3.5 QUICK SORT IMPLEMENTATION WITH STEP-BY-STEP PARTITIONING

Question:

Given an unsorted array 7 43 86 23 1 8 9. Write a program to perform Quick Sort. Choose the first element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted.

AIM

To implement Quick Sort using the first element as the pivot and trace the sorting process step-by-step.

ALGORITHM

1. Initialize a counter missing = 0 to count missing numbers.
2. Traverse natural numbers starting from 1.
   - If the current number exists in arr, skip it.
   - Otherwise, increase missing.
3. Stop when missing == k.
4. Return the current number.

PROGRAM

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    left = [x for x in arr[1:] if x < pivot]
    right = [x for x in arr[1:] if x >= pivot]
    print("Pivot:", pivot, "| Left:", left, "| Right:", right)
    return quick_sort(left) + [pivot] + quick_sort(right)

def run_quick_sort():
    N = int(input("Enter number of elements: "))
    arr = list(map(int, input("Enter array elements: ").split()))
    print("Sorted array:", quick_sort(arr))

run_quick_sort()
```

Inp0ut:

8

76 4 9 23 76 98 34 21

Output:

```
Enter number of elements: 8
Enter array elements: 76 4 9 23 76 98 34 21
Pivot: 76 | Left: [4, 9, 23, 34, 21] | Right: [76, 98]
Pivot: 4 | Left: [] | Right: [9, 23, 34, 21]
Pivot: 9 | Left: [] | Right: [23, 34, 21]
Pivot: 23 | Left: [21] | Right: [34]
Pivot: 76 | Left: [] | Right: [98]
Sorted array: [4, 9, 21, 23, 34, 76, 76, 98]
>>>
```

RESULT:

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

PERFORMANCE ANALYSIS:

- Time Complexity: O(n log n)
- Space Complexity: O(log n)