Insertion Sort:

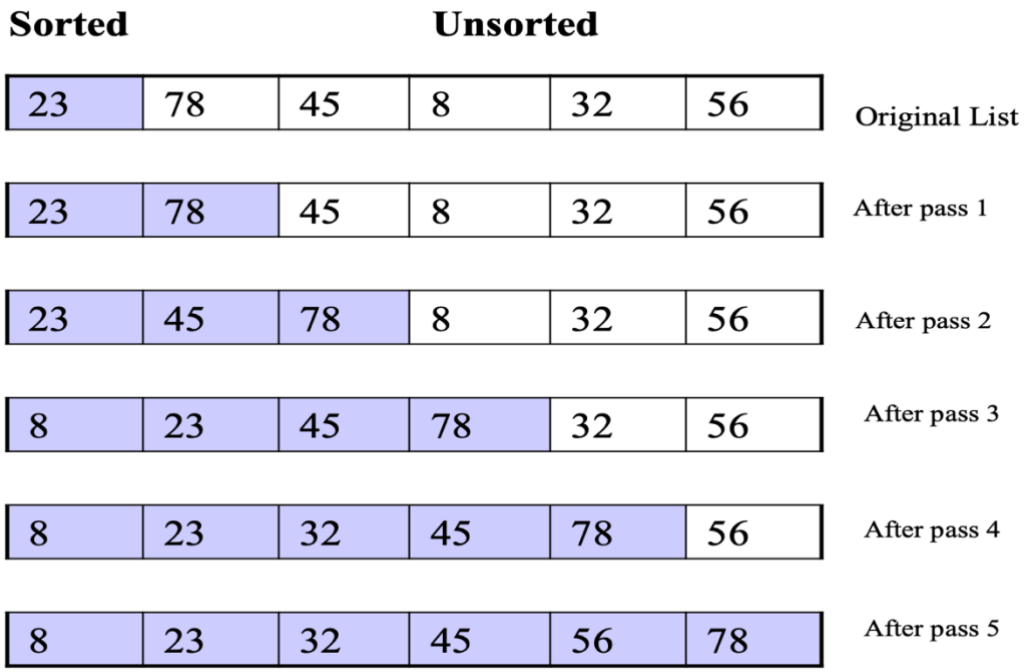
(NOTE: More efficient than bubble sort and is appropriate for small inputs)

An insertion sort compares values in turn, starting with the second value in the list. If this value is greater than the value to the left of it, no changes are made. Otherwise this value is repeatedly moved left until it meets a value that is less than it. The sort process then starts again with the next value. This continues until the end of the list is reached.

Accordingly, the list is polarized into two lists, unsorted and sorted.

In each pass, the first element in the unsorted list is picked up and transferred to the sorted list at the appropriate place.

At most, a list of n elements will take n-1 passes.



Algorithm Python Code Snippet:

```
def insertion_sort(arr):
    for i in range(1, len(arr), 1):
        value = arr[i]
        pos = i
        for j in range(i-1, -1, -1):
            if arr[j] > value:
                arr[j+1] = arr[j]
                pos = j
            else:
                break
        arr[pos] = value
    return arr
```

Time Complexity:

$$O(n^2)$$

Algorithm Complexity Explained:

Not only does the runtime depend on the size of the array, it also depends on the contents of the array.

Best Case →Sorted Array

Inner loop will not be executed.

Number of moves: 2*(n-1) $\boldsymbol{O(n)}$

Number of comparisons: n-1 $\boldsymbol{O(n)}$

<u>Worst Case →Reversed Array</u>

Inner loop → Executed i-1 times

Number of moves: 2*(n-1) + (1+2+...+n-1) = 2 * (n-1) + n*(n-1)/2

$\boldsymbol{O(n^2)}$

Number of comparisons: 1+2+...+n-1 = n*(n-1)/2 $\boldsymbol{O(n^2)}$

<u>Average Case → $\boldsymbol{O(n^2)}$</u>