HACKER RANK PROBLEMS	Description	Solution
Common Child (LCS)	A string is said to be a child of a another string if it can be formed by deleting 0 or more characters from the other string. Letters cannot be rearranged. Given two strings of equal length, what's the longest string that can be constructed such that it is a child of both?	**Longest Common Subsequence** Algorithm using dynamic programming/ memoization. If S[i]==S2[j] ? Matrix[i][j] = Matrix[i-1][j-1] : Matrix[i][j]= max (Matrix[i-1][j], Matrix[i][j-1]. Return Matrix[s1.length()-1][s2.length()-1], where there s1 and s2 are padded with " " because we need a padded row and column initialized to zero to initialize the matrix for dyanamic programming.
Two Strings	**Finds a Common Letter**, actually. Returns YES if there is a common Substring, else NO. A string can be a single letter.	Letters in range a-z,TWO STRINGS O(N) only returns YES if a letter occurs in both strings Traverse each letter in both arrays and when a letter occurs in both strings, we stop and return YES, else we return NO. **Uses s. find_first_of(ch) != std::string::npos for each alphabet letter from a..z for both strings, returning true when a letter is found in both**
Array Manipulation	**Maintain the result in a matrix array** to avoid repeating an operation. Starting with a 1-indexed array of zeros and a list of operations, for each operation add a value to each the array element between two given indices, inclusive. Once all operations have been performed, return the maximum value in the array. Example n=10, queries = a b k 1 5 3 4 8 7 6 9 1 largest value is 10	Maintain a Cumulative operations Array. perform Operation on first element, reverse the operation after the last. ie : Add on first element, and subtract after last element. Returns the greatest value in the manipulation array after all operations. Instead of performing repeated operation on each array entry in range, perform the addition on the first member of the array, and subtraction on the element past the last member of the array. Finally we track a running total on each operation in the n- array!
Minimum Swaps 2	It's a kind of **Insertion Sort**. You are given an unordered array consisting of consecutive integers [1, 2, 3, ..., n] without any duplicates. You are allowed to swap any two elements. Find the minimum number of swaps required to sort the array in ascending order.	For simplicity, i inserted a zero to make the array -1-based. For each number i, from 1..n, j = find(i), j is the position of the number i in the array. while arr[j] != j, swapcontents (arr[arr[j]],arr[j]); swap++ .. track the swaps. when we can't swap any more, check that the next i is in its proper postion. return swaps.
Minimum Bribes	**Bubble Sort!** Determine the minimum number of bribes that took place to get to a given queue order. Print the number of bribes, or, if anyone has bribed more than two people, print Too chaotic.	First, make the vector into zero based for simplicity by inserting a zero. Starting from the back, a person may have been bribed to the end, but nobody will be more than two ahead. For each position, starting from the back, its person is either in place, or requires either one or two swaps ahead to be put in its place. Go through the line and track the swaps. if it is more than two ahead, exit with "Too Chaotic"