Lab 09 Quick and Merge Sort Implementation

Learning Outcomes:

After successfully completing this lab the students will be able to:

1. Understand the divide and conquer mechanism which is at the heart of the two recursive sorting algorithm.
2. Develop programming solutions for Merge Sort and Quick Sort
3. Empirically compare the performance of these two sorting algorithms

Pre-Lab Reading Task:

Quick Sort:

Quick sort is a widely used sorting algorithm developed by C. A. R. Hoare that makes **O(nlogn)** comparisons in the **average case** to sort an array of **n** elements. For the **worst case**, it has a quadratic running time given as **O(n2)**, however its efficient implementation can minimize the probability of requiring quadratic time. The major advantage of using Quick Sort is that it sorts the array ***'in-place'*.** This means that it does not require additional memory space to store data. Compare this to the Merge Sort algorithm which requires **O(n) additional memory** for sorting in **O(nlogn)** time.

Like merge sort, this algorithm works by using a divide-and-conquer strategy to divide a single unsorted array into two smaller sub-arrays. The quick sort algorithm works as follows:

Select an element pivot from the array elements.

Rearrange the elements in the array in such a way that all elements that are less than the pivot appear before the pivot and all elements greater than the pivot element come after it (equal values can go either way). After such a partitioning, the pivot is placed in its final position. This is called the **partition operation**.

Recursively sort the two sub-arrays thus obtained. (One with sub-list of values smaller than that of the pivot element and the other having higher value elements.)

Like merge sort, the base case of the recursion occurs when the array has zero or one element because in that case the array is already sorted.

After each iteration, one element (pivot) is always in its final position. Hence, with every iteration, there is one less element to be sorted in the array.

Thus, the main task is to find the pivot element, which will partition the array into two halves. To understand how we find the pivot element, follow the steps given below in **Algorithm 1**. (We take the last element in the array as pivot.)

```
Partition(A, p, r)
{
    x = A[r]              // x is pivot
    i = p - 1
    for j = p to r - 1
    {
        do if A[j] <= x
            then
            {
              i = i + 1
              exchange A[i] ↔ A[j]
            }
    }                          partition () runs in O(n) time
    exchange A[i+1] ↔ A[r]
    return i+1
}
```

Algorithm 1: *Partition function which chooses the last element as pivot*

The algorithm for the Quick Sort function with recursive calls to itself and the partition function is given below in **Algorithm 2.**

```
P: first element
r: last element    (Indices of)
Quicksort(A, p, r)
{
    if (p < r)
    {
        q = Partition(A, p, r)
        Quicksort(A, p   , q-1)
        Quicksort(A, q+1      , r)
    }
}
```

Algorithm 2*: The Quick Sort algorithm*

In-Lab Tasks

You are given skeleton code for this lab. Your task is to complete the merge() and partition() functions for Merge Sort and Quick Sort respectively.

Post Lab

Study and perform comparative analysis between different sorting algorithms we have implemented in current and previous Lab.

******** End of Document ********