a, b);
      i++;
    }
    return result;
  }
}
```

Example

Working example on JSBin

edited May 23 '17 at 12:18
Community ♦
1 1

answered Aug 10 '15 at 15:52
eljefedelrodeodeljefe
3,233 4 18 46

2 Why? This is not the answer to original question and "the goal" could be solved simply with People.sort((a,b)=>{ return a.Name.name.localeCompare(b.Name.name) || a.Name.surname.localeCompare(b.Name.surname) }) – Tero Tolonen May 3 '16 at 16:02

11

One more option:

```
var someArray = [...];

function generateSortFn(prop, reverse) {
    return function (a, b) {
        if (a[prop] < b[prop]) return reverse ? 1 : -1;
        if (a[prop] > b[prop]) return reverse ? -1 : 1;
        return 0;
    };
}

someArray.sort(generateSortFn('name', true));
```

sorts ascending by default.

answered Jun 26 '16 at 9:10

Ravshan Samandarov
604 8 13

This solution makes for clean code. Works great. – Todd Moses Nov 30 '16 at 18:14

1 Slightly changed version for sorting by multiple fields is here if needed: stackoverflow.com/questions/6913512/… – Ravshan Samandarov Dec 1 '16 at 12:37

10

A simple way:

```
objs.sort(function(a,b) {
  return b.last_nom.toLowerCase() < a.last_nom.toLowerCase();
});
```

See that `'.toLowerCase()'` is necessary to prevent erros in comparing strings.

answered Jan 15 '16 at 13:32

Caio Ladislau
524 5 22

4 You could use arrow functions to let the code a little more elegant: `objs.sort( (a,b) => b.last_nom.toLowerCase() < a.last_nom.toLowerCase() );` – Sertage May 24 '17 at 15:04

This is wrong for the same reason as explained here. – Patrick Roberts Jul 18 '18 at 9:24

2 Arrow functions are not ES5-worthy. Tons of engines still are restricted to ES5. In my case, I find the answer above significantly better since I'm on an ES5 engine (forced by my company) – dylanh724 Jul 21 '18 at 8:50

9

additional desc params for **Ege Özcan** code

```
function dynamicSort(property, desc) {
    if (desc) {
        return function (a, b) {
            return (a[property] > b[property]) ? -1 : (a[property] < b[property]) ? 1 :
0;
        }
    }
    return function (a, b) {
        return (a[property] < b[property]) ? -1 : (a[property] > b[property]) ? 1 : 0;
    }
}
```

answered Sep 16 '12 at 20:22

Behnam Yousefi
798 1 11 13

9

Combining Ege's dynamic solution with Vinay's idea, you get a nice robust solution:

```
Array.prototype.sortBy = function() {
    function _sortByAttr(attr) {
        var sortOrder = 1;
        if (attr[0] == "-") {
            sortOrder = -1;
            attr = attr.substr(1);
```

```
        }
        return function(a, b) {
            var result = (a[attr] < b[attr]) ? -1 : (a[attr] > b[attr]) ? 1 : 0;
            return result * sortOrder;
        }
    }
    function _getSortFunc() {
        if (arguments.length == 0) {
            throw "Zero length arguments not allowed for Array.sortBy()";
        }
        var args = arguments;
        return function(a, b) {
            for (var result = 0, i = 0; result == 0 && i < args.length; i++) {
                result = _sortByAttr(args[i])(a, b);
            }
            return result;
        }
    }
    return this.sort(_getSortFunc.apply(null, arguments));
}
```

Usage:

```
// Utility for printing objects
Array.prototype.print = function(title) {

console.log("*********************************************************************");
    console.log("**** "+title);

console.log("*********************************************************************");
    for (var i = 0; i < this.length; i++) {
        console.log("Name: "+this[i].FirstName, this[i].LastName, "Age: "+this[i].Age);
    }
}

// Setup sample data
var arrObj = [
    {FirstName: "Zach", LastName: "Emergency", Age: 35},
    {FirstName: "Nancy", LastName: "Nurse", Age: 27},
    {FirstName: "Ethel", LastName: "Emergency", Age: 42},
    {FirstName: "Nina", LastName: "Nurse", Age: 48},
    {FirstName: "Anthony", LastName: "Emergency", Age: 44},
    {FirstName: "Nina", LastName: "Nurse", Age: 32},
    {FirstName: "Ed", LastName: "Emergency", Age: 28},
    {FirstName: "Peter", LastName: "Physician", Age: 58},
    {FirstName: "Al", LastName: "Emergency", Age: 51},
    {FirstName: "Ruth", LastName: "Registration", Age: 62},
```

```
    {FirstName: "Ed", LastName: "Emergency", Age: 38},
    {FirstName: "Tammy", LastName: "Triage", Age: 29},
    {FirstName: "Alan", LastName: "Emergency", Age: 60},
    {FirstName: "Nina", LastName: "Nurse", Age: 54}
];

//Unit Tests
arrObj.sortBy("LastName").print("LastName Ascending");
arrObj.sortBy("-LastName").print("LastName Descending");
arrObj.sortBy("LastName", "FirstName", "-Age").print("LastName Ascending, FirstName
Ascending, Age Descending");
arrObj.sortBy("-FirstName", "Age").print("FirstName Descending, Age Ascending");
arrObj.sortBy("-Age").print("Age Descending");
```

answered Apr 23 '13 at 16:07

Mike R
123 1 6

1 Thanks for the idea! By the way, please do not encourage people to change the Array Prototype (see the warning at the end of my example). – Ege Özcan May 10 '13 at 14:51

9

A simple function that sort an array of object by a property

```
function sortArray(array, property, direction) {
    direction = direction || 1;
    array.sort(function compare(a, b) {
        let comparison = 0;
        if (a[property] > b[property]) {
            comparison = 1 * direction;
        } else if (a[property] < b[property]) {
            comparison = -1 * direction;
        }
        return comparison;
    });
    return array; // Chainable
}
```

Usage:

```
var objs = [
    { first_nom: 'Lazslo', last_nom: 'Jamf'     },
```

```
    { first_nom: 'Pig',    last_nom: 'Bodine'   },
    { first_nom: 'Pirate', last_nom: 'Prentice' }
];

sortArray(objs, "last_nom"); // Asc
sortArray(objs, "last_nom", -1); // Desc
```

answered May 28 '18 at 19:54

Francois Girard
165 1 10

8

Acording your example, you need to sort by two fields (last name, first name), rather then one. You can use Alasql library to make this sort in one line:

```
var res = alasql('SELECT * FROM ? ORDER BY last_nom, first_nom',[objs]);
```

Try this example at jsFiddle.

answered Dec 18 '14 at 11:09

agershun
3,106 22 35

8

```
objs.sort(function(a,b){return b.last_nom>a.last_nom})
```

answered Mar 8 '16 at 9:51

Roshni Bokade
263 2 7

1 Actually it didn't seem to work, had to use the accepted answer. It wasn't sorting correctly. – madprops Feb 21 '17 at 11:15

You may need to convert them to the lower case in order to prevent from confusion.

7

```
objs.sort(function (a,b) {

var nameA=a.last_nom.toLowerCase(), nameB=b.last_nom.toLowerCase()

if (nameA < nameB)
  return -1;
if (nameA > nameB)
  return 1;
return 0;  //no sorting

})
```

answered Aug 14 '13 at 10:40

Burak Keceli

768 1 12 25

7

```
function compare(propName) {
    return function(a,b) {
        if (a[propName] < b[propName])
            return -1;
        if (a[propName] > b[propName])
            return 1;
        return 0;
    };
}

objs.sort(compare("last_nom"));
```

answered Oct 29 '15 at 13:09

Evgenii

2,012 2 19 26

1 Please consider editing your post to add more explanation about what your code does and why it will solve the problem. An answer that mostly just contains code (even if it's working) usually wont help the OP to understand their problem. – Drenmi Oct 29 '15 at 18:16

Given the original example:

7

```
var objs = [
    { first_nom: 'Lazslo', last_nom: 'Jamf'     },
    { first_nom: 'Pig',    last_nom: 'Bodine'   },
    { first_nom: 'Pirate', last_nom: 'Prentice' }
];
```

Sort by multiple fields:

```
objs.sort(function(left, right) {
    var last_nom_order = left.last_nom.localeCompare(right.last_nom);
    var first_nom_order = left.first_nom.localeCompare(right.first_nom);
    return last_nom_order || first_nom_order;
});
```

Notes

- `a.localeCompare(b)` is universally supported and returns -1,0,1 if `a<b` , `a==b` , `a>b` respectively.
- `||` in the last line gives `last_nom` priority over `first_nom` .
- Subtraction works on numeric fields: `var age_order = left.age - right.age;`
- Negate to reverse order, `return -last_nom_order || -first_nom_order || -age_order;`

answered Feb 24 '18 at 0:54

Bob Stein
8,229 4 54 77

6

This is a simple problem, don't know why people have such complex solution.
A simple sort function (based on **quick-sort** algorithm):

```
function sortObjectsArray(objectsArray, sortKey)
        {
            // Quick Sort:
            var retVal;

            if (1 < objectsArray.length)
            {
                var pivotIndex = Math.floor((objectsArray.length - 1) / 2);  // middle
index
                var pivotItem = objectsArray[pivotIndex];                    // value in
```

```
the middle index
                var less = [], more = [];

                objectsArray.splice(pivotIndex, 1);                         // remove
the item in the pivot position
                objectsArray.forEach(function(value, index, array)
                {
                    value[sortKey] <= pivotItem[sortKey] ?                  // compare
the 'sortKey' proiperty
                            less.push(value) :
                            more.push(value) ;
                });

                retVal = sortObjectsArray(less, sortKey).concat([pivotItem],
sortObjectsArray(more, sortKey));
            }
            else
            {
                retVal = objectsArray;
            }

            return retVal;
        }
```

Use example:

```
var myArr =
        [
            { val: 'x', idx: 3 },
            { val: 'y', idx: 2 },
            { val: 'z', idx: 5 },
        ];
myArr = sortObjectsArray(myArr, 'idx');
```

answered Nov 19 '15 at 14:27

Gil Epshtain

2,356 4 26 47

5 How is implementing quick sort in js a simple solution? Simple algorithm but not a simple solution. – Andrew Nov 23 '15 at 22:46

It simple since it don't use any outer libraries and it don't change the object's prototype. In my opinion, the length of the code don't have direct impact on the code's complexity – Gil Epshtain Nov 24 '15 at 12:02

2 Well, let me try with different words: How reinventing the wheel is a simple solution? – Roberto14 Dec 9 '15 at 17:36

6

Using Ramda,

npm install ramda

```
import R from 'ramda'
var objs = [
    { first_nom: 'Lazslo', last_nom: 'Jamf'     },
    { first_nom: 'Pig',    last_nom: 'Bodine'   },
    { first_nom: 'Pirate', last_nom: 'Prentice' }
];
var ascendingSortedObjs = R.sortBy(R.prop('last_nom'), objs)
var descendingSortedObjs = R.reverse(ascendingSortedObjs)
```

answered Jul 5 '17 at 7:43

Sridhar Sg
724 7 16

5

I Just enhanced Ege Özcan's dynamic sort to dive deep inside objects. If Data looks like this:

```
obj = [
    {
        a: { a: 1, b: 2, c: 3 },
        b: { a: 4, b: 5, c: 6 }
    },
    {
        a: { a: 3, b: 2, c: 1 },
        b: { a: 6, b: 5, c: 4 }
}];
```

and if you want to sort it over **a.a** property I think my enhancement helps very well. I add new functionality to objects like this:

```
Object.defineProperty(Object.prototype, 'deepVal', {
    enumerable: false,
    writable: true,
    value: function (propertyChain) {
        var levels = propertyChain.split('.');
        parent = this;
        for (var i = 0; i < levels.length; i++) {
```

```
            if (!parent[levels[i]])
                return undefined;
            parent = parent[levels[i]];
        }
        return parent;
    }
});
```

and changed **_dynamicSort**'s *return* function:

```
return function (a,b) {
        var result = ((a.deepVal(property) > b.deepVal(property)) - (a.deepVal(property)
< b.deepVal(property)));
        return result * sortOrder;
    }
```

And now you can sort by **a.a.** this way:

```
obj.sortBy('a.a');
```

See Commplete script in JSFiddle

edited May 23 '17 at 12:03
Community ♦
1 1

answered Jun 29 '15 at 1:10
Morteza Tourani
2,958 4 29 40

protected by Pankaj Parkar Oct 16 '15 at 12:16

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?