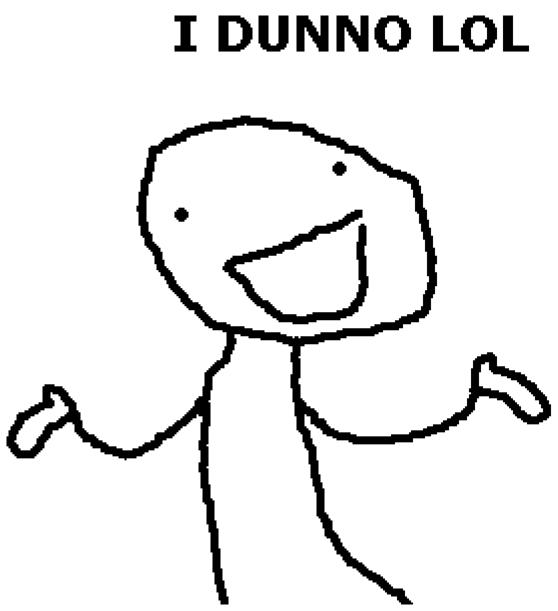
Is Javascript Array.sort() Stable?

Filip Sufitchi Follow
Jan 6, 2017 · 2 min read



I don't mean "stable" as in "does it crash all the time". Javascript's implementations are (hopefully) more reliable than that. I mean stable in the algorithmic sense: *does it preserve the order of otherwise "equal" items*? Consider the following test case:

```
var people = [
  {name: 'Gary', age: 20},
  {name: 'Ann', age: 20},
  {name: 'Bob', age: 17},
  {name: 'Sue', age: 21},
  {name: 'Sam', age: 17},
];

// Sort people by name
people.sort( (p1, p2) => {
  if (p1.name < p2.name) return -1;
  if (p1.name > p2.name) return 1;
  return 0;
});

console.log(people.map(p => p.name));
// ['Ann', 'Bob', 'Gary', 'Sam', 'Sue']

// Re-sort people by age
people.sort( (p1, p2) => {
  if (p1.age < p2.age) return -1;
  if (p1.age > p2.age) return 1;
  return 0;
});

console.log(people.map(p => p.name));
// We're expecting people sorted by age, then by name within age group:
// ['Bob', 'Sam', 'Ann', 'Gary', 'Sue']

// But we might get any of these instead, depending on the browser:
// ['Sam', 'Bob', 'Ann', 'Gary', 'Sue']
// ['Bob', 'Sam', 'Gary', 'Ann', 'Sue']
// ['Sam', 'Bob', 'Gary', 'Ann', 'Sue']
```

js_idunno_sort.js hosted with ♡ by GitHub   view raw

What gives? Well, if you're getting one of the last three results, then you're probably using Google Chrome, or maybe one of an assortment of browsers that does not implement Array.sort() as a "stable" algorithm.

Why, you ask? Well, nowhere in the ES7 specification does it say whether the sort needs to be stable or not. In fact, it tries to be super abstract, allowing as much of the sort to be "implementation defined" as possible. This has resulted in different JS engines (across different browsers) taking different routes to implementing this spec, leading to inconsistent sort stability behavior.

But I need Javascript to have a stable sort!

That's too bad. There is no stable sort in the standard library. You're going to either need to use a 3rd party library, or implement it yourself.

For a 3rd party library, I heartily recommend Lodash, which has great browser-agnostic utility functions for doing a variety of tasks, with a focus on functional programming. In particular, its Collection.sortBy() is stable!

```
people = _.sortBy(people, [person => person.name]);
console.log(people.map(p => p.name));
// ['Ann', 'Bob', 'Gary', 'Sam', 'Sue']

people = _.sortBy(people, [person => person.age]);
console.log(people.map(p => p.name));
// ['Bob', 'Sam', 'Ann', 'Gary', 'Sue']
```

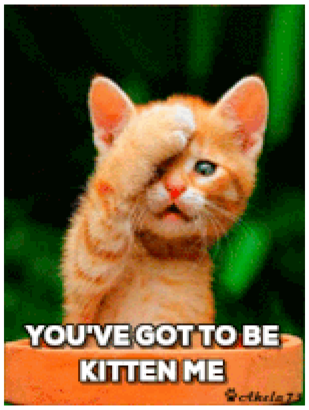
stable_sort_lodash.js hosted with ♡ by GitHub view raw

If, for some reason, you prefer the classic JS sort() semantics (in-place, using a comparator instead of a "key" access), or if you are allergic to using 3rd party libraries, then you could easily implement that yourself by wrapping around Javascript's default sort and adding your new function to the Array prototype:

```
Array.prototype.stableSort = function(cmp) {
  cmp = !!cmp ? cmp : (a, b) => {
    if (a < b) return -1;
    if (a > b) return 1;
    return 0;
  };
  let stabilizedThis = this.map((el, index) => [el, index]);
  let stableCmp = (a, b) => {
    let order = cmp(a[0], b[0]);
    if (order != 0) return order;
    return a[1] - b[1];
  }
  stabilizedThis.sort(stableCmp);
  for (let i=0; i<this.length; i++) {
    this[i] = stabilizedThis[i][0];
  }
  return this;
}

// ----------

people.stableSort( (p1, p2) => {
  if (p1.name < p2.name) return -1;
  if (p1.name > p2.name) return 1;
  return 0;
});

console.log(people.map(p => p.name));
// ['Ann', 'Bob', 'Gary', 'Sam', 'Sue']

people.stableSort( (p1, p2) => {
  if (p1.age < p2.age) return -1;
  if (p1.age > p2.age) return 1;
  return 0;
});

console.log(people.map(p => p.name));
// ['Bob', 'Sam', 'Ann', 'Gary', 'Sue']
```

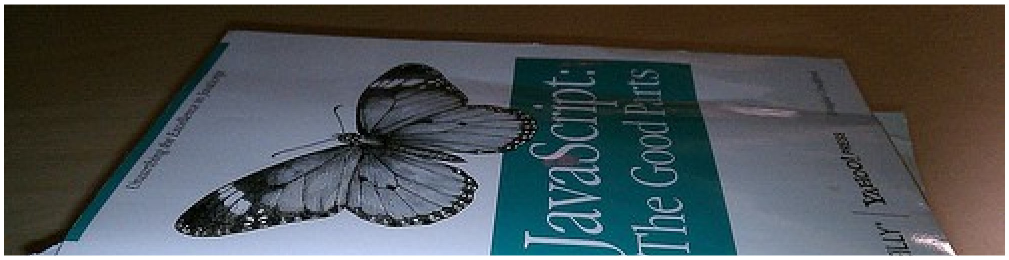
stable_sort_custom.js hosted with ♡ by GitHub view raw

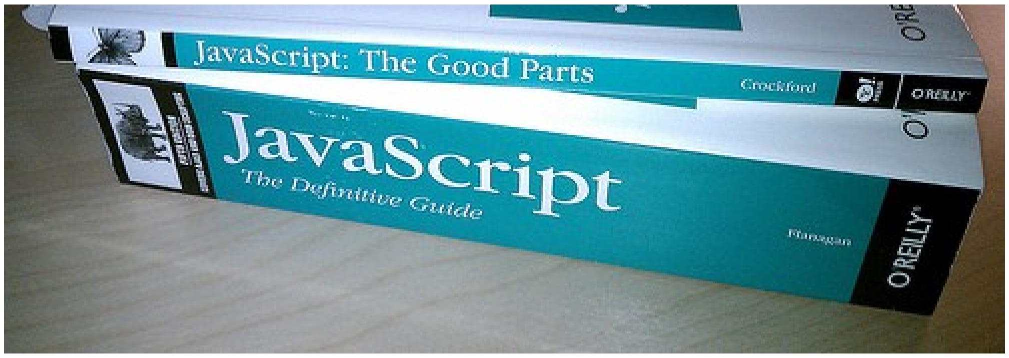
> *If you're looking for a type-safe Typescript "shim", I got that too.*



Yeah, it's terrible. You could write your own sort implementation that's stable from the get-go and avoids some of this mess! Try merge sort! It may be slower than JS's built-in sort, though, given that the latter is likely implemented in the JS engine itself (in C/C++ bytecode).

Hopefully this little rant will have saved you a headache about your data not sorting itself right. If not, well… this probably belongs in the bottom book, not the top one:





JavaScript Functional Programming