**2.Describe the Selection Sort algorithm's process of sorting an array. Selection Sort works by dividing the array into a sorted and an unsorted region. Initially, the sorted region is empty, and the unsorted region contains all elements. The algorithm repeatedly selects the smallest element from the unsorted region and swaps it with the leftmost unsorted element, then moves the boundary of the sorted region one element to the right. Explain why Selection Sort is simple to understand and implement but is inefficient for large datasets. Provide examples to illustrate step-by-step how Selection Sort rearranges the elements into ascending order, ensuring clarity in your explanation of the algorithm's mechanics and effectiveness.**

**Sorting a Random Array:**

**Input: [5, 2, 9, 1, 5, 6]**

**Output: [1, 2, 5, 5, 6, 9]**

**Sorting a Reverse Sorted Array:**

**Input: [10, 8, 6, 4, 2]**

**Output: [2, 4, 6, 8, 10]**

**Sorting an Already Sorted Array:**

**Input: [1, 2, 3, 4, 5]**

**Output: [1, 2, 3, 4, 5]**

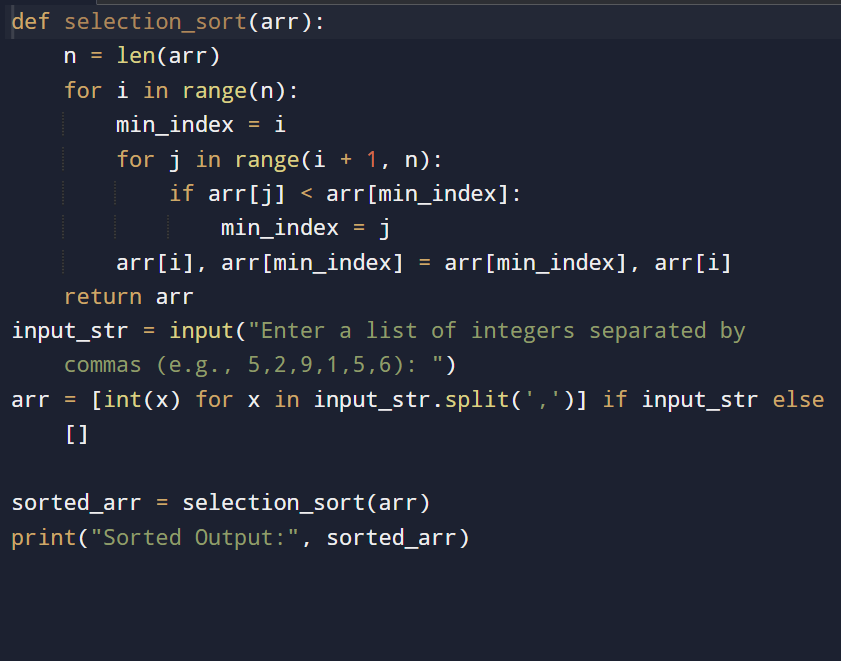
**Aim:**

To understand and implement the Selection Sort algorithm, observe how it works step-by-step on different types of arrays, and analyze its performance and simplicity.

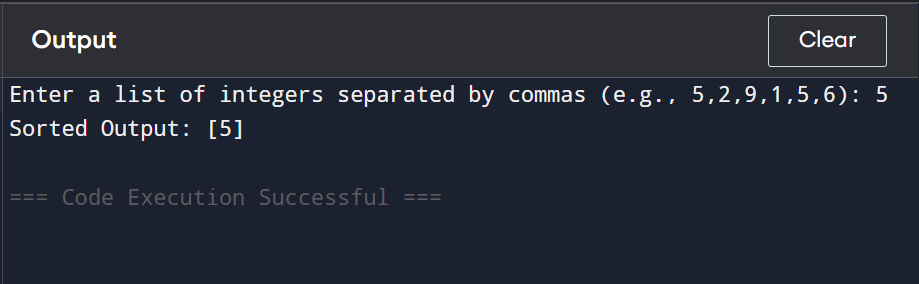
**Algorithm:**

1. Start from the beginning of the array.
2. For each position i in the array:
   * Assume the minimum element is at position i.
   * Traverse the rest of the array (from i+1 to end) to find the actual minimum element.
   * Swap the minimum found with the element at position i.
3. Repeat until the whole array is sorted.

**Code:**

****

**Input and output:**

****

**Result: given Selection Sort Algorithm: Process, Explanation, and Step-by-Step Examples is executed successfully and output is verified**

**Performance analysis:**

**Time Complexity:**

* **Best Case: O(n²)**
* **Average Case: O(n²)**
* **Worst Case: O(n²)**

**Space Complexity:**

* **O(1) — in-place sorting, no extra space used**