Objective: Implementation of Selection Sort

Selection Sort

Selection sort is an algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

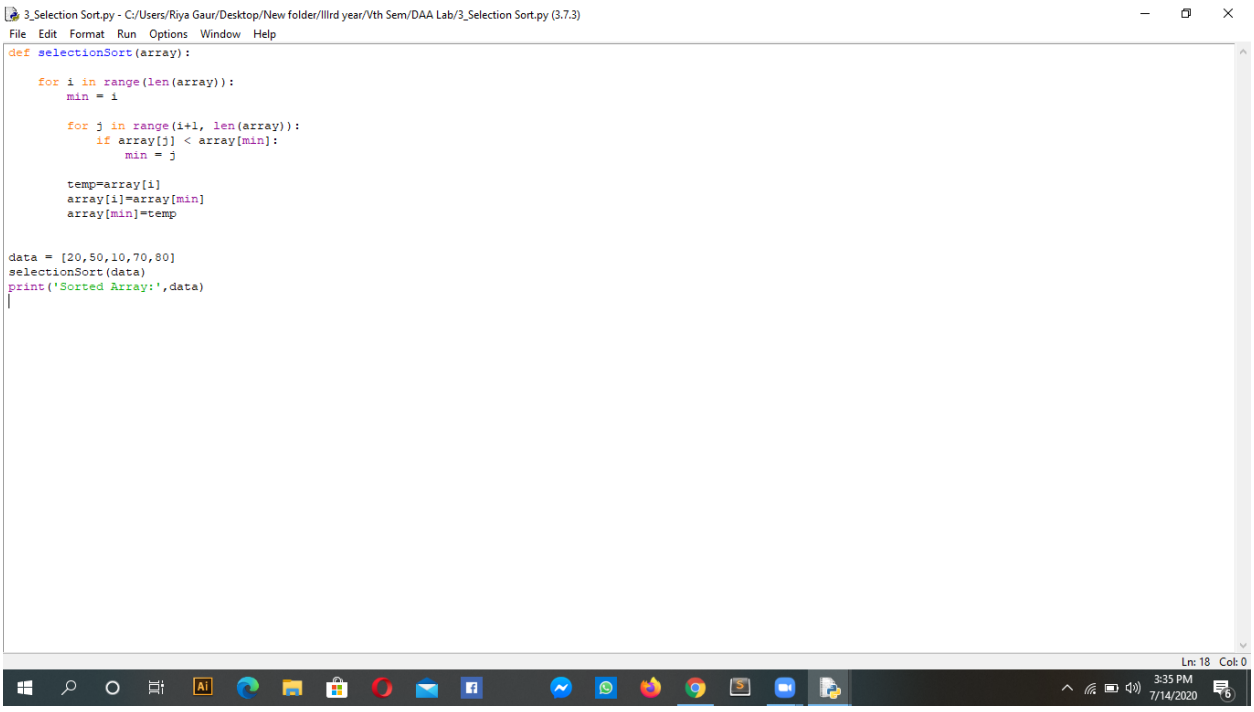
Selection Sort Algorithm:

```
selectionSort(array, size)
  repeat (size - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selectionSort
```

Code:

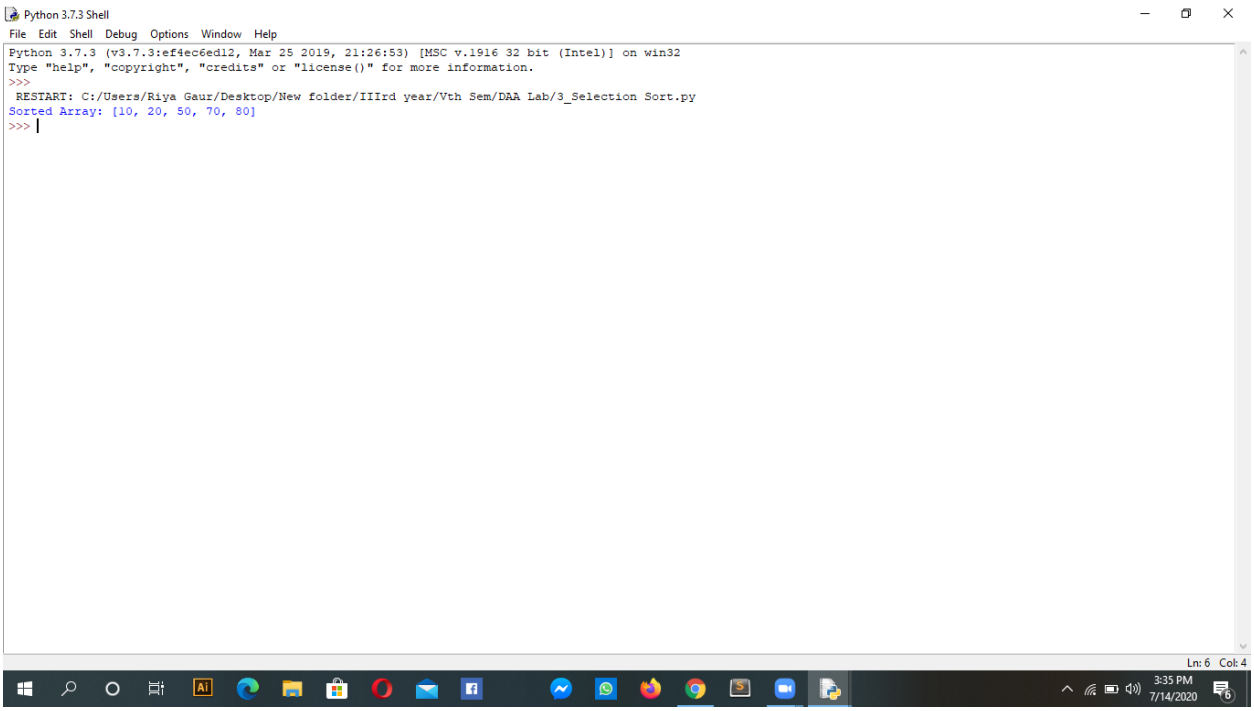
```
def selectionSort(array):

    for i in range(len(array)):
        min = i

        for j in range(i+1, len(array)):
```

```
        if array[j] < array[min]:
            min = j


    temp=array[i]
    array[i]=array[min]
    array[min]=temp


data = [20,50,10,70,80]
selectionSort(data)
print('Sorted Array:',data)
```



Output:

Sorted Array: [10, 20, 50, 70, 80]



Complexity Analysis:

Number of comparisons: `(n - 1) + (n - 2) + (n - 3) + ..... + 1 = n(n - 1) / 2` equals to n^2.

We can analyze complexity by simply observing the number of loops. There are 2 loops so the complexity is `n*n = n`^{2}.

Time Complexities:

- **Worst Case Complexity:** $O(n^2)$

  If we want to sort in ascending order and the array is in descending order then, the worst case occurs.

- **Best Case Complexity:** $O(n^2)$

  It occurs when the array is already sorted

- **Average Case Complexity:** $O(n^2)$

  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

Selection Sort Applications

The selection sort is used when:

- a small list is to be sorted
- cost of swapping does not matter
- checking of all the elements is compulsory
- cost of writing to a memory matters like in flash memory (number of writes/swaps is $O(n)$ as compared to $O(n^2)$ of bubble sort)