Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1. The sub-array which is already sorted.
2. Remaining sub-array which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted sub-array.

arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 **12** 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 **22** 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 **25** 64

Time Complexity: O(n2) as there are two nested loops.

Auxiliary Space: O(1)
The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.

The default implementation is not stable. However it can be made stable.

Selection sort makes O(n) swaps which is minimum among all sorting algorithms

ie Heap,selection,insertion,merge.

```
Unstable Selection Sort:
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    A[i], A[min_idx] = A[min_idx], A[i]



==========================================================================
==
==========================================================================
==
Stable Selection Sort

def stableSelectionSort(a, n):

    # Traverse through all array elements
    for i in range(n):

        # Find the minimum element in remaining
        # unsorted array
        min_idx = i
        for j in range(i + 1, n):
            if a[min_idx] > a[j]:
                min_idx = j

        # Move minimum element at current i
        key = a[min_idx]
        while min_idx > i:
            a[min_idx] = a[min_idx - 1]
            min_idx -= 1
        a[i] = key
```