3.6 QUICK SORT IMPLEMENTATION – MIDDLE ELEMENT AS PIVOT

Question:

Implement the Quick Sort algorithm in a programming language of your choice. Choose the middle 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. Execute your code and show the sorted array.

AIM

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

ALGORITHM

1. Select the middle element of the array as the pivot.
2. Partition the array into:
    - Elements less than the pivot → Left sub-array
    - Elements greater than or equal to the pivot → Right sub-array
3. Recursively apply Quick Sort to both sub-arrays.
4. Display the array after each partition and recursive call.

PROGRAM

```
def quick_sort_mid(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    print("Pivot:", pivot, "| Left:", left, "| Right:", right)
    return quick_sort_mid(left) + middle + quick_sort_mid(right)

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

run_quick_sort_mid()
```

Input:

[76,43,12,8,90,43]

Output:

```
Enter number of elements: 6
Enter array elements: 76 43 12 8 90 43
Pivot: 8 | Left: [] | Right: [76, 43, 12, 90, 43]
Pivot: 12 | Left: [] | Right: [76, 43, 90, 43]
Pivot: 90 | Left: [76, 43, 43] | Right: []
Pivot: 43 | Left: [] | Right: [76]
Sorted array: [8, 12, 43, 43, 76, 90]
>>>
```

RESULT:

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

PERFORMANCE ANALYSIS:

- Time Complexity:
  - Average: O(n log n), Worst: $O(n^2)$
- Space Complexity:
  - O(log n) (due to recursion).