1.

This program implements a parallel quicksort algorithm using the Message Passing Interface (MPI) for distributed computing. The input is provided as an array of integers, where the example input array data[] = {4, 2, 8, 1, 5, 7, 3, 6, 9, 0} consists of ten elements. The program divides the input array into smaller chunks and distributes them among multiple processes using MPI. Each process performs quicksort on its assigned chunk of data. The sorted chunks are then merged iteratively until a single sorted array is obtained. In this case, the outcome of running the code would be the sorted array 0 1 2 3 4 5 6 7 8 9, which represents the elements of the input array in ascending order. This approach demonstrates the efficiency and scalability of parallel sorting with quicksort, leveraging the power of distributed computing using MPI.

2.

This code showcases the application of the fast inefficient ranking algorithm to partially sort a 4x4 character array. The input is represented by the arr variable, which holds the initial array. The program starts by displaying the initial array as the "Initial Array" output. Subsequently, the fast inefficient ranking algorithm is executed, involving multiple passes over the array to compare and swap adjacent elements within each row. The step-by-step process illustrates the comparisons and swaps performed during each pass. Finally, the "Sorted Array" is printed, presenting the partially sorted array as the resulting output. It is important to note that the fast inefficient ranking algorithm employed here is not an efficient sorting method, but rather serves as a demonstrative example to highlight sorting principles rather than emphasizing performance optimization.

Input:  
{'I', 'H', 'Y', 'P'},

{'S', 'B', 'R', 'P'},

{'X', 'T', 'W', 'J'},

{'Q', 'D', 'A', 'B'}

the partially sorted array as the resulting output:

{'H', 'I', 'P', 'Y'},

{'B', 'P', 'R', 'S'},

{'J', 'T', 'W', 'X'},

{'A', 'B', 'D', 'Q'}

4.  
This code demonstrates the pancake sorting algorithm, which is used to sort an array of numbers. The input array contains the values {54, 85, 52, 25, 98, 75, 25, 11, 68}. The pancakeSort function applies the sorting algorithm, which involves finding the largest number in the array and flipping the array multiple times to move the largest number to its correct position. This process is repeated until the entire array is sorted. The code uses OpenMP, a parallel programming framework, to speed up the sorting process by executing some operations concurrently. Finally, the sorted array is printed to the console.