How to sort an associative array by its values in Javascript?

Asked 13 years, 9 months ago Modified 2 years, 3 months ago Viewed 128k times

89

I have the associative array:

```
array["sub2"] = 1;
array["sub0"] = -1;
array["sub1"] = 0;
array["sub3"] = 1;
array["sub4"] = 0;
```

What is the most elegant way to sort (descending) by its values where the result would be an array with the respective indices in this order:

```
sub2, sub3, sub1, sub4, sub0
```

javascript arrays sorting associative

Share
Improve this question
Follow

edited Oct 4, 2020 at 14:19
Penny Liu
17.2k ●5 ●86 ●108

asked Mar 4, 2011 at 22:17
John Smith
8,741 ●14 ●53 ●78

1 Since object properties have no language defined order - you can't (except, perhaps, in some JS engines by depending on the particular way that they implemented properties). – Quentin Mar 4, 2011 at 22:23

11 Answers

Sorted by: Highest score (default)

119

✓

Javascript doesn't have "associative arrays" the way you're thinking of them. Instead, you simply have the ability to set object properties using array-like syntax (as in your example), plus the ability to iterate over an object's properties.

The upshot of this is that there is no guarantee as to the *order* in which you iterate over the properties, so there is nothing like a sort for them. Instead, you'll need to convert your object properties into a "true" array (which does guarantee order). Here's a code snippet for converting an object into an array of two-tuples (two-element arrays), sorting it as you describe, then iterating over it:

```
var tuples = [];

for (var key in obj) tuples.push([key, obj[key]]);

tuples.sort(function(a, b) {
    a = a[1];
    b = b[1];

    return a < b ? -1 : (a > b ? 1 : 0);
});

for (var i = 0; i < tuples.length; i++) {
    var key = tuples[i][0];
    var value = tuples[i][1];

    // do something with key and value
}
```

You may find it more natural to wrap this in a function which takes a callback:

```
function bySortedValue(obj, callback, context) {
  var tuples = [];

  for (var key in obj) tuples.push([key, obj[key]]);

  tuples.sort(function(a, b) {
    return a[1] < b[1] ? 1 : a[1] > b[1] ? -1 : 0
  });

  var length = tuples.length;
  while (length--) callback.call(context, tuples[length][0], tuples[length]
[1]);
}

bySortedValue({
  foo: 1,
  bar: 7,
  baz: 3
}, function(key, value) {
  document.getElementById('res').innerHTML += `${key}: ${value}<br>`
});
```

```
<p id='res'>Result:<br/><br/><p>
```

Run code snippet   Expand snippet

Share

Improve this answer

Follow

edited Aug 4, 2017 at 5:30
bb216b3acfd8f72cbc8f8
99d4d6963
763 ● 11 ● 21

answered Mar 4, 2011 at 22:31
Ben Blank
56.5k ● 28 ● 131 ● 163

2 the tuples.sort function can be cleaned up to tuples.sort(function(a, b) { return a[1] - b[1]; });
– stot Nov 30, 2012 at 17:26

2 @stot — If your values are all numbers (as in the asker's example), absolutely. I appear to have absent-mindedly provided a comparison function which works with strings as well. :-)
– Ben Blank Dec 6, 2012 at 17:52

@Ben: yes you are right, the version provided is more general ;-) – stot Dec 9, 2012 at 11:39

@Ben, thank you very much for this posting. Concerning the String comparison function, does it use an ASCII value comparison? – jjwdesign Jun 7, 2013 at 1:48

@jjwdesign, assuming the string hasn't been encoded, it will sort by Unicode code point.
– Ben Blank Jun 11, 2013 at 15:33

90

Instead of correcting you on the semantics of an 'associative array', I think this is what you want:

```
function getSortedKeys(obj) {
    var keys = Object.keys(obj);
    return keys.sort(function(a,b){return obj[b]-obj[a]});
}
```

for really old browsers, use this instead:

```
function getSortedKeys(obj) {
    var keys = []; for(var key in obj) keys.push(key);
    return keys.sort(function(a,b){return obj[b]-obj[a]});
}
```

You dump in an object (like yours) and get an array of the keys - eh properties - back, sorted descending by the (numerical) value of the, eh, values of the, eh, object.

This only works if your values are numerical. Tweek the little `function(a,b)` in there to change the sorting mechanism to work ascending, or work for `string` values (for example). Left as an exercise for the reader.

Share
Improve this answer
Follow

edited Jul 19, 2021 at 15:05
Keith Hughitt
4,960 ● 5 ● 51 ● 55

answered Aug 4, 2012 at 20:25
commonpike
11.1k ● 5 ● 67 ● 64

2 I should note that most browsers nowadays just support Object.keys() - developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – commonpike Sep 2, 2015 at 19:46

why is object.keys any better? – john k Jun 14, 2018 at 17:17

1 @johnktejik this answer was from 2012 :-) Object.keys() is a standard nowadays, and its shorter and probably quicker. – commonpike Jun 15, 2018 at 11:54

2 ... :) I was tired, let me delete this – manu Sep 4, 2018 at 8:47

21

Continued discussion & other solutions covered at [How to sort an (associative) array by value?](#) with the best solution (for my case) being by [saml](#) (quoted below).

Arrays can only have numeric indexes. You'd need to rewrite this as either an Object, or an Array of Objects.

```
var status = new Array();
status.push({name: 'BOB', val: 10});
status.push({name: 'TOM', val: 3});
status.push({name: 'ROB', val: 22});
status.push({name: 'JON', val: 7});
```

If you like the `status.push` method, you can sort it with:

```
status.sort(function(a,b) {
    return a.val - b.val;
});
```

Share

Improve this answer

Follow

edited May 23, 2017 at 11:55
Community Bot
1 ● 1

answered Oct 8, 2012 at 23:27
PopeJohnPaulII
775 ● 2 ● 7 ● 10

Excellent! To do and alpha sort on name: return a.name > b.name. – mpemburn Mar 20, 2014 at 19:38

2 For the alphabetic sort, compare the string values in the same case because `sort()` treats them differently. It messed me up for an hour until I found this [stackoverflow.com/questions/6712034/…](#) Example; `a.name.toLowerCase() > b.name.toLowerCase()` – Dylan Valade Oct 10, 2016 at 20:39

5

There really isn't any such thing as an "associative array" in JavaScript. What you've got there is just a plain old object. They work kind-of like associative arrays, of course, and the keys are available but there's no semantics around the order of keys.

You could turn your object into an array of objects (key/value pairs) and sort that:

```
function sortObj(object, sortFunc) {
  var rv = [];
  for (var k in object) {
    if (object.hasOwnProperty(k)) rv.push({key: k, value:  object[k]});
  }
  rv.sort(function(o1, o2) {
    return sortFunc(o1.key, o2.key);
  });
  return rv;
}
```

Then you'd call that with a comparator function.

Share Improve this answer Follow

answered Mar 4, 2011 at 22:30

Pointy
413k ●61 ●594 ●625

+1 you beat me to it. I was writing a very similar explanation and piece of code. – gonchuki
Mar 4, 2011 at 22:32

5

The best approach for the specific case here, in my opinion, is the one *commonpike* suggested. A little improvement I'd suggest that works in modern browsers is:

```
// aao is the "associative array" you need to "sort"
Object.keys(aao).sort(function(a,b){return aao[b]-aao[a]});
```

This could apply easily and work great in the specific case here so you can do:

```
let aoo={};
aao["sub2"]=1;
aao["sub0"]=-1;
aao["sub1"]=0;
aao["sub3"]=1;
aao["sub4"]=0;

let sk=Object.keys(aao).sort(function(a,b){return aao[b]-aao[a]});

// now you can loop using the sorted keys in `sk` to do stuffs
for (let i=sk.length-1;i>=0;--i){
 // do something with sk[i] or aoo[sk[i]]
}
```

Besides of this, I provide here a more "generic" function you can use to sort even in wider range of situations and that mixes the improvement I just suggested with the approaches of the answers by *Ben Blank* (sorting also string values) and *PopeJohnPaulII* (sorting by specific object field/property) and lets you decide if you want an ascendant or descendant order, here it is:

```
// aao := is the "associative array" you need to "sort"
// comp := is the "field" you want to compare or "" if you have no "fields" and
simply need to compare values
// intVal := must be false if you need comparing non-integer values
// desc := set to true will sort keys in descendant order (default sort order
is ascendant)
function sortedKeys(aao,comp="",intVal=false,desc=false){
  let keys=Object.keys(aao);
  if (comp!="") {
    if (intVal) {
      if (desc) return keys.sort(function(a,b){return aao[b][comp]-aao[a]
[comp]});
      else return keys.sort(function(a,b){return aao[a][comp]-aao[a][comp]});
    } else {
      if (desc) return keys.sort(function(a,b){return aao[b][comp]<aao[a]
[comp]?1:aao[b][comp]>aao[a][comp]?-1:0});
      else return keys.sort(function(a,b){return aao[a][comp]<aao[b][comp]?
1:aao[a][comp]>aao[b][comp]?-1:0});
    }
  } else {
    if (intVal) {
      if (desc) return keys.sort(function(a,b){return aao[b]-aao[a]});
      else return keys.sort(function(a,b){return aao[a]-aao[b]});
    } else {
      if (desc) return keys.sort(function(a,b){return aao[b]<aao[a]?
1:aao[b]>aao[a]?-1:0});
      else return keys.sort(function(a,b){return aao[a]<aao[b]?1:aao[a]>aao[b]?
-1:0});
    }
  }
}
```

You can test the functionalities trying something like the following code:

```
let items={};
items['Edward']=21;
items['Sharpe']=37;
items['And']=45;
items['The']=-12;
items['Magnetic']=13;
items['Zeros']=37;
//equivalent to:
//let items={"Edward": 21, "Sharpe": 37, "And": 45, "The": -12, ...};

console.log("1: "+sortedKeys(items));
console.log("2: "+sortedKeys(items,"",false,true));
console.log("3: "+sortedKeys(items,"",true,false));
console.log("4: "+sortedKeys(items,"",true,true));
/* OUTPUT
1: And,Sharpe,Zeros,Edward,Magnetic,The
2: The,Magnetic,Edward,Sharpe,Zeros,And
3: The,Magnetic,Edward,Sharpe,Zeros,And
4: And,Sharpe,Zeros,Edward,Magnetic,The
*/

items={};
items['k1']={name:'Edward',value:21};
items['k2']={name:'Sharpe',value:37};
items['k3']={name:'And',value:45};
items['k4']={name:'The',value:-12};
items['k5']={name:'Magnetic',value:13};
items['k6']={name:'Zeros',value:37};
```

```
console.log("1: "+sortedKeys(items,"name"));
console.log("2: "+sortedKeys(items,"name",false,true));
/* OUTPUT
1: k6,k4,k2,k5,k1,k3
2: k3,k1,k5,k2,k4,k6
*/
```

As I already said, you can loop over sorted keys if you need doing stuffs

```
let sk=sortedKeys(aoo);
// now you can loop using the sorted keys in `sk` to do stuffs
for (let i=sk.length-1;i>=0;--i){
 // do something with sk[i] or aoo[sk[i]]
}
```

Last, but not least, some useful references to Object.keys and Array.sort

Share
Improve this answer
Follow

edited Jun 30, 2018 at 16:53

answered Jun 30, 2018 at 16:20
danicotra
1,433 ● 2 ● 18 ● 34

4

Here is a variation of ben blank's answer, if you don't like tuples.

This saves you a few characters.

```
var keys = [];
for (var key in sortme) {
  keys.push(key);
}

keys.sort(function(k0, k1) {
  var a = sortme[k0];
  var b = sortme[k1];
  return a < b ? -1 : (a > b ? 1 : 0);
});

for (var i = 0; i < keys.length; ++i) {
  var key = keys[i];
  var value = sortme[key];
  // Do something with key and value.
}
```

Share
Improve this answer
Follow

edited Dec 18, 2014 at 20:50
Eric Leschinski
153k ● 96 ● 421 ● 335

answered May 2, 2013 at 18:21
donquixote
5,411 ● 3 ● 33 ● 65

No unnecessary complication required...

4

```
function sortMapByValue(map)
{
    var tupleArray = [];
    for (var key in map) tupleArray.push([key, map[key]]);
    tupleArray.sort(function (a, b) { return a[1] - b[1] });
    return tupleArray;
}
```

Share Improve this answer Follow

answered Jan 10, 2015 at 5:07
bop
575 ●3 ●5

The output of this is an array of arrays, not a map – Daniel Jul 2, 2015 at 22:48

`var stuff = {"a":1 , "b":3 , "c":0 } sortMapByValue(stuff) [Array[2], Array[2], Array[2]]` – Daniel Jul 2, 2015 at 22:49

I think this is cleanest. Just add a conversion to a map at the end – Daniel Jul 2, 2015 at 22:59

1

i use $.each of jquery but you can make it with a for loop, an improvement is this:

```
        //.ArraySort(array)
        /* Sort an array
         */
        ArraySort = function(array, sortFunc){
              var tmp = [];
              var aSorted=[];
              var oSorted={};

              for (var k in array) {
                if (array.hasOwnProperty(k))
                    tmp.push({key: k, value:  array[k]});
              }

              tmp.sort(function(o1, o2) {
                    return sortFunc(o1.value, o2.value);
              });

              if(Object.prototype.toString.call(array) === '[object Array]'){
                  $.each(tmp, function(index, value){
                      aSorted.push(value.value);
                  });
                  return aSorted;
              }

              if(Object.prototype.toString.call(array) === '[object Object]'){
                  $.each(tmp, function(index, value){
                      oSorted[value.key]=value.value;
                  });
                  return oSorted;
              }
     };
```

So now you can do

```
    console.log("ArraySort");
    var arr1 = [4,3,6,1,2,8,5,9,9];
    var arr2 = {'a':4, 'b':3, 'c':6, 'd':1, 'e':2, 'f':8, 'g':5, 'h':9};
    var arr3 = {a: 'green', b: 'brown', c: 'blue', d: 'red'};
    var result1 = ArraySort(arr1, function(a,b){return a-b});
    var result2 = ArraySort(arr2, function(a,b){return a-b});
    var result3 = ArraySort(arr3, function(a,b){return a>b});
    console.log(result1);
    console.log(result2);
    console.log(result3);
```

Share Improve this answer Follow

answered Jul 19, 2012 at 13:22

demosthenes

1,191 ● 1 ● 13 ● 18

0

Just so it's out there and someone is looking for tuple based sorts. This will compare the first element of the object in array, than the second element and so on. i.e in the example below, it will compare first by "a", then by "b" and so on.

```
let arr = [
    {a:1, b:2, c:3},
    {a:3, b:5, c:1},
    {a:2, b:3, c:9},
    {a:2, b:5, c:9},
    {a:2, b:3, c:10}
]

function getSortedScore(obj) {
    var keys = [];
    for(var key in obj[0]) keys.push(key);
    return obj.sort(function(a,b){
        for (var i in keys) {
            let k = keys[i];
            if (a[k]-b[k] > 0) return -1;
            else if (a[k]-b[k] < 0) return 1;
            else continue;
        };
    });
}

console.log(getSortedScore(arr))
```

OUPUTS

```
 [ { a: 3, b: 5, c: 1 },
  { a: 2, b: 5, c: 9 },
  { a: 2, b: 3, c: 10 },
  { a: 2, b: 3, c: 9 },
  { a: 1, b: 2, c: 3 } ]
```

Share Improve this answer Follow

answered Jun 27, 2017 at 10:45
Shayan C
1,580 ● 1 ● 13 ● 21

0

A modern approuch to this:

```
Object.fromEntries(Object.entries(data).sort((a,b)=>b[1]-a[1]).slice(0,5))
```

P.S: I did an optional slice, you can remove it if you want.

Share Improve this answer Follow

answered Sep 17, 2022 at 19:18
luizjr8
1

-4

@commonpike's answer is "the right one", but as he goes on to comment...

> most browsers nowadays just support `Object.keys()`

Yeah.. `Object.keys()` is *WAY better*.

But what's ***even better***? Duh, it's it in `coffeescript`!

```
sortedKeys = (x) -> Object.keys(x).sort (a,b) -> x[a] - x[b]

sortedKeys
  'a' :  1
  'b' :  3
  'c' :  4
  'd' : -1
```

> `[ 'd', 'a', 'b', 'c' ]`

Share Improve this answer Follow

answered Nov 3, 2015 at 3:00
Alex Gray
16.5k ● 9 ● 98 ● 121