I'm not a bot

Stable algorithm examples

As of ES2019, sort is required to be stable. In ECMAScript 1st edition through ES2018, it was allowed to be unstable. Simple test case (ignore the heading, second set of numbers should be sequential if the engine's sort is stable). Note: This test case doesn't work for some versions of Chrome (technically, of V8) that switched sorting algorithms based on the size of the array, using a stable sort for small arrays but an unstable one for larger arrays. (Details.) See the end of the question for a modified version that makes the array large enough to trigger the behavior. IE's sort has been stable as long as I've ever used it (so IE6). Checking again in IE8 and it appears to still be the case. And although that Mozilla page you link to says Firefox's sort is stable, I definitely say this was not always the case prior to (and including) Firefox 2.0. Some cursory results: IE6+: stable Firefox < 3: unstable Firefox >= 3: stable Chrome < 70: unstable Chrome >= 70: stable Opera < 10: unstable Opera >= 10: stable Safari 4: stable Edge: unstable for long arrays (>512 elements) All tests on Windows. See also: Fast stable sorting algorithm implementation in javascript This test case (modified from here) will demonstrate the problem in V8 (for instance, Node v6, Chrome < v70) by ensuring the array has enough entries to pick the "more efficient" sort method; this is written with very old JavaScript engines in mind, so without modern features: function Pair(_x, _y) { this.x = _x; this.y = _y; } function pairSort(a, b) { return a.x - b.x; } var y = 0; var check = []; while (check.length < 100) { check.push(new Pair(Math.floor(Math.random() * 3) + 1, ++y)); } check.sort(pairSort); var min = {}; var issues = 0; for (var i = 0; i < check.length; ++i) { var entry = check[i]; var found = min[entry.x]; if (found) { if (found.y > entry.y) { console.log("Unstable at " + found.i + ": " + found.y + " > " + entry.y); ++issues; } } else { min[entry.x] = {x: entry.x, y: entry.y, i: i}; } } if (!issues) { console.log("Sort appears to be stable"); } I think that an example with sorting a struct, rather than a list of integers, is helpful to clarify what the difference is between both. Imagine a list of neighbours in a building, that you construct ordered by the floor where they live. #include #include #include struct Neighbour { int floor; string name; Neighbour(int f, string n) : floor(f), name(n) {} }; std::vector vec = {Neighbour(1,"Bob"), Neighbour(2,"Anna"), Neighbour(3,"Peter"), Neighbour(4,"Bob"), Neighbour(5,"Laura")}; 1 Bob 2 Anna 3 Peter 4 Bob 5 Laura If you now want to sort your list alphabetically, and you use std::sort(vec.begin(), vec.end(), [](Neighbour a, Neighbour b){ return a.name < b.name; }); for(auto v : vec) std::cout