Task 1

The code here achieves its goal to sort through an integer list by at first putting the list through a 'merge_sort' function that divides the list into manageable halves which are then sorted and then merged back through the 'merge' function, which iterates through the lists, each integer being appended into a new list based on their numerical order until all elements are processed.

Task 2

The code finds the minimum value of an array by recursively dividing the array into halves until it reaches a base case from where it figures out maximum values by comparing between 'left' and 'right' until an overall maximum is found.

Task3

The code sorts out the taller cities from the smaller ones by using the the recursive function 'part' which recursively divides and conquers through the array by splitting them into halves. After which it compares between left and right and counts the number of inversions, which it returns.

Task4

The code finds the maximum value of $B[i]+B[j]^2$ in a large array by out first implementing a divide and conquer strategy by breaking the array into halves recursively. It then stores out the largest values of the halves, which it uses ~~to~~ in the function 'calc-sec' to find a secondary value, which is compared to a whole array of secondary values, and of which the maximum is returned.

Task 5

The code here is an implementation of the quicksort algorithm, although it is divided into two functions. First off, the quicksort function divides up the array into smaller subarrays until reaching base case, after which the partition function takes the elements around a chosen pivot, and swaps them - The quicksort function then recursively sorts the partitioned subarrays into a perfectly sorted array.

Task 6