When sorting an array in reverse order, insertion sort typically runs faster than selection sort. Here's the breakdown:

Selection Sort

- **Comparisons:** Selection sort scans through the entire unsorted portion of the array to find the smallest element, performing approximately
$$\frac{n(n-1)}{2}$$
comparisons for an array of size n.
- **Swaps:** Selection sort performs only one swap per element to place the minimum element in its correct position. This means the number of swaps is linear, i.e., $O(n)$.
- **Total Time Complexity:** Regardless of the input order, selection sort performs $O(n^2)$ comparisons and $O(n)$ swaps.

Insertion Sort

- **Comparisons and Shifts:** Insertion sort works by taking elements from the unsorted part and inserting them into their correct position in the sorted portion. For an array sorted in reverse order, every element needs to be compared to all previously sorted elements and shifted. This results in $O(n^2)$ comparisons and shifts.
- **Total Time Complexity:** In the worst case (reverse order), insertion sort also performs $O(n^2)$ comparisons and shifts.

Comparison for Reverse-Ordered Arrays

- **Data Movement:** While both algorithms have $O(n^2)$ time complexity in the worst case (reverse-ordered array), insertion sort often runs faster in practice because:
  1. Insertion sort tends to perform fewer operations overall when the cost of comparisons and data movements are taken into account.
  2. The total number of shifts in insertion sort (moving elements by one position) is generally faster than performing selection sort's repeated scans and swaps.
- **Selection sort's performance is not influenced by the initial order of elements**, meaning it performs the same number of comparisons regardless. Insertion sort, however, benefits from better initial order, but in reverse order, it still does fewer data-intensive operations compared to selection sort.

Conclusion

Insertion sort typically runs faster than selection sort for a reverse-ordered array, despite both algorithms having the same worst-case time complexity of $O(n^2)$. The reason is that insertion sort tends to be more efficient with data movement, making it faster in practice.