Task-1

This Python code defines a merge sort algorithm and uses it to sort a list of numbers read from an input file. The `merge` function takes two sorted lists, `left` and `right`, and merges them into a single sorted list. The `mergeSort` function recursively divides the input list (`arr`) into smaller halves until each sublist has one or zero elements, then merges them back together. The input numbers are read from "Input1.txt," sorted using merge sort, and the sorted result is written to "Output.txt." The file handling code opens and closes files for input and output, respectively.

Task-2

This Python code defines a function `find_max` that takes either two integers or two sorted lists as input and returns the maximum value. The function `find_max_sorted` recursively finds the maximum value in a sorted list using a divide-and-conquer approach. The input numbers are read from "Input2.txt," converted into a list, and the maximum value is found using the `find_max_sorted` function. The result is then written to "Output2.txt." The file handling code opens and closes files for input and output, respectively.

Task-3

This Python code implements an inversion count algorithm using a merge sort approach. The `merge` function merges two sorted arrays while counting inversions (instances where a larger element precedes a smaller one). The `count_inversions` function recursively divides the array into halves, counts inversions in each half, and then merges while counting inversions. The input numbers are read from "Input3.txt," and the inversion count is calculated using the `count_inversions` function. The result is then written to "Output3.txt." The file handling code opens and closes files for input and output, respectively.

Task-4

This Python code defines a function `find_ij_max` that takes an array and its size as input, finds the two maximum values (one negative and one non-negative) with specific conditions, and calculates a result based on their squares. The main loop iterates through the array, updating the maximum values accordingly. The input numbers are read from "Input4.txt," and the result of the function is written to "Output4.txt." The file handling code opens and closes files for input and output, respectively.

Task-5

This Python code implements the quicksort algorithm to sort an array of numbers. The `partition` function selects a pivot element and rearranges the array elements so that elements smaller than the pivot are on the left and elements greater than the pivot are on the right. The `quick_sort` function recursively applies the partitioning process to sort the entire array. The input numbers are read from "Input5.txt," sorted using quicksort, and the sorted result is written to "Output5.txt." The file handling code opens and closes files for input and output, respectively.

Task-6

This Python code finds the kth smallest element in an array using the quickselect algorithm. The `partition` function selects a pivot element and rearranges the array elements. The `find_kth_smallest` function recursively narrows down the search space based on the partitioning until it finds the kth smallest element. Input numbers are read from "Input6.txt," and for each specified k value, the kth smallest element is found and written to "Output6.txt." The file handling code opens and closes files for input and output, respectively.