Quick Sort Implementation in C

Objective: - Learn the Quick Sort sorting algorithm. - Understand the divide-and-conquer technique. - Implement recursive sorting in C.

Algorithm:

1. **Partition Operation:**
    - Choose the leftmost element as pivot (LOC).
    - Initialize `LEFT = BEG` and `RIGHT = END`.
    - Move `RIGHT` leftwards until an element smaller than pivot is found.
    - Swap pivot and `RIGHT` if needed.
    - Move `LEFT` rightwards until an element larger than pivot is found.
    - Swap pivot and `LEFT` if needed.
    - Repeat until `LOC == LEFT` or `LOC == RIGHT`.
    - Return pivot index `LOC`.
2. **Quick Sort Operation:**
    - If `BEG < END`:
        - Partition the array and get pivot index.
        - Recursively call `quickSort` on left sub-array (BEG to PIVOT-1).
        - Recursively call `quickSort` on right sub-array (PIVOT+1 to END).

Program Code:

```c
#include <stdio.h>
void swap(int A[], int i, int j) {
    int temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
int partition(int A[], int BEG, int END) {
    int LEFT = BEG;
    int RIGHT = END;
    int LOC = LEFT;
    int done = 0;

    while (!done) {
        while (A[LOC] <= A[RIGHT] && LOC != RIGHT) {
            RIGHT = RIGHT - 1;
        }

        if (LOC == RIGHT) {
            done = 1;
        } else if (A[LOC] > A[RIGHT]) {
            swap(A, LOC, RIGHT);
            LOC = RIGHT;
            while (A[LOC] >= A[LEFT] && LOC != LEFT) {
                LEFT = LEFT + 1;
            }

            if (LOC == LEFT) {
                done = 1;
            } else if (A[LEFT] > A[LOC]) {
                swap(A, LOC, LEFT);
                LOC = LEFT;
            }
        }
    }
    return LOC;
}
void quickSort(int A[], int BEG, int END) {
    if (BEG < END) {
        int PIVOT = partition(A, BEG, END);
        quickSort(A, BEG, PIVOT - 1);
        quickSort(A, PIVOT + 1, END);
    }
}
void printArray(int A[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", A[i]);
    }
    printf("\n");
}
int main() {
    int A[] = {25, 12, 64, 10, 22, 11, 90};
    int n = sizeof(A) / sizeof(A[0]);

    printf("Original array:\n");
    printArray(A, n);

    quickSort(A, 0, n - 1);

    printf("Sorted array:\n");
    printArray(A, n);

    return 0;
}
```

Sample Output:

```
Original array:
25 12 64 10 22 11 90
Sorted array:
10 11 12 22 25 64 90
```