The time complexity of \*\*Selection Sort\*\* is \( O(n^2) \) in both the best and worst cases. Let's break down why this is the case by analyzing the algorithm step by step.

### How Selection Sort Works:

1. \*\*Basic Idea\*\*:

- The algorithm repeatedly selects the smallest (or largest) element from the unsorted part of the array and swaps it with the first unsorted element.

- After each iteration, the sorted part of the array grows by one element, and the unsorted part shrinks.

### Example of Selection Sort (Array: [5, 3, 8, 4, 2]):

1. \*\*First Pass (i = 0)\*\*:

- Find the minimum element from the whole array `[5, 3, 8, 4, 2]` → 2.

- Swap `2` with the first element `5`.

- Array after first pass: `[2, 3, 8, 4, 5]`.

2. \*\*Second Pass (i = 1)\*\*:

- Find the minimum element from the remaining unsorted part `[3, 8, 4, 5]` → 3.

- No swap needed because 3 is already in the correct position.

- Array remains `[2, 3, 8, 4, 5]`.

3. \*\*Third Pass (i = 2)\*\*:

- Find the minimum element from the remaining unsorted part `[8, 4, 5]` → 4.

- Swap `4` with `8`.

- Array after third pass: `[2, 3, 4, 8, 5]`.

4. \*\*Fourth Pass (i = 3)\*\*:

- Find the minimum element from the remaining unsorted part `[8, 5]` → 5.

- Swap `5` with `8`.

- Array after fourth pass: `[2, 3, 4, 5, 8]`.

5. \*\*Fifth Pass (i = 4)\*\*:

- Only one element remains `[8]`, so the array is now sorted.

### Why the Time Complexity is \( O(n^2) \):

Let's look at the number of comparisons made at each step.

#### Number of Comparisons per Pass:

1. In the first pass (i = 0), the algorithm looks at all \( n \) elements to find the smallest one.

2. In the second pass (i = 1), it looks at \( n - 1 \) elements.

3. In the third pass (i = 2), it looks at \( n - 2 \) elements.

4. This continues until the last pass where it looks at just \( 1 \) element.

The total number of comparisons is the sum of:

\[

(n) + (n - 1) + (n - 2) + \dots + 1

\]

This is a well-known mathematical series, which sums up to:

\[

\text{Total Comparisons} = \frac{n(n - 1)}{2}

\]

This simplifies to \( O(n^2) \), as the highest-order term is \( n^2 \), and constants like \( \frac{1}{2} \) are ignored in Big-O notation.

#### Key Points:

- \*\*Best Case\*\*: Even if the array is already sorted, Selection Sort still has to scan the entire unsorted portion of the array to find the minimum in each pass, resulting in \( O(n^2) \) comparisons.

- \*\*Worst Case\*\*: If the array is completely unsorted, the same number of comparisons occur, as Selection Sort doesn’t benefit from any pre-existing order in the array.

#### Swaps:

- The number of swaps is always \( O(n) \), as the algorithm swaps at most one element per pass. However, since comparisons dominate the time complexity, the overall time complexity remains \( O(n^2) \).

### Summary:

Selection Sort performs \( O(n^2) \) comparisons, regardless of the initial order of the array, because it scans through the entire unsorted part of the array in each iteration to find the minimum element. Therefore, its time complexity is \( O(n^2) \).