Q.4

Brute force Algorithm: It is a sorting technique which will go through all possible solutions extensively without taking anything in consideration. It is a simplest way to explore the space of solutions. In other terms we call brute force as exhaustive search technique.

Implementation Method:

1. Firstly, with the help of Scanner, we are taking the input from user and store it in the str String variable.
2. Pass the string to "longestRepeatedSubSeq" function which will in turn call SubsetsRest method.
3. This method is a Recursion function which will help to generate combinations and subsets, after generating first subset it will call the 'brute' function and it will go on for all combinations.
4. In brute, we will loop over combination generated as outer loop with an inner loop to traverse and find the repeated occurrence of that character. i.e. first char compared with other with the help of inner for loop.
5. While working with inner for loop, we will check for conditions to check it whether this comparison is possible or not. The conditions are as follows:
    - Inner loop which traverse on input string should not execute the same index number which is present in generated combination sequence.
    - We have created an index array where it will store all the index used, so that not to be use again for that sequence, if that index is matched.
    - This index array will be used to check if the current loop index, either of combination sequence or present in user input sequence then do nothing and just increment the inner loop else proceed with inner string matching.
6. After this condition, we will compare both positions, once above conditions are fulfilled and also keep count into consideration such as
    a. We will compare inner and outer loop current position char and count to check whether outer loop has already matched with the inner. If yes, then just increment the inner index without performing any action as char is already matched with other combination index and we don't want to overlap indexes.
7. Now if char matched and count is zero, then add it to temp, count to 1, add that matched inner loop index to index variable, so that we cannot use that again.
8. Finally, result variable length will be checked with temp, if result length is less, than replace with temp variable value and lastly, print the result variable as Invalid or matched char value output.

Dynamic Programming: Dynamic programming (usually referred to as DP) is a very powerful technique to solve a particular class of problems. It demands very elegant formulation of the approach and simple thinking

Implementation Method:

1. Firstly, with the help of Scanner, we are taking the input from user and store it in the str String variable.
2. Call "longestRepeatedSubSeq" function and pass the input string with it/
3. Create 2d matrix with a size of n+1 and value to 0.
4. As Efficient DP uses bottom up approach, Hence, we will keep all element 0 if any of the x or y position in the matrix is 0. Because, while traversing from bottom, we can reach any of these indexes which will help to state that we have found the combination and we don't have to process it further.
5. After that, now in loop, we will compare i&j position index to not to be same. If strings matched, increase the count in the matrix by 1 which will help us to know that we found a match. To again not use the same ith to any of the element further, we have put flag which will help to check the condition, where it says if strings matched

and flag is not set then add 1 to [i][j-1] int, and if set or strings doesn't match then just take the max of [i-1][j],[i][j-1].

6. After Generating the matrix, traverse in bottom up fashion, it follows it ways as:
    a. Compare the current i&j position to i-1&j-1 and add 1 to it, if they both gets equal, that means pair has been added over here, hence get the position value and find the charAt position to that index in input string, and append that char to res. Also, reduce i&j by 1 to move upwards diagonally.
    b. If without adding 1 we get same, then traverse upwards instead of left, as traversing in left with all elements won't be of useful purpose (unnecessary condition check).
    c. If both of them did not work, then shift to left, maybe in the left and its above diagonal, we can find the way to go up quickly.
    d. This way you will reach till 0th value position, which will notify that you have got the maximum repeated sequence pair out of this string.

Q5.

As after executing both the algorithms and looking straight out of its complexity, we can find out that dynamic programming works better even string size increases.

The time complexity for brute force algorithm is O(2^n) whereas time complexity for dynamic programming is O(N^2).

While, space complexity for brute force algorithm is O(1) and space complexity for dynamic programming is O(N^2).

To get a proof, we looked out for time complexity with different cases for the above 2 algorithms, and we got the result as follows:

Test case	Brute	Dynamic	Test case	Brute	Dynamic
ATACTCGGA	40ms	28ms	Aac	27ms	26ms
Babb	29ms	26ms	Aaabbb	30ms	26ms

We can find out, that once the number of characters increase, brute performs badly and even more badly if there are distinct characters, while dynamic is performing consistent, as because dynamic traverse mostly diagonally to find the positions where value has been incremented. As dynamic algorithm with memorization technique, it keeps the result in case future computation may use it, hence it requires less of computation, but as it saves some of the value, it requires more memory than brute force technique, but its good to compensate with, as huge memory is available, but time does not. The reason basically, brute performs all the combinations and traverse through whole string each time whereas, dynamic finds the increment position and move diagonally upwards till it finds the 0th element.

Q6.

Reference: https://stackoverflow.com/questions/8103050/what-exactly-is-the-brute-force-algorithm

http://www.techiedelight.com/longest-repeated-subsequence-problem/

https://rosettacode.org/wiki/Longest_common_subsequence

https://www.codechef.com/wiki/tutorial-dynamic-programming

https://stackoverflow.com/questions/13467674/determining-complexity-for-recursive-functions-big-o-notation