AOA Experiment 01 (A).

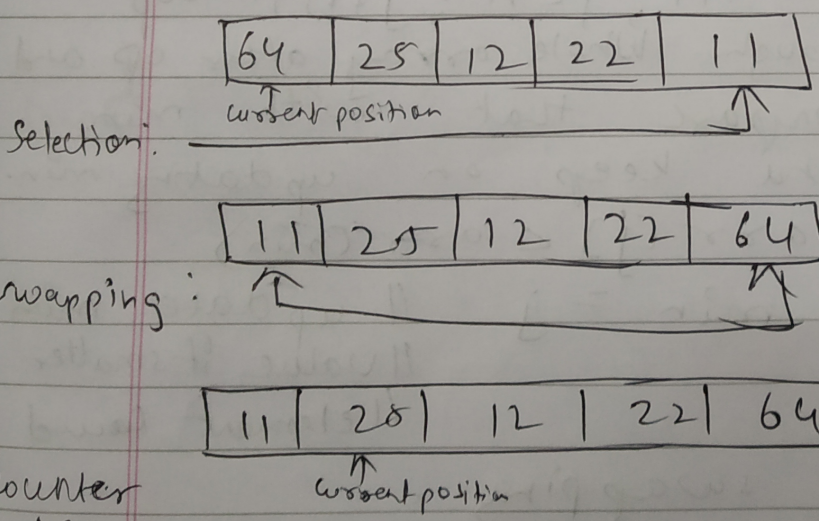
Selection Sort.

Logic:

1. Selection of smallest element.
2. Swapping of current element & smallest.
3. Shifting counter.

Algorithm (Theory):

We will find out the smallest element in the array by looping through every element; then we will swap the current position we are on (0) and swap the smallest element with it.
Then we will increase our position and keep on going till second last as last one is already sorted.



Keep on repeating these steps till cp is (n-1) i.e till 64!

Algorithm (code) :

```
void selectionsort(){
    int i, j, min;
    for(i=0; i<(n-1); i++){
        min = i;
        for(j=i+1; j<n; j++){
            if(arr[min] > arr[j])
                min = j
        }
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}
```

Selection : `min = i // select cp as min.`

```
for(j=i+1; j<n; j++){ // loop
// through whole array after cp and
// compare that with min
// and keep on updating min.
if(arr[j] < arr(min)
    min = j   // update min
              // value if smaller
              // element found
}
```

Swapping : Basic swapping.

Counter shift : `for(i=0; i<(n-1); i++)`

↑ Because last element will already be in sorted order.

Sum :

Q. | 7 | 4 | 10 | 8 | 3 | 1 |

n = 6

A. | 7 | 4 | 10 | 8 | 3 | 1 |

sorted i Current pointer unsorted array

loop 1 | 1 | 4 | 10 | 8 | 3 | 7 |

sorted cp unsorted

loop 2 | 1 | 3 | 10 | 8 | 4 | 7 |

sorted cp unsorted

loop 3 | 1 | 3 | 4 | 8 | 10 | 7 |

sorted cp unsorted

loop 4 | 1 | 3 | 4 | 7 | 10 | 8 |

sorted cp unsorted

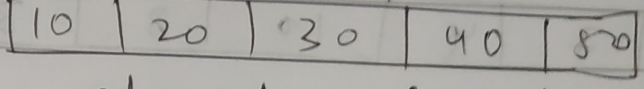
loop 5 | 1 | 3 | 4 | 7 | 8 | 10 |

sorted unsorted

(i < n - 1)
(i < (6 - 1))
hence we stop here.

| 1 | 3 | 4 | 7 | 8 | 10 |

Analysis :

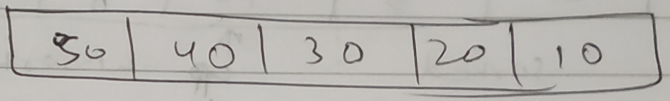
Best case : 

Array is already sorted,
But it will still loop (n-1) times and check (compares) min with remaining elements all the time.
So $1 + 2 + 3 + 4 \ldots (n-1)$.

$$\frac{n(n-1)}{2} = O(n^2)$$

$O(n^2)$ Comparisions
$O(1)$ swaps

Swapping Complexity $= O(1)$

Worst case : 

Same in Worst Case as well.
Only here it swaps as well.
So Time Complexity is $O(n^2)$
But swapping Complexity becomes $O(n)$ here.

Average = $O(n^2)$ (Comparision), $O(n)$ (Swaps)

Conclusion :

SS is not stable by default.
SS is in place.
↓ does not need another array.
↓ can change position of two non-distinct elements.