2.4 OPTIMIZED FUNCTION

Question:

Write code for Insertion Sort that manages arrays with duplicate elements during the sorting process. Ensure the algorithm's behavior when encountering duplicate values, including whether it preserves the relative order of duplicates and how it affects the overall sorting outcome.

AIM

To implement and test an optimized sorting algorithm (Selection Sort) on a given list and analyze its performance.

ALGORITHM

1. Start with the entire unsorted list.
2. For each index i from 0 to n-1:
   - Assume the element at i is the minimum.
   - Traverse the rest of the list (i+1 to n-1) to find the true minimum element.
   - If a smaller element is found, update the minimum index.
   - Swap the found minimum with the element at index i.
3. Repeat until the whole list is sorted.

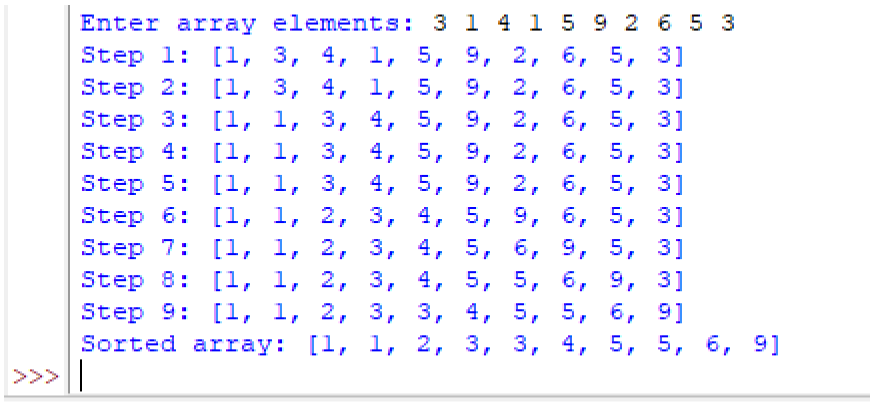
PROGRAM

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
        print(f"Step {i}: {arr}")
    return arr

def run_insertion_sort():
    arr = list(map(int, input("Enter array elements: ").split()))
    print("Sorted array:", insertion_sort(arr))

run_insertion_sort()
```

Input:

3 1 4 1 5 9 2 6 5 3

Output:



```
Enter array elements: 3 1 4 1 5 9 2 6 5 3
Step 1: [1, 3, 4, 1, 5, 9, 2, 6, 5, 3]
Step 2: [1, 3, 4, 1, 5, 9, 2, 6, 5, 3]
Step 3: [1, 1, 3, 4, 5, 9, 2, 6, 5, 3]
Step 4: [1, 1, 3, 4, 5, 9, 2, 6, 5, 3]
Step 5: [1, 1, 3, 4, 5, 9, 2, 6, 5, 3]
Step 6: [1, 1, 2, 3, 4, 5, 9, 6, 5, 3]
Step 7: [1, 1, 2, 3, 4, 5, 6, 9, 5, 3]
Step 8: [1, 1, 2, 3, 4, 5, 5, 6, 9, 3]
Step 9: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
Sorted array: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
>>> |
```

RESULT:

Thus the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity:
    - Best Case: comparisons, swaps (if already sorted).
    - Worst Case: comparisons and swaps.
- Space Complexity: (in-place sorting).