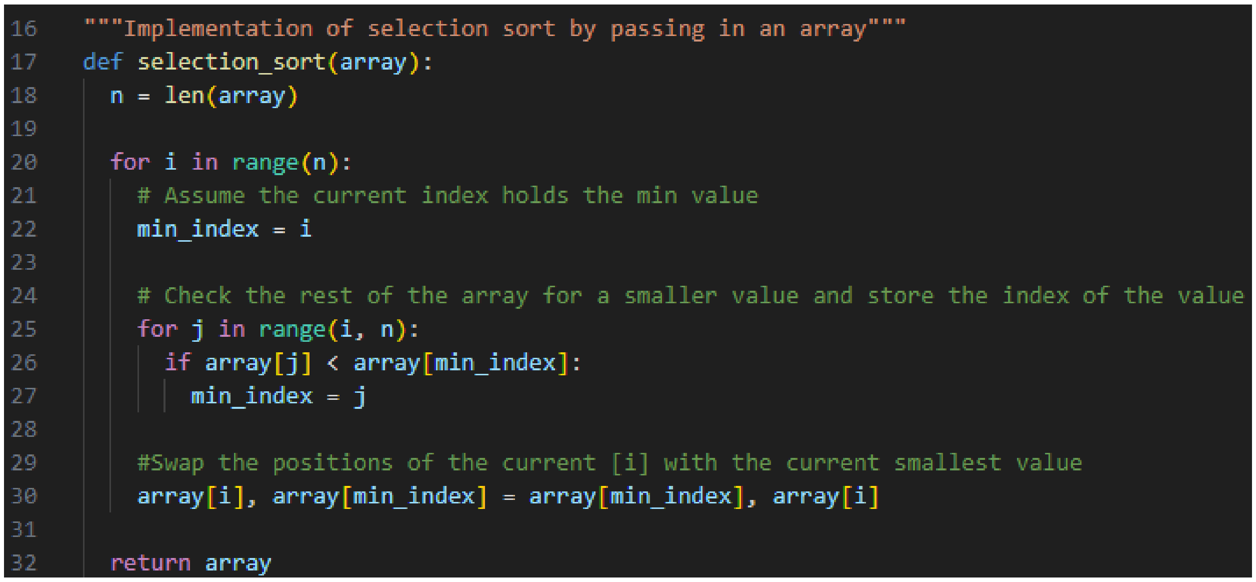
Keiffer Tan

Implementing and Analyzing Selection Sort

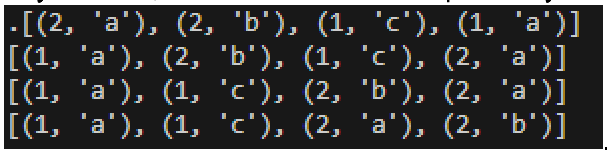


```python
"""Implementation of selection sort by passing in an array"""
def selection_sort(array):
  n = len(array)

  for i in range(n):
    # Assume the current index holds the min value
    min_index = i

    # Check the rest of the array for a smaller value and store the index of the value
    for j in range(i, n):
      if array[j] < array[min_index]:
        min_index = j

    #Swap the positions of the current [i] with the current smallest value
    array[i], array[min_index] = array[min_index], array[i]

  return array
```

The nested for loops will make the worst-case time complexity O(n^2) because if the array is sorted the opposite way, the program will have to go through the entire array n times to find the lowest value which is at the end of the array.

The space complexity of the algorithm is O(1) or constant because we are not creating a new dynamic memory space to hold values. At most we are allocating memory to store index values so that we can alter the original array by rearranging elements. This is why it is considered an in-place sorting algorithm because it sorts using the original array or memory space.

For stability, generally selection sort is not stable since it swaps non-adjacent elements. Given an array of [2, 2, 1, 3], it will swap the 2 at the 0 index to the 3rd index and the order of same items is already gone.

Even in my test file, the 2nd value in the tuple array is being sorted to show how the items move around 

```
.[(2, 'a'), (2, 'b'), (1, 'c'), (1, 'a')]
[(1, 'a'), (2, 'b'), (1, 'c'), (2, 'a')]
[(1, 'a'), (1, 'c'), (2, 'b'), (2, 'a')]
[(1, 'a'), (1, 'c'), (2, 'a'), (2, 'b')]
```

. The original (1, c), (1, a) order isn't saved.