SORTING

Write a C program to take n numbers and sort the numbers in ascending order. Try to implement the same using following sorting techniques.

1. Quick Sort

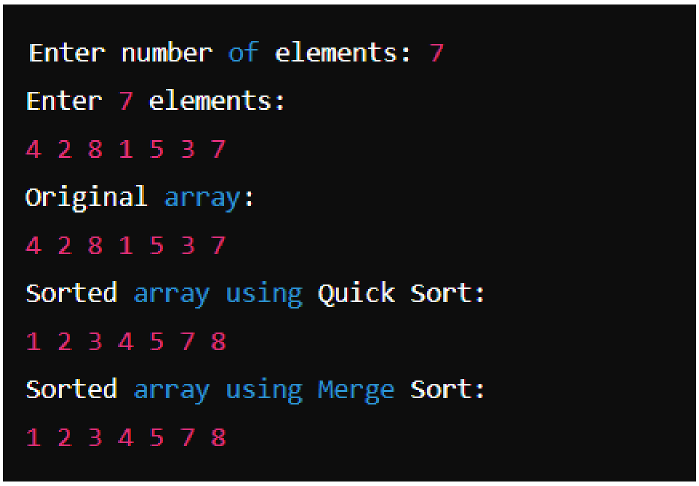
2. Merge Sort

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>


// Function to swap two elements

void swap(int* a, int* b) {

   int t = *a;

   *a = *b;

   *b = t;

}


// Function to perform partition for Quick Sort

int partition(int arr[], int low, int high) {

   int pivot = arr[high];    // pivot

   int i = (low - 1);  // Index of smaller element


   for (int j = low; j <= high - 1; j++) {

      // If current element is smaller than or equal to pivot

      if (arr[j] <= pivot) {

         i++;    // increment index of smaller element

         swap(&arr[i], &arr[j]);

      }

   }

   swap(&arr[i + 1], &arr[high]);

   return (i + 1);

}


// Function to implement Quick Sort
```

```
void quickSort(int arr[], int low, int high) {
   if (low < high) {
      // pi is partitioning index
      int pi = partition(arr, low, high);

      // Separately sort elements before partition and after partition
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}

// Function to merge two subarrays of arr[]
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;

   // Create temporary arrays
   int L[n1], R[n2];

   // Copy data to temporary arrays L[] and R[]
   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];

   // Merge the temporary arrays back into arr[l..r]
   i = 0;  // Initial index of first subarray
   j = 0;  // Initial index of second subarray
   k = l;  // Initial index of merged subarray
   while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
          arr[k] = L[i];
          i++;
      } else {
          arr[k] = R[j];
          j++;
      }
      k++;
   }

   // Copy the remaining elements of L[], if any
   while (i < n1) {
      arr[k] = L[i];
      i++;
      k++;
   }

   // Copy the remaining elements of R[], if any
   while (j < n2) {
      arr[k] = R[j];
      j++;
      k++;
   }
}

// Function to implement Merge Sort
void mergeSort(int arr[], int l, int r) {
   if (l < r) {
      // Calculate mid point
      int m = l + (r - l) / 2;

      // Sort first and second halves
      mergeSort(arr, l, m);
      mergeSort(arr, m + 1, r);
        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

// Function to print an array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Main function to test the above sorting functions
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

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

    // Sort using Quick Sort
    quickSort(arr, 0, n - 1);
    printf("Sorted array using Quick Sort:\n");
    printArray(arr, n);

    // Sort using Merge Sort
    mergeSort(arr, 0, n - 1);
    printf("Sorted array using Merge Sort:\n");
    printArray(arr, n);


    return 0;
}
```

OUTPUT:



```
Enter number of elements: 7
Enter 7 elements:
4 2 8 1 5 3 7
Original array:
4 2 8 1 5 3 7
Sorted array using Quick Sort:
1 2 3 4 5 7 8
Sorted array using Merge Sort:
1 2 3 4 5 7 8
```