Why Selection sort can be stable or unstable

Asked 9 years, 4 months ago Modified 26 days ago Viewed 42k times

29

I know that `selection sort` can be implemented as stable or unstable. But I wonder how it can be. I think sort algorithm can be only stable or only unstable. Can someone explain?

arrays algorithm sorting

Share Improve this question Follow

asked Dec 24, 2013 at 12:58

fortegente

1,351 2 11 22

6 Answers

Sorted by:

Highest score (default)

79

Basically in `selection sort` , swap that occurs at the end of each "round" can change the relative order of items having the same value.

for example, suppose you sorted `4 2 3 4 1` with `selection sort` .

the first "round" will go through each element looking for the minimum element. it will find that 1 is the minimum element. then it will swap the 1 into the first spot. this will cause the 4 in the first spot to go into the last spot: `1 2 3 4 4`

now the relative order of the 4's has changed. the "first" 4 in the original list has been moved to a spot after the other 4.

remember the definition of stable is that

> the relative order of elements with the same value is maintained.

Well, `selection sort` works by *finding the 'least' value in a set of values, then swap it with the first value*:

> Code:
>
> 2, 3, 1, 1 # scan 0 to n and find 'least' value
>
> 1, 3, 2, 1 # swap 'least' with element 0.
>
> 1, 3, 2, 1 # scan 1 to n and find 'least' value
>
> 1, 1, 2, 3 # swap 'least' with element 1.

...and so on until it is sorted.

To make this stable, instead of swaping values, insert the 'least' value instead:

Code:

2, 3, 1, 1 # scan 0 to n and find 'least' value

1, 2, 3, 1 # insert 'least' at pos 0, pushing other elements back.

1, 3, 2, 1 # scan 1 to n and find 'least' value

1, 1, 2, 3 # insert 'least' at pos 1, pushing other elements back.

...and so on until it is sorted.

It shouldn't be too hard to modify an unstable selection sort algorithm to become stable.

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 1

answered Dec 24, 2013 at 13:07

Prashant Shilimkar
8,312 13 54 89

7 Nice answer. Thanks. But if I understand correctly, a selection sorting implemented with "insert" rather than "swap", deviates from its canonical algorithm definition, so that it is not the same animal anymore. The price of being stable is the overhead of "insert" is much larger than "swap". @Alma-Do has a better explanation about this topic. – RayLuo Oct 29, 2014 at 7:06

2 good explanation.Would have been better if the second example(2,3,1,1) was unstable in the normal algorithm and was stable after the modified algo.Its stable in either way.The first example is a better one. – crackerplace Apr 29, 2017 at 19:26

26

In common case - you're not correct. Selection sorting is unstable. That comes from it's definition. So, you're obviously confused with one custom case.

It *can be stable* - normally, only with linked lists. To do that (classic way, O(1) memory) - instead of swaps, minimal element must be linked to the unsorted part, making entire algorithm stable. And this is the difference in "implementation" - which, obviously, will produce stability only in this case due to data structure specifics. It has nothing to do with common case, when selection sorting is **unstable**

Share Improve this answer Follow

answered Dec 24, 2013 at 13:10

Alma Do
36.9k 9 74 104

6

Let's say I have this array:

```
A = {3, 3, 1}
```

Starting at position 0, selection sort will search for the minimum value and swap the current element with the minimum. For `A`, we swap the first `3` with `1`, and do nothing on the second step.

This is not stable as the first `3` should have come before the second. If you use a linked list instead of an array, and insert an element in the correct position instead of swapping, selection sort is stable.

Share Improve this answer Follow

answered Dec 24, 2013 at 13:05

sebii

506 2 6

1

The fact that a sorting routing is a selection sort does not define everything about it. There are still decisions to be made that may be made differently in different implementations, and different choices may produce either a stable or an unstable sort. (Most sorts that can be stable don't have to be; usually, the interesting theoretical question is whether the sort *can* be implemented stably.)

Share Improve this answer Follow

answered Dec 24, 2013 at 13:05

user2357112

256k 28 419 494

0

- First, you have to understand that there is nothing unstable kind of thing with the Selection sort as it vividly depends upon the data structure you are using for the specific.
- For example, consider the fact that we have an array as **4 2 4 1** and upon sorting this array using the traditional method the result will be **1 2 4 4**, simple. But during the first round of the iteration, the 4 which is present at the 0th index will be placed at the last index which is ethically wrong considering the fact of stability as the 4 which is present on the 2nd index should come later than the 4 present on the 0th index. **(1 2 4[2nd index] 4[from 0th index])**
- If asked! You can make a slight modification in the implementation and instead of swapping the numbers, choose the minimum element during the first round and then place that element at its correct position and shift the whole array. I know the shifting operation might cost more than the swapping but this is the best I can think of.
- If you are concerned about the shifting operation, you may use the Linked list rather than the array and insert the appropriate element at its appropriate position in **O(1)** time.

Share Improve this answer Follow

answered May 20, 2021 at 7:22

Vipul Maheshwari

1 1

0

First of all I want to mention that I'm a student, not an expert so I may be wrong.

why is selection sort unstable?

- let *a* be an array of integers such that *a* = [0, 0, 2, 3, 4].
- if *a* is to be sorted using an unstable implementation of selection sort; **first two indexes of *a* (*a*[0] = 0 and *a*[1] = 0) will get swapped**. (because the inner loop makes sure that if *a*[*j*] <= *a*[*i*] then *j* =

smallest_index and therefore should get swapped with the current *i*. (where innerloop iterates *a*[*j* ... *len*] for each index of outerloop: *a*[*i* ... *len* - *1*])).

- if *a* is to be sorted by any stable sorting algorithm, the order of equal members (in this case, 0 and 0) will be preserved. (*a*[0] and *a*[1] won't be swapped.)

is this stable selection sort ?! (NO GUARANTEE :) !)

In your code you must have an if statement like :

```
if a[j] <= a[smallest_index]
    smallest_index = j
```

which is the unstable version.
I simply made it stable by making sure that `smallest_index` is set to `j` if and only if `a[j] < a[smallest_index]` , therefore equal values do not get swapped at all.

simply by making sure that the:

```
if a[j] < a[smallest_index]
    smallest_index = j
```

Share Improve this answer Follow

edited Apr 24 at 4:30

greybeard
2,153 7 28 63

answered Apr 23 at 16:48

no_hope
29 4

`NO GUARANTEE` indeed, especially not showing how to handle the selected index. – greybeard Apr 24 at 4:18

(You focused on the selected value getting swapped in the current iteration. How about the other participant in the swap? Consider case insensitive comparison and [Y, y, x, X] – greybeard Apr 24 at 4:35

@greybeard thank you for comment. about the selected index, the index for smallest element gets set to smallest_index. after that; on each iteration of outer loop right after the inner loop finishes, the a[smallest_index] gets swapped with a[i]. and about the non numerical array. YES YOU ARE RIGHT :), X gets selected in first iteration of outer loop and get's swapped with a[0] causing it to appear before x in final sorted a. do you think my solution works fine for any array of integers ? – no_hope Apr 24 at 10:18

(I think I know it to not work for any kind of item where *stability* / *order of same key items* matters. Otherwise (including *integers*), it avoid dispensable swaps in case of = .) – greybeard Apr 24 at 10:24

@greybeard ok, i get it. hey should i delete this answer? – no_hope Apr 24 at 11:30