Lab7: Sorting

In this lab, you will complete implementations of one simple sorting algorithms and two divide-and-conquer sorting methods. This is an individual assignment; you may not share code with other students.

Specification

The code contains three classes, one to implement the Insert Sort algorithm, one to implement the Merge Sort algorithm and one to implement the Quick Sort algorithm. All algorithms sort a collection of data stored in an array into non-decreasing order.

1. Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

   In the `InsertSort` class, there is a `sort` method that implements the Insert Sort algorithm. Test your method with the supplied `main` method.

2. In the Merge Sort algorithm, we split the array into two halves, sort the two halves recursively using Merge Sort, and then merge the two sorted arrays back together to form our final sorted result. When we merge the two sorted subarrays back into one, we examine the first element in each subarray to determine which value moves into the first position of the merged array. We then examine the first element that remains in each subarray to determine which value moves into the next position of the merged array. We repeat the previous step over and over until we run out of elements from one of the two subarrays. Once this happens, we move all of the remaining elements in the other subarray into the final positions of the merged array (since they're already in order).

   In the `MergeSort` class, there is a `sort` method that implements the steps of the Merge Sort algorithm. Read through this to see how it works. This method calls the `merge` helper method to merge the two sorted sub-arrays. Implement the `merge` method based on the merging algorithm described in class. Test your method with the supplied `main` method.

3. In the Quick Sort algorithm, we choose the first element in the array as the "pivot". We scan the array from the beginning for the first element that is larger than the pivot. Then we scan the array from the end for the first element that is smaller than the pivot. We then swap these two elements. We continue scanning in the array from each end for the next elements that are larger than the pivot (from the beginning of the array) and smaller than the pivot (from the end of the array), swapping these also. We repeat this process until we reach the same element from both ends of the array. Finally, we swap the pivot element with the last element that is smaller than the pivot. This partitions the array into two subarrays: all elements before the pivot are less than the pivot and all elements after the pivot are greater than the pivot. Recursively sort the two subarrays using the Quick Sort algorithm.

   In the `QuickSort` class, there is a `sort` method that implements the steps of the Quick Sort algorithm. Read through this to see how it works. This method calls the `partition` helper method to partition the array around the pivot element. Implement the `partition` method based on the partitioning algorithm described in class. Test your method using the supplied `main` method.

TEST

You can simply test your algorithm implementation in the main method of each class.

Besides, an additional test class is provided, you should use dataset with different sizes to test the implemented sorting algorithms and find the best sorting algorithm for different sizes.

HANDIN

Raise your hand to call for the teaching assistant and demonstrate your results.

Create a zip file named YourStudentID-YourName.zip that contains your code project and upload your zip file to the https://wss.pet/s/3zovqytdiug.