PRACTICAL - 3

AIM : To perform quick sort

PSEUDO CODE :

QuickSort (Array A, low, high)

Step 1: if low < high, then go to step 2, otherwise go to step 6
Step 2: Set pivot_index to the index returned by Partition(A, low, high)
Step 3: Call QuickSort(A, low, pivot_index - 1)
Step 4: Call QuickSort(A, pivot_index + 1, high)
Step 5: Exit

Partition (Array A, low, high)

Step 1: Set pivot_value to A[high]
Step 2: Set i to low - 1
Step 3: for j from low to high - 1, do steps 4-5
Step 4: if A[j] <= pivot_value, then increment i and swap A[i] and A[j]
Step 5: Swap A[i + 1] and A[high]
Step 6: Return i + 1

CODE :

```
#include <stdio.h>

// Function to swap two elements
void swap(int* a, int* b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

// Function to partition the array and return the pivot index
int partition(int arr[], int low, int high) {
  int pivot = arr[high];
  int i = low - 1;
```

```
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            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) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

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

int main() {

    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
```

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

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

    quickSort(arr, 0, n - 1);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

OUTPUT :

```
Rishita Chaubey - A2305221265

Enter the number of elements in the array: 4
Enter the elements of the array:
12
3
33
-1
Original array: 12 3 33 -1
Sorted array: -1 3 12 33
```

COMPLEXITY : O(nlog(n)) - for best and avg cases. O(n^2) for worst case.

PRACTICAL - 4

AIM : To perform merge sort

PSEUDO CODE :

MergeSort (Array A, low, high)

Step 1: if low < high, then go to step 2, otherwise go to step 6
Step 2: Set mid to (low + high) / 2
Step 3: Call MergeSort(A, low, mid)
Step 4: Call MergeSort(A, mid + 1, high)
Step 5: Call Merge(A, low, mid, high)
Step 6: Exit

Merge (Array A, low, mid, high)

Step 1: Set left_size to mid - low + 1
Step 2: Set right_size to high - mid
Step 3: Create two temporary arrays left_array and right_array with sizes left_size and right_size, respectively
Step 4: Copy elements from A[low] to A[mid] into left_array
Step 5: Copy elements from A[mid + 1] to A[high] into right_array
Step 6: Set i to 0, j to 0, and k to low
Step 7: while i < left_size and j < right_size, do steps 8-9
Step 8: if left_array[i] <= right_array[j], then set A[k] to left_array[i] and increment i
Step 9: otherwise, set A[k] to right_array[j] and increment j
Step 10: Copy the remaining elements of left_array and right_array, if any, into A
Step 11: Exit

CODE :

```
#include <stdio.h>


// Function to merge two sorted arrays
void merge(int arr[], int low, int mid, int high) {
  int left_size = mid - low + 1;
  int right_size = high - mid;
  int left_array[left_size], right_array[right_size];
```

```
    for (int i = 0; i < left_size; i++)
        left_array[i] = arr[low + i];
    for (int j = 0; j < right_size; j++)
        right_array[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = low;
    while (i < left_size && j < right_size) {
        if (left_array[i] <= right_array[j]) {
            arr[k] = left_array[i];
            i++;
        } else {
            arr[k] = right_array[j];
            j++;
        }
        k++;
    }

    while (i < left_size) {
        arr[k] = left_array[i];
        i++;
        k++;
    }

    while (j < right_size) {
        arr[k] = right_array[j];
        j++;
        k++;
    }
}

// Function to implement Merge Sort
void mergeSort(int arr[], int low, int high) {
    if (low < high) {
        int mid = low + (high - low) / 2;
```

```
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}

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

int main() {
    printf("Aabhas Kumar Jha - A2305221279\n\n");
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Original array: ");
    printArray(arr, n);
    mergeSort(arr, 0, n - 1);
    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

OUTPUT :

```
gcc P004.c && ./a.out
Rishita Chaubey - A2305221265

Enter the number of elements in the array: 6
Enter the elements of the array:
12
2
1
-2
22
0
Original array: 12 2 1 -2 22 0
Sorted array: -2 0 1 2 12 22
```

COMPLEXITY : O(nlog(n))