Problem: Implement selection sort.

Constraints

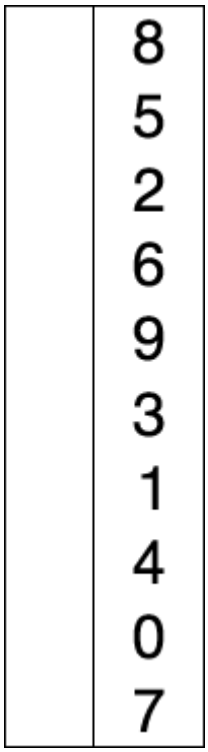
- **Is a naive solution sufficient (ie not stable, not based on a heap)?**
  - **Yes**
- **Are duplicates allowed?**
  - **Yes**
- **Can we assume the input is valid?**
  - **No**
- **Can we assume this fits memory?**
  - **Yes**

Test Cases

- **None -> Exception**
- **[] -> []**
- **One element -> [element]**
- **Two or more elements**

Algorithm

Wikipedia's animation:



We can do this recursively or iteratively. Iteratively will be more efficient as it doesn't require the extra space overhead with the recursive calls.

- **For each element**
    - **Check every element to the right to find the min**
    - **If min < current element, swap**

Complexity:

- **Time: O(n^2) average, worst, best**
- **Space: O(1) iterative, O(m) recursive where m is the recursion depth (unless tail-call elimination is available, then O(1))**
    - **Note: Tail call elimination is not inherently available in Python, see the following StackOverflow post.**

Misc:

- **In-place**
- **Most implementations are not stable, due to swapping of values**

Selection sort might be a good option if moving elements is more expensive than comparing them, as it requires at most n-1 swaps.

The finding of a minimum element can be done with a min heap, which would change the worst-case run time to O(n log(n)) and increase the space to O(n). This is called a heap sort.