SelectionSort can be used when the elements are unsorted.

TimeComplexity = O(n^2)
Space Complexity = O(1)
Swap = O(n)

Selection Sort gives best performance when you are using smaller set of array (size <= 50)
This would never happens in real time.
Another draw back of Selection Sort is, it takes O(n^2) time complexity even the elements in array are sorted.

Insertion Sort :

i = 1{1,2, 3,5}

Inner loop.
Do sorting only when currentIndex - 1 value is greater than current element.
Then repeats the process.

i = 2 {1,2, 3,5}

i = 3 {1,2,3,5}

Insertion Sort :

Insertion sort gives you best time complexity O(n) when the elements are sorted.

Insertion sort divides the given array into two subsets . Sorted and Unsorted.

Ex :

i = 1 {1,2, 3,5} Here current integration is 1 so insertion sort make sure that left part of the current iteration to be sorted.

{1,2 ....}

Insertion Sort do the sorting only when the left part element is greater than current interaction element so that when the array is already sorted inner loop always executes in constant time.

For a sorted array overall time complexity => O(n)

If the array is not sorted then the insertion sort would give the time complexity as => O(n^2)

Insertion Sort :

i = 1 {7,4,5,2}
=> inner loop
{7,4 ...}
=> {4,7,...}

i = 2 { 4,7,5,2}
=> inner loop
{ 4,7,5 ....}
J = i-1
j = 1
while (j>=0 && arr[j] > 5)

{4,5,7,....}

i=3. { 4,5,7,2}

=> inner loop
{ 4,5,7,2}
J = i-1

while (j>=0 && arr[j] > 2)

j = 2 { 4,5,2,7}

j =1 {4,2,5,7}

j=0 {2,4,5,7}

TimeComplexity :
Sorted Array => O(n)
Unsorted Array => O(n^2)
Space Complexity => O(1)
Stability => Stable
Comparison Sort =>Yes
Swap => O(n^2)
NonRecursive
Internal Sort

If the input array has all unsorted elements then insertion sort is not recommended. It's not only takes time complexity O(n^2) even the swap happens n^2 times.