19201089
Section: 9

Task 1 (a)

Here, two loops are run. In the outer for loop, we iterate over every element. The current element is set as num1. In the inner for loop, we iterate over every element after num1. The current element is set as num2. If sum is equal to num1 + num2, the file is written with their values included. The found variable is set to True to help out of the outer loop. If found == False after loop is completed, then 'Impossible' is written in the file. This means no two elements can make the sum.

Task 1 (b)

Here, we initialize a dictionary named 'needed'. It stores the required counterpart to make the sum and the value of this key is the index of the current element. So, if the element is in the needed dictionary during iteration, it means that a previous element exists that can make up the sum along with it. So, we get the previous element's index from the value and the current element's index is the ith index. They are written to the file. The loop is broken and found is set to True. If the element was not in needed, we just store its counterpart and its index in it. If found is False after loop ends, write "IMPOSSIBLE" in the file.

Task 2 (a)

The built in python sort function has time complexity O(nlogn), so we just joined the two arrays and then used the built in sort function.

Task 2 (b)

We run a for loop that runs for the total length of both arrays. Two pointers were initialized for two arrays. If the pointers are within bounds, we check ~~if the~~ which ~~first~~ array's element is larger. It is appended to the arr3 array. That array's pointer is incremented. After finishing traversal, if any element is left out, they are inserted at the end of arr3 array. Then it is written to the file.

Task 3

In the merge sort algorithm, there are two functions - mergeSort and merge. mergeSort is called to run the algorithm. For length 1 arrays, it returns the array. It calculates the middle index of the array and recursively divides the array into left and right parts. Then merge function is called on them. The merge function initializes two pointers for the two parts and checks which one is larger. This element is appended to the merged array. Any left out elements are appended to the end of the array. The merged array is returned. Recursively, the whole array is sorted

Task 4

This is a modification of the merge sort algorithm. Here, in mergeSort, a new value maxValue is introduced that stores the returned maximum value of each of the subarrays. The merge function has a new parameter maxValue that checks to see if between left side's element, right side's element and maxValue which is the largest. If left side's element is smaller than right sides, its pointer increases. otherwise, right side's does. maxValue is updated to the max of the maxValue and left side's element / right side's element (depends on which is larger). Then the two subarrays are joined and returned along with maxValue In the end, we get the maximum number that's written to the file.