Task-1(a)

This code reads data from two files, “Input1(a).txt” and “Output1(a).txt.” It first opens the input file for reading and the output file for writing. It then reads a line from the input file, splits it into two variables, x and y. Next, it reads a list of integers from the input file and stores them in list_1.

The code then proceeds to find pairs of elements in list_1 whose sum equals y. If such a pair is found, it records the indices of these elements and writes them to the output file. It uses nested loops to compare elements in the list and breaks out of the inner loop once a valid pair is found.

Finally, the code closes both the input and output files. It appears to be solving a problem related to finding pairs of numbers in a list that add up to a specific target value and then saving the indices of these pairs to the output file.

Task-1(b)

This code defines a function `find_sum_pair` that takes three arguments: `n` (an integer), `s` (an integer), and `arr` (a list of integers). The function's purpose is to find a pair of indices in the `arr` list such that the elements at those indices sum up to `s`. It uses a set called `num_set` to keep track of numbers encountered in the list while iterating through it. If it finds a complement in `num_set` (i.e., an element whose sum with the current element equals `s`), it returns the indices of those elements. If no such pair is found, it returns "IMPOSSIBLE."

The code then opens an input file "Input1(b).txt," reads `n` and `s` from the first line, and reads the list of integers `arr` from the second line.

It calls the `find_sum_pair` function with the given inputs and stores the result in the variable `result`. Finally, it opens an output file "Output1(b).txt" and writes the result (either the indices of the pair or "IMPOSSIBLE") to the file.

This code appears to be solving a problem related to finding a pair of numbers in a list that add up to a specific target value and then saving the indices of these numbers in the output file.

Task-2(a)

This code defines a function `merge_sorted_lists_nlogn` that merges two sorted lists, `alice_list` and `bob_list`, into a single sorted list `merged_list` with a time complexity of O(n). It initializes empty lists for the merged result and indices for both input lists. Then, it iterates through both lists, comparing elements, and appends the smaller element to `merged_list` until one of the input lists is exhausted. Finally, it appends the remaining elements from both lists to `merged_list` and returns the sorted merged list.

Task-2(b)

This code defines two functions, `merge_sorted_lists_nlogn` and `merge_sorted_lists_n`, for merging two sorted lists, `alice_list` and `bob_list`, into a single merged list.

1. `merge_sorted_lists_nlogn` merges the lists in O(n log n) time by iterating through both lists and comparing elements, appending the smaller one to the result list until one of the input lists is exhausted.

2. `merge_sorted_lists_n` merges the lists in O(n) time by iterating through the lists in reverse order, starting from the end of each list and placing the larger element into the merged list, effectively merging them in reverse order.

The code then opens an input file "Input2.txt.txt," reads the lengths and elements of `alice_list` and `bob_list`, and calls both merging functions to obtain `merged_list_nlogn` and `merged_list_n`.

Finally, it prints the merged lists for both O(n log n) and O(n) complexity, demonstrating two different approaches for merging sorted lists.

Task-3

This code reads data from "Input3.txt," where the first line specifies the number of tasks `x`, and subsequent lines represent task intervals. It defines a function `tasks` that takes a list of task intervals and returns non-overlapping tasks based on their end times.

The code then reads the task intervals, processes them using the `tasks` function to obtain a list of completed non-overlapping tasks, and writes the count of completed tasks and their details to "Output3.txt." It ensures that tasks are sorted by their end times, and only non-overlapping tasks are considered.

Task-4

This code reads data from "Input4.txt," where the first line contains two integers `n` and `p`. It then reads `n` lines, each containing a pair of integers, and stores them in `list_2`.

The code proceeds to sort `list_2` based on the first integer in each pair. It then initializes variables `c1` and `l3` and enters a loop to select `p` non-overlapping pairs from `list_2`. It iterates through `n` times, and for each iteration, it selects a non-overlapping pair, updates `c1`, and appends the selected pair to `l3`.

Finally, the code writes the value of `c1` to "Output4.txt," representing the count of non-overlapping pairs within the given constraints.