

## Explanations

### 1a

The code defines a function 'func' that takes the length of the list 'N', the target sum 'S', and the list of integers 'arr' as inputs. It iterates through each pair of distinct positions in the list using nested loops. If the sum of the numbers at the current pair of positions is equal to the target sum, it returns the positions. If no such pair is found, it returns 'IMPOSSIBLE'.

### 1b

The code keeps track of the numbers it has come across so far using a dictionary 'dic'. It cycles through the list of numbers, computing the complement for each number ( $S$  less the current number). ~~for~~ The presence of the complement in the dictionary indicates that we have discovered two integers that add up to  $S$ . The complements locations and the current number are then returned. We return 'IMPOSSIBLE' if the loop ends up being intractable. Because we only iterate through

the list of integers once, the time complexity of this solution is  $O(N)$ .

**2a**

This program defines a function called 'func' that uses the merge sort algo to combine two sorted lists into a single sorted list. The items are compared when we iterate through both lists at once, adding the smaller element to the combined list. The remaining components for each list are then added to the combined list. The merge sort algo makes sure that the final sorted list is produced in  $O(n \log n)$  time, where  $n$  is the sum of the entries in both lists.

**2b**

It iterates across both lists concurrently, comparing elements and adding them in ascending order to the combined list. It adds the remaining components of the other list to the merged list after

using up all the elements in one of the lists. The output file is then written with the final merged list. This algo has an  $O(n)$  time complexity, where  $n$  is the sum of the elements in the two input lists.

3

The ~~merge~~ 'mergeSort' function separates the input list recursively until each sublist includes just one element, then merges them back together in sorted orders. Then 'merge' function joins two sorted lists together.

4

This algorithm's time complexity, where  $N$  is the list's length, is  $O(N \log N)$ . This is due to the list's repeated division into halves and comparison at each level of the highest value from each half. The recursion has  $\log N$  levels, and since we perform a fixed amount of work at each level, the overall time com. is  $O(N \log N)$ .