Assignment-2

Ans-1)

a) This is a direct implementation of the stated question all possible triplets are made in $O(n^3)$ then their indices are found in O(n), thus having a total

Time Complexity - $O(n^4)$

Space Complexity - O(n)

b) The question is rephrased so that we don't have to find the indices of elements individually thus saving for a loop and having:-

Time Complexity - $O(n^3)$

Space Complexity - O(n)

c) Subsequence containing all elements from $(i+1)^{th}$ element to the $(n-1)^{th}$ element is made and it is checked if it is sorted in ascending order. Checking takes O(n) and this is done for all n elements.

Time Complexity - $O(n^2)$

Space Complexity - O(n)

d) To find if there exists b[i]<b[k]<b[j], for i<j<k, Here we check for all k, iterating through the array forwards. For a given k, a disordered tuple exists only if there exists number b[i], which is the minimum from all 0 to k-1. And b[j], which the number which is just greater than b[i] in the index range from k+1 to n-1. Finding b[i] for a given k is simple as we have to just maintain minimum upto than index.

To find b[j] we use a visited array initially all marked false, and is marked true for each element traversed through the array. We maintain a variable named **up** which b[j]. Now, if up!=b[k] and b[k]>mn Then there exists some number which is in between mn and b[k], i.e.

up. **Thus, 0 is returned**. In this case, the new value of up is found by incrementing the up variable using a loop till **up** is no longer covered in the traversed section of the array.

When mn>b[k]. New value of mn is set to b[k] and also the value of **up** is to be updated (if possible) to a number from b[k] to mn, which is not yet traversed. Using the visited array.

Time Complexity - O(n)

Space Complexity - O(n)

Ans-2) This is a direct example of an **unbounded knapsack** and **memorization** is used. Here the states are **n** and **m.** The logic is we can take any element and if its preference number is less than the current number of crafts we can either take it or and reduce the number of craft left in the following recursion or do not take the element and just proceed to the next element. We take a dp array initialized to -1, and so as to prevent recounting of **overlapping substructures** and previously calculated value is stored in the 2D-dp and is value output whenever the same states are called again.

Also, it is given in the question that each person must buy > 0 crafts, so we initially gave each some the preference number of crafts initially and the value of added to the profit calculated recursively. The constraint so that each person gets at least > 0 craft will be:-

Constraint: (n > Sum of all preference numbers)

Time Complexity - O(n*m)

Space Complexity - O(n*m)